Independent study report

Study of Traffic Engineering Algorithms and Framework

Submitted by,

Shyam Subramanian.

Table Of Contents

Independent Study – Report

1. Introduction

This report discusses the study on various Traffic Engineering algorithms and protocols. The task of mapping traffic flows onto an existing physical topology is called *Traffic Engineering* (TE). Traffic engineering is used to balance the traffic load on the various links, routers, and switches in the network. Traffic Engineering is needed in the Internet mainly because current *Internet Gateway Protocols* (IGP) always use the shortest paths to forward traffic. Using shortest paths conserves network resources, but it causes some resources of the network to be over utilized while the others remain under utilized.

The purpose of traffic engineering is to optimize resource efficiency and network utilization. Network planning is to improve the architecture (topology and link capacity) of a network in a systematic way so that the network is robust, adaptive and easy to operate. Network optimization is to control the mapping and distribution of traffic over the existing network infrastructure to avoid and/or relieve congestion, to assure satisfactory service delivery, and to optimize resource efficiency. This report discusses about the network utilization.

In order to do Traffic Engineering effectively, the *Internet Engineering Task Force* (IETF) introduced *Multi Protocol Label Switching* (MPLS), Constraint-based Routing and an enhanced link state Interior Gateway Protocol.

The report is organized as follows. Section 1 gives a brief introduction. Section 2 introduces the essential background materials needed for traffic engineering. Section 3 discusses the classical traffic engineering algorithms. Section 4 gives a brief overlook on the network simulators. Section 5 shows the simulation results. Section 6 concludes the report. After the conclusion the reference and appendix are given.

2. Background Literature

2.1 Multi-Protocol Label Switching Networks (MPLS)

MPLS is an advanced forwarding scheme. It extends routing with respect to packet forwarding and path controlling [1]. Packet forwarding is a process by which the data

packets are switched from the source to the destination node in a network, while path controlling controls the path taken by each packet.

In MPLS a label identifies each packet. In an *Asynchronous Transfer Mode* (ATM)[18] this label is inserted in the *Virtual Path Identifier* (VPI)[18] and *Virtual Control Identifier* (VCI)[18] fields. In all other environments the label is present in a header called the Shim Header. The Shim header is 32 bits in length. It resides between the *layer 2*[1] (L_2) header and *layer 3* (L_3) header. The shim header consists of

- *Label.* Label value, 20 bits in length. This value contains the MPLS label.
- *Exp,* Experimental use, 3-bits in length. This fields is not yet fully defined.
- *S,* Stacking bit, 1-bits in length. Used to indicate stacking of multiple labels and
- *TTL,* TTL field, 8-bits in length. Places a limit on how many hops the MPLS packet can be forwarded.

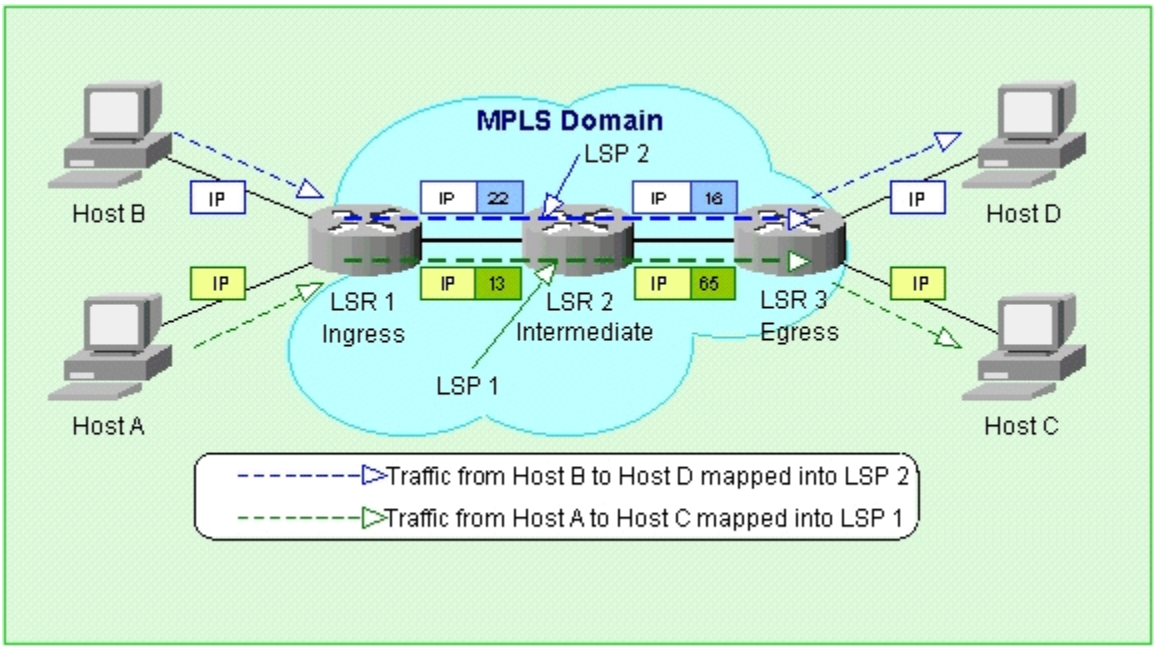

Figure 1.1 MPLS

Figure 1.1 depicts a MPLS domain. There are 4 hosts and 3 routers, which switch packets between them. Routers in MPLS domain are referred to as *Label Switched Routers* (LSR). The first LSR a packet encounters in a MPLS domain is called the *ingress LSR*,

[1] Layer 2 is also called the datalink layer and Layer 3 is also called the network layer

while the last LSR is known as the *egress LSR*. At the ingress LSR, IP packets are classified based on a combination of the information carried in the IP header of the packets and the local routing information maintained by the LSR and a label is assigned to them. The assigned label is then distributed to the neighboring LSR in the MPLS domain. The neighboring LSR then associates a label with this incoming label and distributes it to its neighbors. This process is known as label distribution and it ends at the egress LSR. The incoming label and corresponding outgoing labels are stored in a table, known as the *forwarding table*.

At the ingress LSR a label is then inserted into each packet. Within an MPLS-capable domain, each LSR uses the label to forward the packet. At each LSR (except the egress LSR) the outgoing label replaces the incoming label and the packet is switched to the next LSR. The process of switching the label is known as *Label Swapping.* At the egress LSR the label is removed and the packet is forwarded to the next domain. This whole process is showed in Fig. 1. The path between the ingress LSR and the egress LSR is referred to as *label switched paths* (LSP). MPLS uses some signaling protocol like *Resource reservation protocol* (RSVP) [2][3], *Label Distribution protocol* (LDP) [3] or *Constraint based LDP* (CR-LDP) [4] to set up a LSP and distribute the label.

2.2 RFC and Internet Draft

This section deals with TE related RFC's, Internet drafts and papers that address traffic engineering issues in MPLS.

2.3.1. RFC 2702 – MPLS Traffic Engineering.

RFC 2702[5] is exclusively focused on the Traffic Engineering applications of MPLS. Specifically, the goal of this document is to highlight the issues and requirements for Traffic Engineering in a large internet backbone. A description of the basic capabilities and functionality required of an MPLS implementation to accommodate the requirements is also presented.

RFC discusses the basic functions of Traffic Engineering in the Internet, provides an overview of the traffic Engineering potentials of MPLS (essentially background material). It presents an overview of the fundamental requirements for Traffic Engineering over MPLS, describes the desirable attributes and characteristics of traffic

trunks which are pertinent to Traffic Engineering. It also presents a set of attributes, which can be associated with resources to constrain the routability of traffic trunks and LSPs through them. It also advocates the introduction of a "constraint-based routing" framework in MPLS domains.

2.3.2. Internet Draft – OSPF-TE[6].

This draft specifies a method of adding traffic engineering capabilities to *Open Shortest Path First* (OSPF)[7]. The extensions provide a way of describing the traffic engineering topology (including bandwidth and administrative constraints).

The information made available by these extensions can be used to build an extended link state database. The extended link state database (referred to below as the traffic engineering database) has additional link attributes.

This extension makes use of the Opaque *Link State Advertisements* (LSA)[8]. One new LSA is defined, the Traffic Engineering LSA. This LSA describes routers, point-to-point links, and connections to multiaccess networks (similar to a Router LSA).

2.3.3. CR-LDP Vs RSVP[12]

CR-LDP and RSVP are the two protocols used to set up constraint based LSP. This section compares the signaling procedures of the CR-LDP, RSVP and E-RSVP[12]. E-RSVP is the extension proposed to RSVP to support differentiated service for MPLS networks.

Protocol / Categories	CR-LDP	RSVP	E-RSVP
Protocol Objective	Created to enable Lsp setup for reliable end-to-end differentiated service in MPLS networks	Established to support soft state resource reservation of integrated services of IP networks	Proposed with modifications to support differentiated services with RSVP for MPLS networks.
Network Positioning	Designed for carrier backbone networks	Designed for edge and host services	Revised design for backbone networks
Differentiated Services	Supported	Not Supported	Supported
Routing types	Strict, loose, pinned	Strict, loose, no pinning	Strict, loose, pinned

Scalability	Good	Poor	Marginal
User Security	Low	Low	Low
LSP State	Hard	Soft	Soft
LSP State Refresh	None	Periodic, all nodes	Periodic, all nodes
Resource Request	By sending LER	By Receiving LER	By Receiving LER
LSP Setup Action	Forward Downstream	Backwards Upstream	Backwards Upstream
LSP Architecture	Sink Tree	Source Tree	Source Tree
LSP Failure Detection	Reliable	Unreliable	Unreliable
LSP Failure Recovery	Local and Global	Local and Global	Local and Global
Failure Recovery Traffic	Low	High, all nodes	High, all nodes
Multipoint LSP merging	Yes	Yes	Yes
Multicasting LSP setup	No	Yes	No
Loop Prevention	Yes	Yes	Yes
Path Rerouting	Yes	Yes	Yes
Path Preemption	Yes	Yes, but not reliable	Yes, but not reliable

TABLE 1 – A COMPARISON OF CR-LDP, RSVP AND E-RSVP.

3. Traffic Engineering Algorithms

This section discusses classical the traffic engineering algorithms and framework taken from various recently published papers.

3.1. Profile Based Routing

This is an algorithm and framework for dynamic routing of bandwidth guaranteed flows. The main idea of the algorithm is to use the "traffic profile" of the network obtained by measurements or service level agreements[2] (SLA), as a rough predictor of the future traffic distribution. This algorithm has two phases

1. Preprocessing phase and
2. Online routing phase.

The algorithm uses the "traffic profile" in the preprocessing step (one multi-commodity flow computation) to determine certain bandwidth allocation on the links of the network. The online phase of the routing algorithm then routes the request using a "shortest path" (Shortest Path Forwarding) like algorithm but with the additional information given by the preprocessing phase. The multi-commodity preprocessing phase allows the online algorithm to exercise admission control by rejecting some request because of their blocking effects in the network.

The performance of this algorithm is compared with the shortest path[9][10], widest path[9][10] and minimum interference routing algorithm (MIRA)[9][11]. The details of the algorithm and the simulation results can be found in [9]

3.2. On Path Selection for Traffic with Bandwidth Guarantees

In the paper, "*On Path Selection for Traffic with Bandwidth Guarantees"* the authors, "*Qingming Ma, Peter Steenkist,*", present a systematic evaluation of the four routing algorithms that offers different tradeoffs between limiting the path hop count and balancing the network load. The four routing algorithms discussed in this paper are

1. *Widest-Shortest path:* a path with the minimum hop count among all the feasible paths. If there are several such paths, the one with the maximum reservable bandwidth is selected.
2. *Shortest-widest path:* a path with the maximum bandwidth among all feasible paths. If there are several such paths one with the minimum hop count is selected.
3. *Shortest-distance path:* a feasible path with the shortest distance. The distance function is defined by

$$\text{Dist}(p) = \sum_{j=1}^{k} 1/R_{I_j}$$

where, R_{I_j} is the bandwidth available on link I_j,

I_j is the link identifier, j=1 to k (k=number of links).

4. *Dynamic-alternative routing:* Let n be the hop count of a minimum-hop path when the network is idle. A dynamic alternative path is a widest-shortest path with no more than n+1 hops

[2] SLA is a means by which the service provider assures a certain type of service to the user.

The results from the simulation shows that a routing algorithm that gives preference to limiting hop count performs better when the network load is heavy, while an algorithm that gives preference to balancing network load performs slightly better when the network load is light. The details of the algorithm and simulation results can be found in [10]

3.3. Minimum Interference Routing with Application to MPLS Traffic Engineering(MIRA)

This is an online algorithm, which does not have any priori knowledge of the traffic requests. It just routes request as they come in. This algorithm chooses a path for the request that causes minimum interference for the future potential LSP. In other words a path that maximizes the minimum open capacity between every other ingress-egress pairs (Potential nodes where flow can generate and leave the MPLS domain) is chosen.

In other words this algorithm identifies the "critical links" – these are the links with the property that whenever an LSP is routed over those links the max flow values of one or more ingress-egress pairs decreases.

A detailed mathematical formulation of the algorithm and simulation results can be found in [11]. This algorithm is compared with widest-shortest path[9][11][13] and the Minimum hop algorithm[11].

3.4. QOS routing as a tool of MPLS Traffic Engineering

The paper, "*QOS routing as a tool of MPLS Traffic Engineering*", by "*Karol Kowalik and Martin Collier*", discusses basic algorithms such as the Widest Shortest Path (WSP) and more complicated ones such as the MIRA [11][13] and Profile-Based Routing [9][13]. Then a new QoS routing algorithm for Differentiated-Service [19] flows in MPLS networks is proposed.

The algorithm has two phases the online and the offline phase. In the offline phase of the algorithm the network is modeled as a weighed graph. Then LSPs are setup for different class of traffic based on the weighed graph. In the online algorithm, as the request for flow arrives they are accommodated in the LSPs setup by the offline phase. If it is not possible to accommodate the flow in the predefined LSPs then a graph with modified link

costs is used and the flow is accommodated using Dijkstra's[20] or Belman-Ford algorithms[20]. A detailed analysis of the algorithm and simulation results can be found in [13].

3.5. RATES: A Server for MPLS Traffic Engineering

The article, "*RATES: A Server for MPLS Traffic Engineering*", by *Aukia P, Kodialam M, Koppol P.V.N, Lakshman T.V, Sarin H, Suter, B* [14], describes a software system called the Routing and Traffic Engineering Server (RATES) developed for MPLS traffic engineering. RATES is a software system built on a centralized paradigm.

The centralized server used the OSPF[7] or *Intermediate System to Intermediate System* (IS-IS)[21] with appropriate extensions to get the details of the topology. Then these details are distributed to the ingress-egress nodes as the request for traffic flow comes in.

The online routing algorithm in RATES is a shortest-path algorithm on an appropriately weighted graph. This algorithm takes in to account the location of the ingress-egress routers. This algorithm chooses the path that has minimum interference with other potential LSP flows. The simulation results can be found in [14].

3.6. Load Balancing Algorithms in MPLS Traffic Engineering

This section presents three load-balancing[3] algorithms for MPLS traffic engineering: Topology-Based Static Load Balancing Algorithm (TSLB), Resource-Based Static Load-Balancing Algorithm (RSLB) and Dynamic Load-Balancing Algorithm (DLB). The proposed algorithm is for unicast traffic with no priority.

TSLB is enhanced from Shortest-Path algorithm. In this algorithm, new traffic flow is routed through shortest-path first. If this path has insufficient resource then a longer route that has sufficient resource is selected. This algorithm greatly reduces the congestion in the shortest path. This algorithm has the following limitation [15].

1. Causes low utilization of network resources as this algorithm does not consider the future requests[15]

[3] Load balancing is a process by which the traffic is distributed through more than one path. The path need not be the shortest path.

2. The traffic appears in network in random, and each traffic flow's sending rate can be unstable, it can fluctuate in special range. This also causes low network utilization.

RSLB overcomes the first limitation of TSLB. When a new traffic flow arrives, ingress will select a route whose free capacity is nearest to the traffic flow's bandwidth request (but larger than the request). This algorithm reserves larger capacity route for future arriving traffic flow, which has larger bandwidth. This is still a static algorithm so it has the following limitation [15].

1. The traffic appears in network in random, and each traffic flow's sending rate can be unstable, it can fluctuate in special range. This causes low network utilization.

DLB is a dynamic algorithm and it overcomes all the limitations of TSLB and RSLB. This algorithm takes into account the network topology character and traffic bandwidth request simultaneously. Under light load condition, traffic flows can be mapped on the short and high capacity route; when load changes to heavy, small traffic flow can reroute to another appropriate route so as to reserve high capacity route for large traffic flow. This algorithm greatly enhances the network throughput and resource utilization. The complexity analysis of each algorithm and simulation result can be found in [15].

4. Introduction to Network Simulators

Generally there are two forms of network simulation

1. Analytical modeling
2. Computer simulation

The first is by mathematical analysis that characterizes a network as a set of equations. The mail disadvantage is its over simplistic view of the network and inability to simulate the dynamic nature of a network. Thus the study of complex design system always requires a discrete event simulation package, which can compute the time that would be associated with real events in real life situation. In this section we are going to discuss briefly about two network simulators

1. NS-2 and
2. *Optimized Network Engineering Tool* (OPNET)

4.1. NS-2

NS-2 is a network simulator used for simulating various networking scenarios. NS-2 uses two languages for the simulation.

1. OTcl scripting language and
2. C++.

Otcl is used for

- Configuration, setup, and ``one-time'' stuff.
- Manipulating existing C++ objects.

C ++ is used when

- Doing *anything* that requires processing each packet of a flow.
- Changing the behavior of an existing C++ class in ways that weren't anticipated.

The various functionality of ns is documented as ns-Manual. For detailed notes on ns refer [16]. MPLS is supported by NS-2 and it is documented separately [22]. In the following section gives the list of command used in *MPLS on NS* (MNS).

MNS Commands.

The list of command and a brief description of what each one does.

1. **$lsrJ get-module "MPLS"** – this command is used to attach the MPLS module with node lsrJ.
2. **$lsrJ aggregate-flows I K** – this command is to aggregate flows. It aggregates flow to node I from node J till node K.
3. **$lsrJ send-ldp-withdraw-msg I** – this command is used to withdraw the label that was distributed by node J to node I.
4. **$lsrJ send-crldp-release-msg lspid** – this command is used to release all labels associated with the erlsp, which has an id lspid.
5. **$lsrJ setup-crlsp fec er lspid Bandwidth BufferSize PacketSize SetupPriority HoldingPriority** – this command is used to setup a constrained-lsp (crlsp) from node 1 to node 3 through the erlsp 1_2_3. It attaches an id (lspid) along with this crlsp. The setup path has a bandwidth, buffer size, packet size, setup priority and holding priority associated with that.
6. **$ns configure-ldp-on-all-mpls-nodes** – Configure LDP agents on all the created MPLS nodes.
7. **$lsrJ enable-control-driven** – Let $lsrJ operate as control-driven trigger .

8. **$lsrJ enable-data-driven** – Let $lsrJ operate as data-driven trigger.
9. **$lsrJ enable-on-demand** – Let $lsrJ operate as on-demand-mode.
10. **$lsrJ enable-ordered-control** – Let $lsrJ operate as ordered-control-mode .
11. **$ns enable-control-driven** – Let all the created MPLS nodes operate as control-driven trigger.
12. **$ns enable-data-driven** – Let all the created MPLS nodes operate as data-driven trigger.
13. **$ns enable-on-demand** – Let all the created MPLS nodes operate as on-demand-mode.
14. **$ns enable-ordered-control** – Let all the created MPLS nodes operate as ordered-control-mode.

APIs for ER-LSP

1. **$lsrJ setup-erlsp I J_P_A_T_H_I lspid** – this command is used to setup an explicit route from node J to node I through nodes P,A,T,H and binds an id (lspid) with this erlsp (explicit routed label switched path).
2. **$lsrJ bind-flow-erlsp I priority lspid** – this command is used to bind a flow which is bound to node I with already established erlsp which has an id lspid.

APIs for CR-LSP and CR-LDP

1. **$ns cfg-cbq-for-SBTS $qlim $cbq_qtype $okborrow $bw $maxidle $extradelay** - Configure CBQ on all MPLS nodes so as to support Simple Best-effort Traffic Service.

   Configuration Parameters

   $qlim : The queue size in packets

   $cbq_qtype : The type of a Queue object into the compound CBQ

   $okborrow : A boolean indicating the class is permitted to borrow bandwidth from its parent

   $bw $maxidle : The maximum amount of time a class may be required to have its packets queued before they are permitted to be forwarded

   $extradelay : increase the delay experienced by a delayed class by the specified time

   For more information about configuration parameters refer [16][17].

2. **$ns cfg-cbq-for-HBTS $qlim $cbq_qtype $okborrow $bw $maxidle $extradelay** Configure CBQ on all MPLS nodes so as to support Higher priority Best-effort Traffic Service. The configuration parameters are the same as that of $ns cfg-cbq-for-SBTS.
3. **$ns cfg-cbq-for-STS $qlim $cbq_qtype $okborrow $bw $maxidle $extradelay** – Configure CBQ on all MPLS nodes so as to support Signalling Traffic Service. The configuration parameters are the same as that of $ns cfg-cbq-for-SBTS.
4. **$ns cfg-cbq-for-RTS $qlim $cbq_qtype $okborrow $bw $maxidle $extradelay -** Configure CBQ on all MPLS nodes so as to support Real-time Traffic Service. The configuration parameters are the same as that of $ns cfg-cbq-for-SBTS.
5. **$ns bind-flowid-to-SBTS $id1 [$id2] –** Cause packets containing flow id $id1 (or those in the range $id1 to $id2 inclusive) to be associated with SBTS service
6. **$ns bind-flowid-to-HBTS $id1 [$id2]** – Cause packets containing flow id $id1 (or those in the range $id1 to $id2 inclusive) to be associated with HBTS service
7. **$ns bind-flowid-to-STS $id1 [$id2]** – Cause packets containing flow id $id1 (or those in the range $id1 to $id2 inclusive) to be associated with STS service
8. **$ns bind-ldp-to-SBTS** – Cause LDP packets to be associated with SBTS service
9. **$ns bind-ldp-to-HBTS** – Cause LDP packets to be associated with HBTS service
10. **$ns bind-ldp-to-STS** – Cause LDP packets to be associated with STS service
11. **$MPLSnode setup-crlsp $fec $er $lspid $TRate $BSize $PSize $SPrio $HPrio** – Create an CR-LSP ofth which the FEC is $fec, the specified Explicit Route is $er, and the LSPID is $lspid. By this command, a CR-LDP Request message is sent toward the node with $fec along the $er.

    Configuration Parameter:

    $fec : The value of FEC

    $er : Explicit Route

    $lspid : The value of LSPID

    $TRate : The value of traffic rate

    $BSize : The value of buffer size

    $PSize : The value of packet size

    $SPrio : The value of setup priority

$HPrio : The value of holding priority

12. **$MPLSnode send-crldp-release-msg $lspid** – Send a CR-LDP Release message toward an upstream LSR in order to release the established ER-LSP/CR-LSP of which the LSPID is $lspid.
13. **$MPLSnode send-crldp-withdraw-msg $lspid** – Send a CR-LDP Withdraw message toward downstream LSRs in order to release the established ER-LSP/CR-LSP of which the LSPID is $lspid.
14. **$MPLSnode send-crldp-notification-msg $status $lspid $tr –** Send a CR-LDP Notification message toward upstream LSRs in order to give them a notification information.

    Configuration Parameter

    $status : The status information defined in CR-LDP standards

    $lspid : LSPID

    $tr : Traffic information

15. **$MPLSnode set-flow-prio $fec $flowid $priority** – let the traffic with FEC $fec and flow id $ flowid have priority $priority.

There are three call-back functions in MNS

1. proc notify-erlsp-setup {node lspid} – It is used to notify user that CR-LDP Mapping message with LSPID $lspid arrived at a node $node.
2. proc notify-erlsp-fail {node status lspid tr} – It is used to notify user that CR-LDP Notification message with a status information $status, LSPID $lspid, and traffic information $tr arrived at a node $node.
3. proc notify-erlsp-release {node lspid} – It is used to notify user that CR-LDP Release/Withdraw message with LSPID $lspid arrived at a node $node.

APIs for the constraint-based routing

1. **$ns collect-resource-info $itime** – Used to collect the resource information from all MPLS nodes periodically every the time interval $itime
2. **$MPLSnode constraint-based-routing $dstid $bw** – Used to calculate explicit route. If O.K return an explicit route, else return -1.

    Configuration Parameters

    $dstid : the ID of the destination LSR

$bw : the required bandwidth

APIs for reroute simulation in MPLS Networks

1. **$ns enable-reroute $option –** let all MPLS nodes execute path restoration function. There are five options as follows:
   drop : Not create any new alternative path.
   L3 : make use of L3 routing table
   notify-prenegotiated : The node that detected a link failure transmits a LDP Notification message to its upstream LSRs.
   simple-dynamic : Create new alternative path between The node that detected a link failure and the PML if one does not exist.
   shortest-dynamic: Create new alternative path between The node that detected a link failure and the next node if one does not exist.
2. **$ns set-protection-lsp $stime $itime $lspid** – used to see whether a link failure occurs on all MPLS nodes.
   Configuration Parameters
   $stime : The time to start detecting a lnk failure
   $itime : The time intervals
   $lspid : the LSPID of working LSP
3. **$MPLSnode reroute-lsp-binding $w_lspid a_lspid** - used to bind working LSP to altanative LSP
   Configuration Parameters
   $w_lspid : The LSPID of working LSP
   $a_lspid : The LSPID of altanative LSP
4. **$MPLSnode enable-reroute-egress-lsr** – let $MPLSnode operate as a Protection Merge LSR

APIs for Trace of MPLS/LDP packets and Dump of Tables

1. **$MPLSnode trace-mpls** – Trace packets in a MPLS node, $MPLSnode
2. **$MPLSnode trace-ldp** – Trace LDP packets in a MPLS node, $MPLSnodel
3. **$MPLSnode pft-dump** – Display a table, PFT(Partial Forwarding Table) managed in a MPLS node, $MPLSnode

4. **$MPLSnode erb-dump** – Display a table, ERB(Explicit Route information Table) managed in a MPLS node, $MPLSnode
5. **$MPLSnode lib-dump** – Display a table, LIB(Label Information Base) managed in a MPLS node, $MPLSnode

APIs for utility

6. **$ns ldp-request-color $color** – Set a color for LDP Request message.
   ex)$ns ldp-request-color $green – Set green color for LDP Request message.
7. **$ns ldp-mapping-color $color** – Set a color for LDP Mapping message
8. **$ns ldp-withdraw-color $color** – Set a color for LDP Withdraw message
9. **$ns ldp-release-color $color** – Set a color for LDP Release message
10. **$ns ldp-notification-color $color** – Set a color for LDP Notification message

4.2. OPNET

OPNET is a network simulator that runs on a windows environment. OPNET provides a comprehensive development environment for the specification, simulation and performance analysis of communication networks. A large range of communication systems from a single LAN to global satellite networks can be supported. Discrete events simulations are used as the means of analyzing system performance and their behavior. The key features of OPNET are summarized here as:

Modeling and Simulation Cycle: OPNET provides powerful tools to assist user to go three out of the five phases in a design circle (i.e. the building of models, execution of a simulation and the analysis of the output data)

Hierarchical Modeling: OPNET employs a hierarchical structure to modeling. Each level of the hierarchy describes different aspects of the complete model being simulated.

Specialized in communication networks: Detailed library models provide support for existing protocols and allow researchers and developers to either modify these exisiting models or develop new models on their own

Automatic simulation generation: OPNET models can be compiled into executable code. An executable discrete-event simulation can be debugged or simply executed, resulting in output data.

This sophisticated package comes complete with a range of tools which allows developers specify models in great details, identify the elements of the model of interest, execute the simulation and analyze the generated output data.

5. Traffic Engineering Simulation and Performance

5.1. ER-LSP Setup

The topology show below is created using NS-2. Nodes 0,1,7 and 8 can be considered as computer or non-MPLS supporting device. Nodes 0 and 1 act as sources of traffic, while nodes 7 and 8 are their respective sinks. In other words, nodes 1 and 0 send some data to nodes 8 and 7 through the MPLS network.

When no traffic engineering is done the packets get lost due to congestion at node 2, as both the traffic flow take the shortest path and it gets congested. There are 2 paths between ingress-egress, but only the path with minimum hop count is used (The path through node 2, 9 and 6). The other path (Path via node 2,4,5 and 6) is not utilized, while the shortest path gets over utilized.

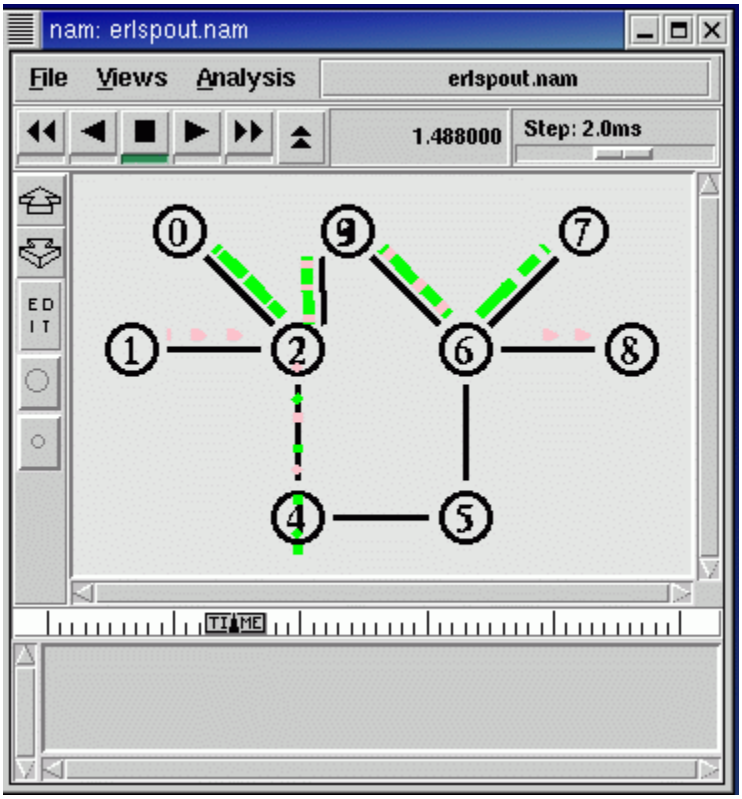

Figure 2: Shortest Path Algorithm – Packet Loss due to congestion

The graph below shows the through put of each flow when no traffic engineering is applied and all the traffic flow via the shortest path.

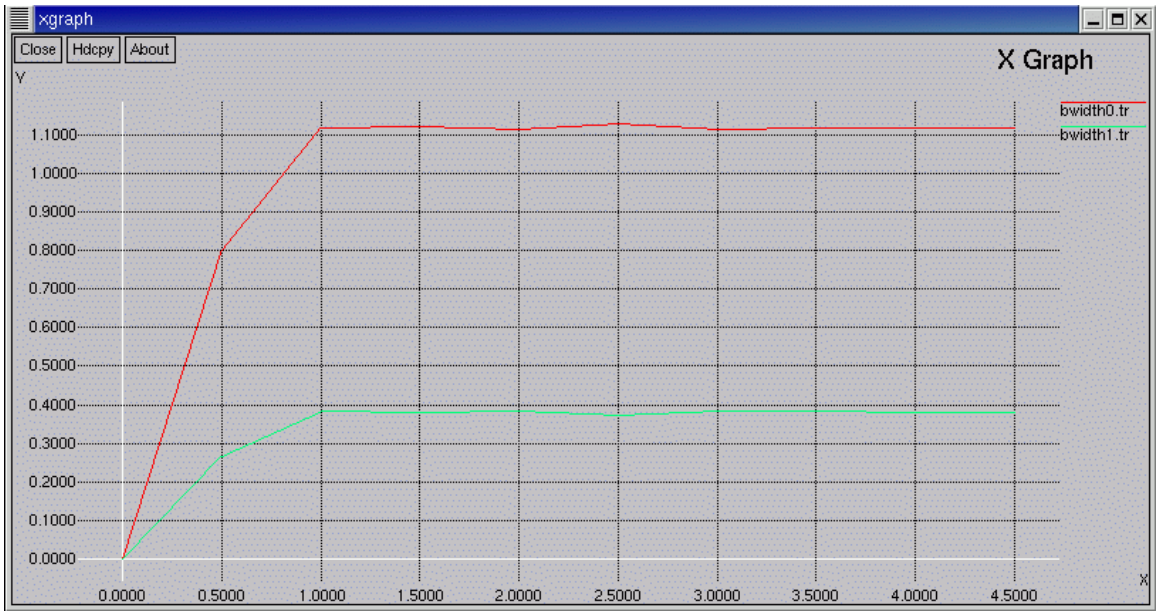

Figure 3: Graph showing the delay for each flow when the shortest path algorithm is used

Now traffic engineering is applied. The point of congestion is identified and rectified, by switching one of the flows (Either flow from 0 to 7 or 1 to 8) through the explicit path via nodes 2,4,5 and 6. When this is there is no packet loss due to congestion and the throughput of the network is increased. This scenario is depicted in figure 4.

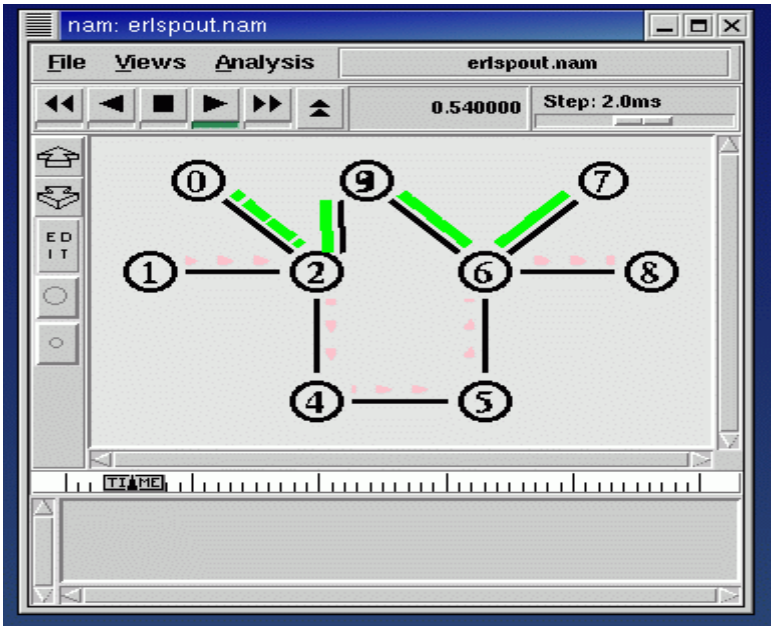

Figure 4: No Congestion – After applying traffic engineering

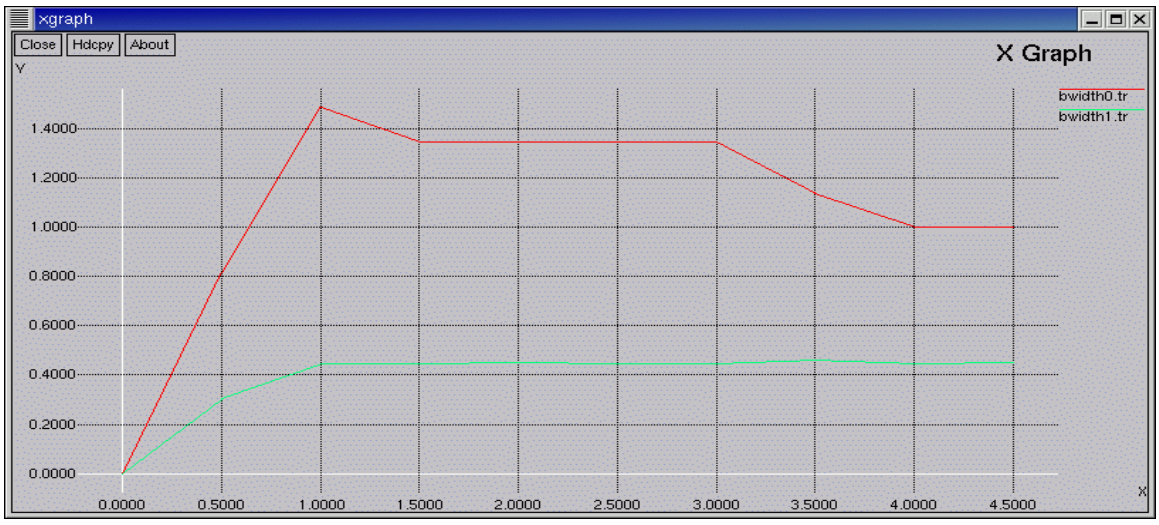

Figure 5: Graph showing the delay for each flow when Traffic engineering is applied.

The delay in the case when traffic engineering is lower then when it is not applied (Refer Figure 3 and Figure 5). This simulation was done to emphasize the significance of Traffic Engineering. The Tcl code is attached in Appendix A.

5.2. UDP and TCP Simulation

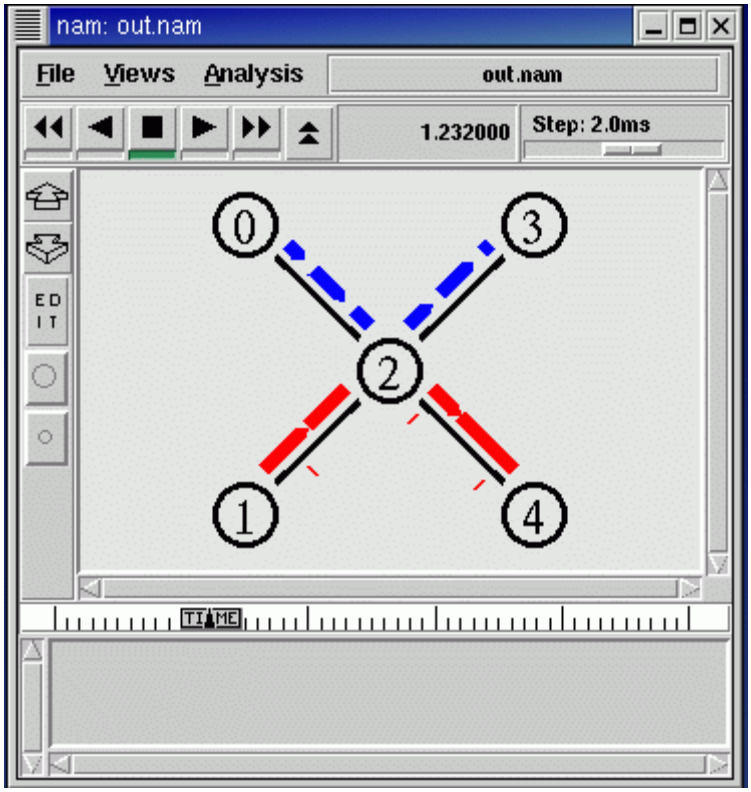

Figure 6: UDP and TCP Simulation

Then above figure is a simple topology consisting of 5 nodes. Nodes 0 and 1 are the source of traffic and nodes 3 and 4 are the sink, while node 2 just routes the traffic to the destination. A constant bit rate traffic using UDP is transported from node 0 to node 3, while an ftp application is set between nodes 1 and 4.

The slow start algorithms implementation is studied from this simulation. The Tcl code of this simulation is attached in Appendix A.

5.3.MPLS Simulation Using OPNET

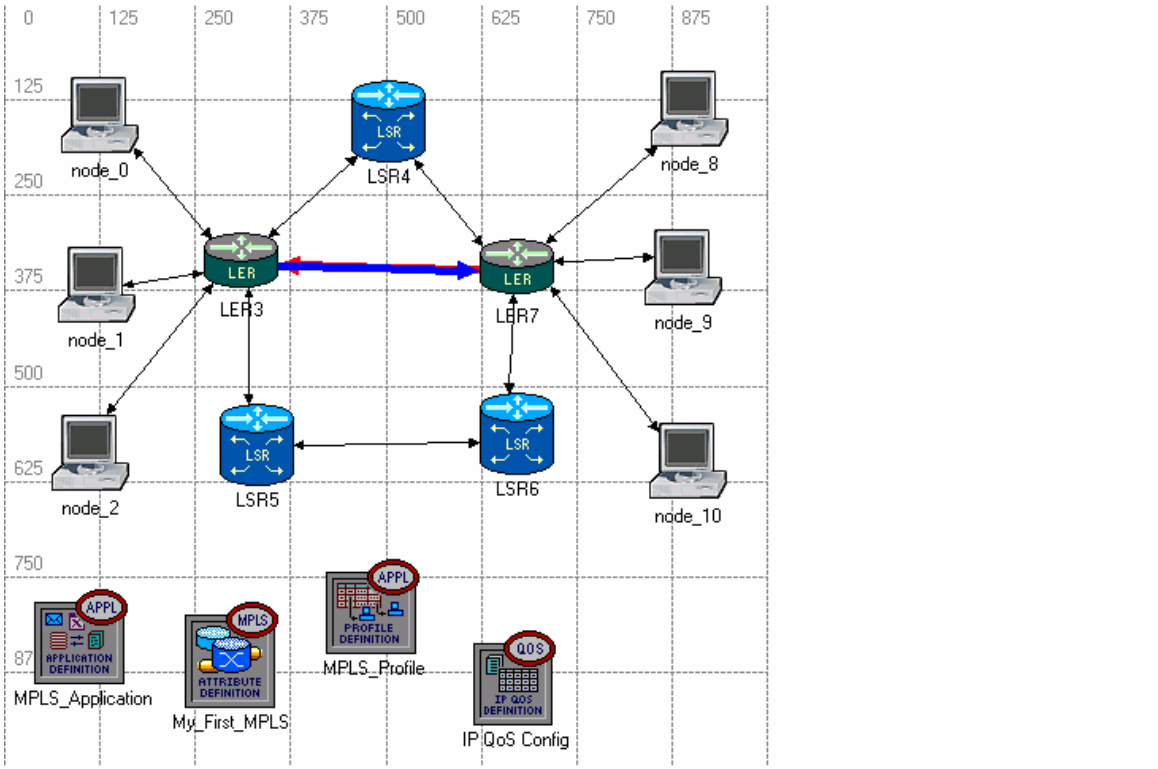

Figure 6: MPLS Simulation in OPNET

The above figure shows the topology that was used to simulate the MPLS environment in OPNET. It contains 6 hosts and 3 traffic pairs (Traffic from node0, node1 and node2 to node8, node9 and node10) are setup between them. The hosts are connected to 2 *Label Edge Router* (LER), viz LER3 and LER7. There are 2 paths between the 2 LERs one us the shortest path through LSR4 (path1) and the other path is through LSR5 & LSR6 (path2).

Two LSPs are setup to switch traffic through path2. The traffic from node0 and node2 are made to flow via these LSPs, thereby distributing the traffic. The delay and the throughput of the network is enhanced. The graphs are shown below.

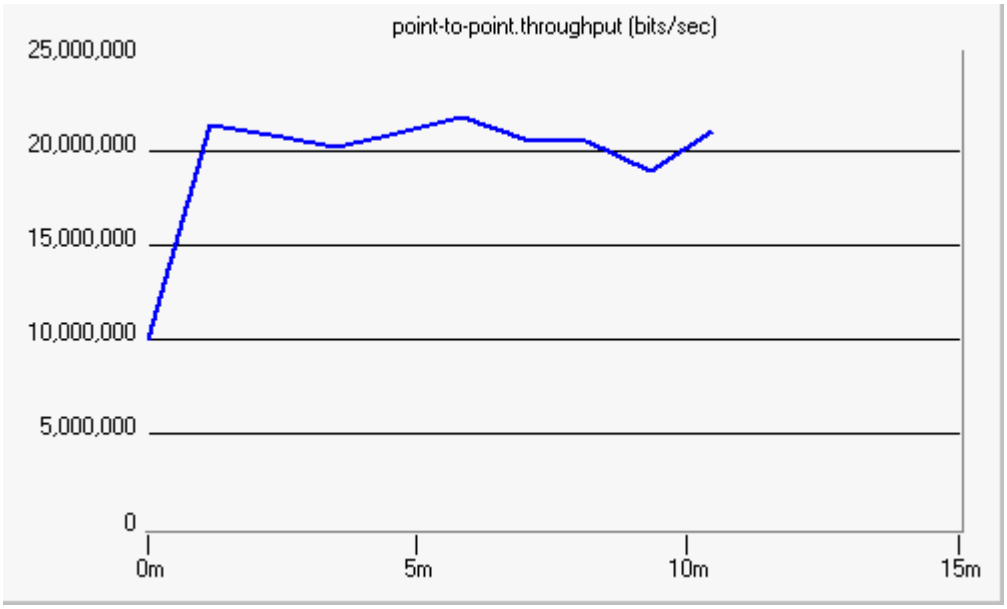

Figure 7: Point to Point throughput when LSP is used

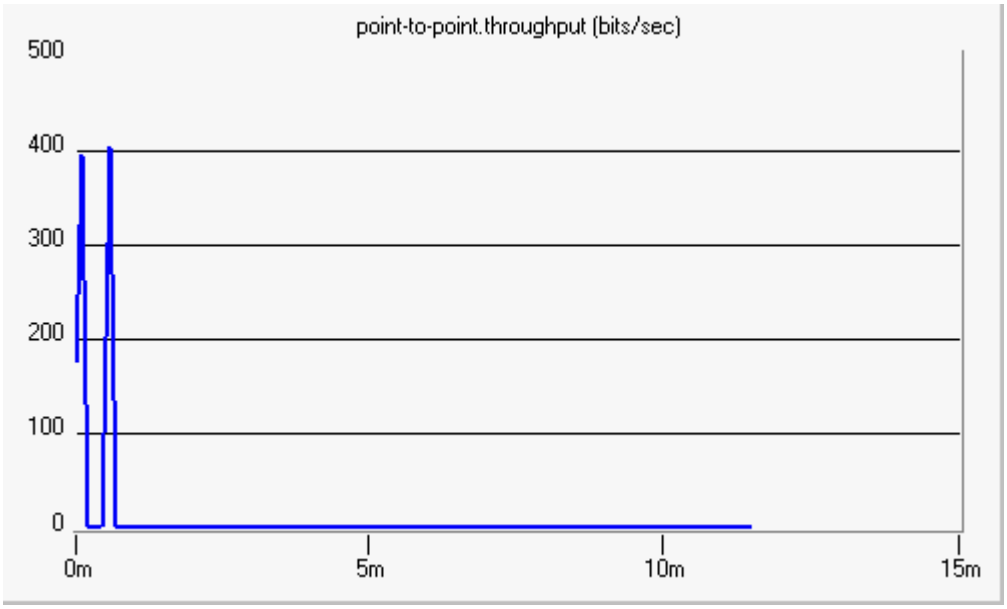

Figure 8: Point to Point throughput when LSP is not used

This is the throughput on the link LER3 to LSR5. The throughput in the case when LSP is not used is almost nil as all the traffic flows via the shortest path (path1), but in the case when LSP is installed, there is throughput in the link LER3 to LSR5. So we can see

that when traffic engineering is applied the traffic gets evenly distributed and load balancing is done. This simulation was carried out to signify the effect of traffic simulation.

6. Conclusion

A detailed study of the various traffic engineering algorithm and different protocols used were studied. From the independent study a better understanding of MPLS and Traffic Engineering was obtained. The future study would be to simulate the various Traffic Engineering algorithm mentioned here and compare their performance.

References:

1. *Rosen, E., Viswanathan, A. and R. Callon*, "Multiprotocol Label Switching Architecture", RFC 3031.
2. *D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow*, "RSVP-TE Extension to RSVP for LSP", RFC 3209.
3. *L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas ,* "**LDP** Specification", RFC 3036.
4. *B. Jamoussi*, "Constraint-Based LSP Setup using LDP", Work in Progress
5. *D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus* "Traffic Engineering in MPLS", RFC 2702.
6. *Dave Katz, Derek Yeung, Kireeti Kompella*, "Traffic Engineering Extensions to OSPF - draft-katz-yeung-ospf-traffic-06.txt ", Work in progress
7. *Moy, J.*, "OSPF Version 2", RFC 2328, April 1998.
8. *Coltun, R.*, "The OSPF Opaque LSA Option," RFC 2370, July 1998.
9. *Subhash Suri, Marcel Waldvogel, Priyank Ramesh Warkhede*, "Profile-Based Routing: A New Framework for MPLS Traffic Engineering".
10. *Qingming Ma, Peter Steenkiste, "*On Path Selection for Traffic with Bandwidth Guarantees".
11. *Murali Kodialam, T.V. Lakshman,*"Minimum Interference Routing with Applications to MPLS Traffic Engineering"
12. *Jong-Moon Chung*, "Analysis of MPLS traffic engineering", Circuits and Systems, 2000. Proceedings of the 43rd IEEE Midwest Symposium on , Volume: 2 , 2000 Page(s): 550 -553 vol.2.
13. *Karol Kowalik and Martin Collier, "*QoS routing as a tool of MPLS Traffic Engineering".
14. *Aukia P, Kodialam M, Koppol P.V.N, Lakshman T.V, Sarin H, Suter, B*, "RATES: a server for MPLS traffic engineering", IEEE Network**,** March-April 2000, Volume: 14 Issue: 2

15. *Keping Long, Zhongshan Zhang, Shiduan Cheng*, "**Load balancing algorithms in MPLS traffic engineering"**, High Performance Switching and Routing, 2001 IEEE Workshop on , 2001
16. Ns-manual http://www.isi.edu/nsnam/ns/ns-documentation.html
17. MNS-manual http://flower.ce.cnu.ac.kr/%7Efog1/mns/mns2.0/manual/manual.htm
18. *Juha Heinanen*, Multiprotocol Encapsulation over ATM Adaptation Layer 5, RFC 1483.
19. *S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang*, An Architecture for Differentiated Services, RFC 2475.
20. *Kershenbaum Aarona***,** Telecommunications network design algorithms, 1993.
21. "Intermediate System to Intermediate System Intra-Domain Routeing Exchange Protocol for use in Conjunction with the Protocol for Providing the Connectionless-mode Network Service (ISO 8473)", ISO DP 10589, February 1990.
22. MNS-Manual - http://flower.ce.cnu.ac.kr/~fog1/mns/

Appendix A

1. This is the code that sets up an explicit route path.

```
###################This code was written by Shyam Subramanian#################
#Create a simulator object
set ns [new Simulator]

#Open the nam trace file
set nf [open erlspout.nam w]
$ns namtrace-all $nf

set f0 [open bwidth0.tr w]
set f1 [open bwidth1.tr w]
#set f2 [open erlspbwidth2.tr w]

#Define a 'finish' procedure
proc finish {} {
        global ns nf f0 f1
        $ns flush-trace
           #Close the trace file
        close $nf
           #Execute nam on the trace file
        exec nam erlspout.nam &
           exec xgraph bwidth0.tr bwidth1.tr  -geometry 800x400 &
        exit 0
}


# Insert your own code for topology creation
# and agent definitions, etc. here


#Node Definitions

set Node0  [$ns node]
set Node1  [$ns node]
set LSR2   [$ns mpls-node]
set LSR3   [$ns mpls-node]
set LSR4   [$ns mpls-node]
set LSR5   [$ns mpls-node]
set LSR6   [$ns mpls-node]
set Node7  [$ns node]
set Node8  [$ns node]
#set Node9  [$ns node]
#set Node10 [$ns node]
```

```
#Link Definitions

$ns duplex-link $Node0 $LSR2 1.5Mb 10ms DropTail
$ns duplex-link $Node1 $LSR2 1Mb   10ms DropTail
$ns duplex-link $LSR2  $LSR3 1.5Mb 10ms DropTail
$ns duplex-link $LSR3  $LSR6 1.5Mb 10ms DropTail
$ns duplex-link $LSR2  $LSR4 1Mb   10ms DropTail
$ns duplex-link $LSR4  $LSR5 1Mb   10ms DropTail
$ns duplex-link $LSR5  $LSR6 1Mb   10ms DropTail
$ns duplex-link $LSR6 $Node7 1.5Mb 10ms DropTail
$ns duplex-link $LSR6 $Node8 1Mb   10ms DropTail
#$ns duplex-link $Node9 $LSR2 1Mb   10ms DropTail
#$ns duplex-link $Node10 $LSR6 1Mb  10ms DropTail


$ns duplex-link-op $Node0 $LSR2 orient right-down
$ns duplex-link-op $Node1 $LSR2 orient right
#$ns duplex-link-op $Node9 $LSR2 orient right-up
$ns duplex-link-op $LSR2  $LSR3 orient right-up
$ns duplex-link-op $LSR3  $LSR6 orient right-down
$ns duplex-link-op $LSR2  $LSR4 orient down
$ns duplex-link-op $LSR4  $LSR5 orient right
$ns duplex-link-op $LSR5  $LSR6 orient up
$ns duplex-link-op $LSR6 $Node7 orient right-up
$ns duplex-link-op $LSR6 $Node8 orient right
#$ns duplex-link-op $Node10 $LSR6 orient right-down

#creating agents and attaching them to the nodes
#Between Node0 ------------ Node7 4mbps traffic

set udp0 [new Agent/UDP]
$ns attach-agent $Node0 $udp0
set Src0 [new Application/Traffic/CBR]
#$Src0 set packetSize_ 5000
#$Src0 set interval 0.0
$Src0 attach-agent $udp0

set Src3 [new Application/Traffic/CBR]
#$Src3 set packetSize_ 5000
#$Src3 set interval 0.0
$Src3 attach-agent $udp0

set Dst0 [new Agent/LossMonitor]
$ns attach-agent $Node7 $Dst0
```

```
$ns connect $udp0 $Dst0

#Between Node1 ------------ Node8 3mbps traffic

set udp1 [new Agent/UDP]
$ns attach-agent $Node1 $udp1
set Src1 [new Application/Traffic/CBR]
#$Src1 set packetSize_ 5000
#$Src1 set interval 0.0000167
$Src1 attach-agent $udp1

set Src4 [new Application/Traffic/CBR]
#$Src4 set packetSize_ 5000
#$Src4 set interval 0.0
$Src4 attach-agent $udp0

set Dst1 [new Agent/LossMonitor]
$ns attach-agent $Node8 $Dst1

$ns connect $udp1 $Dst1


#Between Node9 ------------ Node10 4mbps traffic

#set udp2 [new Agent/UDP]
#$ns attach-agent $Node9 $udp2
#set Src2 [new Application/Traffic/CBR]
#$Src2 set packetSize_ 5000
#$Src2 set interval 0.0000125
#$Src2 attach-agent $udp2

#set Dst2 [new Agent/LossMonitor]
#$ns attach-agent $Node10 $Dst2

#$ns connect $udp2 $Dst2

$ns color 1 Green
$ns color 2 Pink

$udp0 set fid_ 1
$udp1 set fid_ 2

proc record {} {
        global Dst0 Dst1 nf f0 f1
        set ns [Simulator instance]
```

```
        #Set the time after which the procedure should be called again
        set time 0.5

        #How many bytes have been received by the traffic sinks?
        set bw0 [$Dst0 set bytes_]
           set bw1 [$Dst1 set bytes_]

        #Get the current time
        set now [$ns now]
        #Calculate the bandwidth (in MBit/s) and write it to the files
        puts $f0 "$now [expr $bw0/$time*8/1000000]"
           puts $f1 "$now [expr $bw1/$time*8/1000000]"

        #Reset the bytes_ values on the traffic sinks
        $Dst0 set bytes_ 0
           $Dst1 set bytes_ 0

        #Re-schedule the procedure
        $ns at [expr $now+$time] "record"
}

# Creating Ldp Agent on all the MPLSnodes
$ns configure-ldp-on-all-mpls-nodes

# setting colors to different msg type

$ns ldp-request-color blue
$ns ldp-mapping-color red
$ns ldp-withdraw-color gray80
$ns ldp-release-color orange
$ns ldp-notification-color yellow

# Setting trigger Scheme

$ns mpls-node enable-data-driven

#$ns LDP-peer $LSR2 $LSR3
#$ns LDP-peer $LSR3 $LSR6
#$ns LDP-peer $LSR2 $LSR4
#$ns LDP-peer $LSR4 $LSR5
#$ns LDP-peer $LSR5 $LSR6

for {set i 2} {$i < 7} {incr i} {
        set a LSR$i
        set m [eval $$a get-module "MPLS"]
        eval set LSR$i $m
```

```
}

$ns at 0.0 "record"
$ns at 0.1 "$Src0 start"
$ns at 0.1 "$Src1 start"
#$ns at 0.1 "$Src2 start"
$ns at 0.1 "$Src3 start"
$ns at 0.1 "$Src4 start"
$ns at 0.2  "$LSR6 ldp-trigger-by-withdraw 7 -1"
$ns at 0.2  "$LSR6 ldp-trigger-by-withdraw 8 -1"

#$ns at 0.3  "$LSR2 aggregate-flows    7  6"
#$ns at 0.3  "$LSR2 aggregate-flows    8  6"
#$ns at 0.5  "$LSR6 send-ldp-withdraw-msg 6"

$ns at 0.3  "$LSR2 make-explicit-route  6  2_4_5_6   3600  -1"
$ns at 0.4  "$LSR2 flow-erlsp-install   8 -1   3600"
$ns at 3.1  "$LSR2 ldp-trigger-by-release  6 3600"
$ns at 0.4  "$LSR2 make-explicit-route  6 2_4_5_6    3700  -1"
$ns at 3.2  "$LSR2 flow-erlsp-install   7 -1 3700"
$ns at 4.5  "$LSR2 ldp-trigger-by-release  7 3700"

#Call the finish procedure after 5 seconds simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

1. This is the code that simulates a TCP and UDP application.

```
#Create a simulator object
set ns [new Simulator]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
           #Close the trace file
        close $nf
           #Execute nam on the trace file
        exec nam out.nam &
        exit 0
}

# Insert your own code for topology creation
# and agent definitions, etc. here
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail

#Changed to SFQ to implement fair queueing
$ns duplex-link $n3 $n2 1Mb 10ms SFQ
$ns duplex-link $n4 $n2 1Mb 10ms SFQ

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right-up
$ns duplex-link-op $n2 $n4 orient right-down

  #Create a UDP agent and attach it to node n0
  set udp0 [new Agent/UDP]
  $ns attach-agent $n0 $udp0

  # Create a CBR traffic source and attach it to udp0
  set cbr0 [new Application/Traffic/CBR]
  $cbr0 set packetSize_ 500
```

```
 $cbr0 set interval_ 0.005
 $cbr0 attach-agent $udp0

 #Create a TCP agent and attach it to node n1
 set tcp1 [new Agent/TCP]
 $ns attach-agent $n1 $tcp1

 # Create a FTP traffic source and attach it to tcp1
# set ftp1 [new Application/FTP]

 set cbr1 [new Application/Traffic/CBR]
 $cbr1 set packetSize_ 500
 $cbr1 set interval_ 0.005
 $cbr1 attach-agent $tcp1

 set null0 [new Agent/Null]
 $ns attach-agent $n3 $null0

 set sink [new Agent/TCPSink]
 $ns attach-agent $n4 $sink

 $ns connect $udp0 $null0
 $ns connect $tcp1 $sink

#start and stop traffic time
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
#$ns at 1.0 "$ftp1 start"
$ns at 4.5 "$cbr0 stop"
#$ns at 4.0 "$ftp1 stop"
$ns at 4.0 "$cbr1 stop"

#to identify the packets we color them after defining them objects for them
$udp0 set class_ 1
$tcp1 set class_ 2

$ns color 1 Blue
$ns color 2 Red

#to Monitior the queue and find out what is happening on that
$ns duplex-link-op $n2 $n3 queuePos 0.

#Call the finish procedure after 5 seconds simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```