University of Oslo
Department of Informatics

Traffic Engineering And Supporting Quality of Service

Shamshirgaran, Mohammad Reza

Cand. Scient Thesis

February 2003

Acknowledgement

This thesis is written as a part of my graduate studies within the field of communication systems, Informatics.

I would like to start by expressing my gratitude towards professor Pål Spilling for his guidance and collaboration, making it possible for me to advance and extend my knowledge within the field of communication systems. I would also like to thank Boning Feng for his time and insightful comments on this thesis.

Also, the people at UNIK – University Graduate Centre for Technology and department of Informatics at University of Oslo deserve my gratitude for their cooperation. During my time spent there, they provided the right equipment for simulation use.

Last but not least, warm thanks goes to my girlfriend and family for excusing my absence of present spending time with them, because of the much-spent time on this thesis.

ABSTRACT

Traffic Engineering describes techniques for optimising network performance by measuring, modelling, characterizing and controlling Internet traffic for specific performance goals [11]. This is a comprehensive definition. Traffic engineering performance goals typically fall into one of two categories. The first one is traffic related performance objectives such as minimizing packet loss, lowering end-to-end delay, or supporting a contracted Service Level Agreement (SLA). The second category is efficiency related objectives, such as balancing the distribution of traffic across available bandwidth resources. Traffic related performance goals are set in order to meet contracted service levels and offer competitive services to customers. Efficiency related goals, are required by the service provider to minimize the cost of delivering services, especially the cost of utilizing expensive network resources.

The objective of this thesis is to present a description of Multi Protocol Label Switching (MPLS) architecture and its functionality to achieve a tool for performing traffic engineering and QoS support. We simulate traffic engineering with MPLS on a simple network and measure its performance. We analyse measurements related to queuing delay, throughput and other traffic related issues. We then move on fine- tuning the MPLS-TE network to also take into consideration QoS support when aggregating flows through a single label- switching path. We combine differentiated services with MPLS architecture in order to support QoS requirements. The simulation tool used in this thesis is called OPNET Modeler version 8.1[1].

[1] OPNET Modeler 8.1 is a network simulation tool © OPNET Technologies Inc.

CONTENTS

ABSTRACT 3

1 INTRODUCTION 9

2 SHORTEST PATH ROUTING PRINCIPLE 11

2.1 Shortest path routing within an Autonomous System 11

2.2 Shortest path routing principle and its drawbacks 12

2.3 Summary over shortest path routing principle 15

3 TRAFFIC ENGINEERING & QOS SUPPORT WITH MPLS 17

3.1 MPLS 17
3.1.1 MPLS functionality 17

3.2 Traffic engineering with MPLS 20
3.2.1 Distribution of network statistical information 20
3.2.2 Path Selection 20
3.2.3 Signalling for path establishment 21
3.2.4 Packet forwarding 24
3.2.5 Rerouting 24

3.3 Quality of Service support with MPLS 25
3.3.1 Integrated Services 25
3.3.2 IntServ implementation with MPLS 26
3.3.3 IntServ scalability drawbacks 26
3.3.4 Differentiated Services 27
3.3.5 Per-Hop Behaviour (PHB) 28
3.3.6 DiffServ implementation with MPLS 30
3.3.7 Aggregation of traffic flows with MPLS and Diffserv 31

3.4 Summary over MPLS Traffic Engineering and QoS Support 33

4 INTRODUCTION TO SIMULATION 34

4.1 Simulation tool 34

4.2 Network topology 34

4.3 General experimental conditions regarding all simulation scenarios 35

5 SIMULATION EXPERIMENT USING OSPF 37

5.1 Analysing and discussing experimental results 37
5.1.1 Throughput 37
5.1.2 Queuing delay 40

5.2 Concluding remarks 41

6 SIMULATION EXPERIMENT USING MPLS -TE 42

6.1 MPLS Traffic engineering configurations 42

6.2 Analysing and discussing experiential results 44
6.2.1 Throughput 44
6.2.2 Queuing delay 48

6.3 Concluding remarks 50

7 SIMULATION EXPERIMENT USING MPLS-TE AND DIFFSERV 51

7.1 MPLS-TE and QoS support configuration 51

7.2 Analysing and discussing experiential results 53
7.2.1 WFQ delay and buffer usage 53
7.2.2 Flow Delay 54
7.2.3 Throughput 55

7.3 Concluding remarks 56

8 CONCLUSION 58

8.1 Conclusion made from shortest path routing principle 58

8.2 Conclusion made from MPLS traffic engineering 59

8.3 Conclusion made from MPLS traffic engineering with QoS support 60

8.4 Further need for research 62

9 APPENDIX 63

9.1 Dijkstras Algorithm 63

9.2 Shortest Path Routing configuration details within OPNET 64
9.2.1 Application configuration 64
9.2.2 Profile configuration 68
9.2.3 Workstations and Server configuration 68
9.2.4 Router configuration 69
9.2.5 Simulation configuration attributes 71

9.3 MPLS-TE configuration details within OPNET 72
9.3.1 Application configuration 72
9.3.2 Profile configuration 76
9.3.3 Workstations and Server configuration 76
9.3.4 Creating LSPs 77
9.3.5 MPLS configuration 78
9.3.6 Router configuration 79
9.3.7 Simulation configuration attributes 82

9.4 MPLS-TE-QoS supported flows config. details within OPNET 82
9.4.1 Application configuration 82
9.4.2 Profile configuration 82
9.4.3 Creating LSPs 83
9.4.4 MPLS configuration 83
9.4.5 QoS Configuration attributes 84
9.4.6 Workstations and Server configuration 84
9.4.7 Router configuration 85
9.4.8 Simulation configuration attributes 85

10 REFERENCES 86

1 INTRODUCTION

Rapid growth of the Internet has made a huge impact on what type of services requested from consumers and what kind of performance they demand from the services they wish to use. Consequently as service providers encourage businesses on to the Internet, there has been a requirement for them to develop, manage and improve IP- network infrastructure in terms of performance. Therefore, the interest of traffic control through traffic engineering has become important for ISP's.

Today's networks often function with well-known shortest path routing protocols. Shortest path routing protocols as their name implies, are based on the shortest path forwarding principle. In short, this principle is about forwarding IP-traffic only through the shortest path towards their destination. At one point, when several packets destined from different networks start using the same shortest path, this path may become heavily loaded. This will result in congestion within the network. Various techniques have been developed to cope with the shortest path routing protocols shortcomings. However, recent research has come up with another way to deal with the problem. With traffic engineering, one can engineer traffic through other paths than the shortest path. The network carries ip-traffic, which flows through interconnected network elements, including response systems such as protocols and processes. Traffic engineering establishes the parameters and operating points for these mentioned elements. Internet traffic leads to control problem. Therefore a desire and need for better control over the traffic may be accomplished with help of traffic engineering.

The main purpose of traffic engineering is to achieve a certain performance in large IP networks. High quality of service, efficiency, and highest possible utilization of network resources are all driving forces behind the need and desire for traffic engineering. Traffic engineering requires precise control over the routing functionality in the network. To compute and establish forwarding path from one node to another is vital to achieve a desired flow of traffic. Generally, performance goals can be traffic- and/or resource oriented. Traffic oriented performance is usually related to QoS in the network, which concerns prohibit packet loss and delay. Resource oriented performance is related to efficient utilization of network assets. Efficient resource allocation is needed to achieve performance goal within the net. Congestion control is another important goal of traffic engineering. Congestion typically arises under the circumstances such as when network resources are insufficient or inadequate to handle offered load. This type of congestion can be addressed by augmenting network capacity, or modulating, conditioning, or throttling the demand so that traffic fits onto the available capacity using policing, flow control, rate shaping, link scheduling, queue management and tariffs [2]. Other circumstances where congestion appears are when traffic is inefficiently mapped onto resources, causing subset of resources to become over utilized while others remain under utilized. This problem can be addressed by increasing the efficiency of resource allocation. An example would be to route some traffic away from congested resources to relatively under utilized ones [2].

Other purposes with traffic engineering are also reliable network operation and differentiated services, where traffic streams with different service requirements are in contention for network resources. QoS is thus important for those who have signed up for a certain service level agreement (SLA). It is therefore needed to control the traffic so that certain traffic flows can be routed in a way that the required QoS is given. When traffic engineering flows with different QoS requirements, one may want to assign certain flows to a certain path. Since several flows often take the same path to a certain destination, aggregation of traffic flows may reduce number of resource allocations needed [38], reserving resource for each aggregated traffic flow. This gives the opportunity to traffic engineer aggregated traffic flows while at the same time supporting QoS to each of them with minimum overhead for reservation of resources along a certain path.

In order to outline the performance achieved by traffic engineering, we felt it was necessary to starts by giving a description of the shortest path routing principle and its drawbacks. Then, we present the architecture of Multi protocol label switching and differentiated services. Highlighting their functionality and the way they can interact to support quality of service while traffic engineering.

After giving a description of the technologies itself, we move on to our simulation networks to measure their performance. First out, we configure a network to run shortest path routing protocol OSPF. To measure performance outbreaks, we generate TCP and UDP traffic to measure their treatment under a heavily loaded network. Then, we use the same network with its traffic once again, this time installing multi protocol label switching to engineer the flows to separate paths. Results collected from the both networks are then compared. Later we also show of the possibility of traffic engineering, while at the same time taking QoS aspects into consideration. Here, we only compare the QoS support given to flows that are engineered through the same label-switching path.

2 Shortest Path Routing Principle

In this chapter, a description of routing within an autonomous system (AS) based on the shortest path routing principle is given. This chapter concentrates only on the Intra-domain shortest path routing principles within an AS of a service provider's network. We start with a description of an exemplary backbone architecture belonging to an Internet Service Provider (ISP). Furthermore, giving a description of shortest path routing principle and its drawbacks.

2.1 Shortest path routing within an Autonomous System

Ever since the deployment of ARPANET, the forerunner of the present-day Internet, the architecture of the Internet has been constantly changing. It has evolved in response to advances in technology, growth, and offerings of new services. The internet today consists of multiple service providers network connected to each other, forming a global network communication infrastructure. This infrastructure enables people around the world to communicate with each other through interconnected network devices. These devices are set up to process any data that traverse through them. These devices or nodes are often formed in logical and hierarchical way. With customers networks connected to a node or a router often called customer edge router (CE) at one end, and to an Internet service provider's (ISP) network edge router, which is referred to as provider edge router (PE) at the other end. The core routers within the provider's network form the inner routers forwarding packets a step closer to its destination. These often smaller autonomous systems (AS) are then connected to more powerful networking area referred to as the backbone. The backbone often carries the extensive amount of traffic that is to be transmitted or/and received between AS's. An example over such architecture is given in the below figure.

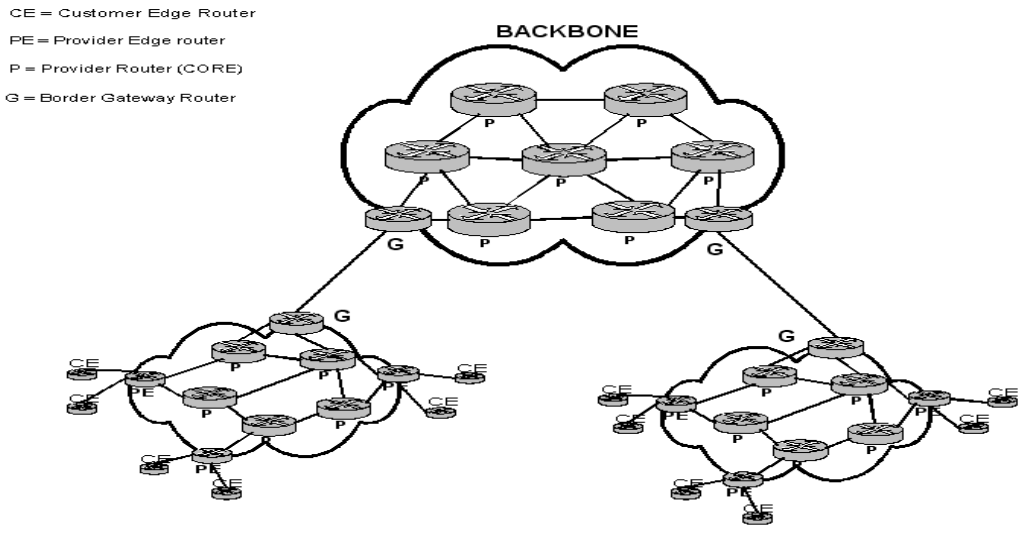

Figure 2.1 *Illustrate architecture over backbone of an ISP.*

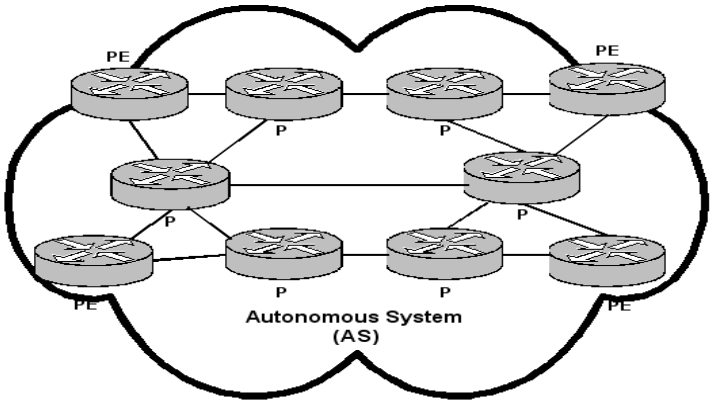

Figure 2.2 *Illustrate an exemplary architecture over an autonomous system.*

Zooming in on our precedence figure, we look at a single clouded area running a shortest path routing protocol as its routing protocol. An AS may look like the one illustrated in figure 2.2. The way an AS handles its traffic using shortest path routing principle is a sophisticated engineering detail that we don't look into. But we thereby give a simple description of its functionality. In order to make right delivery of packets received from the customer's networks, routers must exchange information with each other. The exchange of this information is a complex topic, which we will not get into in this thesis. But in short, the routing and forwarding mechanism is primarily divided into three processes. The first process is mainly responsible for exchanging topology information. This is needed for the second part of the process, which is the calculation of routes. Calculation happens independently within each router to build up a forwarding table. The forwarding table enables processing incoming packets to be forwarded towards its destination. The forwarding table is used when a packet is being forwarded and therefore must contain enough information to accomplish the forwarding function.

Within an AS, routing is based on Interior Gateway Protocols (IGPs) such as Routing Information Protocol (RIP) [27], Open Shortest Path First (OSPF) [13] and Intermediate System-Intermediate System (IS-IS) [28]. RIP is based on the distance vector algorithm and always tries to find the minimum hop route. Routing protocols such as OSPF and IS-IS are more advanced in the sense that routers exchange link state information and forward packets along shortest path based on Dijkstra's algorithm [12]. In short, Dijkstras algorithm computes the shortest path from every node to every other node in the network that it can reach. This is of course a highly simplified description. A complete coverage over the Dijkstras algorithm can be found in appendix 9.1. With help of Dijkstras algorithm, every node can compute the shortest path tree to every destination [12].

2.2 Shortest path routing principle and its drawbacks

The shortest path routing principle imposes some drawbacks within the routing area. A description of these drawbacks is described here. The scenario in Figure 2.3 illustrates the forwarding of packets based on the shortest path

algorithms. Looking at the below figure, imagining the routers 1,2,3, and 4 forming a smaller piece of a larger AS or backbone. Traffic is coming in from both network A and C and destined for the same terminating network through router 4. The interesting part here is that congestion may appear after a while between router1 and router2 since all the packets are sent over the minimum cost (high bandwidth) path to its destination. It uses only one path per source–destination pair, thereby potentially limiting the throughput of the network [12].

To give an example of the impacts this may appose in the network consider this: It is known that TCP connections intend to lower their transfer rate when signs of congestion appears, consequently making more room for UDP traffic to fill up the link and suppress the TCP flows [15]. This will cause the UDP traffic sent by one of the sources suppress the TCP flows sent by the other sources. Clearly, this situation can be avoided if the TCP and UDP traffic choose different non-shortest paths to achieve a better performance.

Congestion in the network is caused by lack of network resources or uneven load balancing of traffic. The latter one is the one that can be remedied by traffic engineering, which is the intention of this thesis to simulate in the coming chapters. If all packets sent from customers use the same shortest path to their destination, it may be difficult to assure some degree of QoS and traffic control. There are of course ways to support every single traffic flow with different technologies to assure QoS. In [39] for example, a signalling protocol is used to reserve resources for a certain flow travelling through the network, but this is only per-flow basis and when many of these are configured it makes it unacceptable for an ISP to manage and administer, since it isn't a scalable solution [36]. This can be proven by a simple formula, which states that if there exist N routers in the topology and C classes of services, it would be needed (N* (N-1) * C) –trunks containing traffic flows [36]. We will not further discuss this issue here, but later show that with another technology this can be reduced to C * N or even N traffic trunks.

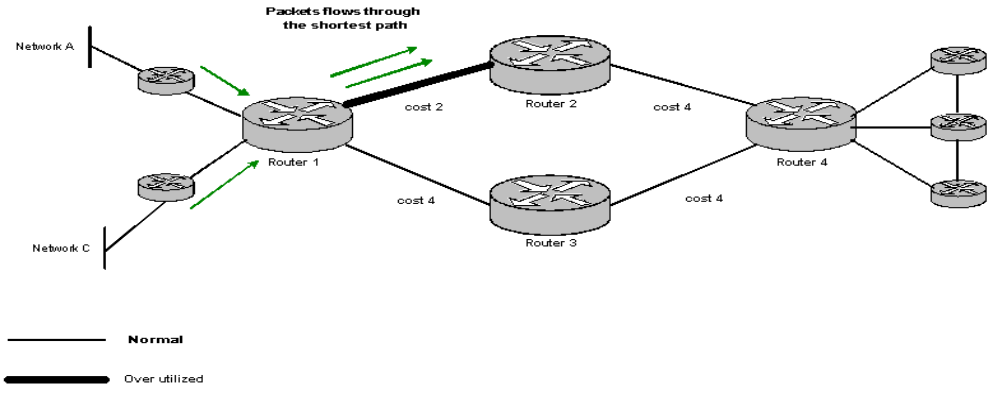

Figure 2.3 *Forwarding based on shortest path (minimum cost)*

The other problem mentioned with the shortest path routing protocols is the lack of ability to utilize the network resources efficiently [2]. This is not achieved by the shortest path routing protocols since they all just depend on the shortest path [2]. This is illustrated in the below figure, where packet from both network A and C traverse through the path with minimum cost, leaving other paths under utilized. Its capability to adapt to changing traffic conditions is limited by oscillation effects. If the routers flood the network with new link state advertisement messages based on the traffic weight on the links, this could result in changing the shortest path route. At one point, packets are forwarded along the shortest path, and suddenly right after exchange of link states advertisement choosing another "shortest" path through the network. The result may again be poor resource utilization [12]. This unstable characteristic has more or less been dealt with in the current version of OSPF, but with the side effect of been less sensitive to congestion and speed of response to it [12].

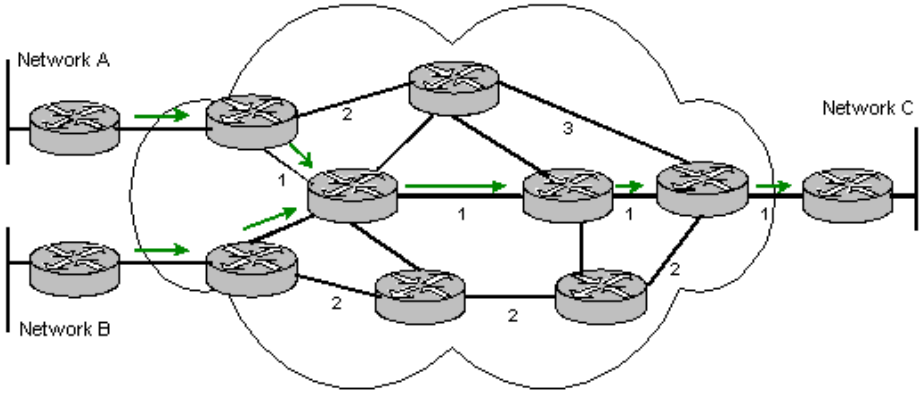

Figure 2.3 *Illustrates under utilized paths in the backbone*

Looking at figure 2.4, one can see that a more balanced network is taken place when traffic from network A and C starts using the under utilized paths in the above figure.

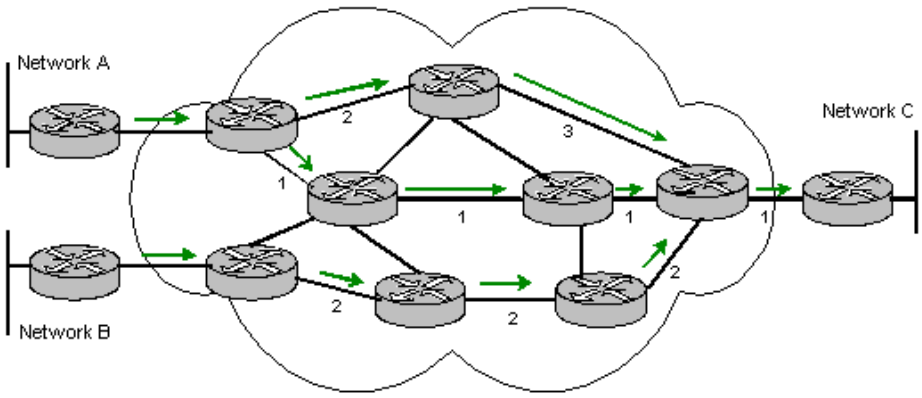

Figure 2.4 *Illustrates optimised backbone link utilization*

The shortest path routing principle cause uneven distribution of traffic, as a result of the shortest path algorithm they depend upon. Various techniques have emerged to cope with the traffic- balancing problem. For example, the equal-cost multipath (ECMP) option of OSPF [13] is useful in distributing load to several equal shortest paths. But, if there is only one shortest path, ECMP does not help. Another method for load-share balancing is the unequal-cost load balancing. In order to enable OSPF unequal-cost load balancing, one can manipulate the link speed of an interface. Since this manipulation doesn't really represent the actual speed of the link, it can be used to manipulate how data is load-shared over different links with varying speeds. This can be done by for example setting the same value across some links. The physical throughput however is unchanged.

For example, in figure 2.5 there are three ways for router A to get to network 10.0.0.1/24 after manipulating two links to the same value:

- A-F-G with a path cost of 84
- A-D-E-G with a path cost of 31
- A-B-C-G with a path cost of 94

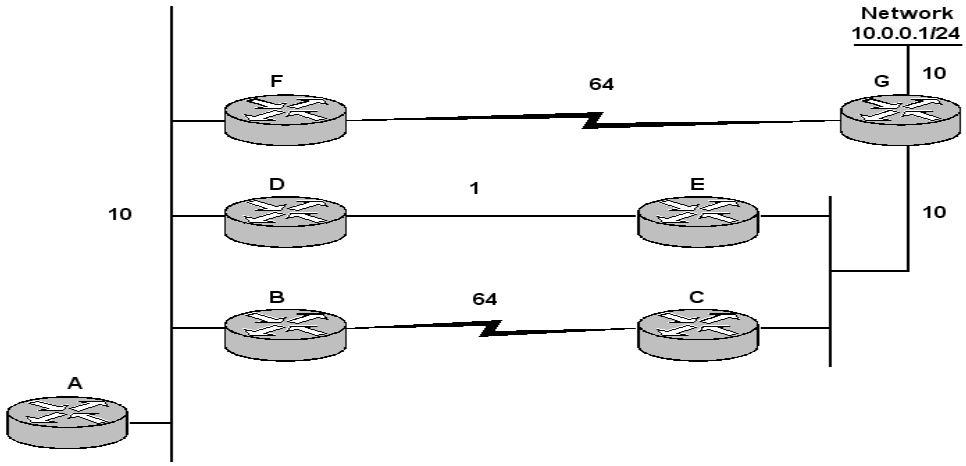

Figure 2.5 OSPF Unequal-Cost Load Balancing

For simple networks, it may be possible for network administrators to manually configure the cost of the links so that traffic can be more evenly distributed. Clearly, for complex ISP networks, this becomes a difficult task to administrate in a larger ASs of a service providers network since they have little or no low-level control over the basic mechanisms responsible for packet scheduling, buffer management, and path selection [7].

2.3 Summary over shortest path routing principle

In summary, making a forwarding decision actually consists of three sets of processes. The routing protocols, routing table and the actual process which

makes the forwarding decision and switches packets. These three sets of processes are illustrated, along with their relationship, in figure 2.6.

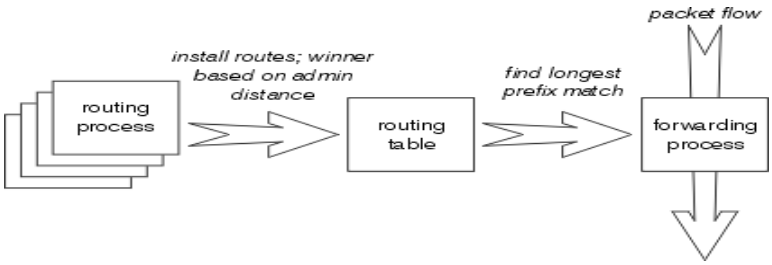

Figure 2.6 Illustrates the three components that describe the routing and forwarding process.

The longest prefix match always wins among the routes actually installed in the routing table, while the routing protocol with the lowest administrative distance always wins when installing routes into the routing table. This is known as shortest path routing principle. As mentioned, the downside of the shortest path routing is its drawbacks when it comes to efficient network utilization and to being able to handle traffic flows in a way so that bottlenecks are avoided within the network. This infer because packets seems to only be forwarded using the shortest path to a certain destination, and as stated in [5], the shortest paths from different sources overlap at some links, causing congestion on those links. As an example we mentioned what impact this had on TCP flows that got suppressed when signs of congestion appeared in the network. This allowed more room for the UDP traffic, thus made it even worst for the TCP traffic.

Before going any further, we summarize the problems concerning the shortest path based routing principles that we will try to simulate and address.

> **As described earlier, when all packets sent from different sources only utilises the shortest path between a pair of ingress and egress routers, the shortest path will become congested. As an example, we mentioned the impact of this on TCP and UDP traffic under heavy load conditions. Thus, our first problem is related to managing to engineer some traffic away from using the shortest path through the network topology. By this way, we aim to avoid congestion and bottlenecks within the network. Furthermore, we will try to address the shortest path routing principle's lack of ability to engineer traffic flows so that a more balanced and efficient utilized network is achieved.**

In the following chapter, MPLS is illustrated as a tool for performing traffic engineering and provisioning QoS. It is further to be seen whether MPLS based traffic engineering and QoS can deal with the mentioned shortest path routing principle drawbacks.

3 Traffic Engineering & QoS Support With MPLS

In this chapter, a description of the architecture that is believed to deal with the need of traffic engineering and QoS provisioning is given. This technology is called MPLS and a complete coverage of it is to be found under the following subchapters. Furthermore, we describe other technologies that are to be complementing the MPLS architecture for QoS provisioning.

3.1 MPLS

MPLS stands for Multi Protocol Label Switching and is basically a packet forwarding technique where the packets are forwarded based on an extra label attached in front of the ordinary payload. With this extra label attached, a path controlling mechanism takes place and a desired route can be established. Although MPLS is a relatively simple technology, it enables sophisticated capabilities far superior to the traffic engineering function in ordinary IP network. When MPLS is combined with differentiated services and constraint based routing, they become powerful and complementary tools for quality of service (QoS) handling in IP networks [2].

3.1.1 MPLS functionality

The functional capabilities making MPLS attractive within traffic engineering in IP networks are described in this section. MPLS functionality can be described by demonstrating the forwarding mechanism in its domain. Starting with its header and how it is constructed, we can slowly but clearly work us through the technology and describe the MPLS functionality. The figure below shows the format of this label, also called the MPLS header. It contains a 20*bit* label, a 3*bit* field for experimental use, a 1*bit* stack indicator, an 8*bit* time to live field. Each entry consists of 4 *octets* in a format depicted below [1]. The label field indicates the actual value of the MPLS label. The EXP field was ment for experimental purpose, and has been used in connection with QoS /CoS support. The stack bit implements MPLS label stacking, wherein more than one label header can be attached to a single IP packet [3]. The stack bit is set to 1 in order to indicate the bottom of the stack. All other stack bits are set to 0. Packet forwarding is accomplished using the label values of the label on the top of the stack. The TTL field is similar to the time-to-live field carried in the IP header. The MPLS node only processes the TTL field in the top entry of the label stack. The IP TTL field contains the value of the IPv4 TTL field or the value of the IPv6 Hop Limit field. Since MPLS nodes don't look at the IP TTL field, the IP TTL field is copied into the MPLS label.

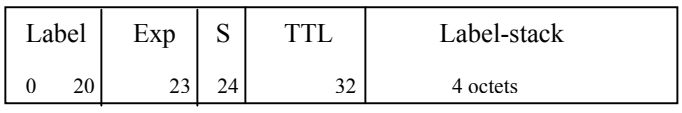

Figure 3.1 *The MPLS header format*

A MPLS header is inserted for each packet that enters the MPLS domain. This header is used to identify a Forwarding Equivalence Class (FEC). The same FEC is associated to packets that are to be forwarded over the same path through the network. FECs can be created from any combination of source and destination IP address, transport protocol, port numbers etc. Labels are assigned to incoming packets using a FEC to label mapping procedure at the edge routers. From that point on it is only the labels that dictate how the network will treat these packets, such as what route to use, what priority to assign, and so on.

Within a domain, a label switching router (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 outgoing label replaces the incoming label, and the packet is switched to the next LSR. Before a packet leaves a MPLS domain, its MPLS header is removed [5]. Figure 3.2 illustrates the mentioned scenario so far. A fundamental concept in MPLS is that two LSRs must agree on the meaning of the labels used to forward traffic between and through them. This common understanding is achieved by using a set of procedures, called a label distribution protocol (LDP), by which one LSR informs another of label bindings it has made [29,30]. Labels are maps of the network layer routing to the data link layer switched paths. LDP helps in establishing an LSP by using a set of procedures to distribute the labels among the LSR peers.

LDP provides an LSR discovery mechanism to let LSR peers locate each other and establish communication. It defines four classes of messages:

- DISCOVERY messages run over UDP and use multicast HELLO messages to learn about other LSRs to which LDP has a direct connection. It then establishes a TCP connection and an eventual LDP session with its peers. The LDP sessions are bi-directional. The LSR at either end can advertise or request bindings to or from the LSR at the other end of the connection.
- ADJACENCY messages run over TCP and provide session initialisation using the INITIALISATION message at the start of LDP session negotiation. This information includes the label allocation mode, keep alive timer values, and the label range to be used between the two LSRs. LDP keep alive are sent periodically using KEEP ALIVE messages. Teardown of LDP sessions between peer LSRs results if the KEEP ALIVE messages are not received within the timer interval.
- LABEL ADVERTISEMENT messages provide label-binding advertisements using LABEL MAPPING messages that advertise the bindings between FECs and labels. LABEL WITHDRAWAL messages are used to reverse the binding process. LABEL RELEASE messages are used by LSRs that have received label- mapping information and want to release the label because they no longer have a need for it.
- NOTIFICATION messages provide advisory information and also signal error information between peer LSRs that have a LDP session established between them.

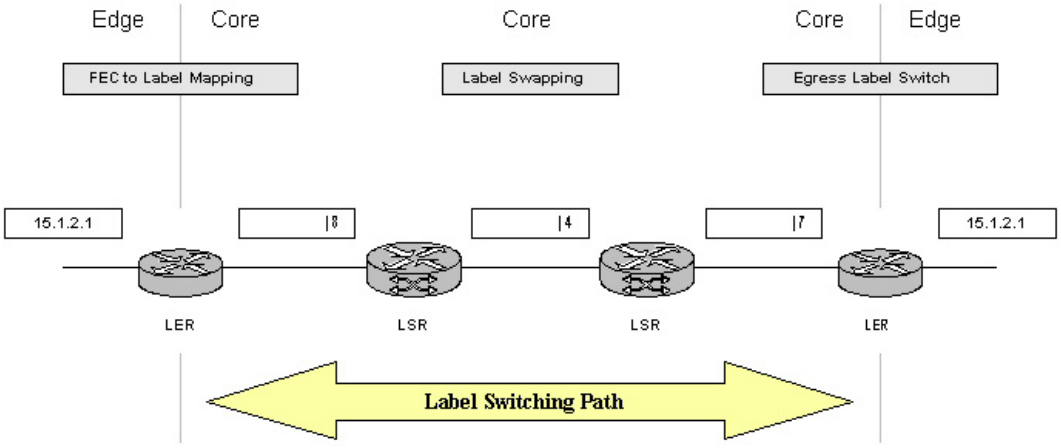

Figure 3.2 *Illustrating the label-switching path scenario*

MPLS allows routing control capabilities introduced in IP networks. These capabilities support connection control through explicit label- switched paths (LSPs). An explicit LSP is determined at the ingress LSR. This kind of connection control permits explicit routes to be established which are independent of the destination based IP shortest path routing mechanism [2]. Once an explicit route is determined, a signalling protocol is then used to set up the path. LDP as described earlier can be used for signalling purpose. A complete coverage of the signalling process is described later in chapter 3.2.3.

In MPLS networks, traffic trunks are set up in the network topology through the selection of routes for explicit LSPs. The terms LSP tunnel [3] and traffic-engineering tunnel (te-tunnel) [4] are commonly used to refer to the combination of traffic trunk and explicit LSPs in MPLS [2]. LSP tunnels are useful when dealing with the congestion problem mentioned. Multiple LSP tunnels can be created between two nodes, and traffic between them can be divided among the tunnels according to some local policy. Figure 3.3 illustrates a scenario where LSP tunnels are configured to redistribute traffic to address congestion problems caused by shortest path IGPs described in chapter 2.

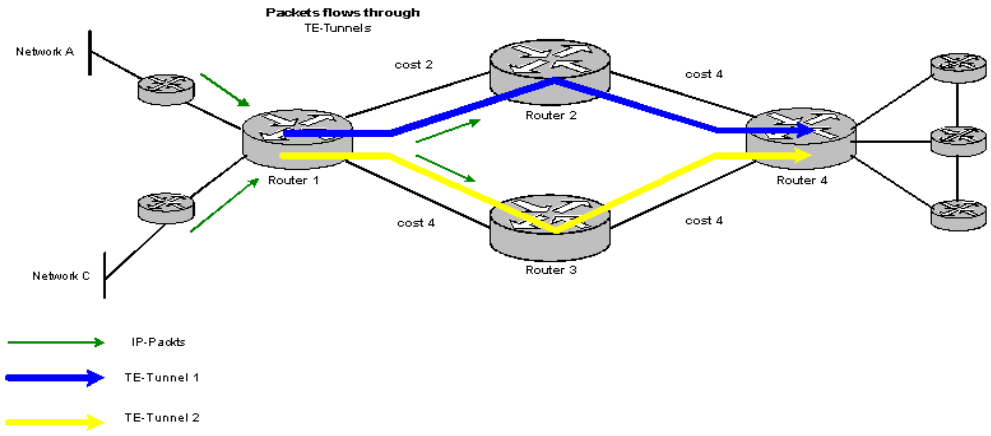

Figure 3.3 *Traffic trunks with LSPs*

3.2 Traffic engineering with MPLS

The challenge of traffic engineering is how to make the most effective use of the available bandwidth in a large IP backbone of an Internet Service Provider's network. MPLS traffic engineering routes IP traffic flows across a network based on the resources the traffic flow requires and the resources available in the network. This is unlike the shortest path routing protocols, which routes packets based on the shortest path to their destination. The main functional components for performing traffic engineering over MPLS are the distribution of network statistical information, path selection, path signalling and finally the packet forwarding mechanism. In this section, each of these components is described, to illustrate how MPLS can be used to perform traffic engineering.

3.2.1 Distribution of network statistical information

To achieve optimised traffic engineering, it is very important having access to up to date topology information. Therefore, distribution of network topology information is central for the remaining components of the functional parts of the MPLS control plane. This component is implemented as an extension to the conventional IGPs, so that link attributes are included as part of each router's link state advertisement. The standard flooding algorithm used by the link state IGP ensures that link attributes are distributed to all routers in the routing domain.
Each LSR maintain network link attributes and topology information in a database. This database is used by the path selection component to compute a desired route. Some of the traffic engineering extensions added to the IGP link state advertisement is maximum link bandwidth, maximum reserve-able bandwidth, current bandwidth reservation, current bandwidth usage, link colouring and interface IP address [8].

3.2.2 Path Selection

The next step in the process of traffic engineering by MPLS is to use the distributed information made by the flooding procedure of the IGP to compute and select the wanted paths. The information needed for this part of the component is collected from the database mentioned in the distribution of network statistical information component. Each LSR uses this database to calculate the paths for its own set of LSPs within the routing domain. The path for each LSP can be constructed either based on strict or loose explicit route. This allows the path selection process to work more freely whenever possible, but to be constrained when necessary.
Path selection must also take in consideration the constrained imposed by administrators of the domain. These constrained are usually related to the topology and resource usability. The path calculated by the path selection component may differ from the shortest path calculated by an IGP. The path selection component may consider several kind of information as input, such as topology link state information learned and stored in the database. Also attributes that consider the state of network resources such as total link bandwidth, reserved link bandwidth, and available link bandwidth are factors that it may consider important for its path selection calculation. Other considered information

attributes may be administrative related and is required to support traffic traversing the proposed LSP such as bandwidth requirements, maximum hop count and administrative policy requirements that are obtained from user configuration.

The result of the path selection is a route consisting of a sequence of LSR addresses that provides the shortest path through the network that meets the constraints. This calculated route is then used by the signaling component which then establishes forwarding state in the LSRs along the LSP.

The path selection component plays a very important role in traffic engineering. Both on-line and off-line calculation can be used for path selection. On-line calculation takes resource constraints into account and calculates one LSP at a time. It can calculate path quickly and adaptive to the change of the topology and resource information. Off-line planning and analysis tool simultaneously examines each link's resource constraints and the requirements of each ingress- to -egress LSP. It performs an over all calculations, compares the results of each calculation, and then selects the best solution for the network as a whole.

3.2.3 Signalling for path establishment

Path selection component described above computes a path that is thought to take into consideration some constraints appointed. However, the path is not operational until the LSP is actually installed by the signalling component. There are two options for the label distribution protocol. These two signalling protocols are defined as Resource Reservation Protocol (RSVP-TE) [34,37] with traffic engineering extensions and Label Distribution Protocol with constrained based extensions (CR-LDP) [31,32].

The first one relies on a number of extensions to the Resource Reservation Protocol (RSVP). The objective of extending RSVP is not only to support the establishment of explicit LSP tunnels with resource reservation, but also to support such attributes as reselecting and sustaining LSP tunnels [6]. It also watches out for loop detection [9]. It can automatically select the path and avoid the congested points and bottlenecks in the network. Three objects are used in this signalling protocol. The Explicit Route Object (ERO) allows an RSVP PATH messages to traverse a sequence of LSRs that is independent of conventional shortest path IP routing. The Label Request Object (LRO) permits the RSVP PATH message to request that intermediate LSRs provide a label binding for the LSP that it is establishing. The Label Object (LO) allows RSVP to support the distribution of labels without having to change its existing mechanisms. Because the RSVP RESV message follows the reverse path of the RSVP PATH message, the Label Object supports the distribution of labels from downstream to upstream nodes.

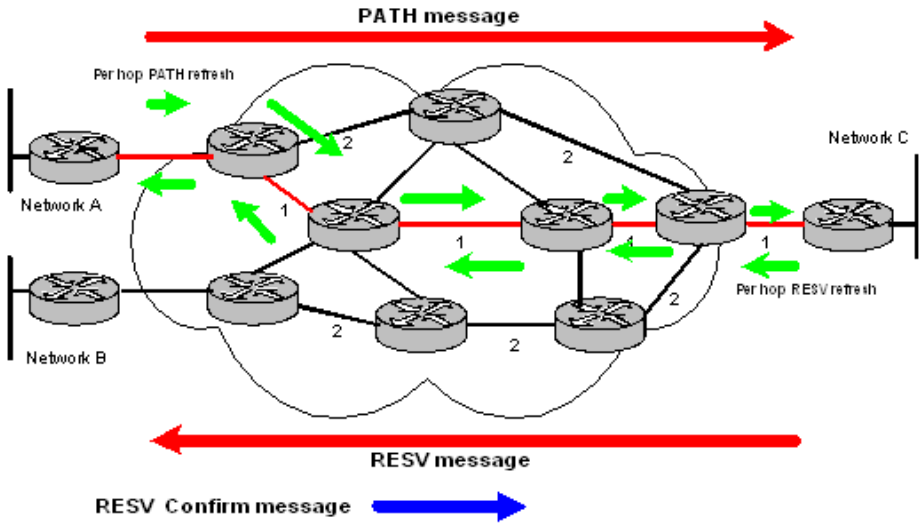

Figure 3.3.2. *Illustrates the RSVP-TE functionality*

In this example, having used BGP to discover the appropriate egress LER to route the traffic to another autonomous system (AS), the ingress LER initiates a PATH message to egress LER through each downstream LSR along the path. Each node receives a PATH message to remember this flow is passing, thus creating a "path state" or session. The egress LER uses the RESV message to reserve resources with traffic and QoS parameters on each upstream LSR along the path session. Upon receipt at the ingress LER, a RESV confirm message is returned to the egress LER confirming the LSP setup. After the loose ER-LSP has been established, refresh messages are passed between LERs and LSRs to maintain path and reservation states. It should be noted that, none of the downstream, upstream or refresh messaging between LER and LSRs is considered to be reliable, because UDP is used as the communication protocol.

TE-RSVP features are robust and provide significant capabilities to provide traffic- engineering functions to MPLS.
These includes:

- QoS and traffic parameters – for traffic management.
- Failure alert – when failing to establish an LSP or loss of an existing one, will trigger an alert message.
- Failure recovery – "make before break" when rerouting.
- Loop detection – required for loosely routed LSPs only, also supported for re-path establishing.
- Multi Protocol support - supports any type of protocol.
- Management – LSP ID identifies each LSP, thereby allowing ease of management to discrete LSPs.
- Record Route Objects – Provide the ability to describe the actual setup path to interested parties.
- Path Pre-emption – The ability to "bump" or discontinue an existing path so that a higher priority tunnel may be established.

The second signalling protocol, which is called the CR-LDP, is specifically designed to facilitate constrained based routing of LSPs [10]. Like Label Distribution Protocols (LDP), it uses TCP sessions between LSR peers and sends label distribution messages along the sessions. If we review figure 3.2, but this time illustrate how the forwarding labels where engineered in the first place, we can understand the functionality behind CR-LDP. Figure 3.4 illustrates the CR-LDP scenario.

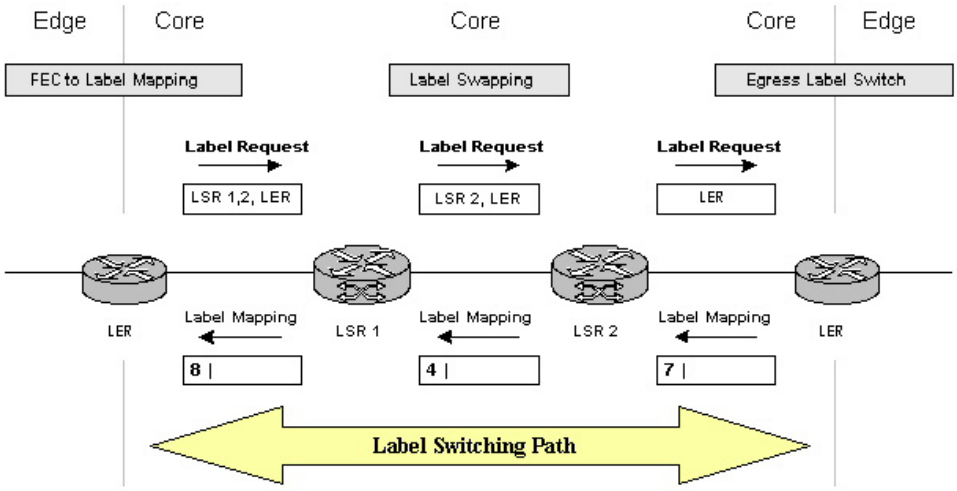

Figure 3.4 *Illustrating the CR-LDP scenario*

As figure 3.4 illustrates, the ingress LER determines that it needs to set up a LSP to egress LER. The traffic parameters required for the session or administrative policies for the network enable LER to determine that the route for the wanted LSP should go through LSR1 and LSR2. The ingress LER builds a label request message with an explicit route of {LSR1, LSR2, LER} and details of the traffic parameters requested for the route. The ingress LER reserves the resources it needs for the LSP, and the forward the label request to LSR1. When LSR1 receives the label request message, it understands that it is not the egress for this LSP and makes the necessary reservation before it forwards the packet to the next LSR specified by the message. The same processing takes place at the LSR2, which is the next LSR along the wanted LSP. When the label request message arrives at the egress LER, the egress LER determines that it is the egress for this LSP. It performs any final negotiation on the resources, and makes the reservation for the given LSP. It allocates a label to the new LSP and distributes the label message to the last know LSR2 where the message arrived from. This label is packed in a message called the label- mapping message, which contains details of the final traffic parameters reserved for the LSP. LSR2 and LSR1, respectively receives the label mapping message and matches it to the original request using

the LSP ID contained in both the label request and label mapping messages. It finalizes the reservation, allocates a label for the LSP, sets up the forwarding table entry, and passes the label to ingress LER in a label- mapping message. The processing at the ingress LER is similar, beside that it does not have to allocate a label and forward it to an upstream LSR or LER since it is the ingress for the LSP.

CR-LDP traffic engineering extensions to LDP feature set is comprehensive and is fairly well defined.
These includes:

- QoS and Traffic Parameters – the ability to define edge rules and per hop behaviours based upon data rates, link bandwidth and weighting given to those parameters.
- Path pre-emption – the ability to set prioritisation to allow or not allow pre-emption by another LSP.
- Path re-optimisation – allows for the capability to re-path loosely routed LSPs based upon traffic pattern changes and includes the option to use route pinning.
- Failure alert – upon failure to establish a LSP, alert is provided with supporting failure codes.
- Failure recovery – mapping policies to automatic failure recovery at each device supporting a LSP.
- Management – LSP ID identifies each LSP, thereby allowing ease of management to discrete LSPs.

3.2.4 Packet forwarding

This component is responsible of forwarding packets. It forwards packets based on the decisions that the path selection and path- signalling component have made. Here, traffic is allocated to established LSP tunnels. This functional component consists of a partitioning function and an apportionment function. The partitioning function partitions ingress traffic according to some principle of division and the apportionment function sends the partitioned traffic to established LSP tunnels according to some principle of allocation [2]. In this way one can achieve load sharing. I refer again to figure 3.2 where forwarding of packets is illustrated. Packets entering the MPLS domain gets assigned MPLS labels while they are switched form one LSR to another, following an established LSP path before they leave the domain with their original destination network layer address.

3.2.5 Rerouting

In a traffic- engineered network, one must expect the network to be able to respond to changes in the network topology and maintain certain stability. Any link or node failure should not disrupt high-priority network services, especially the higher classes of service. Fast routing is a mechanism that minimizes service disruptions for traffic flows affected by an incident, and optimised rerouting re-optimises traffic flows affected by a change in topology.

In MPLS, splicing and stacking techniques are utilized to enable local repair of LSP tunnels. In the splicing technique, an alternative LSP tunnel is pre-established to the destination, from the point of protection via a path that bypasses the downstream network elements being protected. When detecting a failure at a link or a node, the forwarding entry of the protected LSP tunnel is updated to use the label and interface of the bypass LSP tunnel. The stacking technique creates a single alternative LSP tunnel, acting as the replacement for the failed link. It bypasses the protected link. The local router maintains a label that represents the bypass tunnel.

3.3 Quality of Service support with MPLS

Although the original idea behind the development of MPLS was to facilitate fast packet switching, currently its main goal is to support traffic engineering and provide quality of service (QoS). The goal of traffic engineering is to facilitate efficient and reliable network operations, and at the same time optimise the utilization of network resources. MPLS support this goal and enhance traffic oriented performance characteristics. For example, non-shortest paths can be chosen to forward traffic. Multiple paths can also be used simultaneously to improve performance from a given source to a given destination. Since it uses label switching, packets of different flows can be labelled differently and thus receiving different forwarding, and hence different quality of service.

Specific flows of traffic can then become aggregated to achieve a more scalable way to perform QoS support in the backbone of a service provider's network [36]. There are 3 bits dedicated for the QoS in the MPLS header. With these bits set in the header, LSRs can make the proper decision for provisioning QoS. MPLS has actually no functional method for assuring QoS, but it can be combined with Integrated Services or Differentiated Services to become complementing.

3.3.1 Integrated Services

IntServ, as it is also referred to, provides for an end-to-end QoS solution by way of end-to-end signalling [17]. IntServ specifies a number of service classes designed to meet the needs of different application types. RSVP [22] is an IntServ signalling protocol that is used to make requests for QoS using the IntServ service classes. The IntServ model [17] proposes two services classes in addition to best-effort services. The first one is guaranteed service [18] for applications requiring fixed delay bounds. The second one is controlled-load services [19] for applications requiring reliable and enhanced best-effort service. These service classes can be requested with help from the RSVP signalling protocol.

3.3.2 IntServ implementation with MPLS

MPLS can be enabled on LSRs by associating labels with flows that have RSVP reservations. Packets for which a RSVP reservation has been made can be considered belonging to one FEC. A label can identify each FEC. Bindings created between labels and the RSVP flows must be distributed among the LSRs. Figure 3.5 illustrates the scenario, where on receipt of an RSVP PATH message, the host respond with a standard RSVP RESV message. LSR3 recieves the RESV message and allocates a label and sends out an RESV message with a label object and the value of the label 7 to LSR2. The other LSRs in turn assign their label information associated with the FEC. As the RESV message precede, the LSRs and LSP is established along the RSVP path, making it possible for each LSR to associate QoS resources with the LSP.

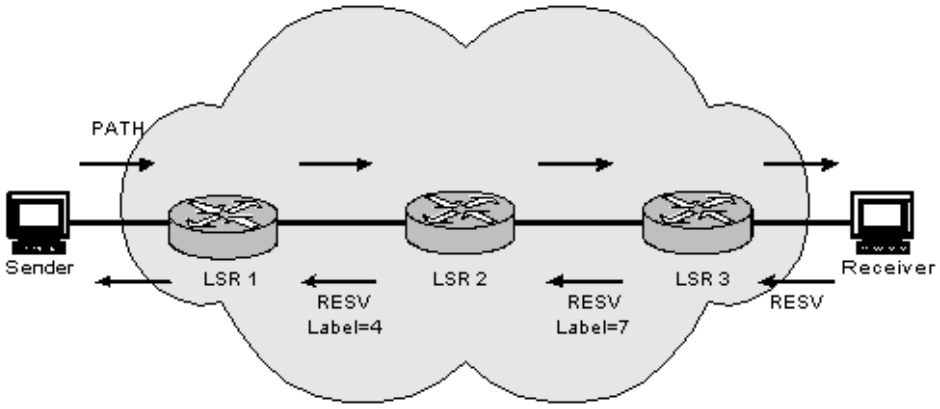

Figure 3.5 *MPLS PATH and RESV message flow*

3.3.3 IntServ scalability drawbacks

The IntServ RSVP per-flow approach to QoS described is clearly not scalable and leads to complexity of implementation. The philosophy of the IntServ model is that there is inescapable requirement for routers to be able to reserve resources in order to provide special QoS for specific user packet flows [16]. A problem with IntServ is the amount of state information stored in each router, which increases proportionally with the number of flows. This places a huge storage and processing overhead on the routers, thus not scaling well in the Internet core.

RSVP is referred to as a “soft state” protocol. After an initial LSP set-up process, refresh messages must be exchanged between peers periodically to notify the peers that the connection is still desired. If the refresh messages are not exchanged, a maintenance timer senses the connection as unwanted to continue

and deletes the state information, returns the label and reserved bandwidth to the resource pool and notifies the effected peers. The "soft state" approach can be viewed as a self –cleaning process since all expired resources eventually are freed.

It is stated in [35], that the RSVP Refresh overhead is seen as a fundamental weakness in the protocol and therefore not scalable. This issue rises when supporting numerous small reservations on high bandwidth links, since the resource requirements on a router increases proportionally. Extensions are made to the RSVP to try to overcome this problem with defined RSVP objects that are sent inside standard RSVP messages. To reduce the volume of exchanged messages between two nodes, an RSVP node can group a number of RSVP refresh messages into a single message. This message is sent to the peer router where it is disassembled and each refresh message is processed. In addition, the MESSAGE_ID and MESSAGE_ID_ACK objects have also been added to the protocol. These objects are used to hold sequence numbers corresponding to previously sent refresh messages. While the peer router receives a refresh message with a non-changing MESSAGE_ID, it assumes that the refresh state is identical to the previous message. Only when the MESSAGE_ID value changes does the peer router have to check the actual information inside the message and act accordingly. To further enhance the summarization process, sets of MESSAGE_ID's can be sent as a group to the peer router in the form of "summary messages". While this strategy will substantially decrease the time spent exchanging information between the peer routers, it does not eliminate the computing time required to generate and process the refresh messages them-selves. Time must still be spent checking timers and querying the state of each RSVP session. In short, the scalability issues of RSVP has some how been addressed, but not fully.

3.3.4 Differentiated Services

DiffServ as it is also referred too emerged because of the drawbacks mentioned with the IntServ model and RSVP. In the Differentiated Service model [21], IPv4 header contains a Type of Service (ToS) byte. In the standard ToS based QoS model, packets are classified at the edge of the network into one of eight different classes. This is accomplished by setting three precedence bits in the ToS (Type of Service) field of the IP header. The three precedence bits are mainly used to classify packets at the edge of the network into one of the eight possible categories listed in table 3.2.

Number	Name
0	Routine
1	Priority
2	Immediate
3	Flash
4	Flash override
5	Critical
6	Internet control
7	Network control

Table 3.2 *IP Precedence values*

IP Precedence	DSCP
IP precedence 0	DSCP 0
IP precedence 1	DSCP 8
IP precedence 2	DSCP 16
IP precedence 3	DSCP 24
IP precedence 4	DSCP 32
IP precedence 5	DSCP 40
IP precedence 6	DSCP 48
IP precedence 7	DSCP 56

Table 3.3 *IP Precedence to DSCP Mapping*

However, choices are limited. Differentiated Services defines the layout of the ToS byte (DS *field*) and a basic set of packet forwarding treatments (per-hop behaviours) [20]. Marking the DS fields of packets differently and handling packets based on theirs DS fields; one can create several differentiated service classes. A 6-bit differentiated service code point (DSCP) marks the packet's class in the IP header. The DSCP is carried in the ToS byte field in the IP header. 6-*bit* can result in the implementation of 64 different classes. As shown in table 3.3, IP precedence levels can be mapped to fix DSCP classes. [20,21], define the DiffServ architecture and the general use of bits within the DS field. This supersedes the IPv4 ToS octet definitions of [25].

In order for a customer to receive differentiated services from its Internet Service Provider (ISP), it must have a service level agreement (SLA) with its ISP. An SLA is a specification of the service classes supported and the amount of traffic allowed in each class. It can be static or dynamic. Static ones are negotiated on a monthly/yearly basis. If dynamic, a signalling protocol such as RSVP must be used to request services on demand.

Differentiated services are significantly different from integrated services. First, there are only a limited number of service classes indicated by the DS field. This makes it more scalable, since the amount of state information is proportional to the number of classes rather than the number of flows. Second, sophisticated classification, marking, policing, and shaping operations are only needed at the boundary of the networks. ISP core routers need only to have behaviour aggregate classification. Therefore, it is more scalable to implement and deploy differentiated services.

3.3.5 Per-Hop Behaviour (PHB)

As illustrated in figure 3.6, network elements or hops along the path examine the value of the DSCP field and determine the QoS required by the packet. This is known as per-hop behaviour (PHB). Each network element has a table that maps the DSCP found in a packet to the PHB that determines how the packet is treated. The DSCP is a number or value carried in the packet, and PHBs are well-specified behaviours that apply to packets. A collection of packets that have the same DSCP value, and crossing a network element in a particular direction, is called a Behaviour Aggregate (BA). PHB refers to the packet scheduling, queuing, policing, or shaping behaviour of a node on any given packet belonging to a BA.

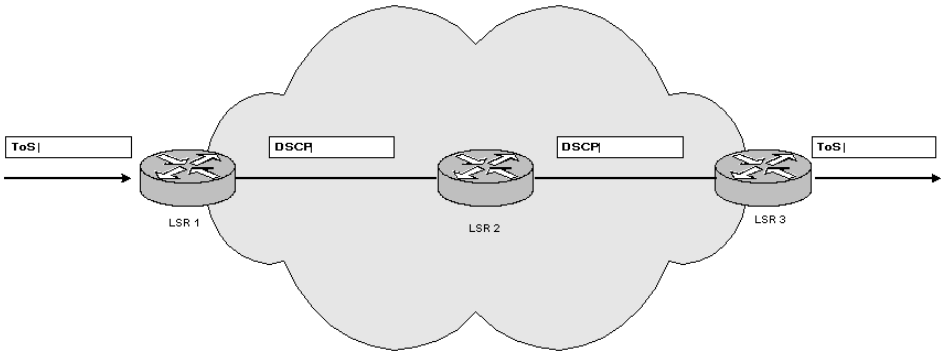

Figure 3.6 *PHB based on DSCP value*

Four standard PHB implementations of DiffServ are available:

Default PHB
The default PHB results in a standard best-effort delivery of packets. Packets marked with a DSCP value of 000000 get the traditional best-effort service from a DS-compliant node. Also, if a packet arrives at a DS-compliant node and its DSCP value is not mapped to any of the available PHBs, it is mapped to the default PHB.

Class-Selector PHB
In order to preserve backward compatibility with ToS based IP QoS schemes, DSCP values of the form xxx000 are defined (where x equals 0 or 1). Such code points are called class-selector codepoints. The default code point 000000 is a class-selector codepoint. The PHB associated with a class-selector code point is a class-selector PHB. These PHBs retain almost the same forwarding behaviour as nodes that implement IP QoS classes based on the ToS classification and forwarding. As an example, packets that have a DSCP value of 101000 (IP ToS = 101) have a preferred forwarding treatment as compared to packets that have a DSCP value of 011000 (IP ToS = 011). These PHBs ensures that DS-compliant nodes can coexist with IP ToS-based aware nodes.

Expedited Forwarding (EF) PHB
The DSCP marking of EF, results in expedited forwarding with minimal delay and low loss of packets. These packets are prioritised for delivery over others. The EF PHB in the DiffServ model provides for low packet loss, low latency, low jitter and guaranteed bandwidth service. EF can be implemented using priority queuing, along with rate limiting on the class. According to [38], the recommended DSCP value for EF is 101110.

Assured Forwarding (AF) PHB
The DSCP marking of AF packets specifies an AF class and drop preference for IP packets. Packets with different drop preference within the same AF class are dropped based on their relative drop precedence values within the AF class [26]. Also [26] recommends 12 AF PHBs representing four AF classes with three drop-preference levels in each.

The Assured Forwarding PHB defines a method by which BAs can be given different forwarding assurance. The AFxy PHB defines four classes: AF1y, AF2y, AF3y and AF4y. Each class is assigned a certain amount of buffer space and interface bandwidth, dependent on the customer's SLA with its service provider. Within each AFx class, it is possible to specify three-drop precedence values. If there is congestion in a DiffServ enabled network element on a specific link, and packets of a particular AFx class need to be dropped, packets are dropped such that dp(AFx1)<=dp(AFx2)<=dp(AFx3), where dp(AFxy) is the probability that packets of the AFxy class will be dropped. The subscript y in AFxy denotes the drop precedence within an AFx class. For example, packets in AF23 get dropped before packets in AF22 and before packets in AF21. Table 3.4 shows the DSCP values for each class, and drop precedence. According to [26], an AFx class can be denoted by the DSCP xyzab0, where xyz is 001, 010, 011 or 100, and ab represents the drop precedence bits.

Drop Precedence	**Class 1**	**Class 2**	**Class 3**	**Class 4**
Low drop precedence	(AF11)	(AF21)	(AF31)	(AF41)
	001010	010010	011010	100010
Medium drop precedence	(AF12)	(AF22)	(AF32)	(AF42)
	001100	010100	011100	100100
High drop precedence	(AF13)	(AF23)	(AF33)	(AF43)
	001110	010110	011110	100110

Table 3.4 *Diffserv AF codepoint table*

3.3.6 DiffServ implementation with MPLS

MPLS LSRs do not examine the contents of the IP header and the value of its DSCP field as required by DiffServ. This means that the appropriate PHB must be determined from the label value. The MPLS shim header has a 3-*bit* field called EXP. It was originally defined for experimental use. This field supports eight different values and is used for MPLS support of up to eight DiffServ classes. As illustrated in figure 3.7, the IP precedence bits from the ToS field or the first 3-*bits* of the DSCP field are copied into the MPLS EXP field at the ingress router. Each LSR along the LSP maps the EXP bits to a PHB. The service provider can also set an MPLS packet's CoS to a different value, as determined by a service offering. This feature allows the service provider to set the MPLS EXP field instead of overwriting the value in the customer's IP ToS or DSCP field. This leaves the IP header intact and available for the customer's use. The customer configured CoS is not changed as the packet travels through the MPLS configured network. The LSPs created this way are known as E-LSPs or explicit-LSPs. E-LSPs are established before any traffic gets to use it. E-LSPs can support up to eight PHBs per LSP.

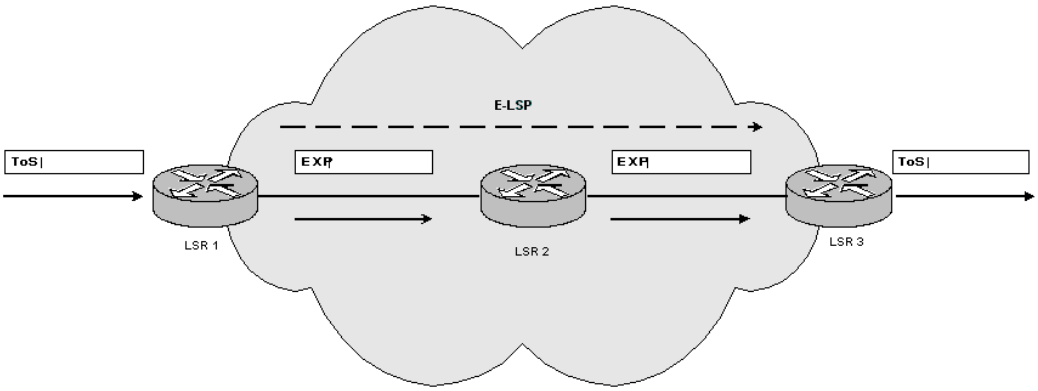

Figure 3.7 *MPLS E-LSP*

As illustrated in Figure 3.8, if more than 8 PHBs are needed in the MPLS network, L-LSPs (Label LSPs) are used, in which case the PHB of the LSR is inferred from the label. The label to PHB mapping is signalled. Only one PHB per L-LSP is possible, except for DiffServ AF. In the case of DiffServ AF, packets sharing a common PHB can be aggregated into a FEC, which can be assigned to an LSP. This is known as a PHB scheduling class. The drop preferences are encoded in the EXP bits of the shim header, as illustrated in figure 3.8.
E-LSPs are more efficient than L-LSPs, because the E-LSP model is similar to the standard DiffServ model. Multiple PHBs can be supported over a single E-LSP. The total number of LSPs created can be limited, thus saving label space.

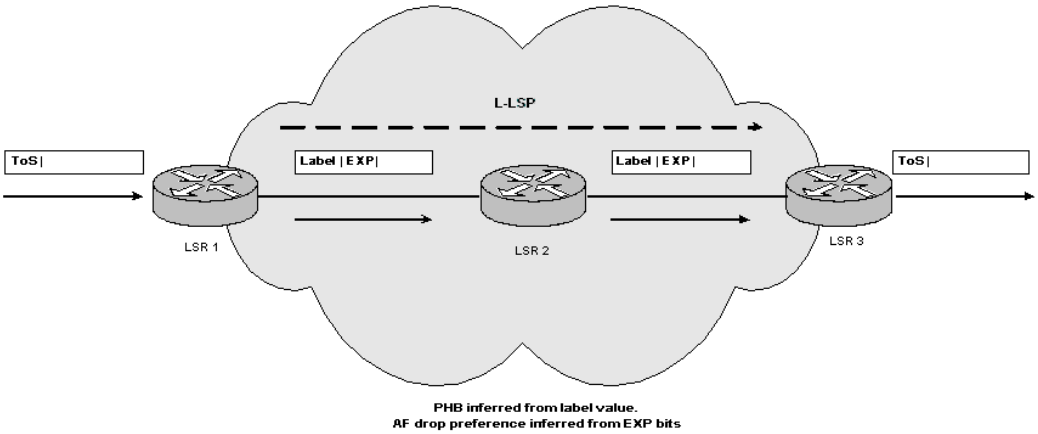

Figure 3.8 *MPLS L-LSP*

3.3.7 Aggregation of traffic flows with MPLS and Diffserv

Traffic flows are referred to as unidirectional stream of packets [36]. Typically a flow has very fine granularity and reflects a single interchange between hosts that communicates. An aggregated flow is a number of flows that share forwarding state and a single resource reservation along a sequence of routers.

With MPLS and differentiated services, packets get classified and forwarded through established LSPs. Traffic classes are separated based on the service level agreements. Priority traffic is likely to come in many flavours, depending on the application. Particularly flows may require bandwidth guarantees, jitter guarantees, or upper bounds on delay. For the purpose of this thesis, we will not distinguish the subdivision of priority traffic. All priority traffic is assumed to have an explicit resource reservation. When flows are aggregated according to their traffic class and then the aggregated flow is placed inside a LSP, we call the result a traffic trunk, or simply a trunk. Many different trunks, each with its own traffic class, may share an LSP if they have different traffic classes.

As described, packets may fall into a variety of different traffic classes. For ISP operations, it is essential that packets be accurately classified before entering the ISP backbone and that it is very easy for a ISP ingress router to determine the traffic class for a particular packet. The traffic class of MPLS packets can be encoded in the three bits reserved for experimental use within the MPLS label. In, addition, traffic classes for IP packets can be classified via the ToS byte, possibly within the three precedence bits within that byte.

As, described above, traffic of a single traffic class that is aggregated into a single LSP is called a traffic trunk, or simply a trunk. Trunks are very useful within the architecture because they allow the overhead in the infrastructure to be decoupled from the size of the network and the amount of traffic in the network [36]. While the size of the traffic scales up, the amount of traffic in the trunks increases, but the number of trunks doesn't. In a given network topology, the worst case would be to have a trunk for every traffic class from each ingress router to each egress router. If there exist N routers in the topology and C classes of service, this would be (N* (N-1) * C) -trunks. To make this more scalable its stated in [36], that trunks with a single egress point which share a common internal path can be merged to form a single tree. Since each sink tree created touches each router at most once and there is one sink tree per egress router, the result is N * C sink trees. Also the number of sink trees can be reduced if multiple sink trees for different classes follow the same path. This works because the traffic class of a sink tree is orthogonal to the path defined by its LSP. This makes it possible for two trunks with different traffic classes to share a label for any part of the topology that is shared and ends in the egress router. This again forces out that the entire topology can be overlaid with N trunks.

MPLS and diffserv are actually very complementing in the process of supporting traffic trunks by aggregating traffic flows and placing these in LSPs established. MPLS can thus make the route for the flows of packet entering a service provider's network. Diffserv in other hand can decide which treatment a packet will get while travelling between routers along the LSPs. Therefore, flows with different CoS can be aggregated and engineered through the backbone by the MPLS and diffserv architecture.

3.4 Summary over MPLS Traffic Engineering and QoS Support

Multi protocol label switching (MPLS) is an emerging technology that aims to address many of the existing issues associated with packet forwarding in today's Internetworking environment. As stated in this chapter, it can be used to engineer traffic, and also combined with diffserv assure QoS support to traffic.

MPLS traffic engineering mechanism takes place by establishing LSPs that can carry traffic through desired path. Packets get classified when entering the ingress router at the edge of the MPLS enabled network. When classified, they are assigned a MPLS header by their FEC class, which helps them to get engineered through the network. When traffic is engineered, the flowspec configured governs the traffic characteristic and requested class of service implied to it. These flowspecs govern the type of class, amount of traffic allowed to enter, and other details of traffic imposed to the ingress router to be engineered through a LSP. Figure 3.9 illustrates the way flowspecs function through a LSP. Combined with differentiated services, one can achieve traffic engineering with QoS support. LSPs are first configured between each ingress-egress pair. For each traffic class, a flowspec may be installed. As the number of transmitting flow increases, the number of flows in each LSP increases. But the number of LSPs or flowspecs does not need to increase.

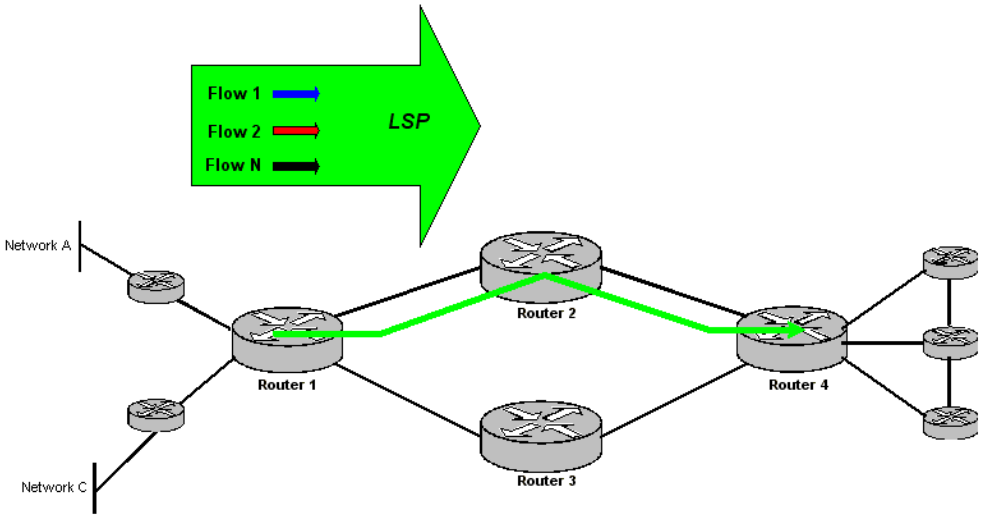

Figure 3.9 *Flows within a LSP*

Traffic engineering is the process of arranging traffic flows through the network so that congestion caused by uneven network utilization can be avoided. Avoiding congestion and providing graceful degradation of performance in congestion are complementary. Traffic engineering therefore complements differentiated services. In summary MPLS will set up a route for a flow and at the same time govern the amount of traffic allowed into the network, it specify the next hop for a packet, while differentiated services will specify the treatment of a packet waiting to make that next hop.

4 Introduction to simulation

To begin with we did an experiment with a network configured to run shortest path routing protocol OSPF. We considered this necessary in order to gain experience with the simulation tool and to highlight some of the shortest path routing principal as mentioned earlier in this thesis. However, we chose not to elaborate the results extensively because of the focus of this thesis on traffic engineering topic and the available time to us. Furthermore, we experimented with MPLS architecture to experience its features of traffic engineering. Intention was to investigating the treatment of this protocol on flows of traffic getting engineered. We then move on fine-tuning the MPLS configured network to also take into consideration the QoS aspects of traffic flows within a traffic-engineered path.

4.1 Simulation tool

Optimised Network Performance (OPNET) [14] is a discrete event simulation tool. It provides a comprehensive development environment supporting the modelling and simulation of communication networks. This contains data collection and data analysis utilities. OPNET allows large numbers of closely spaced events in a sizeable network to be represented accurately. This tool provides a modelling approach where networks are built of nodes interconnected by links. Each node's behaviour is characterized by the constituent components. The components are modelled as a final state machine. We have chosen to use OPNET as our simulation tool. Details of OPNET Modeler 8.1 can be found in [23,24]

Our objective with using this simulation tool for our experiments, were to gain a better understanding of its use for further research and simulation of communication systems. We therefore used a lot of time and energy understanding using it as a simulation tool. The time and energy spent on leaning to master the tool did however not have anything to do with the software user friendliness, contrary it is quite well arranged to provide and represent all the functionality it beholds.

4.2 Network topology

Figure 4.1 illustrates our networking topology. The network topology used in our experiments was designed to be simple. This was chosen due to the time consuming simulation. The network topology cannot be said to be a realistic operational network. However, our intention was to create a networking environment, which could represent a part of an overall network topology of an ISP network. The model suite supported workstations, servers, routers, and link models. We used access routers at the edge of the network where the traffic was transmitted to or received from the sites. The core routers were configured to

handle traffic from the edge routers. We used DS1 links between all networking devices, meaning that the maximum throughput was set to 1,544,000 bits/sec. The sites were actually workstations and server transmitting or/and-receiving data. We have chosen to call them sites, because they could have behaves as theirs own networking environment connected to a service provider edge router.

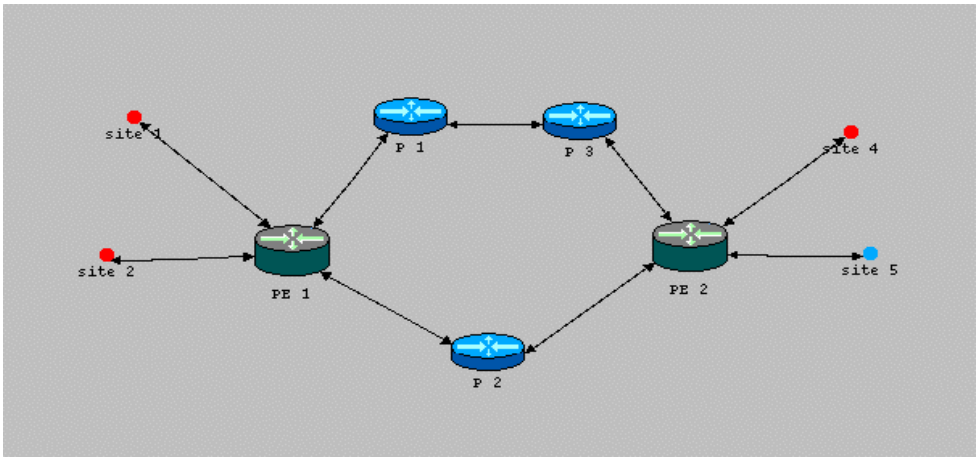

Figure 4.1 *Overview of the experiential network model.*

4.3 General experimental conditions regarding all simulation scenarios

We configured applications, which used TCP and UDP as their transport protocol. With these applications generating traffic, our intention was to measure the treatment of these traffic types when shortest path routing, MPLS-TE and MPLS-TE with QoS support is implied. Since most of the traffics getting transmitted in today's Internet use TCP or UDP as transport protocols, these protocols were the right choice for experiments within our simulations.

We gave the network approximately two minutes before traffic generation was triggered. This was done to make sure the routers had enough time to exchange topology information and building up their routing tables. Of course, we knew that this was not necessary in a small networking environment as the one we configured. However, we did not managed to get the software simulation program to start generating traffic earlier than 100 seconds. From the second minute, file transfer application was triggered to start, making TCP to transport its packets through the network. TCP traffic intensity was set to 1,5 *Mbytes/sec* of files uploaded to the server. This gave us the intensity of 1,500,000 *bits/sec*. The other application was set to start one second later transporting its packets with UDP transfer protocol. There were configured five application of this sort, with exactly same configuration triggered to start one second after each other. The reason for this was that we wanted to cautiously each time increase the same UDP traffic intensity to measure its impact on TCP traffics. UDP packets were set to 37500

bytes/sec. This gave us the traffic intensity of 300,000 *bits/sec* multiply by five applications achieving 1,500,000 *bits/sec* of traffic intensity.

The maximum transmission unit (MTU) was set to the Ethernet value of 1500 *bytes*. The MTU specify the IP datagram packet that can be carried in a frame. When a host send an IP datagram, therefore, it can choose any size that it wants. A reasonable choice is the MTU of the network to which the host is directly attached. Then a fragmentation will only be necessary if the path to the destination includes a network with a smaller MTU. Should the transport protocol that sits on top of IP give IP a packet larger than the local MTU, however, then, the source host must fragment it. The packets sent from the file transfer application, was set to 1,5 *Mbytes*. However, we configured the interfaces on routers and workstations to segment the file in Ethernet values. This was a realistic thing to do, since uploading raw IP packets of such large sizes would not be very realistic. The maximum massage size of TCP was set to auto assigned, meaning that the IP value would be used. For a complete, detail coverage of the TCP configuration parameter we refer to appendix 9.2.

Referring to figure 4.1, site1 was to communicate with site5 using the file transfer application, meaning it would start generating the TCP traffic intensity described above at the second minute. Site2 in other hand, were to use the video conferencing application, thus making it to generate UDP traffic one second later. The UDP traffic was transmitted to site4, which accepted video conferencing related UDP traffic. Table 4.1 summarize the traffic intensity configured for use within all simulation experimentations made within this thesis.

Site	*Supported Protocol*	*Start*	*End time*	*Traffic-intensity*
Site1	TCP	2m:00s	2m: 06s	1,500,000 bits/sec
Site2	UDP	2m:01s	2m: 06s	300,000 bits/sec
Site2	UDP	2m:02s	2m: 06s	300,000 bits/sec
Site2	UDP	2m:03s	2m: 06s	300,000 bits/sec
Site2	UDP	2m:04s	2m: 06s	300,000 bits/sec
Site2	UDP	2m:05s	2m: 06s	300,000 bits/sec
Site4	UDP	N/A	N/A	N/A
Site5	TCP	N/A	N/A	N/A

Table 4.1 *A summarization over the traffic configuration*

Beside these general configurations made, each simulation experiment is also configured with its own set of specific configurations. These simulation specific configuration details are outlined within each simulation experiment chapter. For more detail regarding all simulation scenarios with their respective configuration details within OPNET Modeler, we refer to appendix [9.2-9.4].

5 Simulation experiment using OSPF

The first scenario was created to obtain experience with the simulation tool, while at the same time highlight some of the shortest path routing principal as mentioned in chapter 2.2. Specifically, we aimed to investigate throughput and queuing delay issues, when traffic flows compete for scarce resources under overloaded situations. In this scenario, there were not given any quality of service guarantee to neither of the traffic types. Therefore, no traffic entering the network would be given any quality of service support. The type of service field of the packets was therefore set to (0) precedence class, which qualify for the best effort service class. All the routers were configured using only Open Shortest Path First (OSPF) as their routing protocol. Details over configurations of network nodes and traffic implementations within OPNET can be reviewed in appendix 9.2.

5.1 Analysing and discussing experimental results

The results collected from within OPNET, is shown below. From our experimentation, statistical data were collected concerning throughput and queuing delay measured from the simulated network. Our objective here is to analyse and discuss the results gathered from measurements registered. We are not going to elaborate these results extensively since our focus is concentrated around the traffic engineering part of this thesis and the time available to us was unfortunately scarce.

5.1.1 Throughput

As recalled, site1 was configured to generate TCP traffic from the second minute. The amount of this traffic was 1,500,000 *bits/sec*. Observing collected statistics from figure 5.1, we witnessed that this value was reached. A second later site2 started generating UDP traffic of size 300,000 *bits/sec*, and each second after this intensity was been increased with 300,000 *bits/sec*. Keeping in mind that both traffic utilised their links towards the ingress router, we registered that the UDP traffic intensity had a tremendous effect on the TCP traffic intensity. These effects were registered between sites and the ingress router PE1 every time UDP- traffic intensity was increased. Figure 5.1, shows that the TCP throughput starts falling, when the UDP traffic starts generating traffic. This force's the TCP throughput fall down below 750,000 *bits/sec* from its originating 1,500,000 *bits/sec* within the time frame of this simulation. The UDP traffic does not care about congestion within the network, continuing transmitting its traffic regardless of packets managing to arrive at the intended destination. The UDP traffic starts consuming resources and stabilizes not before it has reached its maximum traffic intensity at 1,500,000 *bits/sec*.

Figure 5.2 shows the amount of packets sent from the clients towards the server. Observing the registered result we witnessed that each times UDP- traffic increases its traffic intensity; the TCP traffic intensity lowers its intensity equally.

However, some increase was registered right after such incidents. We believe that these increases of intensity made by TCP after each decreases are related to the fast retransmit option of TCP RENO implementation. Since TCP registers that it after an intensity decrease manages to receive acknowledgements for some of its transmitted packets, its immediate reaction is to starts transmitting more packets again. Also, there were registered some slightly decrease amount of packets sent from the UDP generating site. This was interesting since we imagined that UDP traffic wouldn't decrease its traffic intensity under overload conditions. However, these decreases is considered to be very small and takes place under a second each time. More time and effort is needed to investigate this phenomenon in more details. Each time such decreases take place we witness some increase from the TCP generator. It all happens in a matter of mille seconds. It would be interesting to investigate the TCP congestion window details and fast retransmit option of it in more details. Unfortunately, we didn't have the time to elaborate further on this issue since our work was to be concentrated on the traffic engineering and QoS aspect of the MPLS architecture. The results presented here should be kept in mind when results from MPLS traffic engineering are presented later on, to be able to acknowledge the performance benefits of traffic engineering.

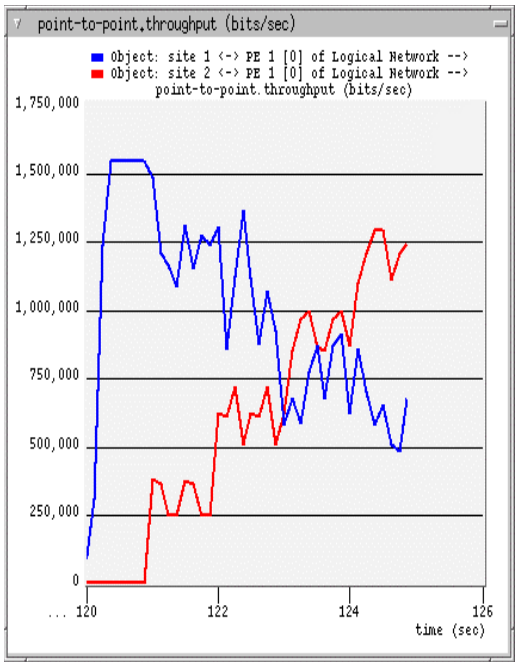

Figure 5.1 *TCP and UDP Throughput (bits/sec)*

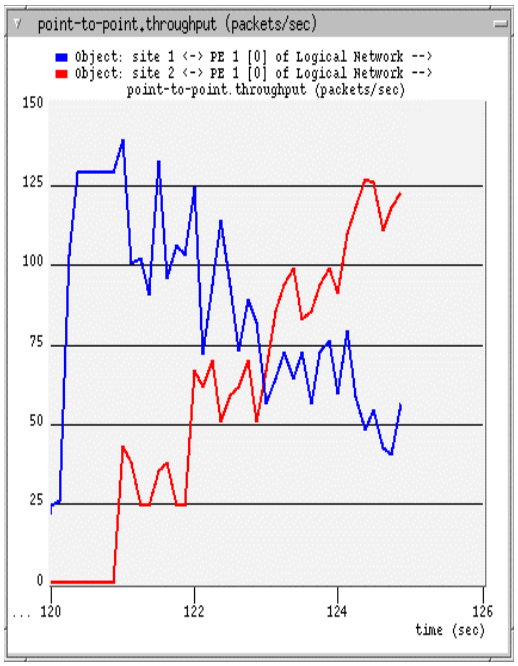

Figure 5.2 *TCP and UDP Throughput (packets/sec)*

The other QoS statistical related result gathered from our simulation were the throughput measured from paths between routers that handled the traffic flows. Figures 5.3 and 5.4 below, shows the results gathered from our simulation. We observed that the throughput between routers combining one path (PE1 → P1→ P3 → PE2), were unutilised, while the other path (PE1 → P2 → PE2) were fully utilised. To us, this indicated weakness of the protocol functionality, when it

came to load balance the traffic. We observed that one path's throughput is nothing compared with the other one, which obviously needs more capacity. From the below figures, we observed that the non-shortest path had a stable amount of zero throughput. The shortest path however, had a throughput of maximum 1,544,000 *bits/sec* that its links allowed it to carry. From figure 5.4 below, we observed a slightly drop off value between the 121 and 122 seconds. We don't know whether this was related to the simulation software or not. However, we find it little interesting to elaborate further on this registered result. If it were to exactly strike at the 121 second, we could have been related it to the time when UDP traffic starts generating traffic. Nevertheless, this could still be related to the registered result, only a fraction of mille second late. The overall picture that we aimed to show here was the fact that the routing protocol did not utilise the network resources efficiently at times were traffic load conditions are heavy, utilising only the shortest path between any pair of ingress and egress routers. With this functionality implied, bottlenecks arise and congestion takes place within the network. If the network topology were more complex and other traffic was forwarded from other routers and utilised this path towards some destination, the results may have been even worst from the ones we registered. In the real world of ISP networks, different traffic types may end up utilising the same shortest path, making it possible to achieve the same negative results at any point between any routers that gets to become part of a shortest path. This force out congestion points and bottlenecks within a network configured with a shortest path routing protocol.

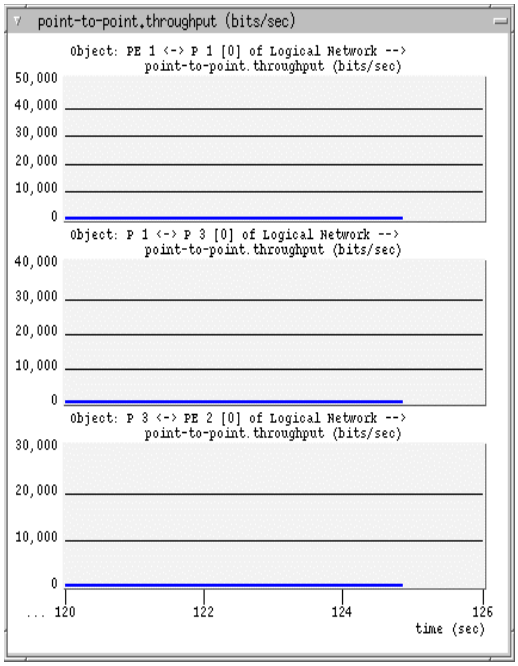

Figure 5.3 *Throughput(bits/sec) PE1→P1→P3→PE2* (non-shortest path)

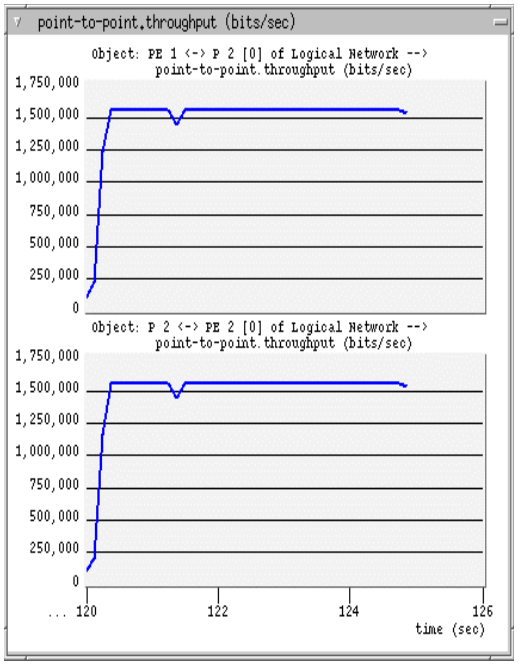

Figure 5.4 *Throughput(bits/sec) PE1→P2→PE2* (shortest path)

5.1.2 Queuing delay

We also collected some statistics concerning queuing delay and throughput from the edge and core routers. From figure 5.5 we registered no activities taking place between routers combining the non-shortest path. This is not a surprising result since this path is never been utilised within the simulation time. In the other hand, the queuing delay from PE1 → P2 grows every time the UDP traffic starts increasing its traffic intensity. The first increase occurs at the 121 second when the UDP traffic starts generating 300,000 bits/sec. Here we witness a small increase of queuing delay value. Each time the UDP traffic increases its intensity; there were registered a higher queuing delay value. This is of course reasonable result since the amount of traffic that exceeds the amount of capacity limit imposed by the links increases each second from the time UDP traffic is generated.

Another explanation for this heavy queuing is that we chose not to implement early packet dropping. However, implementing this would have given other results. Since these traffics are best effort class related they could have been dropped. From figure 5.6 shows that the queuing is much heavier between the ingress router and the first router along the path. From the second router and after, the queuing delay has a stable value of 0.008 seconds, which is lower compared with the earlier queuing along the path. This indicates that heavy queuing only occurs between the first routers along the shortest path. This is quit reasonable since the ingress router forwards enough packets that the link connected to the first core router can carry. Since every other links along the path has the same capacity, extensive queuing is not necessary any more. Therefore we believe that the queuing value registered between the core router P2 and the egress router keeps a normal value when forwarding enough traffic that the links directly attached to it is able to carry.

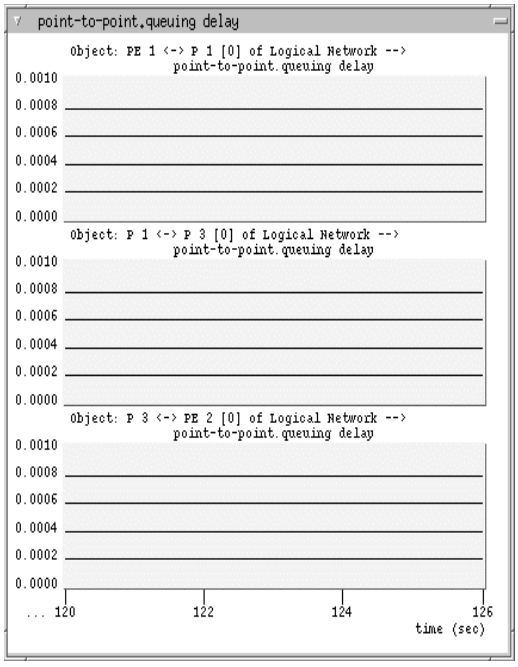

Figure 5.5 *Queuing delay path PE1 →P1 →P3 →PE2*

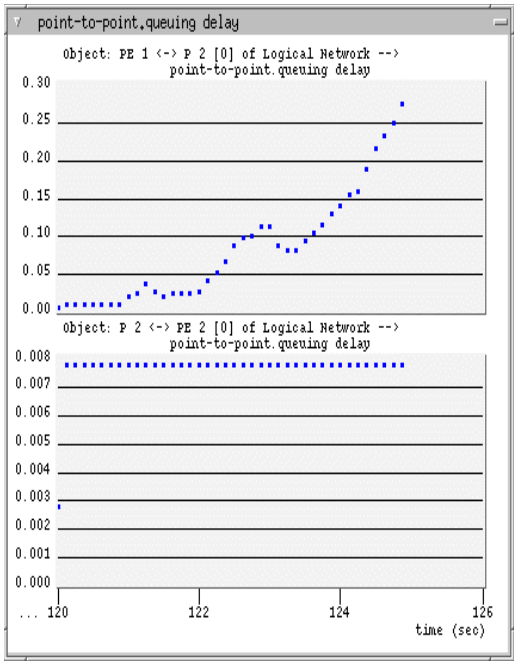

Figure 5.6 *Queuing delay path PE1 →P2 →PE2*

5.2 Concluding remarks

After observing and analysing the results collected from the simulation, we did manage to simulate some of the problems concerning the shortest path routing principal highlighted earlier in this thesis. The simulation showed that UDP traffic tends to suppress TCP traffic when a shortest path configured network becomes heavily loaded. In matter of a few mille second, the UDP traffic out conquers the TCP traffic. The simple answer to this behaviour is that the TCP protocol senses congestion and are bound to its flow control mechanism, therefore slowing down transmitting traffic into the network. It does this even when UDP traffic doesn't have a higher QoS support granted from the network service provider. In our simulation, both traffic flows was set to use best effort service class, but this didn't stop the UDP to just make the network become congested and suppress the TCP traffic flow.

The negative effect on the queuing delay between PE1 → P2 takes place because of the traffic that struggles only to use the shortest path to its destination. Under heavily loaded conditions, this looks like not to be a good choice. The queuing delay grows for one path, while the other path have plenty capacity to deal with traffic and are unutilised. The queuing delay causes the outbreak of the delay growth for both TCP and UDP traffic. Although UDP traffic doesn't understand and don't register whether its packets reaches its destination or not, it continues to keep its traffic intensity high. The TCP traffic intensity does the opposite, suffering from its flow control mechanism making it to become the looser when competing with the UDP traffic. One interesting aspect of queuing that is worthwhile mentioning is that we only observed heavy queuing between the first two routers along the shortest path. After these two routers, packets get to travel normally through the other routers along the path. If we possessed a more complex topology, we could have registered this effect between any ingress and first core router along a shortest path computed path. This could also be the case between any core routers being part of any shortest path carrying traffic path. This shows that if we had a more complex topology, we would need a very precise and fast route computation routing protocol in order to manage to have the right information about the cost of each link at any time. It has not been an easy task to come up with such a shortest path routing protocol. We would still get the oscillation effect even if this were available. We therefore conclude that this is a major flaw with the current shortest path routing protocol.

Also, the shortest path routing comes short when it comes to load balancing traffic in a efficient way. We are of course aware of the load balancing options of OSPF, but as stated in the beginning of this thesis, it cares for much administration and can get awfully complicated in a more complex networking environment. Shortest path routing doesn't imply efficient load balancing of traffic so a more efficient utilized networking environment can take place. The routing protocol is to be blamed, not being intelligent enough to sense when to use under utilized paths when forwarding traffic.

6 Simulation experiment using MPLS -TE

In this chapter, we experiment with traffic engineering with help of MPLS. After the presentation of the architecture itself, our aim was to investigate its performance and treatment of the flows it traffic engineer. We aimed to engineer flows of traffic in a way to secure a more efficient utilized network, while avoiding at the same time bottlenecks within the network. As the preceding experiment, no quality of service support was given to traffic entering the MPLS domain. Traffic engineering was only implied based on which protocol traffics used. Measurements were taken to investigate its performance features concerning delay and throughput between nodes within the network.

6.1 MPLS Traffic engineering configurations

Figure 6.1 illustrates the MPLS traffic- engineering scenario. The preceding network model was copied and the only changes made were the red and blue coloured stretched arrows combining label- switching paths through the experiential network. Below, details over the MPLS traffic engineering related configurations are presented. For a complete and more detail specifications over this experiential network, we refer to appendix 9.3

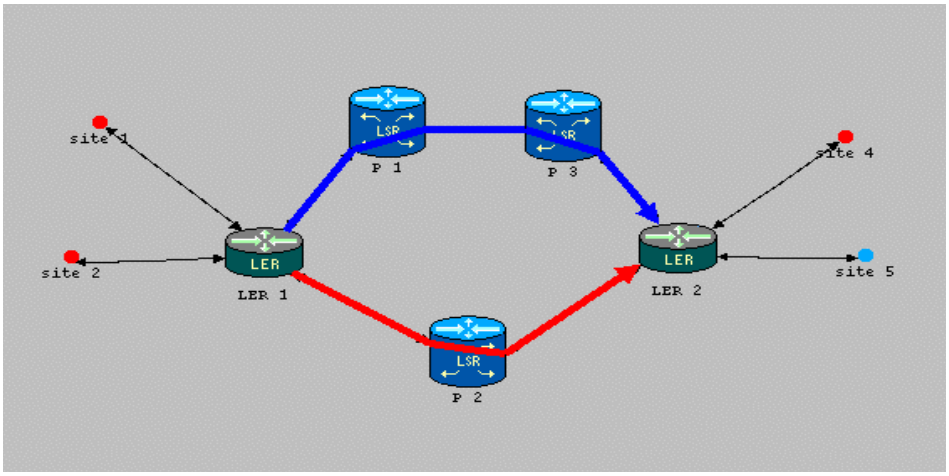

Figure 6.1 *Overview of the MPLS experiential network model*

In order to be able to traffic engineer flows of traffic, label-switching paths (LSPs) had to be installed. With RSVP, which is outlined in chapter 3.2.3, we reserved resources combining the paths for label switching. Static LSPs were established, in order for us to have a more precise control over the path a flow was to use. Flowspecs governed by the ingress router for traffics injected into the network were also specified. Table 6.1 below outlines the two separable parts to

the flowspec, TSpec and RSpec. Flowspec1 for traffic entering the red LSP and its traffic characteristics TSpec was configured with maximum bit rate of 1,544,000 *bits/sec*, average bit rate of 1,500,000 *bits/sec*, maximum burst size of 64,000 *bits/sec*, and its RSpec was best effort service class. A copy of this flowspec were made and configured for the blue LSP. Table 6.1 summarizes the flowspec configuration table. Since we traffic engineered by means of transport protocol type, traffic entering the LSP without the right type of protocol was discarded.

Flows	Max. Bit rate (bits/sec)	Average Bit Rate (bits/sec)	Max. Burst Size (bits)	Out of profile action
Flowspec1	1,544,000	1,500,000	64,000	Discard
Flowspec2	1,544,000	1,500,000	64,000	Discard

Table 6.1 *Flowspec Configuration Table*

The LSPs were installed between the pair of ingress and egress routers called the LER1 and LER2. These routers played a very important role, since they governed and controlled the mappings of the three important MPLS configuration elements called the forwarding equivalence class (FEC), flowspec, and LSP usage. One FEC class was given to one type of flow, in our case the TCP traffic, and the other FEC class was given to our second traffic type, the UDP traffic. Since we had configured traffic flows entering the network from left to right, meaning that site1 and site2 generating traffic towards site4 and site5, ingress router (LER1) interfaces had to be configured right. This meant that LER1 had to be configured to assign FECs based on which interfaces and what kind of traffic that was transmitted to it. Also, in order to assign FECs, other information gathered from incoming packets was inspected too at ingress router LER1. Based on the information, FECs was assigned from governing rules outlined in table 6.2.

FEC name	*Protocol used*	*Destination address*	*LSP Usage*
TCP Traffic	TCP	192.0.13.2 (Site5)	Blue LSP
UDP Traffic	UDP	192.0.11.2 (Site4)	Red LSP

Table 6.2 *FEC specification table*

At the ingress router LER1, packets was categorized and assigned an appropriate FEC. The FECs were then mapped to the right flowspec, which used a certain LSP. This way, the incoming traffic was engineered based on some administrative rule. Since our intention was to remedy the drawbacks experienced with the shortest path experiential network, our MPLS-TE configuration had the objective to measure the performance achieved by traffic engineering TCP traffic and UDP traffic to separate paths within the network. UDP-traffic was therefore configured to utilise the red LSP, while TCP-traffic was to utilise the blue LSP.

6.2 Analysing and discussing experiential results

The statistics collected from within OPNET, is shown below. From our experimentation, we collect statistics concerning MPLS traffic engineering. Our objective here is to analyse and discuss the results gathered from measurements registered. By this, we aim to investigate the MPLS traffic engineering architecture and its benefits. Below, various measurements concerning our findings are analysed and discussed.

6.2.1 Throughput

As recalled, site1 was configured to generate TCP traffic from the second minute. The amount of this traffic was 1,500,000 *bits/sec*. And as we observed from the result shown in figure 6.2, we witnessed that this value was reached and was stable until the UDP traffic started making some activities. A second later after the TCP traffic generation UDP started generating traffic. Keeping in mind that both traffic utilised their own link towards the ingress router, we registered that the UDP traffic intensity had some effect on the TCP traffic intensity. These effects did take place every time UDP- traffic intensity was increased. There were registered transient values between 1,544,000 bits/sec, which is the maximum capacity and all the way down to 1,250,000 bits/sec. These transients values registered may have been taken place because of combination of several factors. Below we outline two factors that we believe might be the cause of values registered.

First, TCP acknowledgements did travelled from the server back towards the client along the shortest path. With this shortest path being severely busy handling the UDP traffic, this would make it difficult for the ack packets to get transmitted when heavy UDP traffic was competing with it for the same shortest path resources. The UDP traffic was guaranteed the bandwidth for its use by the unidirectional LSP, which utilised the same shortest path that the server would be using for transmitting acknowledgment packets towards the FTP client. We found this very interesting since this issue was not been referred to by all the related work we came over to study. It's obvious that the FTP client would suffer, not receiving its acknowledgements in time. It's difficult to point out every single event that could reveal the transient values of both traffic types, even though both traffic types get to use separate paths towards their destination. But when it comes to the TCP traffic, we point out that one of the reasons is the missing in time acknowledgements expected from the sender. When configuring the experiential network, we didn't take this issue in consideration, but it's worth exploiting further to measure its impact on TCP traffic. When traffic engineering TCP traffic, one might take into consideration the path an acknowledgment packet might take towards the transmitter side. Of course, this is not relevant when enough bandwidth is available. We are also fully aware of the fact that the ack packets are relatively small in size, and the links we used were duplex ones. Therefore, we cannot be sure of whether this issue has an impact when it comes to the TCP –traffic intensity drop offs values registered. In our case it was enough bandwidth available to cope with both the UDP traffic and the TCP ack packets until the UDP traffic, which uses the shortest path reached its maximum traffic

intensity at its last traffic intensity increase. It is then when we registered the lowest achieved TCP throughput. After a fraction of a second the throughput however manages to struggle back to its maximum traffic intensity. This jump back does actually occur each time a drop off takes place. We believe that this fast retransmit of traffic is done with help of TCP RENO implementation details. Because of short available time, we did not manage to investigate this issue any further. We let this be an open issue for further research.

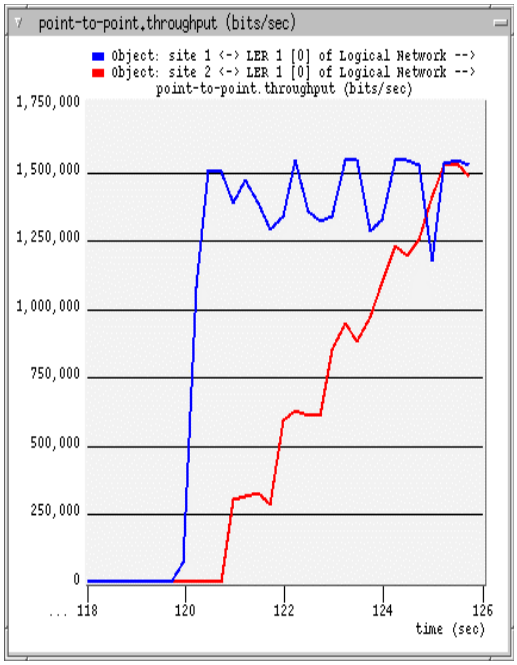

Figure 6.2 *TCP and UDP throughput towards LER1*

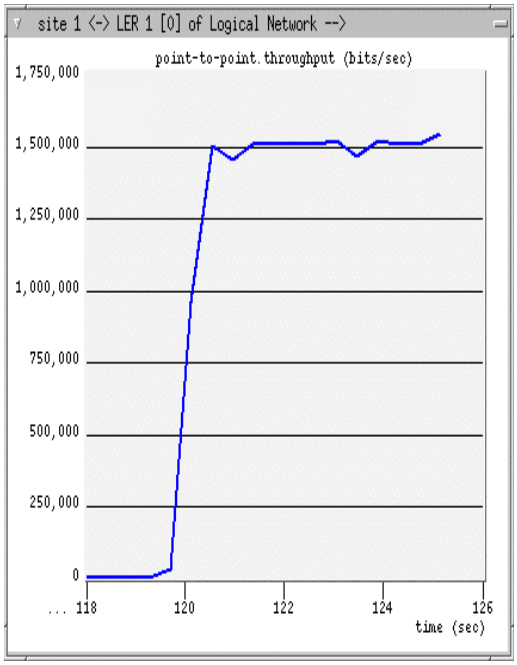

Figure 6.3 Only *TCP throughput towards LER1*

Second, we turn to another factor that is related to the ingress router, which is responsible for the forwarding of traffic transmitted to it. Packets intended to be traffic engineered must follow the policy of and reservation of the label- switching path that it's going to use. When utilising a LSP, the router must keep track of which flowspec established for the LSP the packets gets to use. LER1 which is the ingress router must then govern the amount of average bit rate allowed by the flowspec defined for each LSP. The flowspec which was defined and used by TCP traffic, allowed only an average bit rate of 1,500,000 bits/sec. With TCP traffic exceeding at some points the average bit rate traffic intensity, some queuing at the ingress router had to be taken place to govern the amount of average bit rate limit configured. Figure 6.7, shows the amount of queuing delay between the ingress router LER1 and the first router along the path. The queuing delay has some direct impact on the TCP protocol. TCP protocol would register the delay and lower its intensity, thus causing the traffic intensity drop off values registered. With TCP protocol sensing some delay for acknowledgements, it suffers from its flow control mechanism, thus lowering its transfer intensity. We believe that this is the major factor for this incident. Observing the registered result presented, we noticed that each time the UDP traffic increases its activity its

activity is affected on the queuing delay between ingress and first core router along the path. Consequently, this is again reflected on TCP throughput between the source and ingress router. Every increase of UDP traffic intensity, mark a fall on TCP traffic intensity. It's important to establish that more investigation is needed to exactly point out the reason for this incident. With shortage of time available to us, we believe more research is needed in this area. However, we are amazed by our discoveries and encourage further investigation of the MPLS traffic engineering and its treatments of traffic when engineering them.

But the overall intention was to make the TCP traffic not to suffer from UDP traffic injected into the network, by engineer it to a separate path apart from the UDP traffic. In the beginning of this thesis we explained the suppression the TCP traffic would achieve when competing with the UDP traffic for the same shortest path resources. This was also simulated in the precedent chapter. Since TCP have a flow control mechanism, it would lower its traffic intensity when signs of congestion appear in the network. UDP traffic in other hand doesn't suffer from this flow control mechanism, making it to become the winner of the two protocols under heavy load conditions. Since we traffic engineered these flows to separate paths with MPLS-TE, both traffic types kept growing almost to the maximum available bandwidth capacity. This shows that even with some degradation of TCP traffic intensity registered from the sources towards the ingress router, the TCP- and UDP traffic comes out with an acceptable performance. Achieving almost full utilisation of available network resources

Other results gathered from our MPLS-TE experimentation were the throughput measured from paths between routers that handle the traffic flows. Figures 6.4 and 6.5, shows the results gathered from our simulation. We registered that the throughput between routers combining the shortest path and non- shortest path, were more balanced compared with the shortest path configured network simulated earlier. However, we witnessed some interesting results form measurements taken. From figure 6.4, we witnessed the TCP traffic throughput travelling through the blue coloured LSP. Here, the throughput starts climbing to its intended 1,500,000 *bits/sec*. Thereafter, registering unstable throughput values which occurs approximately each second. Each second, the throughput gets decreased, and then jumps back up to its maximum throughput intensity. Our interpretation of this behaviour is a combination of issues discussed below. However, here we sense a more strongly relation between TCP- and UDP traffic intensity. Each time the amount of UDP traffic intensity is increased, we get to witness its impact on TCP traffic intensity. To recapitulate, we increased the UDP traffic intensity each second within our experimentation. The TCP throughput drop offs takes place approximately at the same times when UDP traffic intensity is increased. This is not a very dramatic effect since the throughput retains its original high throughput right after registering the UDP traffic intensity increase.

From these results registered, we have to confirm that the non-shortest path was more efficiently utilised with the traffic engineering capabilities of MPLS. With no traffic engineering, we had probably witnessed no throughput activity along the non-shortest path as simulated in the preceding chapter. TCP traffic would have then suffered a lot more when competing with its rival UDP traffic along the shortest path.

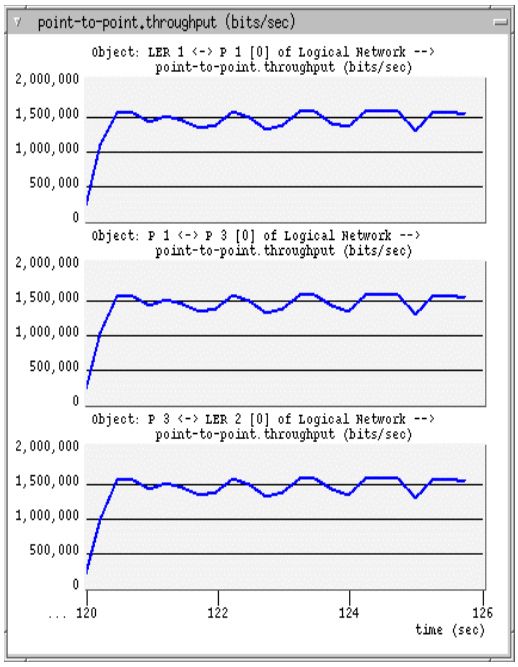

Figure 6.4 *Throughput(bits/sec) LER1→P1→P3→LER2* (Blue LSP)

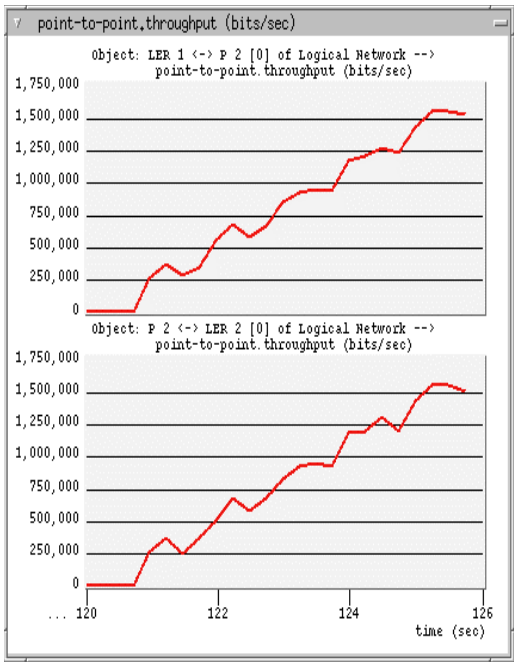

Figure 6.5 *Throughput(bits/sec) LER1→P2→LER2* (Red LSP)

From figure 6.5, we witnessed the throughput intensity from path combining routers (LER1 → P2 → LER2) through the red coloured LSP. The UDP traffic throughput was less interesting. However, here too we registered slightly unstable throughput activity. But these are so small that we find it little relevant and interesting to investigate. In other hand, path utilisation along the shortest path was stepwise utilised relative to the adaptive increase of the UDP traffic in time. Since UDP traffic gets to utilise this path alone, we manage to avoid congestion and bottlenecks within the network. If shortest path routing were configured, we would have over utilised this path, making both traffic streams to suffer from congestion within the network.

We also collected statistical results from traffic amount that tried to enter and the actual amount of traffic managed to exit the LSPs. Figure 6.6 and 6.7 shows the plotted results. From figure 6.6, which shows the amount of traffic entering and exiting the blue coloured LSP, we registered the treatment TCP traffic achieved within this LSP. The black colour line within the graph shows the amount of traffic entering the LSP, while the green colour line indicates the amount of traffic managing to exit the LSP. From the left figure below, we witness that traffic imposed on the LSP is treated well and shows that the same amount of traffic gets to be forwarded using the LSP. Of course, some differences were registered between the two graphs plotted. However, the amount of traffic heading out of the LSP was not registered being lower than the amount of traffic entering the LSP until the two last drop offs registered. These drop offs are however small and neither important nor relative enough to be elaborated further on. The same goes for UDP traffic and its utilisation of the red colour LSP. Here too the amount of traffic entering and exiting the LSP was equal.

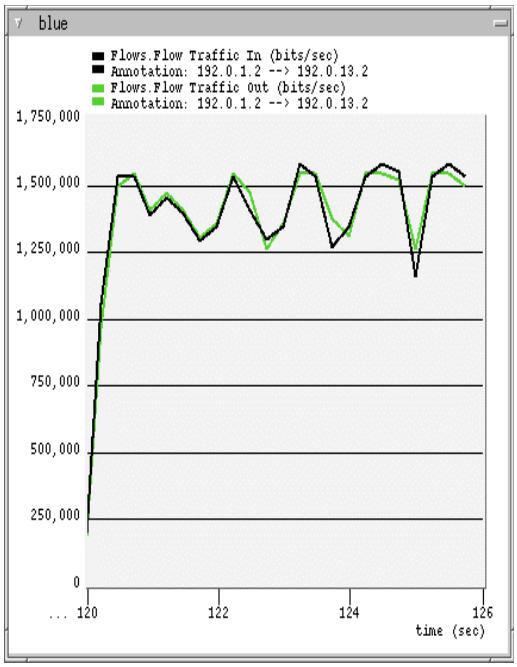

Figure 6.6 *Throughput(bits/sec) In &Out* (Blue LSP)

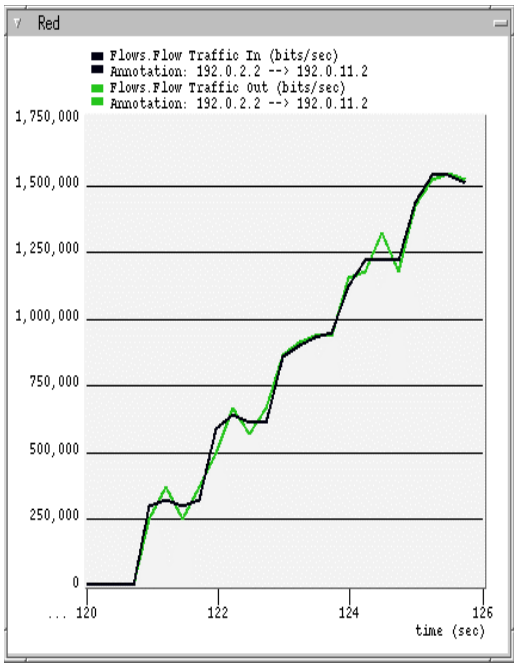

Figure 6.7 *Throughput(bits/sec) In &Out* (Red LSP)

6.2.2 Queuing delay

We also collected some statistics concerning queuing delay from the edge and core routers, which was related to QoS. From figures 6.8 and 6.9, we can observe that the queuing delay is more balanced between the both paths comparing it with the shortest path routing scenario. The most interesting discoveries were done at the ingress router LER1. Here, queuing delay value some how jumped to a higher value each time the amount of UDP traffic passing through the ingress router LER1 were increased. The queuing delay value did each time increase, and then went back to its normal value. Also the amount of this sudden increase was related with the amount of UDP traffic getting increased. Each time the UDP traffic increased its traffic intensity with 300,000 *bits/sec* the queuing delay was affected with higher value on the TCP traffic queuing delay time. A reasonable reason for this effect may be the fact that the ingress router that gets to handle both traffic-types become busier forwarding traffic. This undesired effect takes place even when both traffic types get to use separate paths towards the egress router within the MPLS domain.

UDP traffic, which utilises the path LER1 → P2 → LER2, seems to have a lower queuing delay values than the TCP traffic. This is the case until UDP traffic intensity starts closing in to the maximum link capacity available to it. It keeps a steady value approximately at 0.0075 seconds, until its last additive increase of 300,000 bits/sec, achieving 1,200,000 + 300,000 = 1,500,000 bits/sec. Then it starts sensing its maximum and average bit rate amount allowed by the flowspec defined for traffic intended to utilise the red coloured LSP. It's then when the value starts growing to 0.0125 second at the end of simulation time. Another

interesting detail of it is the slightly queuing delay drop offs at each second along the simulation time. This drop off is related to the time when traffic intensity is increased with 300,000 bits/sec. This could be implementation issue concerning the software simulation tool. However, it isn't a very significant transient value.

The other interesting result registered, were the fact that almost all-significant queuing appeared between the first two routers along both paths. Thereafter, the values kept stable queuing delay values between other routers along the forwarding path. We have a very simple explanation to this phenomenon. Since all the major queuing takes place between the ingress and first router along the path and the queuing values aren't very high, we get a very stable queuing value between other routers along the path. The amount of traffic between these routers are more predictable since the second router along the path get the right amount of traffic that it can forward further closer towards some destination. It's the first router that gets to queue the heavy amount of traffic that the links can't cope to carry immediately.

However our objective wasn't to investigate TCP and UDP characteristics in depth. We were interested to investigate the amount of performance gain from traffic engineering compared with a plain shortest path configured network. Comparing these results with the earlier result from the shortest path routing configured network, the queuing delay keep a much less queuing delay value between the ingress router and the first router along the shortest path, than the shortest path routing scenario.

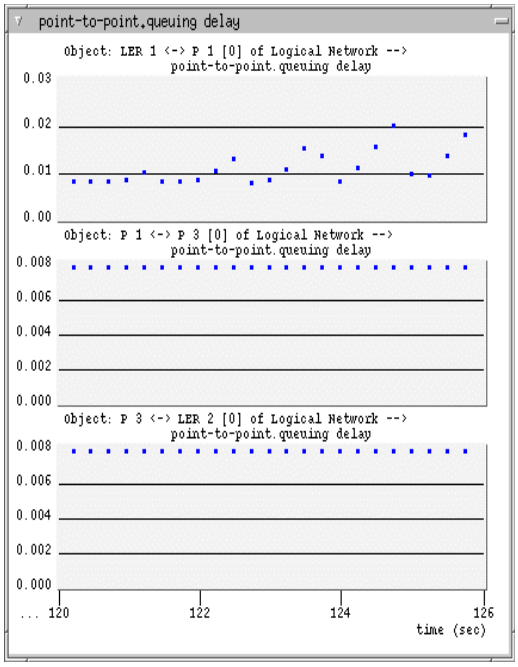

Figure 6.8 *Queuing delay path LER1 →P1 →P3 →LER2* (Blue LSP)

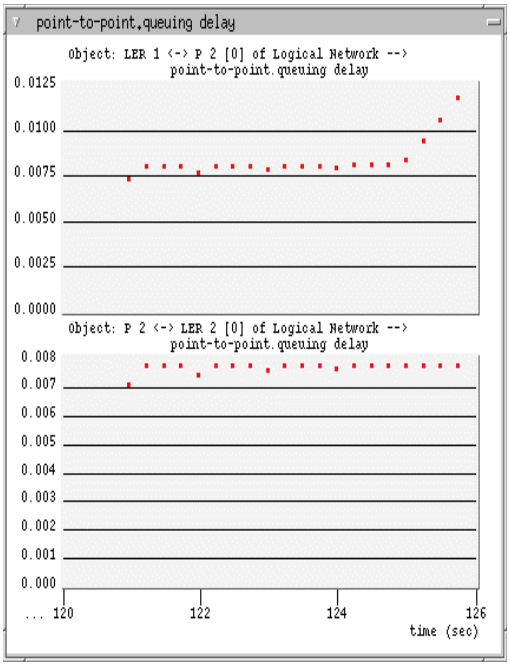

Figure 5.9 *Queuing delay path LER1 →P2 →LER2* (Red LSP)

6.3 Concluding remarks

Our experimentation and analysis of it revealed to us the performance features of MPLS-traffic engineering outlined in the beginning of this thesis, stating that bottlenecks within the network might be avoided by traffic engineering flows through other paths than the shortest path between any ingress and egress routers. By using MPLS technology, traffic engineering can be deployed and performance gains in terms of queuing delay, throughput and path utilisation can be achieved. We are not going to here discuss the technology it self since a complete coverage of it were given in the earlier chapters. However, we can verify that the theoretic assumptions made earlier, stating that it would be a technology worth exploiting to overcome the shortcomings of the shortest path routing protocols were in line with our investigation of the protocol.

The throughput was registered to be of an acceptable value for the TCP traffic in the MPLS-TE scenario. The TCP traffic didn't need to lower its traffic intensity since it didn't compete with the UDP traffic for network resources. We measured TCP traffic throughput between source and ingress router and routers among. We witnessed some TCP throughput drop offs between the sender and the ingress router. An interpretation of these drop offs was given. We stated that two factors combined could be the reason of the throughput result registered. Nevertheless, we think of the throughput outcome registered for the TCP traffic positively, knowing that if traffic engineering was not implemented, TCP traffic would have been suffered competing with UDP traffic for the same shortest path resources. Improvements were also shown in the case of path utilization. Our findings made it clear to us that by traffic engineering one can achieve more efficient network resource utilisation. The simulated network topology didn't represent a service provider's network, but it clearly shows what possibilities MLPS-TE can introduce when it comes to utilizing network resources more efficiently. Service providers can engineer certain traffic flows, by some local administrative policy to utilize its resources more efficiently. We have not investigated any of the positive economic impact of this, but it may be worthwhile further research.

When it comes to queuing delay, lower delay time was registered because flows used both paths available between the ingress and egress routers. Thereby, striking more balanced traffic intensity between paths available towards the egress router. However, we discovered that even with two distinct paths being utilised for UDP and TCP traffic, the UDP traffic intensity increase had some effect on TCP traffic queuing delay and throughput. Our measurements reviled to us that with even traffic engineering TCP traffic through a separate label- switching path, some degradation of performance did take place whenever UDP traffic made some activities. However, we found that the queuing values registered for both traffic types showed a performance gain compared with the shortest path scenario.

7 Simulation experiment using MPLS-TE and Diffserv

After our experimentation with the pure protocol based traffic engineering simulation, we now move forward simulating traffic engineering with quality of service support. Our objective was now to engineer flows of traffic and support QoS with help of differentiated services. Here, we traffic engineer both generated traffics outlined earlier through the same label-switching path. By assigning generated traffics different CoS, we were able to measure performance issues imposed by MPLS-TE and diffserv. Details over configurations of network and traffic implementations within OPNET Modeler can be reviewed in appendix 9.4.

7.1 MPLS-TE and QoS support configuration

Figure 7.1 illustrates the MPLS traffic- engineering with QoS support scenario. The preceding network topology used earlier was copied and the only change made was the utilisation of one of the LSPs configured earlier. The blue coloured stretched arrow is the LSP combining label- switching path configured to be utilised by both generated traffics with different CoS.

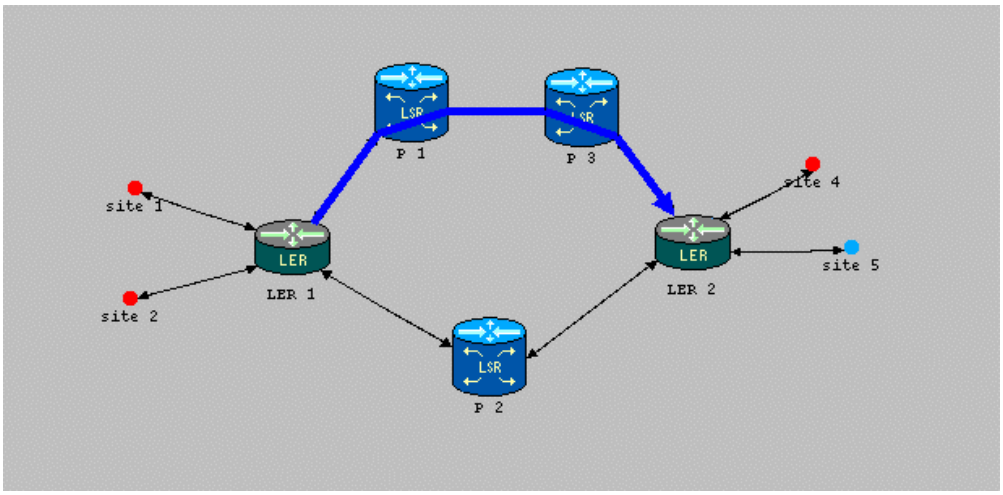

Figure 7.1 *Overview of the MPLS QoS experiential network model.*

In this experiential network, some changes had to be made to secure a traffic engineering networking environment with QoS support. The LSP was to handle two flowspecs governed by the ingress router. Table 7.1 below outlines these two separable parts of the flowspec; TSpec and RSpec. EF_flowspec for traffic entering the blue LSP and its traffic characteristics (Tspec) was configured with maximum bit rate of 1,544,000 *bits/sec*, average bit rate of 1,000,000 *bits/sec*, maximum burst size of 64,000 *bits/sec*, and its RSpec was EF service class. The other flowspec that were to be governed by the ingress router was the AF11_flowspec. This flowspec's traffic characteristic (TSpec) was configured

with maximum bit rate of 1,544,000 *bits/sec*, average bit rate of 500,000 *bits/sec*, maximum burst size of 64,000 *bits/sec*, and its RSpec was AF11 service class. The EF_flowspec was configured to take care of EF CoS traffic and discard traffic other than this particular traffic type entering the LSP. The other flowspec, AF11_flowspec was configured to take care of AF11 CoS traffic and discard traffic other than this particular traffic type entering the LSP. Table 7.1 summarizes the flowspec configuration table.

Flows	Max. Bit rate (bits/sec)	Average Bit Rate (bits/sec)	Max. Burst Size (bits)	Out of profile action	Traffic class
EF_ flowspec	1,544,000	1,000,000	64,000	Discard	EF
AF11_flowspec	1,544,000	500,000	64,000	Discard	AF11

Table 7.1 *Flowspec Configuration Table*

One FEC class was given to one type of flow, in our case the TCP traffic with EF CoS, and the other FEC class was given to our second traffic type UDP traffic with AF11 CoS. Since we had configured traffic flows entering the network from left to right, meaning that site1 and site2 generating traffic towards site4 and site5, LER1 interfaces had to be configured right. LER1 had to be configured to assign FECs based on which interface that was handling the incoming traffic, plus other information gathered from the incoming packed header information. In our case FECs was assigned from governing rules outlined in table 7.2.

FEC name	*DSCP*	*Protocol used*	*Destination address*
Site1	EF	TCP	192.0.13.2 (Site5)
Site2	AF11	UDP	192.0.11.2 (Site4)

Table 7.2 *FEC specification table*

At the ingress router LER1, packets was categorized and assigned an appropriate FEC. The FECs was then mapped to the right traffic trunk, which used a certain LSP. This way, the incoming traffic was engineered based on an administrative rule. Since our intention was to measure the treatment of these two traffic flows with different QoS requirement, our MPLS-TE was now been configured to exploit the QoS architecture of differentiated services. We used diffserv`s Weighted Fair Queuing (WFQ) combined with the DSCP code mapping to govern QoS requirement by the flows. A higher WFQ value was given to the EF CoS traffic over the AF11 CoS traffic. EF CoS traffic was given the weight value of 55 and the opportunity to use the low latency queue, while the AF11 CoS traffic was with its weight value of 5 configured to use the default queue. To measure the performance outcome of this configuration, we collected data from WFQ delay, WFQ buffer usage, flow throughput and flow delay measured from both traffic flows.

7.2 Analysing and discussing experiential results

The statistics collected from within OPNET, is shown below. From this experimentation, we collected statistics concerning MPLS traffic engineering combined with differentiated services WFQ quality of service support. Our objective here is to analyse and discuss the data collected to explore the benefits of the MPLS traffic engineering architecture and its QoS support. Below, we have discussed various measurements concerning our findings.

7.2.1 WFQ delay and buffer usage

Statistical data was collected concerning Weighted Fair Queuing delay from the interface output of the ingress router. We wanted to investigate how much time the AF11 CoS traffic gets to be queued compared with the EF CoS traffic. The results are shown in figure 7.1, where EF CoS traffic achieved a WFQ delay of below 0.025 seconds. Registering its unstable values, it still kept a much lower values, even with its transient values. AF11 CoS traffic, in other hand kept an overall irregular but higher value. It achieves a very sparse WFQ delay values within the simulation time. With its highest one time registered value of approximately close to 0.125 seconds, it holds an overall higher sparse values than the EF CoS traffic. This is an expected result, since the AF11 differentiated services code point was configured with a lower priority value than its rival.

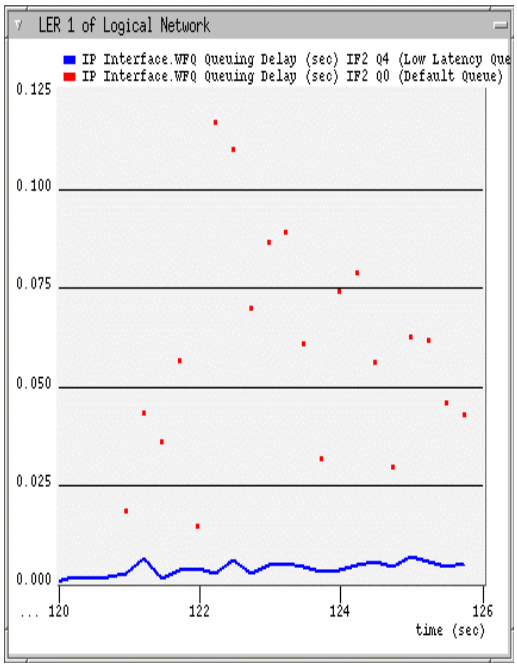

Figure 7.1 *WFQ delay on LER1 output interfaces*

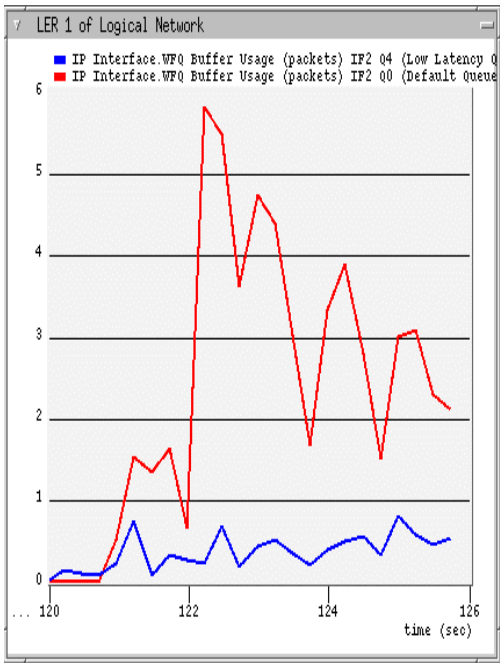

Figure 7.2 *WFQ buffer usage on LER1 output interfacs*

Figure 7.2 shows the WFQ buffer usage by the two types of flows described. The blue colour graph represents as earlier the EF CoS traffic, while the red colour graph represents the AF11 CoS traffic. The blue graph shows that the EF CoS packets achieves a desired queuing delay time. Here, the EF CoS packets

gets to utilise the low latency queue at the router, while the AF11 CoS packets were to utilise the default queue. The low latency queue was configured being processed before any other queue. This meant that any packet residing in this low latency queue were to be processed first and forwarded before packets residing in the default queue. Therefore, we observe from the above figure that no more than one EF CoS packet resided within the queue before getting processed. The down side of this effect is affected on the AF11 CoS packets, which achieves a value of between one and six packets been queued at the ingress router.

7.2.2 Flow Delay

Flow delay values describe the amount of delay imposed to flows getting transmitted through the LSP configured. Figure 7.3, shows the result from flow delay measurements gathered from the simulation. Each flow travelling through the LSP got imposed to certain amount of delay. This delay was taken place because of the QoS support that was given to each of the flows getting engineered. Since the AF11 CoS traffic were given a lower QoS support, it was imposed a higher flow delay value than the EF CoS traffic. The EF CoS possessed an almost stable delay value of 0,025 seconds. The AF11 CoS traffic kept an overall higher flow delay value. The graph shows that it's more difficult to achieve a stable flow delay value with the AF11 CoS, because this kind of traffic utilises the default queue at the routers. However, other types of queues can be configured to have a more precise and calculated queuing policy.

These values were somehow expected from the results interpreted earlier. The effect of WFQ option of differentiated services imposes a better QoS support to the EF CoS traffic, thus achieving a better flow delay for this class of service traffic. AF11 CoS isn't delay sensitive like the EF CoS traffic, therefore residing and spending more time within the queue. These results were therefore in line with results expected to achieve with WFQ implemented within this simulated network.

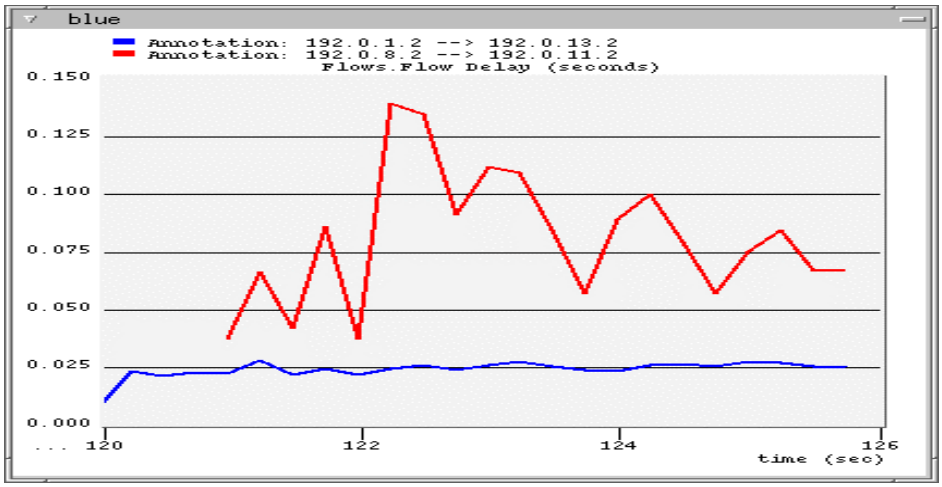

Figure 7.3 *Flow delay within the blue LSP*

7.2.3 Throughput

The other QoS related results registered from the MPLS-TE experimentation were the throughput measured from traffic entering, and the amount that exited out of the LSP. Since we had installed a separate flowspec for each of the two types of flows, we were interested to observe their throughput in this simulation when getting engineered through the same LSP. We allowed the EF flowspec to cope with an average bit rate of 1,000,000 *bits/sec*. The AF11 flowspec was to cope with an average bit rate of 500,000 *bits/sec*. These amounts equalled a value of 1,500,000 *bits/sec*, which is almost the maximum amount of link capacity of each link along the path. Both flowspecs were configured with a maximum bit rate of 1,544,000 *bits/sec*, and maximum burst size value of 64,000 *bits/sec*.

Figures 7.4 and 7.5 below, shows the results gathered from our simulation. We registered that the throughput of the entering traffic was almost exactly the same throughput exiting from the LSP. Our configurations allowed only fixed amounts of traffic to flow through the LSP. The preceding queuing measurements outlined earlier revealed that not all of the traffics imposed to the ingress router were allowed to enter the LSP immediately. The ingress router manages to queue up extensive amount of traffic transmitted to it because of link capacities and the LSP with its respective capacity limit. The ingress router transmits only the amount of traffic that the LSP are configured to process. Figure 7.4, shows that the EF CoS traffic which is represented with the blue coloured graph, has an approximate bit rate average of 1,000,000 *bits/sec*. While the AF11 CoS traffic, which is represented with the red coloured graph, possesses an approximate average bit rate of 500,000 *bits/sec*. These bit rate values takes place because of the average bit rate limitation configured by the flowspecs. However, several transient values were registered which indicated that the router couldn't exactly calculate to govern the capacity limit of the average bit rate values imposed by the flowspecs. The first traffic intensity degradation by the TCP EF CoS traffic takes place at the same time the UDP traffic exceeds its average bit rate limit governed by the AF_flowspec. This indicates that the UDP has some slightly impact on TCP traffic once both flows starts getting engineered through the same LSP with different CoS. It also seemed being easier for the ingress router to keep the average bit rate value governed by the flowspecs when it came to UDP traffic than the TCP traffic. UDP traffic keeps a more stable value around its flowspec average bit rate value of 500.000 *bits/sec*.

Figure 7.5 shows, the amount of traffic managing to exit the trunk. Comparing both figures, we observed that they are almost exactly the same. This indicated to us that the amount of traffic heading in and out of the LSP was approximately equal. The figure also shows that the EF CoS throughput keeps a higher average bit rate value compared with the AF11 CoS traffic. This takes place because of the fact that the flowspec of EF CoS traffic was due to utilise and entitled a higher throughput capacity limit. The overall picture indicates that flows with different CoS can with help of flowspecs become controlled not to over utilise the network resources while at the same time be given different quality of service.

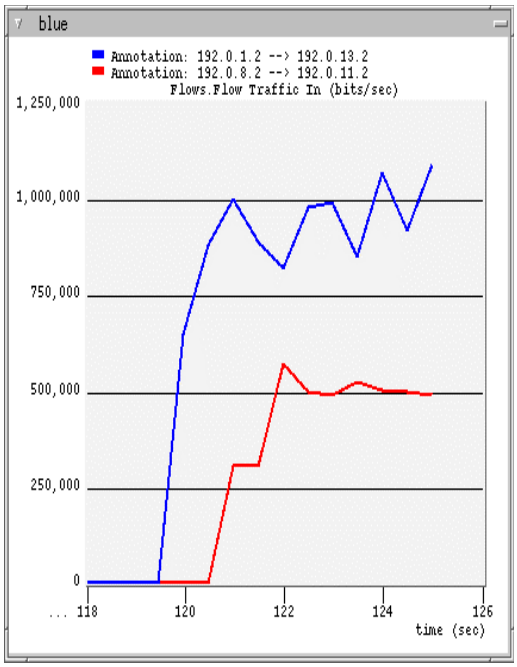

Figure 7.4 Traffic into LSP *Throughput(bits/sec)*

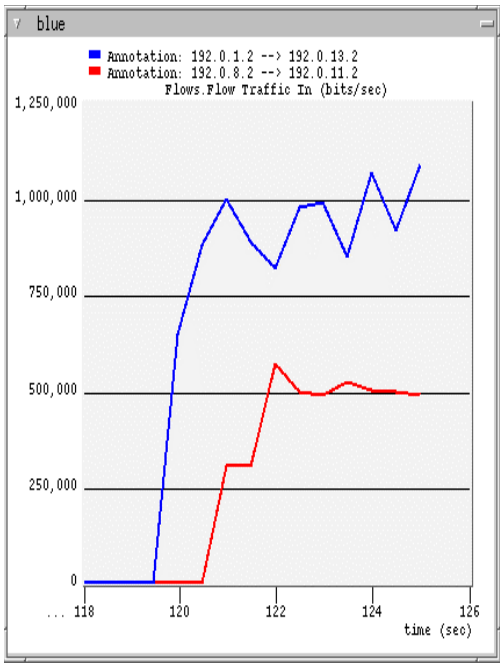

Figure 7.5 Traffic out of LSP *Throughput(bits/sec)*

7.3 Concluding remarks

Our experimentation and analysis with the MPLS traffic engineering combined with differentiated services WFQ revealed to us the possibilities and performance gains achieved when traffic engineering flows with different QoS requirements. These two architectures combined offer a comprehensive traffic control and QoS support. MPLS traffic engineering governs the amount of traffic imposed on the network resources, and controls the path different kind of traffic is to take towards its destination. While weighted fair queuing with help of DSCP of differentiated services governs the QoS requirements of flows getting traffic engineered. We are not going to here discuss the technology it self since a complete coverage of it were given in the earlier chapters. However, we can verify that the theoretic assumptions made earlier, stating that these technologies combined together can perform traffic engineering with QoS support was inline with our results registered from experiments conducted in this chapter.

Results gathered from our simulation showed that a higher priority CoS traffic managed to get a better quality of service from the resources along its path. EF CoS traffic did achieved a lower flow delay time, than the AF11 CoS traffic. This had to do with the fact that the AF11 CoS traffic wasn't delay sensitive as its competitor EF CoS traffic. It did also utilise the low latency queue rather than the default queue of which AF11 CoS packets was forced into. This low latency queue was configured being processed before any other queues were processed. Therefore, we registered no more than a single EF CoS packet residing in this queue at any time during the simulation. Since we used Weighted Fair Queuing to

manage the quality of service treatment of packets, higher priority or weight were given to the EF CoS traffic at the routers. Packets with EF differentiated service code points were given a higher weight, thus residing less time in the queues. Both WFQ delay and WFQ buffer usage values were lower for the EF CoS traffic, indicating that the time EF CoS packets spent in queues at the routers were less than the AF11 packets used. Less time spending in queues helps arriving at the final destination faster. This gives a faster recognition from ack packets to make the sender keep high traffic intensity. Thus, making the EF CoS transmitter to keep a high throughput. However, we registered some directly impact of AF11 CoS UDP traffic upon the EF CoS TCP traffic. The impact was not as much as if they were going to compete directly with each other for the same network resources as simulated earlier in this thesis. Nevertheless, its worthwhile mentioning that with both flows engineered through the same LSP, small impact are eminent not to occur. The values we registered was however not alarming.

The amount of traffic imposed on the network was governed by the flowspec specifications for flows getting engineered through the LSP. In this way, no more traffic was been able to get into the core network than allowed. By traffic engineering traffic through a LSP, we managed to control the amount of traffic intensity throughput within the MPLS domain, while at the same time supporting quality of service to traffic getting forwarded with help of differentiated services.

8 CONCLUSION

We hereby, give our conclusion based on the experimentations and analysis of the simulations made within this thesis and the theoretic description of technological architectures presented. In this thesis, we have simulate three experiential networks. We experimented with shortest path routing, MPLS- traffic engineering and MPLS traffic engineering combined with differentiated services to support QoS. Below, we give a comprehensive conclusion of each of these experiments combined with the technologies presented.

8.1 Conclusion made from shortest path routing principle

Shortest path routing principal in short is based on routing traffic through the shortest path known towards any destination. The routing protocol is not to take into consideration other under utilised non-shortest paths when forwarding traffic. In times where several sources use the same shortest path, that path may eventually become congested. Congestion may appear at some ingress router, which gets to handle all the traffic destined to some destination beyond a single egress router. With this approach introduced by the routing protocol, lack of efficient network resource utilisation is difficult to avoid. Non-shortest paths will at time be under utilised while shortest path will become over utilised.

To highlight these mentioned drawbacks, we simulated a network to run shortest path routing protocol OSPF. This protocol is basically designed to route traffic using only the shortest path to forward traffic through the network. It computes routes based on the link states on the routers and calculate the minimum cost or metric towards any known destination. From our simulation, we managed to measure performance issues concerned with queuing delay, throughput and link utilisation. The results were, as we imagined quite poor, making the traffic to suffer from the shortcomings of the routing protocol. We registered that the routing protocol without purpose treated the UDP traffic better than the TCP traffic under heavy load conditions. This was the case because the TCP have a flow control mechanism and senses the appearance of congestion, making it to lower its traffic intensity. Even though both traffic types were configured with equal best effort service class, the UDP traffic came out with better performance treatment in the shortest path configured network. We measured their queuing delay, throughput and link utilization. From these results, we confirmed that the shortest path under heavy load conditions wasn't as "short" as the routing protocol might believe. The shortest path was over utilised and the performance registered from both traffic types were to show that the shortest path routing protocol didn't impose them any good performance. Worthwhile mentioning that the TCP traffic did suffer most compared with the UDP traffic basically in all measured performance issues. These negative registered results were related to the fact that all traffic was routed through the shortest path between the ingress and egress router. The non-shortest path was left unutilised. Unfortunately even in the case of being able to achieve better performance by using a non-shortest path when the shortest path is under heavy load condition, the routing protocol continued forwarding traffic through the shortest path.

As described earlier in this thesis, there are ways to engineer traffic in IP-networks to cope with the load-balancing problem of shortest path routing protocols. A way to traffic engineer is to manipulate the link metrics that is presented to the link-state IGPs such as OSPF. But this mechanism potentially leads to several problems. First, by changing the link's metric can force changing the path of all packets traversing the link. Second, this does not make any room for dynamic redundancy and do not consider the characteristics of offered traffic and network capacity constraints when making routing decisions. Last but not least, one can imagine how much administration this will cause and making room for human failure.

Traffic Engineering is difficult with IGP in large networks for the following reasons:

1. Between the Equal-Cost Multi-Path (ECMP) from a source, every path will have an equal share of load. This equal ratio cannot be changed. Therefore, one of the paths may end up carrying significantly more traffic than other paths because it also carries traffic from other sources.
2. Load sharing cannot be done among multiple paths of different cost, without a lot of administration and manual link metric manipulation.

We did however not conduct these techniques within our simulation, knowing that a new emerging technology called multi protocol label switching has been developed to cope with the difficulties of shortest path routing protocols when it comes to traffic engineering. With the shortcomings simulated within this thesis, we moved forward using multi protocol label switching, which is to be the future solution to traffic engineering Internet traffic. We give a conclusion of this technology alongside with our simulation experience below, comparing its performance issues with the shortest path routing simulation.

8.2 Conclusion made from MPLS traffic engineering

Traffic Engineering is the process of controlling how traffic flows through one's network so as to optimise resource utilisation and network performance. Traffic Engineering is needed in the Internet mainly because current IGPs always use the shortest path to forward traffic. Using shortest paths conserves network resources, but it may also cause the following problems:

1. The shortest paths from different sources overlap at some links, causing congestion on those links.
2. The traffic from a source to a destination exceeds the capacity of the shortest path, while a longer path between these two routes is under-utilised.

MPLS is an advanced forwarding scheme. It extends routing with respect to packet forwarding and path controlling. MPLS traffic engineering remedies these insufficient by allowing any label-Switched Path (LSP) to be dynamically shifted from a congested path to an alternative path. This allows the ISPs to operate their network at much higher capacity under normal circumstances, knowing that when congestion is about to occur the network will look for alternatives to avoid congestion. It also replaces the need to manually configure the network devices to set up explicit routes. Instead, one can rely on the MPLS traffic engineering functionality to understand the network topology and the automated signalling process.

In our case however, we choose to use the static option of the MPLS LSP establishment. Meaning, that we mapped explicitly the two different traffic types experimented with within the earlier shortest path routing experiment to their separate paths. We made the TCP flow to take the non-shortest path while letting the UDP traffic consume the shortest path resources. Then, we compared the results gathered with the shortest path routing experiment, comparing queuing delay, throughput and link utilization. We registered that both traffic types gained performance when getting traffic engineered to separate paths towards their destination. This made TCP flow to keep up its traffic intensity without facing suppression from the UDP traffic. However, there were registered some traffic intensity degradation with TCP throughput from the transmitter towards the ingress router. This was not to be blamed on MPLS-TE and its protocol behaviour, but the fact that the ingress router became busier forwarding more packets. Also, in the case of efficient resource utilization, the results were much more satisfying. Both paths between the ingress and egress router were now utilized compared with the shortest path routing scenario. This verified to us that with several paths available between any pair of ingress and egress routers, traffic engineering could be done with MPLS to avoid bottlenecks and congestion within the network. It can also help utilising network resources more efficiently by utilising other paths than the shortest path. The simulation network we experimented on was a small and simple network, but it could represent any part of a larger part of any autonomous system. Therefore, we believe that the results presented in this thesis are representative.

From experimentations conducted with MPLS, we believe traffic engineering can be supported in order to control the traffic to utilise desired paths through a network. With this control, it can expect to deliver a more accurate service level agreement to its customer and minimize the cost of delivering services, especially the cost of utilizing expensive network resources.

8.3 Conclusion made from MPLS traffic engineering with QoS support

The default service offering associated with the Internet is characterized as a best-effort variable service response. Within this service profile the network makes no attempt to actively differentiate its service response between the traffic

streams generated by concurrent users of the network. As the load generated by the active traffic flows within the network varies, the network's best effort service response will also vary.

The objective of various Internet Quality of Service (QoS) efforts is to augment this base service with a number of selectable service responses. These service responses may be distinguished from the best-effort service by some form of superior service level, or they may be distinguished by providing a predictable service response which is unaffected by external conditions such as the number of concurrent traffic flows, or their generated traffic load.

Any network service response is an outcome of the resources available to service a load, and the level of the load itself. To offer such distinguished services there is not only a requirement to provide a differentiated service response within the network, there is also a requirement to control the service-qualified load admitted into the network, so that the resources allocated by the network to support a particular service response are capable of providing that response for the imposed load. As a general observation of QoS architectures, the service load control aspect of QoS is perhaps the most troubling component of the architecture. While there are a wide array of well understood service response mechanisms that are available to IP networks, matching a set of such mechanisms within a controlled environment to respond to a set of service loads to achieve a completely consistent service response remains an area of weakness within existing IP QoS architectures.

This is where MPLS technology gets to be combined with differentiated services to offer this control. MPLS will set up a route for a flow and specify a next hop, while differentiated services will specify the treatment of a packet waiting to make that next hop. MPLS can therefore be the one to control the amount of traffic imposed on a router by its trunk reservation capability, admitting no more traffic within each trunk as there are assumed to be resources in the network to handle the traffic load.

In our last experiential network, we took advantage of the positive outcomes of MPLS traffic engineering to combine it with the QoS architectural abilities of differentiated service. This time, we aimed to assure QoS to flows of traffic, which demanded some level of QoS at the same time, as they were traffic engineered through a LSP. The two flowspecs were to take care of the EF CoS traffic and AF11 CoS traffic. We managed to configure the MPLS-TE network to cooperate with diffserv’s weighted fair queuing to administer the QoS aspect of the network. By assigning a higher weight value to the EF CoS traffic, we found out how different CoS flows were treated while they were traffic engineered. The registered measurements revealed that the amount of queuing delay, queuing buffer usage, flow- delay and throughput of these different CoS traffic flows differed in favour of the higher CoS traffic. The higher priority flow received in all measurements a better QoS support compared with the lower valued CoS flow. Also, if further flows of traffic with the same CoS were imposed to the ingress router, no further trunks would be needed. Traffics with the same CoS would become aggregated to use the same CoS traffic trunk.

We conclude that the MPLS-TE and differentiated services architecture combined is a useful tool for performing traffic engineering with quality of service support. Allowing a service provider to control the path a flow would use, plus the amount of traffic allowed into the network and at the same time providing it with the level of quality it requires.

8.4 Further need for research

We leave two areas related to our simulation of MPLS traffic engineering and QoS support with diffserv to be further investigated. First, failures within the network should be researched on to see how MPLS tackles them. We know that MPLS rerouting option is available to help with using backup LSPs configured. However, one should examine the impact of failures within the network to measure the impact on the traffic that are being engineered. It would be useful to research on the amount of timing and other interesting matters of the shifting of traffic from one LSP to another.

Furthermore, it would be interesting to do some more research on traffic aggregation capability of the MPLS and diffserv. Even though we managed to traffic engineer flows with QoS support, we wanted to do some more extensive and concentrated research on this area, but the shortage of time made us to concentrate upon the discoveries outlined. This is definitely one of the most important areas of concern for network service providers.

Other areas left out from our work, but related to MPLS are VPN and Multicast. Both of these are subject to be functional with MPLS. For the interested ones, these are also subject to be further elaborated on to measure their performance.

9 APPENDIX

9.1 Dijkstras Algorithm

The general method to solve the single-source shortest-path problem is known as Dijkstra's algorithm. This thirty-year old solution is a prime example of a greedy algorithm. Greedy algorithm, generally solve a problem in stages by doing what appears to be the best solution at each stage. At each stage, Dijkstra's algorithm selects a vertex v, which has the smallest distance d_v among all the unknown vertices, and declares that the shortest path from the source node to v is known. The reminder of a stage consists of updating the values for the distance d_w. The value of d_w gets lower if shorter path is discovered, thus setting $d_w = d_v + C_{v,w}$ if this new value for d_w would be an improvement. To put it simply, the algorithm decides whether or not it is a good idea to use v on the path to w. The original cost, d_w, is the cost without using v; the cost calculated above is the cheapest path using v (and only known vertices and not ∞). We therefore set $d_w = d_v + 1$ if $d_w = \infty$. The below figure illustrates the stages of Dijkstra's algorithm when executed from node v_1. The figure graphically shows how edges are narked known and vertices updated during Dijkstra's algorithm.

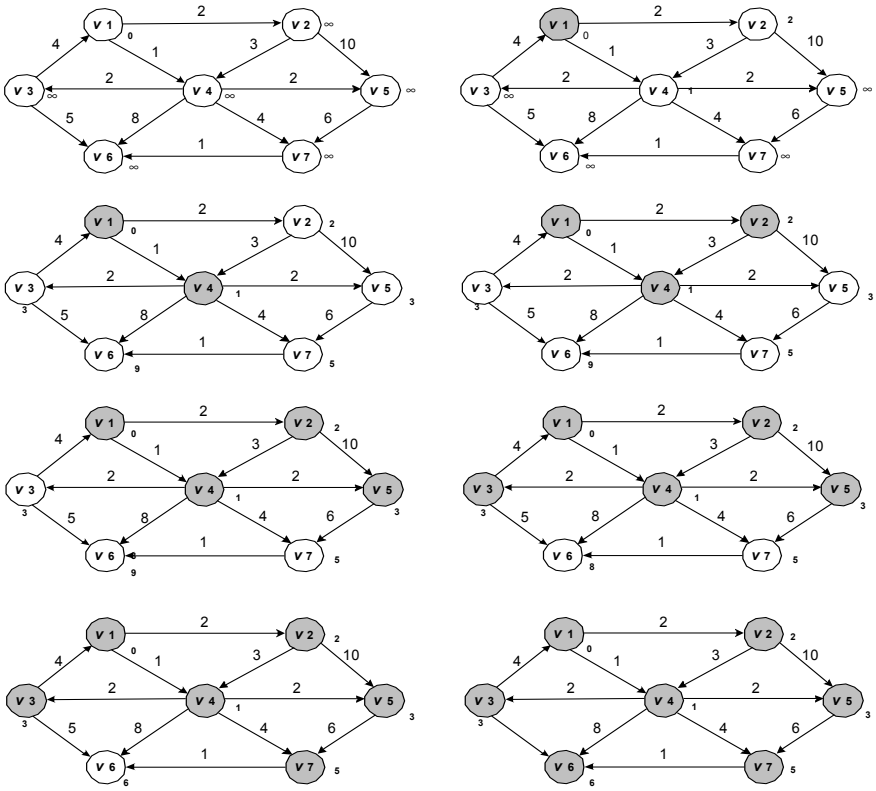

Figure 9.1 *Illustrates stages of Dijkstra's algorithm.*

9.2 Shortest Path Routing configuration details within OPNET

9.2.1 Application configuration

In order to generate TCP and UDP traffic within our simulation, we chose file transfer and video conferencing as application. File transfer was a good choice in our simulation due to the fact that we were able to define how large the file/packet size, which was going to be uploaded, would be. We therefore defined a traffic intensity of the one stated in the below table. The traffic intensity was 1,5 *Mbytes* of packets being uploaded every second. This was done to keep the pipe almost fully busy. The table below gives a description of the FTP configuration parameters.

Attribute	Value	Details
Command Mix (Get/Total)	0%	Only uploading.
Inter Request Time (seconds)	Constant (1)	Constant every second
File Size (bytes)	Constant (187500)	1,5 Mbytes file size
Symbolic Server Name	FTP Server	
Type of Service	Best Effort (0)	

Table 9.1 *FTP Table configuration parameters*

TCP Parameters table:

MSS: Auto assigned

Maximum Segment Size (MSS) that the underlying network can carry without any fragmentation. Used to determine the size of segments sent by TCP. If "Auto-Assigned", TCP will calculate this parameter based on the MTU size of the first IP interface on the surrounding node. In case of more than one interfaces, it will compute the MSS based on the first interface type.

Received buffer (bytes): 8760

Size of the buffer holding received data before it is forwarded to the higher layers (e.g. applications). Note that the advertised window is the amount of space available in the receive buffer. When set to "Default", this parameter is set to at least four times the "negotiated" MSS, with a maximum size of 64 KB unless a window scaling option is in effect.

Transceiver buffer usage threshold: 0.0

Threshold used to determine the limit on the usage of receive buffer before transferring segments out to the socket buffer.
Setting this value to 0.0 is equivalent to modelling a TCP implementation in which the receiver always advertises a constant receive buffer size (e.g., some versions of BSD)

Delayed ack mechanism: segment/clock

Specifies the scheme used to generate dateless ACKs:

1. Clock Based: TCP sends a dateless ACK if no data is sent for "max_ack_delay" time interval.

2. Segment/Clock Based: Generates an ACK every other received segment, or every "max_ack_delay" time interval, if two segments are not received within this interval.

Note that for most Sun implementations, it should be set to "Clock Based", whereas for "Miscrosoft Windows" implemetations, it should be set to "Segment/ Clock Based."

Maximum ACK Delay (sec): 0.200

Maximum time the TCP waits after receiving a segment before sending an ACK. Note that the acknowledgment may be piggybacked on a data packet. For most SUN systems implementations, it value is 50 msec (configurable) whereas for Windows TCP implementation it is set to 200 msec.

Show-start initial Count (MSS): 1

Specifies the number of MSS-sized TCP segments that will be sent upon slow-start. This also represents the value of the initial congestion window (or "cwnd"). RFC-2414 upper bounds this initial window as:
min[4*MSS,max(2*MSS,4380bytes)]

ECN Capability : Disabled

Specifies if TCP implementation supports explicit congestion notification (ECN). Both sides must exchange support for ESN before making use of this feature (documented in details in RFC-3168).

Fast Retransmit Enabled

RENO

Fast Recovery: Disabled

Indicates whether this host uses Fast Retransmir Algorithm as described in RFC 2001. If "Disabled" then slow start and congestion control algorithm along with Fast Retransmit (if enabled) will be executed. If set to "Reno", fast retransmit as defined in RFC 2001 will be executed once the node receives n-th duplicate acknowledgement. If set to "New Reno", fast retransmit as described in RFC 2001 will be executed with the two modifications to the algorithm- fast retransmit will never be executed twice within one window of data- if a partial acknowledgement (acknowledgement advancing snd_una) is received, the process will immediately retransmit the next unacknowledget segment.

Window scaling: Disabled

Indicates whether this host sends the Window Scaling enabled option in its SYN. If the option is both sent and received, Window Scaling will proceed as detailed in RFC 1323.

Selective ACK (SACK): Disabled

Indicates whether this host sends the Selective Acknowledgement Permitted option in its SYN. If the option is both sent and received, SACKs will be sent as detailed in RFC 2018.

Segment send treshold: Byte Boundary

Determines the segment size, and granularity of calculation of slow start threshold (ssthresh) variable. When set to "Byte Boundary":- a segment with any size allowed by the segment send algorithm can be sent, and- during fast retransmission slow start threshold will be set to half of the congestion window When set to "MSS Boundary":- a segment is sent only if its size equals the maximum segment size except when it is the last segment, and- the granularity of slow start threshold is one maximum segment size. Thereby, the ssthresh value after fast-recovery will be set to "((int) (cwnd/2)) * mss"

Nagle's SWS Avoidance: Disabled

Enables or disables use of Nagle's algorithm for sender-side Silly Window Syndrome (SWS) avoidance.

Karn's Algorithm: Enabled

Enables or disables the use of Karn's Algorithm for calculating retransmission timeout (RTO) values.

Retransmission Thresholds: Attempts based

Specifies the criteria used to limit the time for which retransmission of a segment is done.

Initial RTO (sec): 1.0

Retransmission timeout (RTO) value used before the RTO update algorithms come into effect.

Minimum RTO (sec): 0.5

Lower bound on the retransmission timeout (RTO) value.

Maximum RTO (SEC): 64

Upper bound on the retransmission timeout (RTO) value.

RTT Gain: 0.125

Gain used in updating the round trip time (RTT) measurement.

Deviation Gain: 0.25

Gain used to update the mean round trip deviation.

RTT Deviation Coefficient : 4.0

Coefficient used to determine the effect of mean deviation on the final calculated retransmission timeout (RTO) value.

Timer Granularity (sec): 0.5

Represents TCP slow timer duration (used to handle all timers except maximum ACK delay timer). Timer events are scheduled at multiples of the value assigned to this attribute.

Persistent Timeout (sec) : 1.0

Duration of the persistence timeout. This allows the local socket to receive a window update when the receiver window is very small.

The other application we specified in our simulation was the video conferencing. This application was specifically chosen because of its use of UDP as transfer protocol. The frame size was set to constant value of 3750 *bytes*. This gives us the traffic intensity of 3750 *bytes* x 10 *frames/sec* = 37500 *bytes/sec*, which gives the value of 37500 x 8 = 300,000 *bits/sec*. Which again multiplied with five such applications equals 1,500,000 *bits/sec.*

Attribute	Value	Details
Frame Interval Time Information	10 frames/sec	Constant
Frame size Information	3750 bytes	Constant
Symbolic Destination Name	Video Destination	
Type of Service	Best Effort (0)	
RSVP Parameters	None	
Traffic Mix (%)	All Discrete	

Table 9.2 *Video Conferencing table configuration parameters*

9.2.2 Profile configuration

In order to use the applications installed, we configured two profiles. The first profile was named TCP generator, set to use the file transfer application from the second minute. The profile was to start only once and the duration was set to the end of the simulation. The start time was set to constant distribution with the value of 120 (starting from the 2 min). The second profile was named UDP generator and was configured to start using the video conferencing one second later. Then, each second executing one extra video conferencing application. Executing total number of 5 applications. Table 9.3 and 9.4 summarizes.

Profile Name	Applications	Operation-mode	Start-time(sec)	Duration(sec)	Repeatability
TCP generator	(…)	Serial(ordered)	constant(120)	end of sim.	Once at start
UDP generator	(…)	Simultaneous	constant(121)	end of sim.	Once at start

Table 9.3 *Profile Configuration Table*

Name	Start Time Offset	Duration (seconds)	Repeatability
File Transfer (heavy)	No Offset	end of profile	Once at start
Video conf. (heavy)	No Offset	end of profile	Once at start
(UDP generator executes 5x video conf with 1 sec between each)			

Table 9.4 *Applications Table*

9.2.3 Workstations and Server configuration

In order to generate TCP and UDP traffic to measure their impact within the network, we configured file transfer service between site1 and site5. Site1 was set to use engineer1 as its profile, meaning that everything that we have described under the profile configuration section was now being used by site1. Site5 is a server, which accepts the uploaded traffic destined from site1. This means that site1 which is a workstation initiates an upload to site5. Site5 was configured to only respond to file transfer application. Site2 were configured to use engineer2 as its profile. It was configured to send video conferencing traffic to site4. Site4, were only configured to accept the traffic. This was done to control the traffic from one end of the network to another. Both, the file transfer and video conferencing services are based on best effort service, meaning that their ToS-values in the IP header were set to best effort (0) precedence. To summarize, table 9.5 shows the sites with their respective configuration.

Site	*Supported Protocol*	*Start*	*End time*	*Traffic-intensity*	*ToS*
Site1	TCP	2m:00s	10m:00s	1,500,000 bits/sec	Best effort(0)
Site2	UDP	3m:00s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	3m:45s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	4m:30s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	5m:15s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	6m:00s	10m:00s	300,000 bits/sec	Best effort(0)
Site4	UDP	N/A	N/A	N/A	N/A
Site5	TCP	N/A	N/A	N/A	N/A

Table 9.5 *A summarization over the traffic configuration*

We assigned IP-address in order to let the FEC- classes function.

Site	*IP-Address*	*Subnet Mask*
Site1	192.0.1.2	255.255.255.0
Site2	192.0.2.2	255.255.255.0
Site4	192.0.11.2	255.255.255.0
Site5	192.0.13.2	255.255.255.0

Table 9.6 *IP addressing of sites*

9.2.4 Router configuration

The ethernet2_slip8_gtwy node model represents an IP-based gateway supporting up to two Ethernet interfaces and up to 8 serial line interfaces at a selectable data rate. IP packets arriving on any interface are routed to the appropriate output interface based on their destination IP address. The Routing Information Protocol (RIP) or the Open Shortest Path First (OSPF) protocol may be used to automatically and dynamically create the gateway's routing tables and select routes in an adaptive manner. This gateway requires a fixed amount of time to route each packet, as determined by the "IP Forwarding Rate" attribute of the node. Packets are routed on a first-come-first-serve basis and may encounter queuing at the lower protocol layers, depending on the transmission rates of the corresponding output interfaces.

Protocols:

RIP, UDP, IP, Ethernet, Fast Ethernet,
Gigabit Ethernet, OSPF

Interconnections:

1) 2 Ethernet connections at a data rate of 10 Mbps, 100 Mbps, or 1000 Mbps.
2) 8 Serial Line IP connections at a selectable data rate

Attributes:

"IP Forwarding Rate": specifies the rate (in packets/second) at which the gateway can perform a routing decision for an arriving packet and transfer it to the appropriate output interface.

IP Processing information:
Datagram switching rate:500,000

Rate at which the traffic is switched at this node. Note that switching is only done for labeled packets (MPLS). All other packets are routed and undergo the IP Forwarding delay

Datagram forwarding rate: 50,000
Number of packets or bytes that are processed by the "forwarding processor" in one second. The unit associated with this value is specified in the Forwarding Rate Units attribute.

Forwarding rate units: packets/second

Memory size (bytes): 16MB

IP slot info:
Processor speed: 5000
This attribute sets the processing (forwarding) capacity of this slot's processor in packets or bits per second, depending on the value of the "Forwarding Mode" attribute. Alternatively, it can be thought of as the "service rate" of this slot's processor.

Processing mode: packet/second

Input and output buffer capacity: 8MB (shared)

Attribute	Value
Router ID	Auto Assigned
Autonomous System Number	Auto Assigned
Interface Information	(…)
Loopback Interfaces	(…)
Default route	Auto Assigned
Load Balancing Options	Destination-Based
Administrative Weights	Default

Table 9.7 *IP Routing Parameters Table*

Name	Status	Address	Subnet-mask	MTU(bytes)	Metric-info	Routing-Protocol	QoS-info
IF0	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF1	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF2	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF3	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF4	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF5	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF6	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF7	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF8	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF9	Active	Auto	Auto	Ethernet	Default	OSPF	None

Table 9.8 *Interface Information Table*

Attribute	Value
Start Time	uniform (5.0, 10.0)
Interface Information	(…)
Area Summarization	No Address Aggregation
Routing Table Interval (seconds)	60
SPF Calculation Parameters	Periodic

Table 9.9 *OSPF parameters Table within the routers*

9.2.5 Simulation configuration attributes

The following simulation attributes were modified in addition to those listed in chapter 9.2.5. The ones listed below are related to the MPLS experimentation scenario.

Attribute	Value
Duration	126 sec
Values per Statistic	1000
IP Dynamic routing protocol	Default
IP Interface addressing mode	Auto Assigned

Table 9.10 *Simulation configuration attributes*

9.3 MPLS-TE configuration details within OPNET

9.3.1 Application configuration

In order to generate TCP and UDP traffic within our simulation, we chose file transfer and video conferencing as application. File transfer was a good choice in our simulation due to the fact that we were able to define how large the file/packet size, which was going to be uploaded, would be. We therefore defined a traffic intensity of the one stated in the below table. The traffic intensity was 1,5 *Mbytes* of packets being uploaded every second. This was done to keep the pipe almost fully busy. The table below gives a description of the FTP configuration parameters.

Attribute	Value	Details
Command Mix (Get/Total)	0%	Only uploading.
Inter Request Time (seconds)	Constant (1)	Constant every second
File Size (bytes)	Constant (187500)	1,5 Mbytes file size
Symbolic Server Name	FTP Server	
Type of Service	Best Effort (0)	

Table 9.11 *FTP Table configuration parameters*

TCP Parameters table:

MSS: Auto assigned

Maximum Segment Size (MSS) that the underlying network can carry without any fragmentation. Used to determine the size of segments sent by TCP. If "Auto-Assigned", TCP will calculate this parameter based on the MTU size of the first IP interface on the surrounding node. In case of more than one interfaces, it will compute the MSS based on the first interface type.

Received buffer (bytes): 8760

Size of the buffer holding received data before it is forwarded to the higher layers (e.g. applications). Note that the advertised window is the amount of space available in the receive buffer. When set to "Default", this parameter is set to at least four times the "negotiated" MSS, with a maximum size of 64 KB unless a window scaling option is in effect.

Transceiver buffer usage threshold: 0.0

Threshold used to determine the limit on the usage of receive buffer before transferring segments out to the socket buffer.

Setting this value to 0.0 is equivalent to modelling a TCP implementation in which the receiver always advertises a constant receive buffer size (e.g., some versions of BSD)

Delayed ack mechanism: segment/clock

Specifies the scheme used to generate dateless ACKs:

1. Clock Based: TCP sends a dateless ACK if no data is sent for "max_ack_delay" time interval.

2. Segment/Clock Based: Generates an ACK every other received segment, or every "max_ack_delay" time interval, if two segments are not received within this interval.

Note that for most Sun implementations, it should be set to "Clock Based", whereas for "Miscrosoft Windows" implemetations, it should be set to "Segment/ Clock Based."

Maximum ACK Delay (sec): 0.200

Maximum time the TCP waits after receiving a segment before sending an ACK. Note that the acknowledgment may be piggybacked on a data packet. For most SUN systems implementations, it value is 50 msec (configurable) whereas for Windows TCP implementation it is set to 200 msec.

Show-start initial Count (MSS): 1

Specifies the number of MSS-sized TCP segments that will be sent upon slow-start. This also represents the value of the initial congestion window (or "cwnd"). RFC-2414 upper bounds this initial window as:
min[4*MSS,max(2*MSS,4380bytes)]

ECN Capability : Disabled

Specifies if TCP implementation supports explicit congestion notification (ECN). Both sides must exchange support for ESN before making use of this feature (documented in details in RFC-3168).

Fast Retransmit Enabled

RENO

Fast Recovery: Disabled

Indicates whether this host uses Fast Retransmit Algorithm as described in RFC 2001. If "Disabled" then slow start and congestion control algorithm along with Fast Retransmit (if enabled) will be executed. If set to "Reno", fast retransmit as defined in RFC 2001 will be executed once the node receives n-th duplicate acknowledgement. If set to "New Reno", fast retransmit as described in RFC 2001

will be executed with the two modifications to the algorithm- fast retransmit will never be executed twice within one window of data- if a partial acknowledgement (acknowledgement advancing snd_una) is received, the process will immediately retransmit the next unacknowledget segment.

Window scaling: Disabled

Indicates whether this host sends the Window Scaling enabled option in its SYN. If the option is both sent and received, Window Scaling will proceed as detailed in RFC 1323.

Selective ACK (SACK): Disabled

Indicates whether this host sends the Selective Acknowledgement Permitted option in its SYN. If the option is both sent and received, SACKs will be sent as detailed in RFC 2018.

Segment send treshold: Byte Boundary

Determines the segment size, and granularity of calculation of slow start threshold (ssthresh) variable. When set to "Byte Boundary":- a segment with any size allowed by the segment send algorithm can be sent, and- during fast retransmission slow start threshold will be set to half of the congestion window When set to "MSS Boundary":- a segment is sent only if its size equals the maximum segment size except when it is the last segment, and- the granularity of slow start threshold is one maximum segment size. Thereby, the ssthresh value after fast-recovery will be set to "((int) (cwnd/2)) * mss"

Nagle's SWS Avoidance: Disabled

Enables or disables use of Nagle's algorithm for sender-side Silly Window Syndrome (SWS) avoidance.

Karn's Algorithm: Enabled

Enables or disables the use of Karn's Algorithm for calculating retransmission timeout (RTO) values.

Retransmission Thresholds: Attempts based

Specifies the criteria used to limit the time for which retransmission of a segment is done.

Initial RTO (sec): 1.0

Retransmission timeout (RTO) value used before the RTO update algorithms come into effect.

Minimum RTO (sec): 0.5

Lower bound on the retransmission timeout (RTO) value.

Maximum RTO (SEC): 64

Upper bound on the retransmission timeout (RTO) value.

RTT Gain: 0.125

Gain used in updating the round trip time (RTT) measurement.

Deviation Gain: 0.25

Gain used to update the mean round trip deviation.

RTT Deviation Coefficient : 4.0

Coefficient used to determine the effect of mean deviation on the final calculated retransmission timeout (RTO) value.

Timer Granularity (sec): 0.5

Represents TCP slow timer duration (used to handle all timers except maximum ACK delay timer). Timer events are scheduled at multiples of the value assigned to this attribute.

Persistent Timeout (sec) : 1.0

Duration of the persistence timeout. This allows the local socket to receive a window update when the receiver window is very small.

The other application we specified in our simulation was the video conferencing. This application was specifically chosen because of its use of UDP as transfer protocol. The frame size was set to constant value of 3750 *bytes*. This gives us the traffic intensity of 3750 *bytes* x 10 *frames/sec* = 37500 *bytes/sec*, which gives the value of 37500 x 8 = 300,000 *bits/sec*. Which again multiplied with five such applications equals 1,500,000 *bits/sec.*

Attribute	Value	Details
Frame Interval Time Information	10 frames/sec	Constant
Frame size Information	3750 bytes	Constant
Symbolic Destination Name	Video Destination	
Type of Service	Best Effort (0)	
RSVP Parameters	None	
Traffic Mix (%)	All Discrete	

Table 9.12 *Video Conferencing table configuration parameters*

9.3.2 Profile configuration

In order to use the applications installed, we configured two profiles. The first profile was named TCP generator, set to use the file transfer application from the second minute. The profile was to start only once and the duration was set to the end of the simulation. The start time was set to constant distribution with the value of 120 (starting from the 2 min). The second profile was named UDP generator and was configured to start using the video conferencing one second later. Then, each seconds executing one extra video conferencing application. Executing total number of 5 applications. Table 9.13 and 9.14 summarizes.

Profile Name	Applications	Operation-mode	Start-time(sec)	Duration(sec)	Repeatability
TCP generator	(…)	Serial(ordered)	constant(120)	end of sim.	Once at start
UDP generator	(…)	Simultaneous	constant(180)	end of sim.	Once at start

Table 9.13 *Profile Configuration Table*

Name	Start Time Offset	Duration (seconds)	Repeatability
File Transfer (heavy)	No Offset	end of profile	Once at start
Video conf. (heavy)	No Offset	end of profile	Once at start
(UDP generator executes 5x video conf with 45 sec between each)			

Table 9.14 *Applications Table*

9.3.3 Workstations and Server configuration

In order to generate TCP and UDP traffic to measure their impact within the network, we configured file transfer service between site1 and site5. Site1 was set to use engineer1 as its profile, meaning that everything that we have described under the profile configuration section was now being used by site1. Site5 is a server, which accepts the uploaded traffic destined from site1. This means that site1 which is a workstation initiates an upload to site5. Site5 was configured to only respond to file transfer application. Site2 were configured to use engineer2 as its profile. It was configured to send video conferencing traffic to site4. Site4, were only configured to accept the traffic. This was done to control the traffic from one end of the network to another. Both, the file transfer and video conferencing services are based on best effort service, meaning that their ToS-values in the IP header were set to best effort (0) precedence. To summarize, table 9.5 shows the sites with their respective configuration.

Site	*Supported Protocol*	*Start*	*End time*	*Traffic-intensity*	*ToS*
Site1	TCP	2m:00s	10m:00s	1,500,000 bits/sec	Best effort(0)
Site2	UDP	3m:00s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	3m:45s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	4m:30s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	5m:15s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	6m:00s	10m:00s	300,000 bits/sec	Best effort(0)
Site4	UDP	N/A	N/A	N/A	N/A
Site5	TCP	N/A	N/A	N/A	N/A

Table 9.15 *A summarization over the traffic configuration*

We assigned IP-address in order to let the FEC- classes function.

Site	*IP-Address*	*Subnet Mask*
Site1	192.0.1.2	255.255.255.0
Site2	192.0.2.2	255.255.255.0
Site4	192.0.11.2	255.255.255.0
Site5	192.0.13.2	255.255.255.0

Table 9.16 *IP addressing of sites*

9.3.4 Creating LSPs

The journey of a MPLS based packet starts and ends within a LSP. We therefore had to first install LSPs in our MPLS based experiential scenario. A full description of how this was done is given below. We start by explaining some of the important LSP attributes described below. Most of these attributes may also be configured through the LSP browser in OPNET. The update LSP Details operation creates traffic profiles and forward equivalence class (FECs) for the LSPs, which one can modify later as one fine tune the model. Both static and dynamic LSPs are supported in the MPLS module within OPNET. In our experiment, we did not however use dynamic LSPs, since the network model was not very large and we were interested to have a better control over the LSP establishment. This made it easier for us to have a better control over the network. We therefore defined by clicking on different routers, drawing our static LSPs between LERs.

In this scenario, we have chosen to keep the traffic intensity as the shortest path scenario. The only change we imposed in the model was that we forced TCP flows between site 1 and 5 use the red coloured LSP, while we force UDP flows between site 2 and 4 to use the blue coloured LSP. That is, we completely separate the TCP and UDP flows between ingress and egress routers along their path to destination sites. Figure 9.2 illustrates this. Our objective as stated earlier, were now to try to utilize the network resources more efficient while trying to impose a better chance for the TCP traffic to keep up its throughput while transmitting traffic.

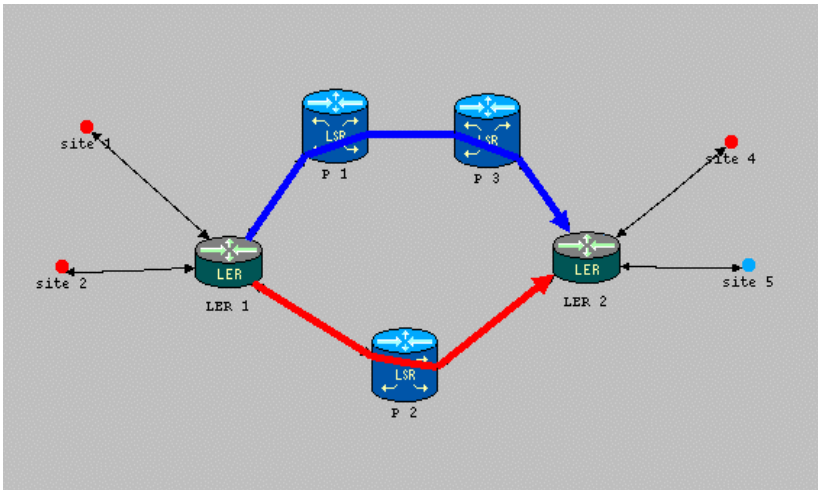

Figure 9.2 *LSPs from LER1 → LER 2*

The traffic is configured as before between sites, with the same intensity and type of service. Here we don't take into consideration any quality of service requirements, since our objective here is only to engineer traffic. Later we also take the QoS requirements into consideration when we engineer traffic, to measure its performance within the network.

9.3.5 MPLS configuration

Traffic engineering bindings governs how packets are labelled and forwarded in a network, by using FECs and traffic trunks to classify packets. These two important MPLS configuration attributes configured are described below. The first one, which is called FEC specifies the Forwarding Equivalence Class (FEC). FECs classify and group packets so that all packets in a group are forwarded in the same way. FECs are defined based on any of the IP header fields such as ToS, protocol, source address range, destination address range, source port, and destination port. When defining a FEC in the FEC details table, you can use any combination of IP header field configuration. We assigned four types of FECs based on destination address and protocol type used. Table 9.10 below describes the FEC configuration.

FEC name	*Protocol used*	*Destination address*
Site1	TCP	192.0.13.2
Site2	UDP	192.0.11.2

Table 9.17 *FEC specification table*

The traffic trunk profile attributes specifies out-of-profile actions and traffic classes for traffic trunks in the network. Traffic trunks capture traffic characteristics such as peak rate, average rate, and average burst size. To function correctly, the model requires that at least one default traffic trunk be configured. Additional trunks can be configured to handle prioritised flows. Two different traffic trunks were defined in our experiment. This was done to separate and apply equally amount of resources to the flows travelling through the MPLS configured network.

Flow	Max. Bit rate (bits/sec)	Average Bit Rate (bits/sec)	Max. Burst Size (bits)	Out of profile action
Flow1	1,544,000	1,500,000	64,000	Discard
Flow2	1,544,000	1,500,000	64,000	Discard

Table 9.18 *Trunk Configuration Table*

We only assigned IP-address in order to let the FEC- classes function.

Site	*IP-Address*	*Subnet Mask*
Site1	192.0.1.2	255.255.255.0
Site2	192.0.2.2	255.255.255.0
Site4	192.0.11.2	255.255.255.0
Site5	192.0.13.2	255.255.255.0

Table 9.19 *IP addressing of sites*

9.3.6 Router configuration

The ethernet2_slip8_gtwy node model represents an IP-based gateway supporting up to two Ethernet interfaces and up to 8 serial line interfaces at a selectable data rate. IP packets arriving on any interface are routed to the appropriate output interface based on their destination IP address. The Routing Information Protocol (RIP) or the Open Shortest Path First (OSPF) protocol may be used to automatically and dynamically create the gateway's routing tables and select routes in an adaptive manner. This gateway requires a fixed amount of time to route each packet, as determined by the "IP Forwarding Rate" attribute of the node. Packets are routed on a first-come-first-serve basis and may encounter queuing at the lower protocol layers, depending on the transmission rates of the corresponding output interfaces.

Protocols:

RIP, UDP, IP, Ethernet, Fast Ethernet,
Gigabit Ethernet, OSPF

Interconnections:

1) 2 Ethernet connections at a data rate of 10 Mbps, 100 Mbps, or 1000 Mbps.
2) 8 Serial Line IP connections at a selectable data rate

Attributes:

"IP Forwarding Rate": specifies the rate (in packets/second) at which the gateway can perform a routing decision for an arriving packet and transfer it to the appropriate output interface.

IP Processing information:
Datagram switching rate:500,000

Rate at which the traffic is switched at this node. Note that switching is only done for labeled packets (MPLS). All other packets are routed and undergo the IP Forwarding delay

Datagram forwarding rate: 50,000
Number of packets or bytes that are processed by the "forwarding processor" in one second. The unit associated with this value is specified in the Forwarding Rate Units attribute.

Forwarding rate units: packets/second

Memory size (bytes): 16MB

IP slot info:
Processor speed: 5000
This attribute sets the processing (forwarding) capacity of this slot's processor in packets or bits per second, depending on the value of the "Forwarding Mode" attribute. Alternatively, it can be thought of as the "service rate" of this slot's processor.

Processing mode: packet/second

Input and output buffer capacity: 8MB (shared)

Attribute	Value
Router ID	Auto Assigned
Autonomous System Number	Auto Assigned
Interface Information	(…)
Loopback Interfaces	(…)
Default route	Auto Assigned
Load Balancing Options	Destination-Based
Administrative Weights	Default

Table 9.20 *IP Routing Parameters Table*

Name	Status	Address	Subnet-mask	MTU(bytes)	Metric-info	Routing-Protocol	QoS-info
IF0	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF1	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF2	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF3	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF4	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF5	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF6	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF7	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF8	Active	Auto	Auto	Ethernet	Default	OSPF	None
IF9	Active	Auto	Auto	Ethernet	Default	OSPF	None

Table 9.21 *Interface Information Table*

Attribute	Value
Start Time	uniform (5.0, 10.0)
Interface Information	(…)
Area Summarization	No Address Aggregation
Routing Table Interval (seconds)	60
SPF Calculation Parameters	Periodic

Table 9.22 *OSPF parameters Table within the routers*

The routers had to be configured to function properly in the MPLS capable networking environment. The edge routers specially had to be configured. The traffic engineering configuration attribute specifies bindings between FECs and LSPs. Each traffic engineering binding specifies the FEC, traffic trunk, and LSP that is applied to the label of the incoming packet. When an unlabeled packet arrives at the ingress LER, the following sequence occurs to determine the appropriate label for the packet:

1. The TE binding is selected based on the packet's FEC and the incoming interface.
2. The packet is checked to make sure that its traffic characteristics conform to those specified for the TE binding's traffic trunks.
3. The packet is then assign a label and sent through the primary LSP specified for the TE binding.

Each of the two FECs was mapped to their own traffic trunks. This was a very easy task within OPNET. This task would probably consume much more time in the real world of router configuration. After creating the LSPs, FECs and traffic trunks, we created TE bindings that governed which packets would be sent to which LSPs. Table 9.13, shows the configuration of the MPLS parameters table within the LER1 router. It shows how the interface to FEC, trunk and LSP combination are established within the ingress router.

Interface In	FEC	Traffic flow	LSP
0	FEC Site 1	flow 1	Red
1	FEC Site 2	flow 2	Blue

Table 9.23 *LER1 MPLS parameter table*

9.3.7 Simulation configuration attributes

The following simulation attributes were modified in addition to those listed in chapter 9.2.5. The ones listed below are related to the MPLS experimentation scenario.

Attribute	Value
Duration	125sec
Values per Statistic	1000
IP Dynamic routing protocol	Default
IP Interface addressing mode	Auto Assigned
LSP Routing Protocol	IGP
LSP Signaling Protocol	RSVP
LSP Start Time	90

Table 9.24 *Simulation configuration attributes*

9.4 MPLS-TE-QoS supported flows config. details within OPNET

9.4.1 Application configuration

Applications used in this experiment, differs not from the ones used in our earlier shortest path and MPLS-TE experiment. The only changes we made were to assign them better class of service. We therefore refer to earlier description of application used in the shortest path routing configured experimentation for further details on the applications themselves.

9.4.2 Profile configuration

The same goes for the profile configuration. Here too, we have chosen to use our earlier defined profiles configured in the shortest path routing and MPLS-TE experiential network.. Table 9.25 and 9.26 outlines the configuration made.

Profile Name	Applications	Operation-mode	Start-time(sec)	Duration(sec)	Repeatability
TCP generator	(…)	Serial(ordered)	constant(120)	end of sim.	Once at start
UDP generator	(…)	Simultaneous	constant(121)	end of sim.	Once at start

Table 9.25 *Profile Configuration Table*

Name	Start Time Offset	Duration (seconds)	Repeatability
File Transfer (heavy)	No Offset	end of profile	Once at start
Video conf. (heavy)	No Offset	end of profile	Once at start
(UDP generator executes 5x video conf with a second between each)			

Table 9.26 *Applications Table*

9.4.3 Creating LSP

Referring to the MPLS-TE Creating LSPs section 9.3.3.
Only this time establishing one of those LSPs installed in the MPLS-TE network.

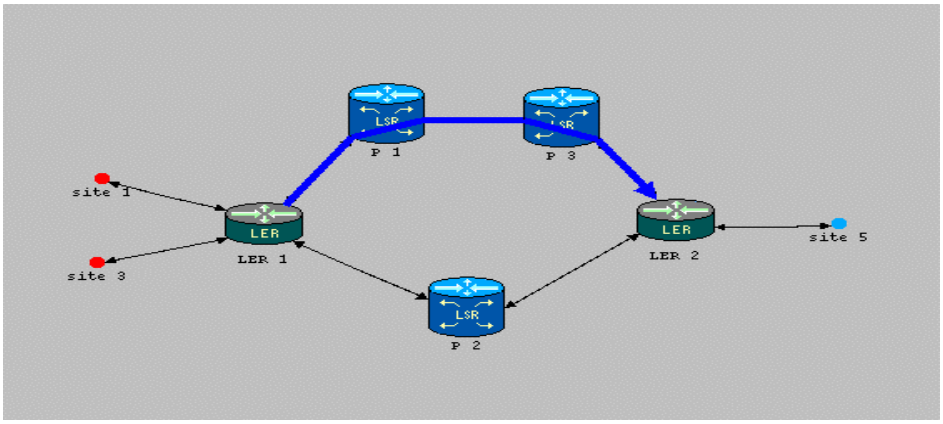

Figure 9.3 *LSPs from LER1 → LER 2*

9.4.4 MPLS configuration

Referring to the MPLS-TE MPLS configuration section 9.3.4.
The changes on FECs and traffic trunk concerned QoS support.

FEC name	*DSCP*	*Protocol used*	*Destination address*
EF TCP	EF	TCP	192.0.13.2
AF11 UDP	AF11	UDP	192.0.11.2

Table 9.27 *FEC specification table*

Flow	Max. Bit rate (bits/sec)	Average Bit Rate (bits/sec)	Max. Burst Size (bits)	Out of profile action	Traffic class
EF flow	2,000,000	1,000,000	64,000	Discard	EF
AF11flow	1,000,000	500,000	64,000	Discard	AF11

Table 9.28 *Trunk Configuration Table*

EXP	*PHB*
0	AF11
6/7	EF

Table 9.29 *EXP to PHB mappings*

9.4.5 QoS Configuration attributes

To configure Weighted Fair Queuing (WFQ), we present the table below.

Weight	Max queuing	classification scheme	queuing category
5	100	AF11	Default Queue
55	500	EF	Low Latency queuing

Table 9.30 *Weighted Fair Queuing details*

Weight is only applicable for WFQ. Weights are attributed to each queue. The weight indicates the allocated bandwidth for the queue. A higher weight indicates larger allocated bandwidth and shorter delays. If a queue is configured as a Low Latency Queue the Weight attribute of this queue is not used and the WFQ scheduler will ignore the value. Max queuing controls the maximum number of packets per queue. Used when the interface is congested (when the total number of buffered packets in all the queues is reached). Classification scheme compare DSCP values, source/destination addresses, protocols and incoming interface values to combine the right weight and queue with the packet. The Queue Category attribute determines whether the queue has the Default Queue and/or Low Latency Queue property. Low Latency Queuing introduces strict priority into WFQ. The Low Latency Queue enables use of a single, strict priority queue for delay-sensitive traffic. Traffic in this queue gets the highest priority, and only if this queue is empty, are other queues allowed to send traffic according to the traditional WFQ mechanism. The Default Queue receives all traffic that does not match the classification criteria of any of the existing queues. Only one Default Queue can be configured for the given queuing environment. If an incoming packet doesn't comply with any of the user-defined criteria, it is put in the default queue (0: Best-Effort).

9.4.6 Workstations and Server configuration

Referring to the MPLS-TE Workstation and server configuration section 9.3.5.

Site	*Supported Protocol*	*Start*	*End time*	*Traffic-intensity*	*ToS*
Site1	TCP	2m:00s	10m:00s	1,500,000 bits/sec	Best effort(0)
Site2	UDP	3m:00s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	3m:45s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	4m:30s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	5m:15s	10m:00s	300,000 bits/sec	Best effort(0)
Site2	UDP	6m:00s	10m:00s	300,000 bits/sec	Best effort(0)
Site4	UDP	N/A	N/A	N/A	N/A
Site5	TCP	N/A	N/A	N/A	N/A

Table 9.31 *A summarization over the traffic configuration*

Site	*IP-Address*	*Subnet Mask*
Site1	192.0.1.2	255.255.255.0
Site2	192.0.8.2	255.255.255.0
Site4	192.0.11.2	255.255.255.0
Site5	192.0.13.2	255.255.255.0

Table 9.32 *IP addressing of sites*

9.4.7 Router configuration

Referring to the MPLS-TE Router configuration section 9.3.6, we modified the FECs and their respective flowspec and LSP usage. Also, the routers interfaces had to be configured to be aware about the per- hop behaviour of the packets travelling through them. Therefore, QoS information attribute was been enabled. The below tables highlights these modifications made.

Interface In	FEC	Traffic flow	LSP
0	EF TCP	EF flow	Blue
4	AF11 TCP	AF11 flow	Blue

Table 9.33 *LER1 MPLS parameter table*

Buffer size(bytes)	Queuing Scheme	Queuing Profile
100000	WFQ	DSCP based

Table 9.34 *Every routers interface QoS information configuration*

9.4.8 Simulation configuration attributes

Referring to the MPLS-TE simulation configuration attributes section 9.3.7.

10 REFERENCES

[1] E. Rosen, A. Viswanathan, R. Callon, " Multi Protocol Label Switching Architecture" RFC 3031, January 2001.

[2] Daniel O. Aweduche, "MPLS and Traffic Engineering in IP Networks," IEEE Communication Magazine December 1999, UUNET (MCI Worldcom).

[3] E. Rosen, D. Tappen, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, A. Conta, " MPLS Label Stack Encoding", RFC 3032, January 2001.

[4] N. Shen and H. Smit, "Calculating IGP Routes over Traffic Engineering Tunnels," IETF Internet draft, work in progress, June 1999.

[5] Xipeng. Xiao, Alan. Hannan, Brook Bailey, Lionel M. Ni, "Traffic Engineering with MPLS in the Internet," IEEE Network March/April 2000, GlobalCenter inc., Michigan State University.

[6] D. Aweduche et al., "Requirements for Traffic Engineering over MPLS," RFC 2702, Sept. 1999.

[7] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, and Jennifer Rexford, et al. "NetScope: Traffic Engineering for IP Networks", IEEE Networks, March/April 2000.

[8] Hang Lu, Ruifeng Wang, Yugeng Sun, "An Architecture of Traffic Engineering," IEEE 2000, 0-7803-6253-5/00.

[9] D Aweduche, L Berger, D Gan, T Li, G Swallow and V Srinivasan, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, December 2001.

[10] Byeongsik Kim, Woojik Chun, Jaeho Yoo, "Constraint-based LSP Setup by Message Reversing of CR-LDP", IEEE 2001, 0-7695-0951-7/01

[11] Aweduche, et al., IEEE Communications, 12/1999, 42-47.

[12] D Bertsekas & R Gallager, "Data Networks", Second Edition, Prentce Hall 1992.

[13] J. Moy,"OSPF version 2", RFC 2328, Ascend Communications Inc. April 1998.

[14] OPNET Online Manual, www.opnet.com

[15] Wei Sun, Praveen Bhaniramka, Raj Jain, ”Quality of Service using Traffic Engineering over MPLS: An analysis”, IEEE 2000, 0-7695-0912-6/00.

[16] Xipeng Xiao and Lionel M. Ni, ”Internet QoS: A Big Picture,” Michigan State University, IEEE Network, March/April 1999.

[17] R. Braden, D. Clark, and S. Shenker, “Integrated Services in the Internet Architecture: An Overview,” Internet RFC 1633, June 1994.

[18] S. Shenker, C. Partridge and R. Guerin, ”Specification of Guaranteed Quality of service,” RFC 2212, Sept. 1997.

[19] J. Wroclawski, “Specification of the controlled-Load Network Element Service,” RFC 2211, Sept. 1997.

[20] K. Nichols et al., “Definition of the Differentiated Services Field (DS *field*) in the IPv4 and IPv6 Headers,” RFC 2474, Dec. 1998.

[21] S. Blake et al., ”An Architecture for Differentiated Services,” RFC 2475, Dec. 1998.

[22] R. Braden et al., ”Resource Reservation Protocol (RSVP) – Version 1, Functional specification,” RFC 2205, September 1997.

[23] OPNET Technologies Inc., "OPNET Modeler Modeling Manual", Bethesda, MD, release 8.0.c, June 2001

[24] OPNET Technologies Inc., "OPNET Protocol Model Documentation", Bethesda, MD, release 8.0.c, June 2001

[25] P. Almquist, “Type Of Service in the Internet protocol suite”, RFC 1349, July 1992.

[26] J. Heinanen, F. Baker,Weiss, W. And J. Wrocklawski, ” Assured Forwarding PHB Group”, RFC 2597, June 1999.

[27] C. Hedrick,”Routing Information Protocol”, RFC 1058, Rutgers University, June 1988.

[28] D. Oran, “OSI IS-IS Intra-domain Routing Protocol”, RFC 1142, Digital Equipment Corp., February 1990.

[29] L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas, ” LDP Specification”, RFC 3036, January 2001.

[30] B. Thomas, E. Gray, “LDP Applicability”, RFC 3037, January 2001.

[31] B. Jamoussi et al., ”Constraint-Based LSP Setup Using LDP”, RFC 3212, January 2002.

[32] J. Ash, M. Girish, E. Gray, B. Jamoussi, G. Wright, “Applicability Statement for CR-LDP”, RFC 3213, January 2002.

[33] J. Ash, Y. Lee, P. Ashwood- Smith, B. Jamoussi, D. Fedyk, D. Skalecki, L Li, “LSP Modification Using CR-LDP”, RFC 3214, January 2002.

[34] D. Awduche, A. Hannan, X. Xiao, “Applicability Statement for Extensions to RSVP for LSP-Tunnels”, RFC 3210, December 2001.

[35] A. Mankin et al., ” Resource Reservation Protocol (RSVP) Version 1 Applicability Statement Some Guidelines on Deployment”, RFC 2208, September 1997.

[36] T. Li, Y. Rekhter, ”A Provider Architecture for Differentiated Services and Traffic Engineering (PASTE) ” RFC 2430, October 1998.

[37] D Aweduche, L Berger, D Gan, T Li, G Swallow and V Srinivasan, “RSVP-TE: Extension to RSVP for LSP Tunnels”, RFC 3209, December 2001.

[38] V. Jacobson, K. Nicholes, K. Poduri, “An Expedited Forwarding PHB”, RFC 2598, June 1999.

[39] A. Apostolopoulos, R. Guerin, S. Kamat, Orda, T. Przygienda, and D. Williams, “QoS Routing Mechanisms and OSPF Extensions”, RFC 2676, August 1999.

Department of Informatics
University of Oslo, Norway
Gaustadalleen 23
Postboks 1080 Blindern
0316 Oslo
Norway

UNIK – University Graduate Centre of Technology
P.O BOX 70
N-2007 Kjeller
Norway

01-03