ABSTRACT

Kulkarni, Amit Narayan. An Investigation of Forwarding in the MPLS support for Differentiated Services. (Under the guidance of Dr. Mladen A. Vouk.)

The changing nature of the Internet-based applications is imposing stricter demands on the performance of the Internet. As the Internet resources become more and more constrained, the Best Effort (BE) model is increasingly proving less capable of providing the required Quality of Service (QoS). One of the solutions recently proposed by IETF is the Differentiated Services (DiffServ) architecture, which can provide different levels of QoS to each class by aggregating traffic into different classes at the network edge, and by giving differential treatment for each class within the core of the network. DiffServ, however, performs within the limits of the resources along the shortest path, and hence its performance is a function of resource availability along that path. Another standard proposed by IETF is Multi Protocol Label Switching (MPLS), a fast switching based technique that offers new capabilities for IP based networks. It combines the control of IP routing with efficiency of layer 2 switching. Traffic engineering (TE), or the ability to map traffic flows onto an existing physical topology is an example of key application of MPLS. MPLS and DiffServ, though independently developed, are complimentary technologies in the pursuit of end to end QoS.

An IETF RFC provides a guideline and requirements for MPLS support for Differentiated services. We studied this RFC in terms of what it does and does not specify about MPLS-DiffServ . We investigate the issues involved in implementation of the forwarding component of MPLS-DiffServ and evaluated the implementation vis-à-vis its functional requirements, its performance, and its ability to deliver better QoS. We conclude that MPLS-Diffserv does provide IP services a greater control over the network while simultaneously being able to deliver Different service levels.

An Investigation of Forwarding in the MPLS support for Differentiated Services.

By

Amit N Kulkarni

A thesis submitted to the graduate faculty of
North Carolina State University
in partial fulfillment of the requirements for the degree of
Master of Science.

Computer Science

Raleigh

2002

APPROVED BY

Dr Gorge Rouskas

Dr. Rudra Dutta

Dr Mladen A. Vouk
(Chair of Advisory Committee)

Dedication

To my Father, Late Mr. Narayan Y. Kulkarni

Biography

Amit Kulkarni was born and brought up in Nashik, a city in western Indian state of Maharashtra. He completed his bachelors from Government College of Engineering Pune. He then joined North Carolina State University to obtain Masters degree in Computer Science. He worked as a Research Assistant to Dr Vouk in the field of Differentiated Services and MPLS for around two years.

Acknowledgements

I take this opportunity to express deepest gratitude to Dr Mladen Vouk. It was an honor and privilege to work with him. His knowledge and patience are only few of the many of his qualities, which I would try to imbibe in myself. I thank him for giving me this opportunity. I would also like to thank Dr Dutta and Dr Rouskas for agreeing to be on my committee despite such short notice. I thank Dr Mike Westall of Clemson University, from the bottom of my heart for his guidance and Linux expertise which helped me surmount numerous kernel crashes.

I thank Marhn Fullmer for providing me the facilities and supporting our research needs in the networking lab. I wish to thank Kesava Narasimhan. His knowledge and expertise are a source of inspiration. He has always been there for me to help and guide. I thank fellow members of the research team Zyad Dwekat, Mukul Senapati for allowing me to bounce my ideas off them and keeping me sane. I thank Mrugendra Singhai. He's been my partner in crime for last two years and a constant source of inspiration. I would also like to thank friend and co-worker Aric Lambert at IP Infusion for constantly encouraging me to complete my write-up.

Mere words are insufficient to express my gratitude and love for my parents my two sisters and my wife Nivedita. Without their incredible support, this would not have been possible.

Table of Contents

List of Figures

1 Introduction

1.1 Need for QoS

Internet has grown at a frenetic pace ever since the first network was constructed in 1965 [History] and with it has changed the way people look at Internet. No one considers Internet as a research product privy of some government institutes anymore, on the contrary Internet has become an integral part of our lives and nerve center of the global economy. From being a file transfer mechanism and a means for instant communication Internet has played host to a variety of real time multimedia applications, Video conferencing, Voice over IP and many more. These applications have not only put strain on the existing network resources but also their varied nature has brought with it differences in requirements to be serviced. For example a voice application is extremely sensitive to delay and delay variation but can tolerate some loss, a video application can sustain higher delay than voice applications but data loss may have noticeable effects where as data application is tolerant to higher delay and delay variation.[Williams et.al.]

The classic IP is unable to cater to the “Quality of Service” requirements of these demanding applications. Ironically, the very features that has helped it succeed, connection less protocol, Best effort model, ease of implementation, had proved insufficient to meet the new challenges and general perception was that change was required.

1.2 Solutions for QoS

Some technologies e.g. Type of Service (TOS) and Integrated Service (IntServ) were created in an attempt to provide some quality of service (QoS) control. However their limitations have restricted their general application and they have been unable to provide a framework for provision of service to meet service level agreements (SLA's).

One of the earliest techniques to be implemented, Type of Service allowed network to distinguish between network control traffic and user traffic based on the TOS byte field in the IP header [RFC791]. It provides a coarse-grained classification and identification of limited number of flows. However since the traffic classes were defined much earlier they do not reflect well the current needs of the network and with no provision to define more classes this byte is often not fully supported in routers.

Another technique that was defined was IntServ [RFC-INTSERV]. IntServ provides well-defined end-to-end QoS for point to point and point to multi point applications. In this architecture the application initiates a session on demand with the network using Resource Reservation Signaling Protocol (RSVP). This session identifies the service requirements of the application including the bandwidth and delay, source of data. RSVP is a soft state protocol that allows merging of resource requests. While this is very powerful for small networks in terms of guaranteeing, the overhead of maintaining state per flow state and the processing power required makes this solution not very attractive for the core which can have thousands of such flows.

It was obvious something different was required. Two protocols proposed recently by IETF, Differentiated Services (DiffServ) [RFC-DSARCH] and MultiProtocol Label Switching (MPLS)[RFC-MPLSARCH] are generating a lot of interest. Both aggregate the traffic at the edge and process on the aggregate at the core. MPLS though cannot offer different level of service within the same class and DiffServ cannot provide the ability to engineer the traffic with consideration for the resource constraints. Thus despite their advantages these technologies have limitations in delivering end to end QoS.

1.3 Goals and Objectives

The goal of the project was to experimentally evaluate the emerging IETF proposals and standards related to delivery of end-to-end QoS, specifically where it concerns Differentiated Services (DiffServ) support for QoS sensitive traffic engineering based on MPLS [RFC_MPLSDS].

The first objective was to construct a re-usable test-bed where the experiments and evaluations could be conducted. We had an existing Linux-based test–bed for DiffServ evaluation and research [Narasimhan, Dwekat]. This test-bed was to be extended to include MPLS, and MPLS support for DiffServ, based on IETF standards.

The second objective was to use the test-bed to understand a specific element of an end-to-end architecture that arises in MPLS assisted PHB and PDB formulation – the function of forwarding. MPLS forwarding was to be studied in relation to the implementation of the existing and proposed MPLS, QoS modifications and extensions, and with respect to DiffServ based support. The basic test-bed we have was used previously to study DiffServ issues [Dwekat]. However, MPLS offers additional challenges, as well as an attractive range of capabilities since it separates forwarding and signaling work and each can be independent of the other as long as the interfaces are well defined. Traffic Engineering via MPLS is generating a lot of interest especially in the prevailing market conditions where the cost of network upgrade for over provisioning may be prohibitive. This forces network service providers to efficiently utilize the existing infrastructure and improve its performance.

So the basic practical challenge was to implement forwarding functionality of MPLS in a DiffServ environment, and gain an understanding of the issues relating to MPLS support for DiffServ architecture. Design goal was to enable the user to choose either protocol depending upon his/her requirements. Consideration was given to the fact that this code might be used in the future as a base to implement Label Distribution Protocol (LDP). Since the code had to conform to existing design architecture, performance at times was sacrificed for the sake of design consistency.

The next chapter provides the background for MPLS, DiffServ and concepts and issues in MPLS support for DiffServ. Chapter 3 Discusses Linux operating system support for MPLS, QoS and networking. Chapter 4 discusses our implementation chapter 5 discusses tests and evaluation of results. Chapter 6 describes the Network simulator and simulates the topology for detailed results. Chapter 7 concludes with summary and identifies future work.

2 Background

In this chapter we discuss the background materials related to DiffServ and MPLS.

2.1 Differentiated Services

The main goal of DiffServ architecture [RFC-DSARCH] was to provide a scalable framework for Quality of Service support without the need to maintain per flow state. This is mainly achieved by aggregating number of flows and giving it similar treatment. DiffServ nodes at the ingress of a domain process and mark the TOS byte in IP header of the packet by a code (called DiffServ Code Point or DSCP), based on a negotiated contract and other routers in the domain that receive the packet look at only the DSCP value to impart a particular treatment to the packet. This particular treatment is called a "Per-Hop Behavior"(PHB).

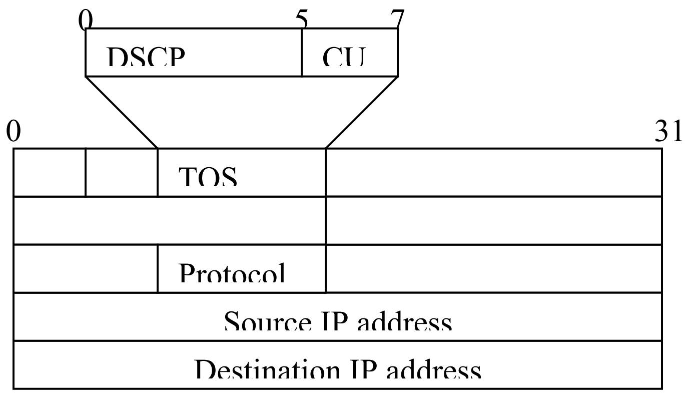

Fig 2.1 IP header fields

2.1.1 Per Hop Behavior:

PHB is defined as the externally observable behavior applied at a DS compliant node on a flow. It forms the basic building block of the DiffServ architecture. PHB is used to identify the treatment that will be given to a particular flow (or aggregate if the router is a core router). This treatment includes selection of queues and schedulers at the interface egress. The defined PHBs are

2.1.1.1 Expedited Forwarding (EF-PHB):

The aim of the EF PHB is to provide low delay and virtually no loss to some flows without per flow queuing. Loss, latency and jitter are all due to the queues traffic experiences while transiting the network. Therefore providing low loss, latency and jitter for some traffic aggregate means ensuring that the aggregate sees no (or very small) queues. Queues arise when (short-term) traffic arrival rate exceeds departure rate at some node. Thus a service that ensures no queues for some aggregate is equivalent to bounding rates such that, at every transit node, the aggregate's maximum arrival rate is less than that aggregate's minimum departure rate. The EF –PHB is useful for applications that are delay sensitive like voice and video. [RFC_EFPHB]

2.1.1.2 Assured Forwarding (AF- PHB)

This forwarding mechanism is intended for urgent data that requires controlled load service. Most network applications can do with some excess bandwidth if available. However the least they want is some guaranteed bandwidth even in times of heavy congestion. AF PHB provides for such a traffic demand.

AF PHB can offer different levels of forwarding assurances. The current AF specification provides delivery of packets in four classes each with three (two are also allowed)drop precedences. Packets in one class must be forwarded independently of the packets in another AF class [RFC-AFPHB].

A DS node should implement all four general use AF classes. Packets in one AF class must be forwarded independently from packets in another AF class, i.e., a DS node must not aggregate two or more AF classes together
Packets with the lowest drop precedence are assumed to be within the subscribed profile.
An AF compliant node allocates resources (buffer space and bandwidth) equal to atleast achieve the configured service bandwidth

2.1.1.3 Default PHB

Default PHB is the common best effort forwarding behavior, which is available in existing routers. This PHB is used when DSCP does not match to any other PHBs. A default PHB can be implemented by a queuing discipline that sends packets of this aggregate whenever the output link is not used by any other PHB. Network Dimensioning should ensure that this aggregate should not be starved. That way, senders that are not DiffServ aware can continue to use the network in the same manner as they do today.

2.1.2 Components of DiffServ architecture

Fig 2.2 shows a logical DiffServ router. DiffServ router consists of the following components

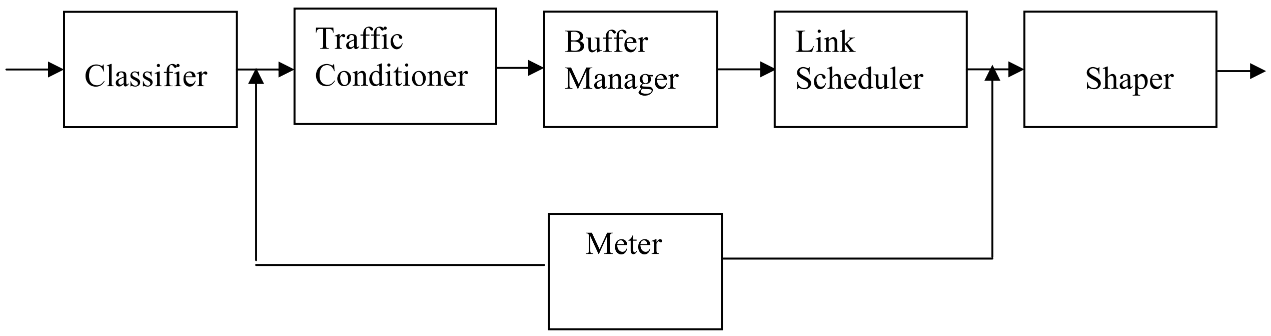

Fig 2.2 Logical components of DiffServ router

A classifier classifies the incoming packet into an appropriate behavioral aggregate and identifies the PHB treatment to be imparted to the flow. A traffic conditioner may condition the incoming flow to ensure the traffic meets the profile agreed upon in the SLA or equivalent agreement. A Buffer Manager and a Link Scheduler ensure hat appropriate treatment is imparted to the flow. Before packet leaves the DiffServ domain, it can be optionally shaped so that it is within the bounds of agreement with the next domain service provider. A more detailed explanation and its components can be obtained from [Narasimhan, Dwekat].

2.1.3 Advantages of DiffServ

2.1.3.1 Scalability:

Scalability is very important concern as a network core can have large number of flows and any protocol which requires to maintain per flow state or computational complexity does not scale well. DiffServ aggregates flows and hence can handle large number of flows. Also since PHBs are essentially kept simple, DiffServ lends itself well to use at high speeds making it scalable in terms of speed.

2.1.3.2 Ease of administering

In a Differentiated Services framework, different DiffServ domains can implement PHBs as they see fit as long as the bilateral agreements that it makes with the other domain are met. This gives the service providers a freedom to choose their implementation as a consequence they can provide Differentiated Services with minimal change in their infrastructure.

2.1.3.3 Simplicity

The DiffServ implementation does not diverge a lot from the basic IP. Hence it maintains simplicity and ease of implementation /upgradation at the cost of granularity.

2.1.3.3 Measurable

Since at each hop in a DiffServ domain, the traffic conditioners and shapers are constantly measuring arrival data and the link schedulers are monitoring packets to be sent, not much effort is required to procure vital information about the behavior of the network. The service providers can use the information to best allocate bandwidths and make service level agreements with the user.

2.1.4 Limitations

There appear to be two principal limitations to the DiffServ architecture as defined by [RFC-DSARCH]:

- DiffServ architecture suggests only mechanisms for relative packet forwarding treatment to aggregate flows, traffic management and conditioning. However it does not provide an architecture for end-to-end QoS.
- DiffServ framework does not lend itself to handle link failures. For example if a link carrying EF traffic, in DiffServ domain goes down, there is no way for the provider to quickly send the traffic through alternate link and ensure minimum packet loss.

Furthermore, there is no traffic engineering provision in DiffServ. As a result some links in the domain might experience congestion while other links go unutilized.

2.2 Multi Protocol Label Switching

MultiProtocol Label Switching (MPLS) is a versatile solution to address the problems faced by the present-day networks: speed, scalability, quality-of-service (QoS) management, and traffic engineering. MPLS has emerged as an elegant solution to meet the bandwidth-management and service requirements for next-generation Internet protocol (IP) based backbone networks. MPLS addresses issues related to scalability and routing (based on QoS and service quality metrics) and can exist over existing asynchronous transfer mode (ATM) and frame-relay networks. It eliminates the complexities of IP over ATM and provides IP routing with hitherto unavailable control. As shown in Fig 2.3, MPLS separates the forwarding and control, so each can develop independently of the other.

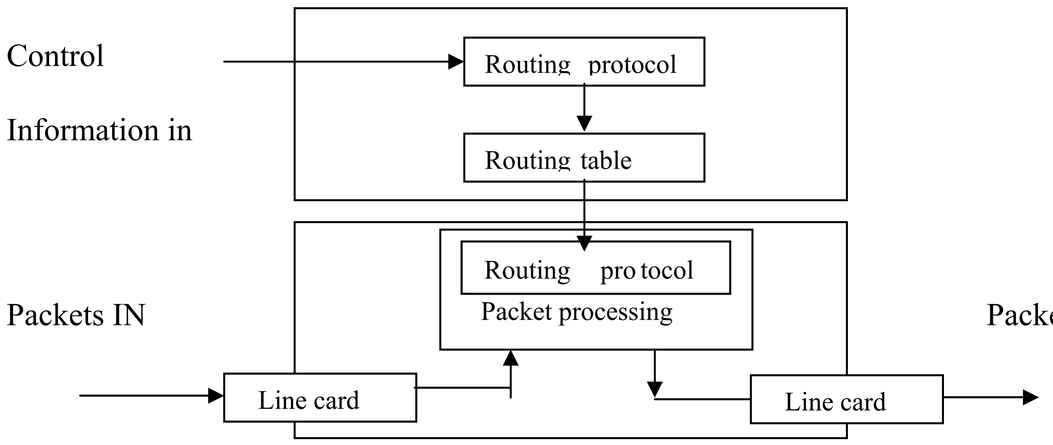

Fig 2.3 MPLS router schematic view

When a packet is received at an Ingress Label Switched Router (LSR), also known as Label Edge Router (LER), it identifies the Forwarding Equivalence Class (FEC) of the packet and assigns the label for that FEC based on the mapping in the Incoming Label Map (ILM also known as the Label Information Base or LIB). The ILM can be populated either manually at each node or by means of some signaling protocol like LDP. Subsequent LSRs in the path of packet then make their forwarding decisions based on the label of the packet and corresponding action for the label stipulated by either signaling or manual configuration in the ILM. The last node in the MPLS domain known as, Egress LER pops out the label so that the next router receives the packet in its earlier form. Fig 2.4 describes the MPLS domain.

[Semeria] presents a good discussion about evolution of MPLS and terminologies in MPLS. [RFC-MPLSARCH] proposes the MPLS architecture and contains the information and explanation for various aspects and design decisions in MPLS implementation. Interested reader is referred to [Mpls_charter] for latest information in standards development undertaken by the IETF.

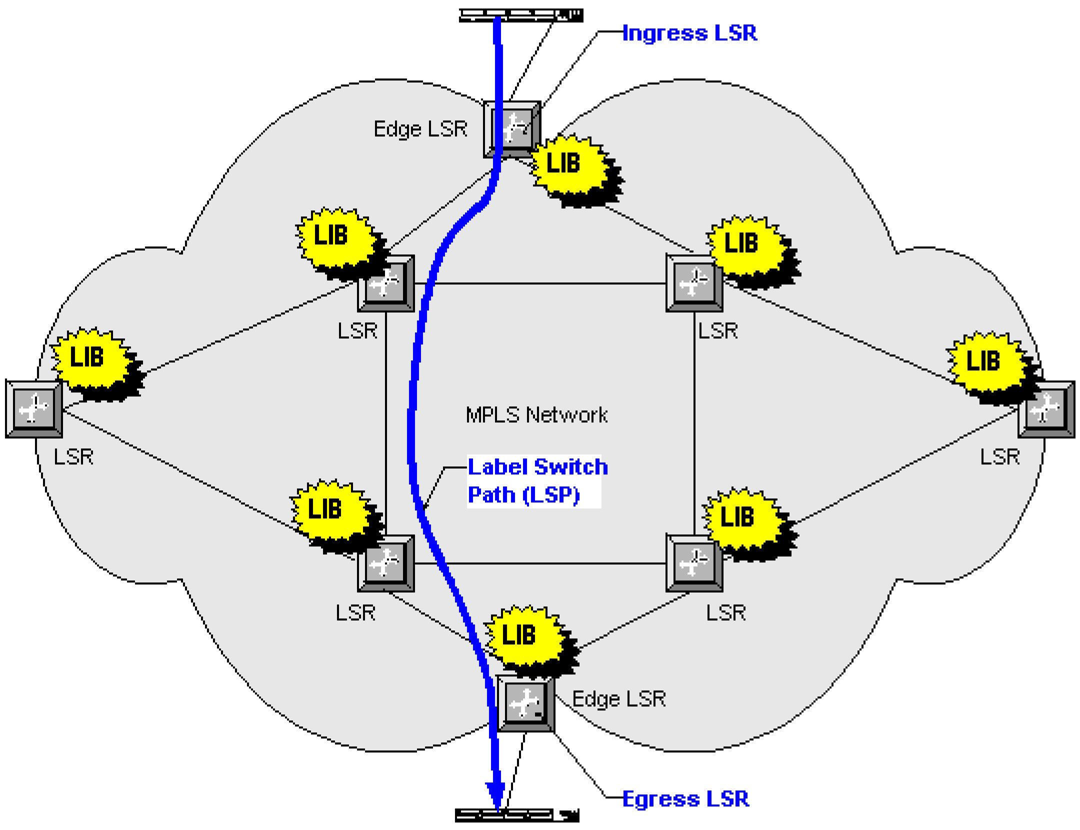

Fig 2.4 MPLS Domain

3 MPLS and DiffServ

3.1 Motivation

MPLS and DiffServ share some common points. Both models do aggregation of traffic at the edge and processing of the traffic only at the core. Both models are scalable. MPLS offers many advantages to service providers. However, it is incapable of providing differentiated service levels in a single flow. Hence MPLS and DiffServ seem to be a perfect match and if they can be combined in such away to utilize each technology's strong points and counter the other's weaknesses, it can lead to a symbiotic association that can make the goal of end to end QoS feasible.

Note that either DiffServ or MPLS can be used to offer some services with differing QoS. Any routing scheme can be used in a DiffServ network and some level of service differentiation will be perceived by the users due to the way packets with different codepoints are treated at DiffServ nodes. MPLS networks can be configured to offer different QoSs to different paths through the network. If the two technologies are combined, then *standardized* DiffServ service offerings can be made and MPLS can facilitate great control over the way these services are implemented. Such control means that it is more likely the operator will be able to offer services within well-defined QoS parameters.

DiffServ aids MPLS in following ways.

1. MPLS only aids layer3 QoS and does not introduce a new QoS architecture. So DiffServ can help MPLS by providing the QoS architecture to MPLS networks.

2. MPLS being a path-oriented mechanism, when used in backbone networks can give rise to scalability problems especially with RSVP-TE.MPLS + DiffServ combination gives rise to networks where there is no per-flow state to be maintained in core routers. Only per-LSP state is to be maintained. If DiffServ is not used, and IntServ is used with MPLS (as is proposed in a new draft), There will be the overhead of maintaining both per-flow state and per-LSP state. With LSP aggregation, one can reduce the number of LSPs.

3. DiffServ can provide differentiation of service within each flow.

4. The aggregated flow scheme of DiffServ not only reduces the flow state overhead, but also enhances the performance of MPLS by reducing the number of labels to be managed.

MPLS aids DiffServ in many ways.

1. When link failures happen, MPLS-based fast rerouting aids DiffServ in guaranteeing much stricter QoS. Of course, link failures are not day-to-day occurrence in backbone networks.

2. Traffic Engineering is provided by MPLS to DiffServ. You can visualize different paths for different PHB groups, resource-preemption, different protection levels for different PHBs etc.

3. When you want to use DiffServ in heterogeneous link-layer environments, for example, in ATM networks, MPLS is pretty much the best option to go for. Of course this may not be a great need, given the excellent QoS guarantees supported by ATM.

To date most of the work in DiffServ and MPLS has focused on defining technologies. IETF has standardized an RFC for MPLS support of Differentiated Services [RFC-MPLSDS]. Raghavan et.al. [Law et.al.] have simulated MPLS +DiffServ and so also Murphy [Murphy et.al]. But there is not much actual implementation around. This research attempts to explore the issues and functional capabilities possible in the implementation of Differentiated Services Support for MPLS in a soft router implementation.

3.2 Concepts

DiffServ and MPLS do not operate at the same layer in protocol stack. Consequently these two technologies cannot work together without some effort. [RFC-MPLSARCH] defines the MPLS header to be a 32-bit quantity that contains,

Label	EXP	S	TTL

Fig 3.1 MPLS header

Label: 20 bits. This value contains the MPLS label

EXP: experimental use 3 bits

S: Stacking bit, used to stack multiple labels

TTL: time to live 8 bits places a limit on number of hops a MPLS packet can traverse.

Recall that the LSR does not examine IP header where the DSCP information resides. So some means must be available to correlate the DiffServ PHB with the packet. The three-bit EXP field could be used for the purpose but the DSCP field is 6 bits in length (2 bits are currently unused- CU). Even if we discount the one bit to indicate whether traffic is in profile or out of profile, we have 5 bits that need to be mapped to three bits in the MPLS shim header. There are two solutions defined in [RFC-MPLSDS] to remedy these problems I) EXP inferred LSP (E-LSP) and II) Label only inferred LSP L-LSP

3.2.1 Types of LSPs

3.2.1.1 E –LSPs

A single LSP can be used to support up to eight BAs of a given FEC, by using the three-bit EXP field. These LSPs are called EXP inferred LSPs or E-LSPs since the PSC of the packet depends solely on the three bit EXP value. Thus the label is used to make the forwarding decisions and the EXP value used to determine the treatment the packet receives. The limitation to this approach is it can support a maximum of eight PHBs per LSP.

3.2.1.2 L-LSPs

To counter this a separate LSP can be established for the <FEC, OA> pair. With such LSPs the PSC is explicitly signaled at label establishment so that after label establishment, the LSR can infer exclusively from the label value the PSC to be applied to a labeled packet and the drop precedence of the same is determined by the EXP value. This approach is called Label only inferred LSP or L-LSP. An arbitrarily large number of PHBs can be supported.

3.2.2 Label forwarding Model in DiffServ LSR

In an L-LSP different Ordered Aggregates of a given FEC may be transported over different LSPs, the Label swapping decision for the DiffServ LSR clearly depends on the Behavior aggregate of the packet concerned. Also since IP DSCP field is not always available to the LSR, an MPLS DiffServ router behaves differently than a non-MPLS

DiffServ router. The DiffServ LSR label switching behavior as defined by [RFC-MPLSDS] has four stages

3.2.2.1 Incoming PHB determination

For E-LSP the EXP-PHB mapping can either be pre-configure or explicitly signaled during E-LSP establishment. This mapping is then used by the LSR to determine the PHB treatment to be given to the incoming packet.

For L-LSP the PHB to be applied is the function of PSC and is set up during LSP establishment. Therefore the PSC is already known to the LSR based on the label and it then determines the drop precedence (and hence the PHB) by looking up the value of EXP field in the EXP-PHB mapping.

3.2.2.2 Out going PHB determination

A DiffServ LSR may perform marking, policing, and shaping on the incoming traffic streams, potentially changing the outgoing PHBs associated with non-conforming packets in the incoming traffic streams. Thus the incoming and out going PHBs might be different.

3.2.2.3 Label forwarding

Each LSR must know the DiffServ context for a label, which is stored in the NHLFE for each outgoing label. The DiffServ context consists of

- LSP type
- Supported PHBs

- EXP-PHB mapping for incoming label
- PHB-EXP mapping for outgoing label.

This information is populated by the ILM and FTN at the time of label set up

3.2.2.4 Encapsulation of DS information

For E-LSP the PHB-EXP mapping can either be pre-configured or explicitly signaled during establishment of the E-LSP. The LSR determines the EXP value to be written to the outgoing packet label from the PHB-EXP mapping.

For L-LSP the PSC information is carried by the label and is set up during establishment. The EXP value to be written is determined by looking up the PHB-EXP mapping.

Obviously, to enforce the service differentiation, the LSR must apply the forwarding treatment pertinent to the supported PHB specification.

3.2.3 Implementation Models

[DIFF-TUNNEL] describes how DiffServ behaves with IP tunnels of various forms. MPLS is not a type of IP tunnel as the encapsulating header is a MPLS label and not an IP header. However MPLS is still a form of a tunnel and has certain similarities with respect to IP tunnels like

- Intermediate nodes (those along the LSP) operate only on the basis of outer DiffServ information (the one encoded in the label).
- LSPs are also unidirectional.

IP Tunnels have no penultimate hop popping described in [RFC-MPLSARCH]. However for those implementations, where this information is not meaningful, it is a non-issue.
The two conceptual models defined in [DIFF-TUNNEL] are applicable for MPLS DiffServ with some changes; namely pipe model and the uniform model.

In a Pipe model the MPLS tunnels hide the intermediate MPLS nodes from the DiffServ perspective. There are two types of DiffServ information to be conveyed by the tunneled packets, one that is useful to intermediate nodes in the LSP (LSP DiffServ) and the other, which is meaningful beyond the LSP (tunneled DiffServ).

With a pipe model LSP DiffServ information needs to be conveyed to LSP egress so that it can apply forwarding treatment based on the same and also tunneled DiffServ information is required to be conveyed, since it (tunneled DiffServ information) is to be passed beyond the egress. [RFC-MPLSDS] requires the support of pipe model as *"For support of the Pipe Model over a given LSP without PHP,* an LSR performs the Incoming PHB Determination and the DiffServ information Encoding in the following manner

- when receiving an unlabelled packet, the LSR performs Incoming PHB Determination considering the received IP Header.

- when receiving a labeled packet, the LSR performs Incoming PHB Determination considering the outer label entry in the received label stack. In particular, when a pop operation is to be performed for the considered LSP, the LSR performs Incoming PHB

Determination BEFORE the pop.

- when performing a push operation for the considered LSP, the LSR:

> *o encodes DiffServ Information corresponding to the OUTGOING PHB in the transmitted label entry corresponding to the pushed label.*
>
> *o encodes DiffServ Information corresponding to the INCOMING PHB in the encapsulated header (swapped label entry or IP header).*

- when performing a swap-only operation for the considered LSP, the LSR encodes DiffServ Information in the transmitted label entry that contains the swapped label - when performing a pop operation for the considered LSP, the LSR does not perform Encoding of DiffServ Information into the header exposed by the pop operation (i.e. the LSR leaves the exposed header "as is"). "

[RFC-MPLSDS] also defines the uniform model but since it is not supported in our implementation, it is beyond the scope of the current discussion. [RFC-MPLSDS] mandates the support of the pipe model but the uniform model is optional. For more information the reader is referred to [RFC-MPLSDS].

4 Linux

Linux is a widely popular Unix like operating system Its source code is available freely on the internet. The presence of a large number of developer base gives ‘never-before’ support for Linux. Originally developed by Linus Torvalds and contributed by developers worldwide Linux has become the favorite operating system among university researchers. Linux already supports the basic routing functionality, moreover with, kernels 2.4.1 onwards, implementing a variety of queuing methods and kernel 2.4.5 onwards, implementing support for MPLS, it forms a good base for soft router implementation. The following chapter aims to discuss various Linux features used in the implementation. Section 3.1 describes the networking support in Linux and section 3.2 describes the QoS support inherent in Linux section 3.3 aims to throw light on the mpls-linux implementation.

4.1 Networking support for Linux

The best way to understand the networking support in Linux is to trace journey of a packet inside Linux [PKT_JOURNEY]. If the network card receives an Ethernet frame that matches the local MAC address or is a link layer broadcast, it issues an interrupt. The network driver for this particular card handles the interrupt, fetches the packet data via DMA / PIO into RAM. It then allocates a local structure sk_buff and calls a function of the protocol independent device support routines: **net/core/dev.c:netif_rx(skb).**

If the driver didn't already timestamp the skb, it is time-stamped now. Afterwards the skb gets enqueued in the appropriate queue for the processor handling this packet. If the queue backlog is full the packet is dropped at this place. After enqueuing the skb, the receive soft interrupt is marked for execution via **include/linux/interrupt.h: __cpu_raise_softirq().**

The interrupt handler exits and all interrupts are re-enabled. Further handling of our packet is done in the network receive softirq (NET_RX_SOFTIRQ) which is called from **kernel/softirq.c:do_softirq().**

NET_RX_SOFTIRQ calls **net/core/dev.c:net_rx_action()**. Here the skb is dequeued from this cpu's receive queue and afterwards handed to the appropriate packet handler. In case of IPv4 this is the IPv4 packet handler. (**linux/net/ip_input.c ip_rcv()**).

ip_rcv performs sanity checks on the packet (checksum , header length etc) and depending upon whether the packet is for the host machine or meant to be forwarded to some other machine it is sent to either **net/ipv4/ip_local_deliver()** or **net/ipv4/ip_forward()**. If the packet is in error ip_err is called. ip_forward calls **ip_send()** which calls **ip_queue_xmit()** which prepends the hardware layer header to the skb and transmits the packet out through the hardware output function pointer typically **dev_queue_xmit()** in linux/net/core/dev.c. [LINUX-IPNET]

4.2 QoS Support in Linux

QoS support in Linux is implemented through the Traffic Control code (TC). The TC code resides in the kernel and the different blocks can be compiled in as modules or straight into the kernel [QoS-LINUX].

Basic principle of TC is to condition the traffic after next hop has been decided i.e. the forwarding code has decided which interface the packet will go out on. This means only the out going packets are subjected to TC. Linux traffic control can be used to build and array of complex queuing mechanisms and classes and filters that control the packets sent to the output interface.

The TC consists of three building blocks

4.2.1 queuing discipline

The **queuing discipline** can be thought of as the traffic/data-packet manager for a device. It *encapsulates* within it the two other major TC components and controls how data flows through them. Only one such managing component can be attached to a device. Queuing disciplines form a basic building block for QoS support of Linux. When a Linux kernel configured for QoS support is booted up, the function net_dev_init (in net/core/dev.c) calls the function pktsched_init (in net/sched/sch_api.c) to initialize the traffic control unit in the Linux kernel. In pktsched_init(), the queuing disciplines that have been compiled into the kernel are all registered and initialized. Other queuing disciplines (like the one we are using –described later) can be loaded as modules in the kernel.

When an outbound packet on the device is queued for transmission by calling dev_queue_xmit, the enqueue function of the device's queuing discipline (if present) is called. The queuing discipline is pointed to by field qdisc in the device structure (include/linux/net/device.h). Soon after the packet is enqueued dev_queue_xmit calls qdisc_run which calls qdisc_restart in /net/sched/sch_generic.c. qdisc_restart polls continuously to check if a packet is ready to be sent. It first tries to obtain the packet from the dequeue function of the qdisc and if it succeeds it calls hard_start_xmit function for the device driver to actually send the packet. If the packet could not be sent for some reason it calls the requeue function for the qdisc.

4.2.2 Classes

The **class (es)** are managed by the device queuing discipline. A class consists of rules for messaging data owned by that class. For example, all data packets in a class could be subjected to a rate limit of 1Mbps and allowed to overshoot up to 3Mbps between the hours of midnight and 6AM. Several queuing disciplines can be *attached* to classes, including FIFO (First-In-First-Out), RED (Random Early Detection), SFQ (Stochastic Fair Queuing) and Token Bucket. If no queuing discipline is attached to a device, basic FIFO is used. In the example shown later, no specific class queuing disciplines are attached, thus defaulting to simple FIFO. CBQ, CSZ and Priority can also be used for classes and allow for sub-classing within a class. This shows how easily very complex scenarios using TC can be built. The queuing disciplines managing classes are referred to as class queuing disciplines. Generally, the class queuing discipline manages the data and queues for that class and can decide to delay, drop or reclassify the packets it manages.

4.2.3 classifiers

Classifiers or filters describe packets and map them into classes managed by the queuing disciplines. These normally provide simple description languages to specify how to select packets and map them to classes. Currently, several filters (depending on your needs) are available in conjunction with TC, including the route-based classifier, the RSVP classifier (one for IPV4 and another for IPV6) and the u32 classifier. All of the firewalling filters can be used subject to their internal filtering tags. For example, **ipchains** could be used to classify packets.

Queuing disciplines and classes are tied to one another. The presence of classes and their semantics are fundamental properties of the queuing disciplines. In contrast, filters can be arbitrarily combined with queuing disciplines and classes, as long as the queuing disciplines have classes. Not all queuing disciplines are associated with classes. [QoS-API].

4.3 MPLS support in Linux

There is ongoing work in the research community in the field of MPLS. "MPLS for Linux" is a project under source forge [source-forge] to implement MPLS stack for Linux kernel and also the portable versions of signaling protocols. For more information about the project refer to [source-forge]. At the time of writing the project had completed MPLS kernel stack covered under GNU public license [GPL] and LDP implementation.

When an MPLS packet is received on the interface, net_rx_action() receives the packet and passes it to **mpls_rcv()** ,which gathers the required information about the packet concerning its label space and the MPLS label itself and calls **mpls_input()**.

This function obtains the corresponding entry for the label in the Incoming Label Map (ILM) and accordingly either does a PUSH or a POP on the label stack. If the entry requires the label to be deleted then the function calls the appropriate underlying protocol handler function for the packet. If the entry requires to forward the packet then **mpls_output2()** is called which gets the output label for the corresponding incoming label and transmits the packet using either **hh_output()** or **dst->neighbor->output()** which is a pointer to output function for this route, typically **dev_queue_xmit()**.

The ILM is set up by either a signaling protocol like LDP, CR –LDP or through manual configuration using a utility mplsadm. This utility creates an FEC for the IP address specified and then binds the label value and the appropriate action (push pop or delete) to the FEC. Appendix A explains the utility and its usage in greater detail.

When an unlabelled packet is received, it traverses through the ip stack like any other packet. **net_rx_action()** hands the packet to **ip_rcv()**.This function performs the initial checks on the packet and then calls **ip_route_input()** to identify the next hop for the packet. In this regard **rt_set_nexthop()** is called. If the packet is a member if a FEC for which a label is assigned (bound), rt_set_nexthop hands the packet over to **mpls_output()** function passing the FEC information in dst_proto_data field of the packet. This function extracts the corresponding entry for the FEC from the FTN and calls **mpls_output2()** for transmission to the next hop.

The functions mpls_opcode_peek ,mpls_opcode_push and mpls_opcode_pop perform the required action on the label namely looking at the label ,pushing a new label or popping out a existing label.

5 Implementation

5.1 Soft router implementation of DiffServ

5.1.1 Introduction

We used the soft router DiffServ implementation by [Narasimhan] as a base. It uses the basic forwarding functionality of the Linux router. The code resides at the output interface. A packet received at an input interface is forwarded to a particular output interface based on the routing table and the decisions made by the Linux forwarding code. When it reaches the output interface the DiffServ code by [Narasimhan] treats it with various modules like classification, buffer manager based on a user-friendly script.

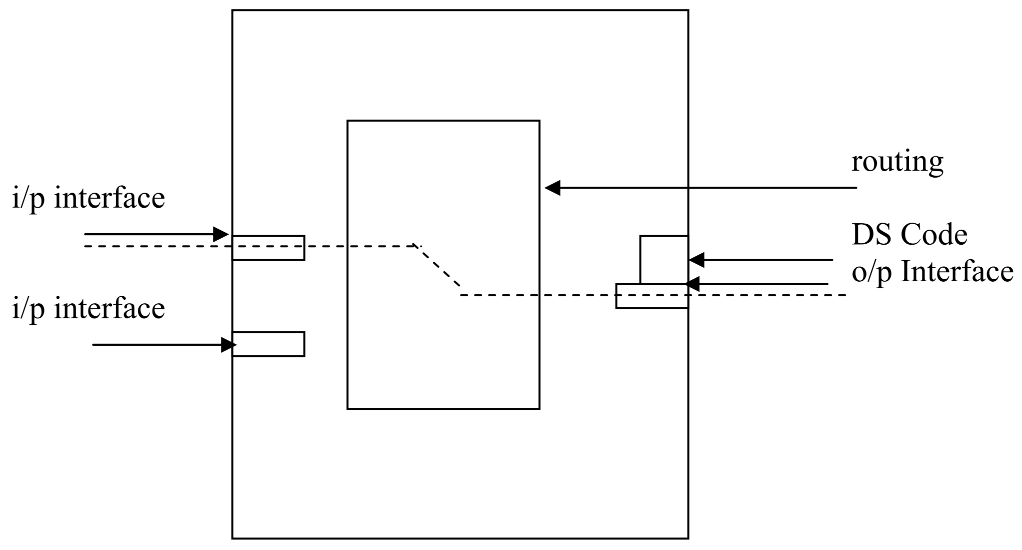

Fig 5.1 DS router

Fig 5.1 shows this high level path of packet through DS router.

The code is loaded as a module in the kernel where it registers itself as the Queuing discipline(3.2.3) and thus gets attached to the output interface specified. The DiffServ module thus runs in the context of the kernel space. A simple configuration script provides the parameters of the PHB the packet receives, such as classifiers, traffic conditioners, buffer managers, and link scheduler. This data however is present in the user space but is required by the initialization of the DiffServ module in the kernel space. Another program init_qd reads the file from the user space and communicates to the DiffServ module through a "dummy driver".

5.1.2 Usage

The usage of the program as recommended in [Narasimhan] is

"The user needs to first create the configuration file "qdisc.rc". In this file, he specifies how he wants the DiffServ modules to be initialized. It includes specifying the type of classifier, traffic conditioner, buffer manager, link scheduler etc. He then calls load <device> to load the DiffServ modules at the output interface device. The DiffServ router is now ready for traffic. The DiffServ modules can be unloaded by unload device. To re-initialize the DiffServ modules, the user needs to unload it first, edit the qdisc.rc file and then reload it using load. While running tests, the user can view various statistics that are captured, by executing the dump_config program. The dump_config program not only shows the state of the system, but it also displays the statistics that are captured like average buffer size etc."

5.1.3 Extensions

Since the DiffServ module is so closely associated with the kernel , it is only to be expected that change in kernel versions could jeopardize its functioning. We made some changes in the code so that it is compatible with Linux kernel 2.4.* (earlier version was Linux 2.2.*). The details of the changes and explanation for porting is given in Appendix B . We note that even this version is kernel 2.4 specific and might require some porting effort, should there be any major changes in the kernel.

The DiffServ module used to send all unclassified packets and ARP packets through control queue. The following diagram in [Narasimhan] describes the behavior.

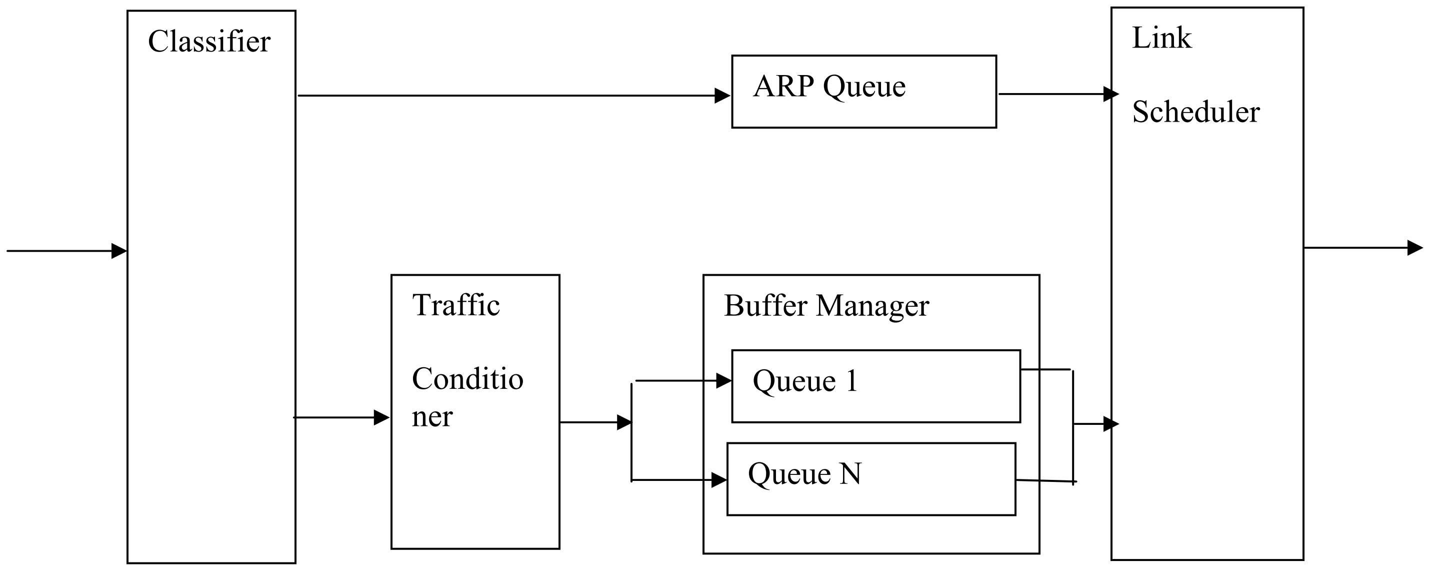

Fig 5.2 Architecture Model of [Narasimhan]

5.1.3.1 Default classifier

ARP packets do not go through various DiffServ modules but are sent through a separate queue called the ARP Queue. All other packets, which cannot be classified, are also sent through ARP queue. This behavior is not acceptable if MPLS support is required, as there has to be a provision for Best Effort traffic. We added a new classifier Default Classifier which if defined would accept all the packets for which no other classifier was found. The ARP packets were classified and sent through the high priority ARP queue.

5.1.3.2 Port Classifier

We also implemented classification by port. The port classifier assumes the transport layer protocol is either TCP or UDP, which is true in most port-based applications. To handle any exceptions to the rule, we already have an existing protocol based classifier.

5.1.3.3 DSCP Marking

DiffServ Code Point or DSCP encodes the PHB requirements of the packet. Hence it is essential that the DiffServ module be able to mark outgoing packets with the specified DSCP. We implemented the DSCP marking at flow level i.e. each outgoing flow can mark packets. The user can ignore, retain or mark DSCP as per requirements of the network.

5.1.3.4 PHB Map

For DiffServ LSR module to provide the functionality to treat an incoming packet based on its PHB, it must be aware of what PHBs are provided for what flows. i.e. it needs a

Map between PHBs and the flows so it can pass a packet belonging to a particular PSC through a particular flow which provides the PHB.

5.1.3.5 Multiple Interfaces

The earlier DS implementation was generic and would attach itself to the interface specified when the module is loaded in the kernel. The problem with this approach is it lends itself unsuitably to be used on multiple interfaces of a same machine. Not only is this non-conformant with DiffServ standards [DS_RFC] but also hinders traffic engineering. We took the simplistic approach of making the code interface specific instead of other better options in terms of design, because of the inherent inflexibility in the earlier architecture.

5.2 Approach

There are two types of events that change the state of a DiffServ LSR

1) Receipt of an un-Labeled packet and
2) Receipt of a labeled packet

5.2.1 Unlabeled packet

This packet is received when the pervious hop of the packet does not belong to MPLS domain or doest not have an LSP set for this path. Whenever such a packet arrives at a DiffServ LSR, it is treated based on the DiffServ configuration for it i.e. it is classified based on the appropriate classifier and given the appropriate PHB. If a LSP is not set for the packet, then it is similar to only DiffServ being configured. However if a LSP is set a MPLS label is pre-pended to the packet. The EXP value for the label alone, currently, maps the DiffServ Code Point information, as only E-LSPs are being supported. To support L-LSPs, the label value would have to be considered while mapping. Also the incoming label map (ILM) also should contain a field that specifies whether the entry specifies an E –LSP or an L-LSP.

5.2.2 Labeled packet

This packet is received when the previous hop of the packet belongs to MPLS domain. The DiffServ LSR checks its ILM for the incoming label entry. Based on the operation specified in the label, action is taken for example. If the label entry specifies SET for a particular device the packet is sent to that device. The outgoing label value stored in the auxiliary data field in **dst** entry in the **skbuff**, which acts as a place holder. A control bit is set in the dst entry to help identify the packets in the IP layer. A bit is also set for delete option of a label as a special case as the delete option specifies the delivery of the packet to IP layer. When the packet does arrive at the outgoing interface, on which DiffServ module is loaded, the DiffServ LSR gives the PHB treatment to the packet recommended by the label value. The outgoing packet label can have either the PHB mapping value of the incoming label (i.e. the intended PHB) or can have the actual PHB value that is being imparted. The current implementation encodes the intended PHB mapping value.

The following figure shows schematic, simplified view of the DiffServ LSR. When labeled packet is received by **mpls_ds_rcv()**, it passes it to **mpls_ds_input()** which looks up the ILM for the label and sets the proper device and sends it to DiffServ module for PBH treatment.

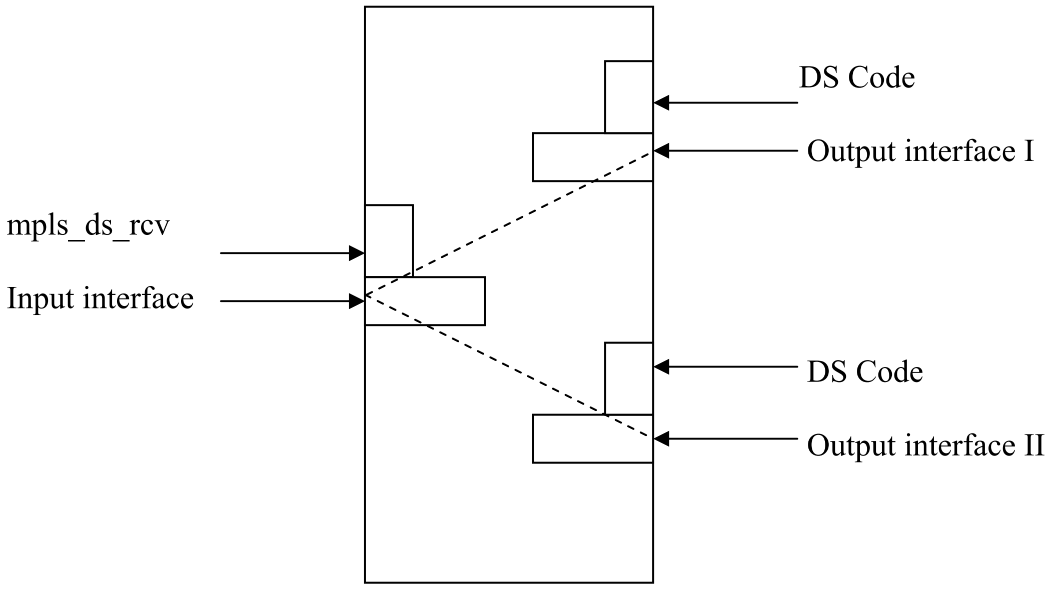

Fig 5.4 Schematic view of DiffServ LSR

5.3 Issues

5.3.1 PHB not present

If an incoming packet requests particular PHB which is not supported by the router it is tempting to classify the packet based on the IP fields or impart the "best available" PHB next to the requested PHB. But as [RFC-MPLSARCH] states that any intermediate MPLS node (core) should refer to only the label at the top of stack to make forwarding decisions. Also if best available PHB is imparted, the user would be expecting a different class of service than being provided with neither parties being informed of the same, thus making the problem difficult to debug. This solution would also be difficult to implement. One other alternative is to impart the default PHB treatment if default PHB is specified, otherwise drop the packet. The simplest option is to drop the packets. This makes it easier to track the problem however can be disconcerting in requiring every PHB supported to be defined. In our implementation we decided to drop the packets whose PHB requirements cannot be met by the router.

5.3.2 Performance requirements not defined

[RFC-MPLSDS] defines the standard for MPLS support. However it does not specify performance requirements or conformance requirements for implementation

5.3.3 Model Implemented

[RFC-MPLSDS] solution provides for three types of models Pipe Model, short Pipe Model and the Uniform Model. Pipe Model is mandatory and the other two models are optional [RFC-MPLSDS]. Our implementation supports the Pipe model currently but can be changed easily to support the uniform model.

5.3.4 Simplification of PHB value handling

For implementation purposes we have simplified the handling of PHB specification and mapping values. The PHB specifications for a particular flow would be dependant on the norms and policies in the domain being used, while the PHB mapping would be based on the requirements of the service provider.

5.3.5 Processing issue.

The current implementation adds an extra overhead in processing a given packet.

5.4 Software components

qdisc_mod_ifindex.o : This object file contains main DiffServ modules code .This is specific to the interface (e.g. for eth0 the file is qdisc_mod_eth0.0).It will be loaded as kernel module.

init_qd [<interface name>]: This executable is common for all the interfaces. It reads the data from the script file in user space and writes it through the dummy driver in kernel space. It takes optional parameter as the interface name the qdisc module is attached to. If interface name is passed it writes the data through the device driver specific to the interface passed.

dum_drv_ifindex.o: This is the dummy driver for that particular interface (e.g. for interface eth the file is dum_drv_eth1.o) . This is also loaded as a kernel module.

dump_config[<interface name>]: This executable denotes the current state of the system. Can be optionally passed the interface name to print the statistics of qdisc loaded on that particular interface. dump_config assumes that particular module is loaded.

clear_stats [<interface name>]: This executable clears all counters and resets the statistical counters. Can be optionally passed the interface name to clear statistics for that particular interface. It assumes that particular module is already loaded.

Load <interface name>: This shell script loads the corresponding module in the kernel and adds tc. It also loads the dummy driver module to pass the information from user space to kernel space.

5.5 Usage

5.5.1 mplsadm utility:

The user first has to set up an LSP according to his/her traffic requirements. 'mplsadm' is a utility that helps manually set up the mpls tables. These values can also be set automatically by using either LDP or some other signaling protocol. The runtime values are displayed by the /proc utility in files **/proc/net/mpls_in** **/proc/net/mpls_out** and **/proc/net/mpls_fec** for the ILM, FTN and FEC tables respectively. Appendix A discusses the usage for the mplsadm utility and Appendix C is a small shell script written for all label settings during testing.

5.5.2 Load script file

The user then has to set up DiffServ control parameters like the traffic conditioning to be applied for a particular flow, the classification required the buffer manager and the scheduling technique to be used. All the information is written in a file '*qdisc.rc'* The program parses this file and sets up its values. A detailed description of the script file and its syntax is explained in [Narasimhan]. Appendix B lists a few basic things about the script file and few example script files.

The user would need to specify the PHB of the flow in the qdisc.rc and optionally indicate the DSCP marking for the outgoing packet on that particular flow.

5.6 Advantages and Limitations

5.6.1 Advantages

Modular: Since the code is modular, the user can specify lot of parameters for each module and thus control the behavior of network to a greater extent.

Optional Set up: The user can set up either MPLS or DiffServ or DiffServ over MPLS depending on the requirements, thus the implementation is backward compatible and can work in an existing test bed.

Traffic engineering ability gives lot of control over network performance and helps avoid congestion through certain paths enabling it to deliver a better QoS.

5.6.2 Limitations

The implementation puts additional processing overhead per packet. Even the MPLS labels are handed over to the IP layer, hence performance of an individual machine is equivalent or worse than only DiffServ but not better.

However the performance of MPLS-DS domain, because of the additional flexibility, might be better than only DiffServ domain.

6 Tests and Results

We tested the functionality of each module. A smartbits SMB 200 was used both as source and sink. The machines used as routers were AMD 686 128 MB 900MHz Gateway machines were equipped with 3COM 3c905 10/100 Mbps PCI Network Interface Cards, running Linux.

The aim of testing was to verify and check functionality of each module in a DiffServ LSR and also to understand the possible interaction between LSPs and PHBs. We also demonstrate how the Traffic Engineering (TE) ability can be combined effectively with Differentiated Services to implement an End-End QoS framework.

6.1 Test I

Aim : Test I was carried out to test the basic functionality and co-existence of the DiffServ and MPLS components.

Setup : Smartbits was used as traffic generator generating two types of streams. Both these streams were applied different DiffServ PHB and different MPLS Labels at one machine and were stripped off at the other machine to deliver back to Smartbits port.

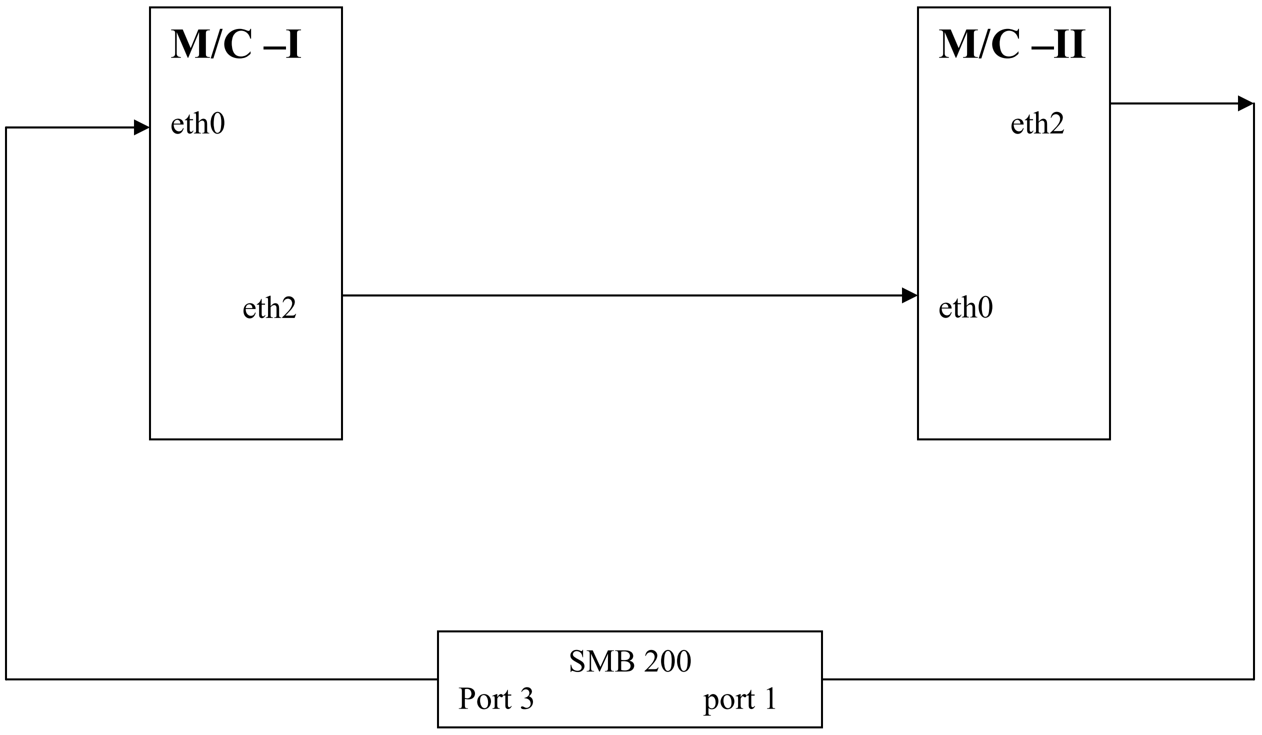

Fig 6.1 Schematic view of Test I setup

Configuration

Smartbits

Packet size : 128 bytes   Test Duration: 60 secs

Flow ID	Source IP	Destination IP	Packets Sent	Packets received	Loss %
1	10.5.4.30	10.3.4.50	253378	120504	52.44
2	10.5.4.40	10.3.4.50	253378	253378	0

DiffServ

Flow ID	Classifier			Traffic Conditioner	Buffer Manager	Link Scheduler
	Flag	Value	PHB			
1	SRC_ ADDR	10.5.4.30	EF	SRTCM (cbs = 2Mbps)	Normal	Static priority (priority=10)
2	DEF	-	DEF	DUMMY	Normal	Static priority (priority= 5)

MPLS

Flow ID	FEC	Label	LSP Path
1	10.3.4.50	16	M/c I -> M/c II

Conclusion

The basic MPLS DiffServ functionality works and co–exists. The software can push and pop out labels and the PHB value is propagated to the next hop where appropriate action is taken

6.2 Test II

Aim : Test II was carried out to test the basic functionality and co-existence of the DiffServ and MPLS components.

Setup : Smartbits was used as traffic generator generating three streams. These streams were applied different DiffServ PHB. Different MPLS Labels applied at one machine were swapped at second machine and finally stripped off at third machine to deliver back to Smartbits port.

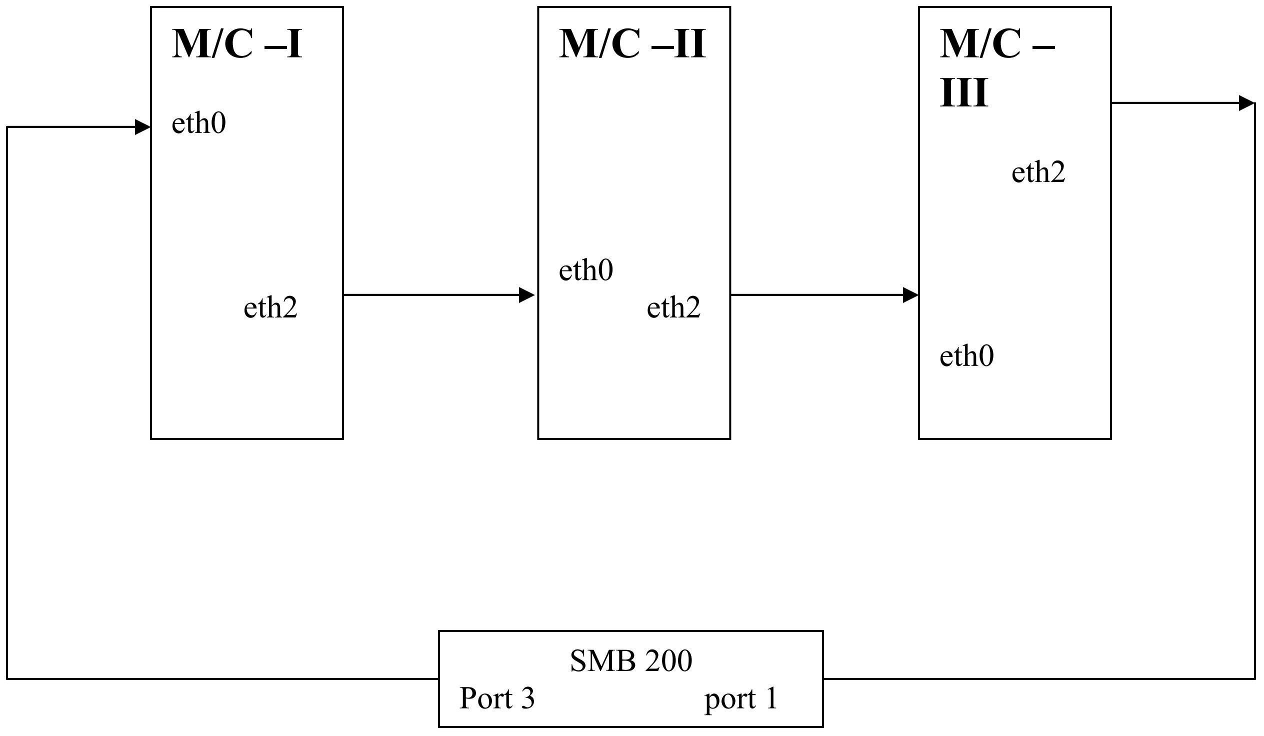

Fig 6.2 Schematic view of Test II setup

Configuration

Smartbits

Packet size : 128 bytes Test Duration: 60 secs

Flow ID	Source IP	Destination IP	Packets Sent	Packets received	Packets lost (%)
1	10.5.4.30	10.7.6.90	168918	30954	81.675
2	10.5.4.40	10.7.6.80	168918	118941	29.586
3	10.5.4.50	10.7.6.80	168918	167125	1.061

DiffServ

Flow ID	Classifier			Traffic Conditioner	Buffer Manager	Link Scheduler
	Flag	Value	PHB			
1	SRC_ ADDR	10.5.4.40	EF	SRTCM cir = 2.5MB/s	Normal	Static priority (100)
2	DEF	-	DEF	DUMMY	Normal	Static priority (50)

MPLS

Flow ID	FEC	Label at I	Label at II	LSP Path
1	10.7.6.80	16	21	M/c I -> M/c II -> M/c III
2	10.7.6.90	32	37	M/c I -> M/c II -> M/c III

Conclusion

The basic MPLS DiffServ functionality works and co –exists. The elementary test bed set-up demonstrates that the DiffServ LSR can pop, push, and swap the label and take appropriate action based on the label value. The PHB value is propagated and acted upon by subsequent routers.

6.3 Test III

Aim : Test III was carried out to test the ability to perform traffic engineering and to demonstrate its effectiveness.

Setup : Smartbits was used as traffic generator generating three types of streams. Initially these three streams share the traffic resulting in packet loss. Traffic engineering was then applied on M/C I and results noted

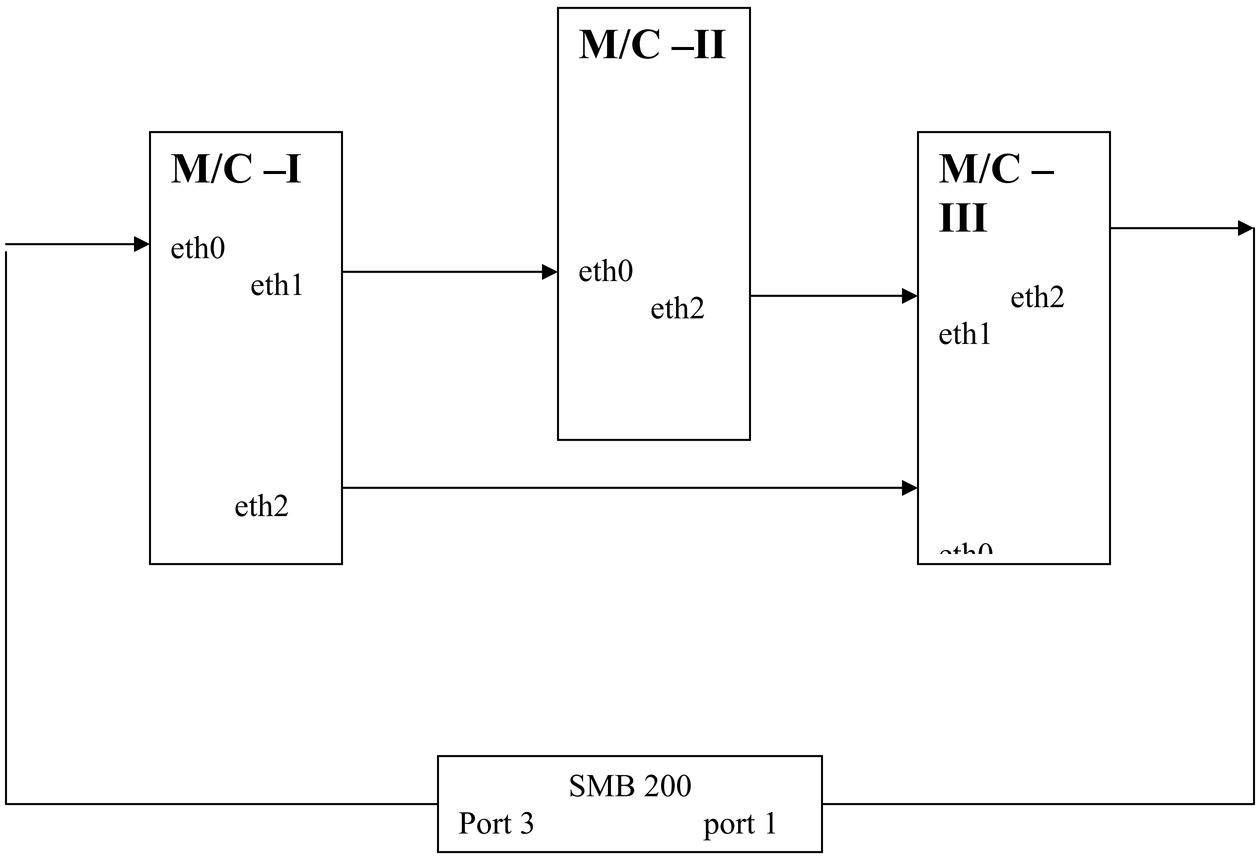

Fig 6.3 Schematic view of Test III setup

Configuration

Smartbits

Packet size : 128 bytes Test Duration: 60 secs

Flow ID	Source IP	Destination IP	Packets Sent	Packets received	Loss %
1	10.5.4.30	10.7.6.90	168918	120199	28.84
2	10.5.4.40	10.7.6.80	168918	168918	0
3	10.5.4.50	10.7.6.80	168918	168918	0

DiffServ

Flow ID	Classifier			Traffic Conditioner	Buffer Manager	Link Scheduler
	Flag	Value	PHB			
1	SRC_ ADDR		EF	SRTCM	Normal	Static priority
2	DEF	-	DEF	DUMMY	Normal	Static priority

MPLS

Flow ID	FEC	Label	LSP Path
1	10.7.6.80	16	m/c I -> m/c II-> m/c III
2	10.7.6.90	32	m/c I -> m/c II-> m/c III
3	10.7.6.90	42	m/c I -> m/c III

Conclusion

The basic MPLS DiffServ functionality works and co –exists. The above experiment demonstrates traffic engineering. Packets destined for 10.7.6.90 , that had earlier traversed the link M/C I-M/C II ->M/C III are directly routed to Machine III

6.4 Discussion of Results

As we note from the tests I and II, the basic functionality of the DiffServ LSR is satisfied. Test I demonstrates “label push” and “label delete (pop)” while test II shows in addition to the aforementioned, “label swap” and “interface set” functionality. The DiffServ PHB indicated in the label is preserved. It is interesting to observe that when Traffic Engineering is applied on M/C I in test III, and default traffic is sent through a different route, better performance is obtained. Also service differentiation can be applied to the diverted traffic as shown. MPLS supports traffic engineering and DiffServ supports service differentiation respectively, the combination of the two combines both the advantages, and hence gives the service provider lot of flexibility and control over the performance of the network.

We note that since demonstration of functionality was the aim of the experiment and not performance token DiffServ control parameters have been used. A better performance can be achieved by better tuning of the DiffServ parameters like traffic conditioner, Buffer Manager, Link Scheduler etc. interested reader is referred to [TEST-REPORT] for details.

7 Conclusion

7.1 Summary

We described the protocols DiffServ and MPLS among the recently proposed protocols to provide Quality of Service. We described how the advantages of both could be combined in MPLS support of Differentiated Services that could potentially provide considerable flexibility and ability to provide service differentiation to the service provider. We further described our extensions to the existing DiffServ implementation and our implementation of MPLS support for Differentiated Services. Next we discussed issues involved in our implementation. We created scenarios to test our implementation and we observe that while DiffServ can provide service differentiation and guaranteed bandwidth, it does not handle overloaded traffic or link failures. Similarly while MPLS can provide control and ability to fast re-route and traffic engineering, it does not offer service level differentiation. MPLS support of DiffServ combines the advantages of both protocols. We conclude that MPLS combined with DiffServ could be an important step towards providing end-to-end QoS in IP based networks.

7.2 Future Work

In our experiments we used token DiffServ parameters to demonstrate functionality. In the future, a comprehensive experimentation could be done to determine ways to optimize performance using a combination of various DiffServ parameters. Our implementation supports only E-LSPs. Support of L-LSPs could also be implemented, which would enable support of more than eight PHBs.

Multicast packets form an increasing portion of the Internet traffic with the advent of P2Pand movie broadcasts. Multicast support could also be implemented both in MPLS and DiffServ. The current implementation is able to interact only with Ethernet frames. Support of other mechanisms like ATM and Frame relay can also be implemented in future.

References

[Adisheshu et.al] Hari Adisheshu , Guru Parulkar , Raj Yavatkar, A state management protocol for Int-Serv DiffServ and label Switching, IEEE ICNP'98, Oct 1998, pp 272-281.

[Awduche et.al.] D.O. Awduche, MPLS and Traffic Engineering in IP Networks, IEEE communications magazine, vol. 37, no. 12, Dec 1999, pp 42-47.

[Bux et.al.] Werner Bux, Wolfgang E. Denzel, Ton Engbersem, Andreas Herkersdorf, and Ronald Luijten, Technologies and Building blocks for fast packet forwarding, IEEE Communication Magazine, Volume: 39 Issue: 1, Jan. 2001, pp.70 – 77.

[Cole] Bernard Cole, Defining Network API, www.embedded.com, Dec 2001 http://www.embedded.com/story/OEG20011207S0099,

[Crowley et.al.] Patrick Crowley, Marc E. Fiuczynski, Jean-Loup Baer and Brian N. Bershad, Characterizing Processor Architectures for Programmable Network Architecture Proceedings of the 2000 International Conference on Supercomputing, Santa Fe, N.M., May 2000, pp. 54-65.

[DIFF-HEADER] Nichols, K., Blake, S., Baker, F. and D. Black, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, RFC 2474, December 1998.

[DIFF-NEW] Grossman, D., New Terminology and Clarifications for DiffServ, RFC 3260, April 2002.

[DIFF-TUNNEL] Black, D., Differentiated Services and Tunnels,RFC 2983, October 2000.

[Dwekat] Dwekat Zyad Ahmed, Construction and Evaluation of a Service Level Agreement Test-Bed, Thesis (M.S.)--North Carolina State University, 2001, ix, 99 p.

[GPL] GNU Public License, www.gnu.org/copyleft/gpl.html

[Herity et.al.] Dominic Herity Network Processor Programming Embedded Systems Programming, July 2001, http://www.embedded.com/story/OEG20010730S0053

[History] Barry M. Leiner, Vinton G. Cerf, David D. Clark Robert Kahn, Leonard Kleinrock, Daniel Lynch, Jon Postel, Larry G. Roberts Stephen, A Brief History of the Internet, Wolff Internet Society, http://www.isoc.org/internet/history/brief.shtml

[Horlait et.al] Eric Horlait, Nicolas Rouhana, Differentiated Services and Integrated Services Use of MPLS, IEEE Symposium on Computers and Communications, 2000. Pg 194-199.

[Laubach] M Laubach Classical IP and ARP over ATM RFC 1577 January 1994

[Law et.al.] Law Raymond, Raghavan Srihari, DiffServ and MPLS –Concepts and Simulation, Department of Electrical and Computer Engineering Virginia Polytechnic Institute and State University

[LINUX-IPNET] Glenn Herrin, Linux IP Networking: A Guide to the Implementation and Modification of the Linux Protocol Stack TR 00-04 *May 31, 2000*

[Mpls_charter] Multiprotocol Label Switching (MPLS) Charter www.ietf.org/html.charters/mpls-charter.html

[Murphy et.al] S. Murphy, D. Botvich, T. Curran, On design of DiffServ/MPLS networks to support VPNs 16th UK Teletraffic Symposium, May 2000

[NGN_INTEL] Network Processor Division, Intel Corporation Next Generation Network Processor Technologies Enabling cost Effective Solutions for 2.5 Gbps to 40 Gbps Network Services, Intel Corporation, Oct 2001, http://www.intel.com/design/network/papers/279050.htm

[Nie et.al.] Xiaoning Nie; Gazsi, L.; Engel, F.; Fettweis, G A new Network Processor architecture for High Speed Communications Signal Processing Systems, 1999. SiPS 99. 1999 IEEE Workshop on, 1999 Page(s): 548 –557.

[Narasimhan] Narasimhan, Kesava Prasad, An Implementation of Differentiated Services in A Linux Environment. Thesis (M.S.)--North Carolina State University, 2000, viii p 116.

[Peirre et.al.] Peirre Paulin, Karim, Network Processors :A Perspective on Market Requirements Processor Architectures and Embedded S/W Tools, Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings, 2001
Page(s): 420 –427.

[PKT_JOURNEY] Harald Welte, The journey of a packet through the Linux 2.4 network stack , www.gnumonks.org/ftp/pub/doc/packet-journey-2.4.html

[QoS_LINUX] Saravanan Radhakrishnan, Linux - Advanced Networking Overview, Department of Electrical Engineering & Computer Science The University of Kansas, http://qos.ittc.ukans.edu/howto

[QoS-API] Gopi Vaddi & Pramodh Malipatna, An API for Linux QoS Support, Department of Electrical Engineering & Computer Science The University of Kansas, http://www.ittc.ukans.edu/~pramodh/courses/linux_qos/mainpage.html

[Redford] Rob Redford, Enabling Business IP services with Multi Protocol Label Switching, Multiservice Switching Business Unit, Cisco Technologies, www.cisco.com/warp/public/cc/so/neso/vvda/ipatm/mpls_wp.htm

[RFC-791] Information sciences Institute, University of Southern California, Internet Protocol DARPA internet program protocol specification, RFC 791, September 1981.

[RFC-AFPHB] Heinan et.al. Assured Forwarding PHB group, RFC 2597, June 1999.

[RFC-DSARCH] Blake S et.al. An Architecture for Differentiated Services, RFC 2475, December 1998.

[RFC-EFPHB] Jacobson et.al. An Expedited Forwarding PHB, RFC 2598,June 1999.

[RFC-INTSERV] R Braden et.al. Integrated Services in the Internet Architecture :an Overview Internet, RFC 1633, June 1994.

[RFC-MPLSARCH] E. Rosen, A Vishwanathan, R. Callon, Multiprotocol Label Switching Architecture, RFC 3031, January 2001 .

[RFC-MPLSDS] B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, J. Heinanen, Multi-Protocol Label Switching (MPLS) Support of Differentiated Services, RFC 3071 May 2002.

[Semeria] Chuck Semeria, MultiProtocol Label Switching: Enhancing Routing in the New Public Network, Juniper Networks, Inc. White Paper, http://www.juniper.net/techcenter/techpapers/200001.html

[status_report] Bill Michael, MPLS Breakthrough a Status report, Computer Telephony Network Magazine, May 2001 http://www.cconvergence.com/article/CTM20010425S0001/1

[source-forge] www.source-forge.net/projects/linux-mpls

[tech_guide] Ennovate Networks Inc , www.techguide.com Innovations in IP networking

[TEST-REPORT] Kulkarni Amit, Singhai Mrugendra, A comprehensive Test Report For Differentiated Services Implementation

[Trimintzois et.al.] Panos Trimintzios, llias Andrikopouls, George Pavlou, and Paris Flegkas, University of Surrey, U.K. David Griffin, Panson Georgatsos, Algonet S.A., Danny Goderis and Yves T'Joens, A management and control architecture for providing IP Differentiated Services in MPLS –based Networks, IEEE Communications, special

	issue in IP-Oriented Operations and Management, IEEE, May 2001, Vol. 39, No. 5, pp. 80-88.
[Williams et.al.]	Brian Williams, Quality of Service Differentiated Services and Multi Protocol Label Switching, Ericsson Australia White Paper, March 2000, http://www.securitytechnet.com/resource/rsc-center/vendor-wp/ericsson/qoswhite_paper317.pdf