Providing Quality of Service to Internet Applications using Multiprotocol Label Switching

By
Radu-Călin Dragoş

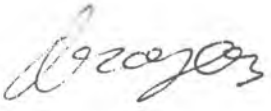
Thesis directed by: Dr. Martin Collier

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

September 2006

School of Electronic Engineering
Dublin City University

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy *is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.*

Signed: 

ID number: 50161199

Date: 25/09/2006

Acknowledgements

I am grateful to many people for help, both direct and indirect, in completing this thesis. It would never have become reality without the help and suggestions of many supportive friends, colleagues and family.

I would especially like to thank my supervisor Dr. Martin Collier for tremendous help, support and encouragement.

An important source of inspiration and knowledge have been my colleagues in the *Switching and Systems Laboratory*. Thank you all! I am particularly grateful to Dr. Karol Kowalik, Dr. Kalaiarul Dharmalingam and Yan Li for their valuable comments and suggestions. Their friendship and professional collaboration meant a great deal to me.

I have learnt a great deal and found great support in many friends and colleagues from the *Dublin City University*. The long talks in labs or on corridors, during and after basketball, volleyball, tennis and ping-pong matches, fire-alarm breaks and uncounted parties had their invaluable contribution to this end result. Most thanks should, however go to Dr. Gabriel-Miro Muntean and Dr. Cristina Muntean who offered us (me and my wife) unconditional friendship and support.

Support and encouragements from my colleagues at *Babeş-Bolyai University* is gratefully acknowledged. I specially want to thank Prof. Florian Mircea Boian and the head of the BBU's *Communication Centre*, Gabriel Ciplea, for believing in me. The help of my colleagues to improve my skills in both the field of networking and 'game theory' is also appreciated.

The most important support I have received from my beloved wife, Sanda Dragoş. She both read and commented on my texts, and encouraged the work through discussions and through positive reinforcements. Here, I want to mention the immense emotional support that I have got from my daughter Alexandra.

Last but not least, I thank to my parents and to my parents-in-law for unconditional support and encouragement to pursue my interests. To my brother Cristian and his wife Simona for having faith in me.

Exprim recunoştiinţă şi mulţumire tuturor prietenilor şi familiei, pentru susţinerea şi înţelegerea pe care mi-au acordat-o pe parcursul acestor ani de studiu. Fără ajutorul lor această teză de doctorat nu ar fi existat. De aceea doresc să le dedic lor această lucrare.

My apologies if I have inadvertently omitted anyone to whom acknowledgement is due.

Dublin, Ireland, September, 2006 *Radu-Călin Dragoş*

Abstract

The growth of the Internet and the range of applications it now supports has created a need for improved traffic engineering techniques. One protocol which shows promise in this regard is Multiprotocol Label Switching (MPLS). MPLS inherits a mix of attributes from earlier protocols such as IP and ATM, and potentially combines the simplicity of IP and the Quality of Service (QoS) capabilities of ATM. MPLS is now a mature standard widely deployed in the Internet. This thesis concerns the development of new mechanisms that can further extend the MPLS capabilities for traffic engineering.

Web service remains a key application in today's Internet. The traffic demands at popular Web-sites and the requirements of redundancy and reliability can only be met by using multiple Web servers. A new solution to Web server load balancing based on MPLS is presented in this thesis. This solution features a novel Web switching architecture featuring switching at layer two. An extended solution for providing differentiated Web services is also proposed. It has been implemented in a soft MPLS router using the Linux operating system.

The performance of soft routers is significantly affected by the packet processing time. An MPLS-based framework to increase the average packet size and consequently reduce the traffic frame-rate is described in the thesis. This has been implemented in a Linux-based soft router and its performance evaluated experimentally. As transmission rates continue to rise, such aggregation techniques will be needed if packet processing time is not to become a bottleneck. The switching technology at the core of tomorrow's Internet, featuring GMPLS and optical switching using, perhaps, optical burst switching technology, will not work efficiently with short packets.

A new class of scheduling algorithms is also described, intended for deployment in MPLS networks. Their operation is based on an analogy with the workings of the human heart. This class of algorithms achieves the optimal fairness for packet based schedulers and has low hardware complexity. It can be combined with the packet aggregation mechanism above to provide an effective interface between the edges of tomorrow's Internet and its high-speed core.

List of Publications

- **R. DRAGOŞ** AND M. COLLIER, *Multiprotocol Label Switching Meta-Frames*, in Proceedings of the International Multi-Conference on Computing in the Global Information Technology (ICCGI), Bucharest, Romania, August 2006.
- **R. DRAGOŞ** AND M. COLLIER, *Heart-like fair queuing algorithms (HLFQA)*, in Proceedings of the 7th International Symposium On Computer Networks, Istanbul, Turkey, June 2006.
- S. DRAGOŞ AND **R. DRAGOŞ**, *WinNet - a network tool*, in Proceedings of the International Conference on Computers, Communications and Control (ICCCC), Oradea, Romania, June 2006.
- S. DRAGOŞ AND **R. DRAGOŞ**, *Modern routing techniques for future QoS enabled networks*, Carpathian Journal of Mathematics, 21(1-2), pp. 51–59, 2005.
- **R. DRAGOŞ** AND M. COLLIER, *An MPLS based architecture for differentiated Web service*, in Proceedings of the International Conference on Software, Telecommunications and Computer Networks (SOFTCOM), Dubrovnik-Split-Venice-Ancona, October 08-11, 2002, pp. 132-136.
- **R. DRAGOŞ**, S. DRAGOŞ AND M. COLLIER, *Design and implementation of an MPLS based load balancing architecture for Web switching*, in Proceedings of 15th ITC Specialist Seminar, Würzburg, Germany, July 2002, pp. 24–32.
- S. DRAGOŞ, **R. DRAGOŞ**, AND M. COLLIER, *Bandwidth Management in MPLS Networks*, in First Joint IEI/IEE Symposium on Telecommunications Systems Research, Dublin, Ireland, November 2001.

CONTENTS

List of Publications

List of Figures v

List of Tables vii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Thesis contributions . . . 3
  - 1.2.1 Problem description . . . 3
  - 1.2.2 Summary of contributions . . . 3
- 1.3 Thesis outline . . . 4

2 Internet QoS overview 6
- 2.1 What is Internet Quality of Service ? . . . 6
- 2.2 Components of Internet QoS . . . 8
  - 2.2.1 QoS Metrics . . . 8
  - 2.2.2 Classes of service and service level agreements . . . 9
  - 2.2.3 Traffic scheduling . . . 11
  - 2.2.4 QoS routing (constraint-based routing and policy-based routing) . . . 14
  - 2.2.5 Signalling protocols . . . 17
- 2.3 QoS in ATM networks . . . 17
  - 2.3.1 Features of ATM . . . 18
  - 2.3.2 Traffic control in ATM networks . . . 20
  - 2.3.3 Signalling . . . 20

2.3.4 PNNI . . . 22
2.3.5 Limitations of ATM . . . 24
2.4 QoS in IP networks . . . 26
2.4.1 Best effort routing . . . 26
2.4.2 Adaptive routing in ARPANET . . . 26
2.4.3 ToS routing . . . 27
2.4.4 Integrated services . . . 28
2.4.5 RSVP . . . 29
2.4.6 Differentiated services . . . 30
2.4.7 Explicit routing and route pinning . . . 31
2.5 Internet traffic engineering . . . 32
2.5.1 Traffic engineering optimisation functions . . . 32
2.5.2 Traffic engineering control functions . . . 34
2.6 Performance of Internet routers . . . 36
2.7 Application level QoS . . . 37
2.7.1 Quality of Web service (QoWs) . . . 38
2.7.2 Dimensioning Web clusters . . . 43
2.8 MPLS . . . 45
2.9 Label switching paradigm . . . 46
2.10 MPLS and traffic engineering . . . 48
2.11 MPLS in the global QoS picture . . . 51
2.12 Concluding remarks . . . 51

3 MPLS 54
3.1 The MPLS label switching paradigm . . . 54
3.2 Other label switching technologies . . . 56
3.3 The MPLS architecture . . . 63
3.3.1 Label encapsulation . . . 63
3.3.2 MPLS label stack . . . 64
3.3.3 Forwarding tables . . . 65
3.3.4 MPLS routing and signalling . . . 66
3.3.5 Service differentiation in MPLS networks . . . 70
3.4 QoS and traffic engineering topics . . . 70
3.4.1 The MPLS traffic engineering problem . . . 71
3.4.2 Generalised MPLS (GMPLS) . . . 71
3.4.3 Protection and recovery . . . 72
3.4.4 MPLS and differentiated services . . . 73
3.4.5 Bandwidth allocation, reallocation and load balancing . . . 74

3.4.6 MPLS-based end-to-end QoS architectures 75
3.4.7 MPLS implementations and deployment 76
3.5 Concluding remarks 77

4 Exploiting the large scale deployment of MPLS **79**
4.1 Web server load balancing 80
4.1.1 Overloading a Web server 80
4.1.2 MPLS approach to Web server load balancing 84
4.1.3 Summary 94
4.2 An MPLS framework to provide differentiated Web services 95
4.2.1 Dynamic weighted load balancing 95
4.2.2 Simulation results 101
4.2.3 Summary 102
4.3 Increasing router performance using MPLS meta-frames 103
4.3.1 The average packet size in the Internet 104
4.3.2 The effects of small packet size on router performance . . . 105
4.3.3 Target MTU for meta-frame 107
4.3.4 Meta-frames overview 108
4.3.5 Frame format 110
4.3.6 Performance results 112
4.3.7 Summary 115
4.4 Heart-like fair queuing algorithms (HLFQA) 115
4.4.1 The atrium-ventricle model 117
4.4.2 Evaluating the algorithm 119
4.4.3 Weighted scheduling 122
4.4.4 Implementing the algorithm 122
4.4.5 Simulation results 123
4.4.6 Simplified HLFQA (s-HLFQA) 124
4.4.7 Complexity of s-HLFQA 127
4.4.8 Weighted s-HLFQA 127
4.4.9 A comparison of HLFQA and WFQ 128
4.4.10 Summary 129
4.5 Conclusions 130

5 Conclusions **131**
5.1 Contributions 132
5.2 Future work 133
5.3 Concluding remarks 134

Bibliography **136**

LIST OF FIGURES

2.1 ATM UNI and NNI Signalling 21
2.2 Elements of an MPLS cloud 47
2.3 The big QoS picture 52

3.1 (a) An IP Switching Network (b) The structure of an IP Switch . . . 58
3.2 IBM ARIS Switched Paths 60
3.3 The MPLS "shim header" 63
3.4 MPLS label stack entry 64
3.5 MPLS planes 67

4.1 The number of active connection for 1 server 81
4.2 The number of active connection for 2 servers 82
4.3 The number of active connections for 3 servers 83
4.4 A framework for MPLS Web switching 88
4.5 Elements of the MPLS based Web switching implementation 89
4.6 Distributed requests 92
4.7 Execution times for concurrent connections 101
4.8 Execution times for premium and basic requests 103
4.9 The variation of the payload with the packet size 106
4.10 Throughput bit/s rate on 100Mb/s links 107
4.11 The throughput performance for no assembly, 2 packet meta-frame and respectively 3 packet meta-frame. 113
4.12 Atrium-ventricle model 116
4.13 Using a shared output FIFO as aorta 119
4.14 3 FECs sharing equally 0.33 of the link 124

4.15 3 weighted FECs sharing respectively 0.16/0.33/0.5 of the link then 2 flows 0.33/0.66 then 1 flow all the bandwidth 125

LIST OF TABLES

2.1 QoS requirements for different types of applications 7
2.2 Comparison of scheduling algorithms 14

4.1 Round-robin load balancing for large files 93
4.2 Round-robin load balancing for small files 94
4.3 Generic MPLS encapsulation 110
4.4 MPLS meta-frame encapsulation 110
4.5 The average meta-frame size and the packetisation delay for MTUs of 1500, 4500 and 9000 bytes 114
4.6 Mean overhead before and after meta-frame encapsulation for MTUs of 1500, 4500 and 9000 bytes 114

LIST OF TABLES

CHAPTER 1

Introduction

Two hundred years ago the first internal combustion engine was created. It took one hundred years until mass production of the automobile began. In those early stages (19^{th} century) no traffic laws were required. Today, with over 600 million cars in the word and a production of over 60 million per year, driving would be impossible without traffic rules. Breaking the rules may result in accidents and traffic disruption. Still, there are drivers that misbehave and drive by their own rules.

But what happens with the Internet traffic when some of it misbehaves? And how misbehaviour can be defined in a network without rules? The Internet evolved so quickly that few regulations could keep up with the change. Therefore, apart from some identity information (e.g. IP addresses, domain names, etc.) which is centrally managed, the Internet traffic is apparently chaotic. Everybody sends and receives traffic as much as he can when he wants and to/from whoever he wants. Paradoxically, the Internet continues to evolve and expands despite this "Brownian motion" of bits.

It is arguable whether the traffic in the Internet should ever be regulated. Ap-

parently, increasing the bandwidth to satisfy users' need for speed may seem enough to keep the Internet alive and customers happy. However, malicious traffic such as floods or denial-of-service attacks can consume the bandwidth or bring down network components such as routers and servers. Moreover, applications such as distributed peer-to-peer file sharing will use up a high proportion of the bandwidth, impairing the functionality of other Internet applications.

In this context, many customers are willing to pay a premium for guaranteed services and the Internet service providers (ISPs) need system tools to be able to provide such guarantees, by means of traffic engineering. Traffic engineering is more than a set of rules for data traffic. It also aims to reduce congestion (which may result in traffic loss) and optimise the network, which consequently makes the business of the ISP more profitable.

The process of providing premium service to customers, or of managing a network for traffic engineering purposes, requires migration from the traditional best-effort service model, where all bits transported by the Internet were (in principle) treated alike. If the traffic engineering tools are standardised, when ISPs deploy them in the Internet they can inter-operate in order to provide a common framework for other services including end-to-end Quality of Service (QoS).

1.1 Motivation

The Internet does not only need to be traffic engineered, but must also be able to provide QoS guarantees when appropriate to its customers. Many mechanisms (as discussed in Chapter 2) have been proposed to provide these facilities but none have prevailed. This is because of the extremely heterogenous network environment in the Internet. In the 1990s Asynchronous Transfer Mode (ATM) was developing as a promising technology for the next generation of heterogeneous telecommunication networks. Its embedded QoS capabilities and high transfer rates made it a candidate as the universal carrier for the Internet. The main fac-

tors that prevented this are described in Section 2.3.5.

Multiprotocol Label Switching (MPLS) evolved from the need to integrate QoS capabilities like those of ATM into the Internet. MPLS features a simple yet effective forwarding mechanism, on top of which many existing and future QoS schemes can be deployed and inter-operate. The MPLS forwarding plane can help the convergence of local QoS mechanisms into a general Internet QoS scheme. MPLS is also attractive for traffic engineering and reduces the need for manual intervention in network administration using advanced protection and fast reroute mechanisms. MPLS is surveyed in Chapter 3.

MPLS is now a mature standard widely deployed in the Internet and used as a framework for deploying QoS. This thesis concerns the development of new mechanisms that can further extend the capabilities of MPLS for traffic engineering and QoS.

1.2 Thesis contributions

1.2.1 Problem description

This thesis concentrates on the advantages of using MPLS as a traffic engineering tool to provide QoS in the Internet. The main focus of this work is twofold:

- To check the existing QoS and traffic engineering technologies and to investigate which are feasible for deployment in the Internet;
- To analyze the role of MPLS in an overall QoS architecture;
- To develop new mechanisms that can further extend the MPLS capabilities for traffic engineering.

1.2.2 Summary of contributions

The main contributions of this thesis are listed below:

- A new solution to Web server load balancing based on MPLS. This solution features a novel Web switching architecture featuring switching at layer two. It has been implemented in a soft MPLS router using the Linux operating system.

- An MPLS based solution to provide different levels of Web service is also described. I have designed, implemented and evaluated a Web switching architecture for next-generation QoS enabled IP networks, based on a Linux implementation of MPLS.

- An MPLS-based framework to increase the average packet size and consequently reduce the traffic frame-rate is described in the thesis. This has been implemented in a Linux-based soft router and its performance evaluated experimentally. As transmission rates continue to rise, such aggregation techniques will be needed if packet processing time is not to become a bottleneck in routers. The switching technology at the core of tomorrow's Internet, featuring GMPLS and optical switching using, perhaps, optical burst switching technology, will not work efficiently with short packets.

- A new class of scheduling algorithms is also described, intended for deployment in MPLS networks. Their operation is based on an analogy with the workings of the human heart. This class of algorithms achieves the optimal fairness for packet based schedulers and has low hardware complexity. It can be combined with the packet aggregation mechanism above to provide an effective interface between the edges of tomorrow's Internet and its high-speed core.

1.3 Thesis outline

The remainder of this thesis is organised as follows:

Chapter 2 describes the main components at the various levels in the overall QoS architecture. It identifies the role of traffic engineering and the importance of MPLS and label switching architectures for Internet traffic engineering.

Chapter 3 presents the architectural details of MPLS that will make this technology a universal framework for building end-to-end Internet QoS schemes.

Chapter 4 describes new techniques for exploiting the large scale deployment of MPLS. This issue is addressed at various levels. At the application level, a framework for load balancing Web servers and providing differentiated level of service that exploit MPLS traffic engineering capabilities is presented . At the network level, a technique for increasing router performance using MPLS meta-frames is presented. At the control layer of QoS routing, a new class of weighted fair queuing algorithms is proposed to complement the existing QoS provisioning mechanisms available to MPLS networks.

Chapter 5 summarises the work, presents future research trends and concludes this thesis.

CHAPTER 2

Internet QoS overview

2.1 What is Internet Quality of Service ?

The quality of the Internet service is difficult to define because the Internet is used to provide a large variety of services for different classes of users and applications. Therefore, it is difficult to measure the level of service. For instance, carrying an electronic mail from one end of the world to another in a matter of minutes is satisfactory. But the echo effect during a voice conversation over the Internet or fuzzy images while watching live video broadcasts may be unacceptable. Mapping these application-level requirements into a set of network constraints is, in general, difficult.

However, for some applications there are specific requirements that must be satisfied in order to make them run over the Internet. Most often, these requirements are bandwidth, delay, jitter and reliability (e.g. packet loss) [147, 159]. Therefore, some sort of metrics are need to specify the requirements and to be able to verify if the network meets them. Allowing users and applications to specify various requirements for data transmission over the Internet and being able

to satisfy their conditions partially or in full means that the network no longer provide *best-effort services* but various services at different levels of quality. In *best-effort service* the network makes no distinction between classes of traffic and but makes an equal (best) effort to deliver all packets.

One of the proposed goals of the Internet Protocol was to provide different levels of service for Internet traffic as it can be seen from the Type of Service field in the Internet Protocol (IPv4) header [123]. However, until recently the Internet was providing almost exclusively best-effort services. This means that providing even limited quality of service guarantees (such as bandwidth, delay or jitter) over the Internet infrastructure is not a trivial task.

Therefore, there is a hot debate about whether to invest in QoS technologies or to increase the network capacity. The tradeoffs between the benefits offered by QoS mechanisms and the overhead associated with these mechanisms are at the root of the controversy that has always surrounded the discussion of QoS mechanisms [28].

Whatever the arguments against providing Internet QoS support, there are a large variety of applications demanding different treatments based on their stringent performance requirements. Here is Tanenbaum's classification [147] of major Internet application and their performance requirements.

Application	reliability	delay	jitter	bandwidth
E-mail	high	low	low	low
File transfer	high	low	low	medium
Web access	high	medium	low	medium
Remote login	high	medium	medium	low
Audio on demand	low	low	high	medium
Video on demand	low	low	high	high
Telephony	low	high	high	low
Videoconferencing	low	high	high	high

Table 2.1: *QoS requirements for different types of applications*

Therefore, as long as there are various classes of applications requiring different levels of service, there will always be someone willing to pay more for some

sort of QoS guarantee. Currently, the ISPs can only provide long term bandwidth guarantees for subscribers. Thus, for example, a subscriber might sign up for premium service a month at a time. The future Internet may be expected to feature pricing and signalling mechanisms to provide on demand QoS guarantees for ad hoc high requirements applications (e.g. a videophone call). Hence, ISPs need tools for differentiating and guaranteeing the level of service and managing various classes of service.

After this short introduction and motivation for Internet QoS, an overview of QoS mechanisms is presented in the rest of this chapter.

2.2 Components of Internet QoS

There is no single technology able to guarantee end-to-end quality of service over the Internet. In order to be able to satisfy the QoS requirements of a traffic flow, a combination of techniques and algorithms must be used. Hardware and software changes in the network elements are also required. There are numerous strategies for how to implement and deploy local or end-to-end QoS mechanisms over the Internet [16, 166]. Before discussing the most important Internet QoS related projects, some of the components of such mechanisms are introduced in this section.

2.2.1 QoS Metrics

QoS metrics are used to express the level of QoS required or received by a traffic flow. Applications can specify one or more requirements to be met by the network. The metrics are classified in three categories and defined as follows [42, 79]:

Let $m(r1, r2)$ be a metric for a link between routers $r1$ and $r2$. For a path $P = (r_1, r_2, \ldots, r_{i-1}, r_i)$, metric m is:

- **additive**, if $m(P) = m(r_1, r_2) + m(r_2, r_3) + \ldots + m(r_{i-1}, r_i)$ Examples are

delay, jitter, cost and hop-count. For instance, the delay of a path is the sum of the delay of every hop.

- **multiplicative**, if $m(P) = m(r_1, r_2) \times m(r_2, r_3) \times \ldots \times m(r_{i-1}, ri)$. An example is reliability, in which case $0 < m(r_i, r_j) < 1$.

- **concave**, if $m(P) = \min\{m(r_1, r_2), m(r_2, r_3), ..., m(r_{i-1}, r_i)\}$. An example is bandwidth, which means that the bandwidth of a path is the value of the link with the minimum available bandwidth.

2.2.2 Classes of service and service level agreements

Internet applications have various requirements that can be specified using the above-mentioned metrics. Routers along the path must be able to guarantee some level of QoS for the requested service. Therefore, an ISP may define classes of service based upon an application or user requirement.

The Internet protocol itself provides a way of specifying the Internet service quality by the mean of the type of service (ToS) field in the IP header [122, 123]. The future version of the IP (i.e. version 6) is also using dedicated header fields such as traffic class (initially called the priority field [51]) and flow label to allow the specification of various classes of QoS [134]. ATM has defined its own classes of service for the most common types of applications (see Section 2.3). In the integrated services Internet QoS model, one can distinguish between three classes of service, namely best effort, controlled-load and guaranteed service [135, 162]. More recent technologies require changes to the standard IPv4 and IPv6 implementation in order to provide their own support for QoS classification. Hence, in the differentiated services (Diffserv) [31] approach, ToS bits (IPv4) and traffic class bits (IPv6) respectively are replaced by the Differentiated Services Code Point (DSCP) field that is intended to map a traffic class to a particular forwarding treatment at each node along the path.

Other authors propose a QoS scheme with three classes of services for the Internet backbone [166]. The classes of traffic provided are:

- Premium service to provide reliable, low-delay and low-jitter service for real-time traffic such as voice over IP, video conferencing or financial traffic;
- Assured service to provide reliable and predictable traffic such as non-real-time VPN;
- Best Effort service for traditional Internet traffic (e.g. WWW, e-mail, etc.).

The number of classes of service provided may vary depending on the targeted applications for each class of service, how clearly a class can be distinguished from another, and depending on the service agreement between customers and service provider.

Service level Agreements (SLA) between ISP and customers can be used to define the level of service offered by the provider, and some sort of billing scheme. A customer may be a user organisation or another provider domain (upstream domain). The agreement typically spells out measures for performance and consequences for failure. SLAs can be classified as follows : [165].

Static SLAs are negotiated on a regular (e.g., monthly or yearly) basis.

Dynamic SLAs require the customer to use a signalling protocol (e.g., RSVP) to request services on demand.

The service performance level must be reviewed regularly by the two parties. Therefore, each service provided should be measurable by using monitoring, measuring and benchmarking tools. The requirements can be specified using QoS metrics or other quantifiable bounds.

An agreement could for example specify a service like this one: "128 Kbps of traffic will be carried from source S to destination D with near zero packet loss rate. Each packet will be delivered from S to D in less than 100 milliseconds."

2.2.3 Traffic scheduling

One important feature in *packet-switching* (*store-and-forward*) networks is the mechanism that determines which packet will be transmitted next on the output link. This mechanism is referred to as the *traffic scheduling algorithm* [143].

The role of traffic scheduling in the Internet QoS scheme is to guarantee the requirements specified in SLAs (Service Level Agreements). Hence, traffic schedulers must assure predictable delays as well as a fair share of the link bandwidth for concurrent traffic classes[1]. Such mechanisms must be able to guarantee the reserved traffic rate without packet loss, independent of the behaviour of other classes.

Traffic scheduling is mostly required in one of the following situations:

- When multiple organisations share bandwidth over the same link;
- When different communication protocols share the same link;
- When traffic types with different QoS requirements share bandwidth on the same link.

Since this last situation describes the traffic mix on most links in today's Internet, it suggests that traffic scheduling of QoS streams should be an intrinsic part of the Internet.

2.2.3.1 Traffic classes

In [52], Demers *et al.* apply the term *"user"* to identify individual traffic classes that compete for the same resource (e.g. output interface). *User* could refer to the source address of a packet, the destination address, the pair source-destination, a TCP conversation, etc. What defines a *user*, is irrelevant for a traffic scheduler. The behaviour of a traffic scheduler remains the same whatever the interpretation of *user*.

[1]The concept of "traffic class" in this context will be explained in subsection 2.2.3.1

However, the effectiveness and complexity of a scheduler depends on the number of *users*. The execution time of a scheduling algorithm increases with the number of concurrent *users*. Reducing the number of users will consequently increase the performance of a traffic scheduler.

QoS technologies such as *diffserv* [31] solve the above mentioned scalability issue by grouping *users* into classes and at any router/switch along the path, each user inside a class receives the same behaviour. Therefore, a whole class of *users* becomes a single *user*. In Multiprotocol Label Switching (MPLS) [133], a class of users forwarded in the same manner and carrying the same label is called a Forwarding Equivalence Class (FEC). I will refer to competing classes of *users* as FECs by analogy with MPLS.

2.2.3.2 Best-effort traffic scheduling

In best-effort Internet service, packets that need to exit a router (or switch) through an interface share the same output queue. They are processed in a FCFS (first come first served) manner. This is the least complex and easiest to implement queuing discipline. However, it cannot offer fair or preferential services for traffic flows. Moreover, one bursty FEC will have a negative impact on all competing FECs.

Although there are proposals to alleviate this issue whilst maintaining FCFS service (such as RED [61] and FRED [91]), fair bandwidth allocation can only be provided using multiple output queues.

2.2.3.3 Fair traffic scheduling

In order to prevent malicious FECs from affecting the well behaved ones, some level of isolation must be provided. This can be performed using a separate FCFS queue for each FEC.

The simplest approach to provide fair queuing is *round robin* processing of queues (RR) [111]. The main advantage of this method is its simplicity. A packet

from each queue is processed in a round-robin fashion (empty queues lose their turn). However, if a queue consistently has larger packets than the others, that particular FEC will get a larger portion of the bandwidth. Improvements to the basic RR scheme include *Deficit Round Robin (DRR)* [138] and *Hierarchical-Round-Robin* [81].

Several other fair queuing mechanisms have also been proposed, all of which use a separate FCFS queue for each FEC. They are classified as work-conserving and non-work-conserving:

- **Work-conserving** schedulers are never idle when a packet is buffered in the system. Such algorithms include *Generalised Processor Sharing (GPS)* [121], *Weighted Fair Queueing (WFQ)* [52], *VirtualClock* [167], *Delay-Earliest-Due-Date (Delay-EDD)* [59] and *Deficit Round-Robin (DRR)* [138].

- **Non-work-conserving** schedulers may remain idle even if there are available packets to transmit if higher priority packets are expected to arrive. Non-work-conserving schedulers include *Hierarchical-Round-Robin* [81] and *Stop-and-go queueing* [68].

2.2.3.4 Fairness of a scheduling algorithm

The fairness of a scheduling algorithms is measured by comparing it with the fairness of an ideal scheme called *Generalised Processor Sharing (GPS)*. In GPS packets are considered infinitely divisible and during one cycle, an equal amount of data is processed from each queue. While this is an ideally fair algorithm, it is not suitable for packet switched networks where packets have various sizes and they are not divisible.

Therefore, the perfect fairness of GPS can not be achieved in a packet based network. However, the best approximation to GPS algorithm is achieved when the difference in throughput at any time in any queue for any arrival pattern between the algorithm and the GPS discipline will never exceed MAX (MAX is

the maximum packet size) [136]. For example, the fairness of WFQ is MAX, of DRR is 3MAX and of FQRR (Fair Queuing with Round Robin [136]) is 2MAX.

In Section 4.4 a new family of fair, work conserving, traffic scheduling mechanisms that imitate the behaviour of the human heart in the cardiovascular system is proposed. The algorithms have MAX fairness and $O(\log N)$ complexity and thus compare favourably with existing algorithms. The algorithms are simple enough to be implemented in hardware. Table 2.2 shows the relation between fairness and complexity of our algorithm and other popular scheduling algorithms.

	FCFS	**DRR**	**WFQ**	**FQRR**
fairness	-	3MAX	MAX	2MAX
complexity	O(1)	$O(1)$	$O(\log N)$	$O(1)$

Table 2.2: *Comparison of scheduling algorithms*

2.2.4 QoS routing (constraint-based routing and policy-based routing)

"QoS-based routing has been recognised as a missing piece in the evolution of QoS-based service offerings in the Internet." [49]

Due to the importance of QoS-based routing, the IETF set up a QoS Routing Working Group [75] to define a framework and techniques and to guide the research for QoS-based routing in the Internet .

QoS routing has been defined as a method for finding feasible paths based on the QoS requirements of a traffic flow [49]. The algorithm must have knowledge of resource availability in the network.

Traditional routing protocols such as RIP and OSPF use a single metric to compute the shortest path toward a destination. This metric is usually hop-count or administrative weight. QoS routing is needed for applications that demand a guaranteed amount of network resources like bandwidth, buffer space, etc.

Therefore, given a set of quality-of-service (QoS) requirements for a connection, the routing algorithm should be able to find a path which satisfies the requirements [66].

The reason for designing and deploying QoS based routing is to solve problems that cannot be solved using best-effort routing. Hence, the main goals of QoS routing are [79]:

1. to meet the QoS requirements of end users;
2. to increase the network efficiency by optimising the network resource usage;
3. to avoid drastic performance degradation during congestion.

Nevertheless, introducing constraints in the optimisation problem to satisfy user QoS requirements increases the *computational cost*. Typically there are two types of constraints [42]: *link constraints* and *path constraints*.

Link constraints restrict the use of some links that do not satisfy traffic requirements. Link constraints use concave metrics (see Section 2.2.1) such as bandwidth. Performing QoS routing based on link constraints is relatively straightforward since one has only to remove from the network graph the links that do not satisfy the constraints. Then, a shortest path through the remaining topology can be computed.

Path constraints refer to the combined (added or multiplied) value of a performance metric along the path. Hence, path constraints use additive or multiplicative metrics such as end-to-end delay or packet loss. A shortest path problem with even a single path constraint is intractable (NP-complete) for large networks [65].

Various heuristic algorithms can be used to solve the complexity problem. One such method, called *sequential filtering*, is described in RFC 2386 [49]. Performing some of the computations in advance can also reduce router computation load [120].

The process of distributing information about link state, reserving resources along the path and maintaining per flow state information also increases the **communication cost**. This is a major issue when evaluating the overhead of QoS routing [11].

In order to reduce the communication cost, QoS routing technologies must minimise the frequency of routing information advertisements. This will inevitably introduce imprecision in the network state information. This is another challenge for QoS routing because **inaccuracy** can degrade the the performance of QoS routing and reduce the network throughput [12].

Other issues in developing QoS routing such as the *increased size of routing tables*, the level of *routing granularity*, topology aggregation for more than one QoS metric in *hierarchical QoS routing* and *the lack of implementation support mechanisms* (e.g. QoS scheduling) are discussed in [43, 94].

QoS routing is sometimes referred to as *policy-based routing (PBR)* [146] or *constraint-based routing (CBR)* [20]. However, the research community makes a distinction between the two concepts.

Policy-based routing is a concept related to QoS routing and commonly means that routing decisions are not based on the knowledge of the network topology and metrics, but on some administrative policies. These policies represent security constraints and are usually statically configured [79]. One such example is routing based on source IP address (*source routing*).

Constraint-based routing is considered as a generalisation of QoS routing because when making routing decisions, it takes into account traffic attributes, network constraints along with policy constraints [18]. One example of CBR is *Constrained Shortest Path First (CSPF)* which is an extension to shortest path algorithms such as RIP, OSPF and IS-IS, and which computes the shortest path after pruning the links that do not satisfy a set of constraints.

2.2.5 Signalling protocols

When the network is required to provide a certain level of service, network nodes must be able to communicate, negotiate, reserve resources along the path and maintain state information. To achieve all these tasks *signalling protocols* are required. Signalling protocols are a means for routers to exchange and maintain state information about network and QoS constraints.

In connection-oriented networks such as ATM, signalling is used to initiate virtual circuits before any data transmission can occur. In the connectionless Internet, signalling protocols can be used to discover a suitable path for a connection and to reserve resources along the path. In MPLS, signalling protocols are used to distribute label information in order to initiate and maintain Label Switched Paths (LSPs). The IETF *Next Steps in Signalling Working Group* [76] was created in order to standardise an IP signalling protocol to be used in QoS-enabled networks. In this thesis I will provide an overview of signalling protocols for ATM (Section 2.3.3), IP (Section 2.4.5) and MPLS (Section 3.3.4).

2.3 QoS in ATM networks

The early phone network consisted of a purely analogue system that connected telephone users directly by a mechanical interconnection of wires. The "digitalisation" process began in the 1960s and in the 1980s, telecommunication companies gradually introduced the Integrated Services Digital Network (ISDN) [142]. However, ISDN, with its limited set of supported bit rates, was a poor fit to emerging high-bit-rate applications with diverse bandwidth requirements [46]. To address these concerns, ITU-T [2] and other standards groups started, in the 1980s, to establish a series of recommendations for the transmission, switching, signalling and control techniques required to implement an intelligent fiber-based

[2]The Telecom Standardisation Sector of the International Telecommunication Union, formerly known as the Consultative Committee for International Telephone and Telegraph (CCITT)

network that could solve current limitations and would allow networks to be able to efficiently carry emerging services. By the end of the 1980s, Asynchronous Transfer Mode (ATM) [84] was developed as a promising technology for the next generation of heterogeneous communication networks, because of its embedded QoS capabilities and high transfer rates. ATM represents the transition from digital circuits to packet based communication networks.

2.3.1 Features of ATM

Designed for voice, video and data communications, ATM uses a 53 byte long packet called the *ATM cell*. It was felt at the time of its standardisation that it would not be possible to build a fast packet switch for variable-length packets. The fixed cell length was chosen to be short because ATM would be used for telephony (and new unknown services) and long packets would cause excessive packetization delay. 53 bytes was picked as an awkward compromise between European (32 bytes) and American (64 bytes) preferences. There was no specific intention to support IP (as it was not very popular at the time).

ATM is a connection-oriented and label switching technology [46], using a fixed length label field (VPI/VCI)[3] inside its forwarding table. This makes its routing simpler and faster [164] than IP's longest prefix match. A *virtual channel* (VC) is set up before any data is sent through the network. VCs are uniquely identified on a link by the pair of VPI/VCI values. The VPI specifies the path (or "bundle") through the network and the VCI identifies a single VC within the path.

QoS requirements are specified when a connection is established and remain in place until the connection is terminated. Regarding traffic requirements, ATM defines a few classes of service such as:

Constant Bit Rate (CBR) for applications generating traffic at fixed rate (e.g. un-

[3]Virtual Path Identifier/Virtual Circuit Identifier

compressed audio and video streaming);

Variable Bit Rate (VBR) for applications that know in advance they will have variable traffic rate (e.g. compressed audio and video rate depends on the amount of input during each sample). There are two subclasses of service for VBR: one for real-time traffic such as videoconferencing and one for non real-time traffic such as watching video or audio broadcasts.

Available Bit Rate (ABR) for bursty applications that do not know in advance the rate at which they will generate data (e.g. Web browsing or FTP).

Guaranteed Frame Rate (GFR) to provide a minimum rate guarantee to VCs at the frame level. The GFR service also allows for the fair usage of any extra network bandwidth.

Many of these applications were not widely used when the service classes were defined.

The QoS requirements must be specified before the connection is established. The connection is then accepted only if all switches along the path can meet the requirements; otherwise the request is rejected. This is because ATM is a connection-oriented technology and therefore, its behaviour is similar to telephone network.

ATM uses virtual circuits to establish connections between the sender and receiver like frame relay and X.25. Connection-oriented architectures are attractive for QoS because they require state information at each network element, and this control information can enable the support of services that are impractical within a pure datagram network [90]. ATM's virtual circuit switching allows both traffic aggregation and disaggregation. Aggregated data travelling along a specific path can receive the same level of QoS. Alternatively, different connections with the same destination can be routed along different paths whereas in datagram routing, packets for the same destination are bound to use the same next hop.

Another advantage in a connection-oriented network is that service restoration can be provided easier and faster by redirecting effected connections. In a datagram network, following a node or link failure, the routing protocol must converge before the service can be restored.

2.3.2 Traffic control in ATM networks

ATM aims to provide QoS guarantees in a connection-oriented environment. It therefore needs mechanisms to process incoming traffic requests as well as to control the existing traffic behaviour as part of an ATM congestion control scheme.

Connection Admission Control (CAC) is an important traffic control component of ATM networks. Any connection request is passed to the CAC which decides whether the connection set-up should be accepted or rejected. The decision is based on resource allocation schemes used for each node and link. If a connection is accepted, during its life time, the *Usage Parameter Control (UPC)* checks whether the actual transmission rate is compliant with the requested/negotiated rate.

From the QoS perspective, CAC is a preemptive congestion control mechanism. A more complete survey of CAC in ATM networks is presented in [153].

2.3.3 Signalling

One advantage of the ATM technology is that it does not require routing at each node. The ATM cells are switched according to their VPI/VCI label and QoS requirements. Nevertheless, before any data transmission a VC must be set-up. The process of initiating a VC, negotiating QoS parameters for that connection and distributing VPI/VCI information is called *signalling*. Signalling is also responsible for maintaining and tearing down the VCs.

There are two sets of signalling standards for ATM (see Fig 2.1):

UNI signalling is performed between end stations and a private ATM switch or between a private ATM switch and the public IP network. UNI signalling

is relatively simple since it does not involve routing. The UNI standards developed by the ATM forum are UNI 3.1 [14] and its successors. UNI 3.1 is derived from ITU's Q.2931 protocol which also evolved from the Q.931 protocol used in ISDN and Frame Relay.

NNI signalling is performed between switches in an ATM public network. An important component of NNI signalling is finding a feasible path for the VC through the ATM network. Therefore, more complex signalling mechanisms are needed. There are two major standards for NNI signalling: Integrated Interswitch Signalling Protocol (IISP) and Private Network-to-Network Interface (PNNI) [15]. IISP is simple because it uses static routing and is therefore suitable for small ATM networks. PNNI is a hierarchical signalling protocol designed to scale well for very large ATM networks.

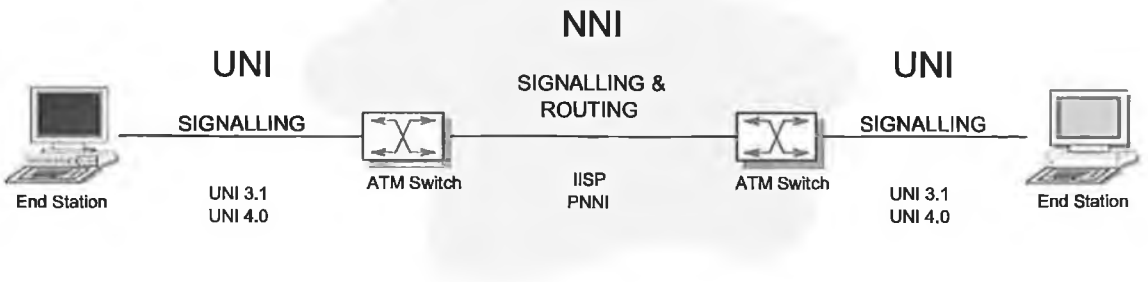


Figure 2.1: *ATM UNI and NNI Signalling*

IISP, also known as PNNI Phase 0, uses UNI procedures to forward signalling requests across an ATM port based on the longest prefix match lookup. IISP uses static, hop-by-hop routing tables. Since it was an interim protocol it will not be discussed here. PNNI is discussed here both as a signalling protocol for ATM and as a reference model for hierarchical routing in the Internet .

2.3.4 PNNI

PNNI is a complex NNI routing and signalling protocol. Complexity is the price paid for its advantages over best-effort routing protocols. Some of these advantages are:

Scalability. PNNI is a hierarchical routing protocol where nodes are grouped to form logical nodes and only aggregated network topology has to be maintained. Therefore, PNNI can scale to potentially tens of thousands of ATM switches.

QoS routing support. Shortest paths are computed based on QoS constraints and VCs are set-up only if the traffic requirements can be satisfied.

Stability. PNNI uses source routing. Hence, loops in the paths are easier to avoid when the aggregated topology data is inaccurate.

Reliability. Signalling is performed via dedicated bandwidth-guaranteed VCs.

The most powerful feature of PNNI is that it is a hierarchical routing protocol. This means that routing and signalling are performed at various virtual layers. The network administrator creates *peer groups* (PG) of ATM switches each of which will form a *logical group node* at the next hierarchical level. Logical nodes are connected by *logical links* which are mapped to VCs in the physical network. Within each peer group, a *group leader* is elected to perform topology aggregation and advertisement on behalf of the group members. Logical nodes are in turn grouped into logical peer groups and so on recursively for an arbitrary number of hierarchical levels.

Topology information and state parameters such as the *available cell rate* (ACR) or *cell loss ratio (CLR)* are flooded between peer groups and dedicated logical links are used to propagate routing information between different hierarchical layers. Complex algorithms for summarising and compressing topology state information are used at each hierarchical level to reduce the amount of routing information.

When an ATM switch is turned on, it sends HELLO messages to its neighbours using a dedicated VC called the Routing Control Channel (RCC). Its neighbours respond with HELLO messages containing information about their neighbours. The HELLO process is in continuous operation and its role is to advertise topology changes such as new nodes and links or broken nodes and links. Peer group leaders are responsible for distributing/receiving topology information to/from higher level logical groups.

The topology information flooded by the HELLO messages also contains link and node attributes such as maximum CTD (Cell Transfer Delay), maximum CDV (Cell Delay Variation), maximum CLR (Cell Loss Ratio), administrative weight, ACR (available cell rate), CRM (cell rate margin), etc. Topology information is stored in PNNI Topology State Elements (PTSEs). Once a node's PTSE is consistent with its neighbours, it will share its routing database with all peer group members. In this way each node maintains routing knowledge about the network: detailed topology information about its peer group and aggregated information about the other logical nodes.

PNNI uses source routing, which means that upon receiving a request to establish a VC the ingress router (the switch connected to the UNI setting up a connection), using the same shortest path first algorithm as OSPF (Open Shortest Path First) and IS-IS (Intermediate System to Intermediate System) routing algorithms, will find a feasible path to the destination. The route is stored in a stack of Designated Transit Lists (DTLs) [15] with detailed information for the local PG and summarised information for higher level PGs that will be expanded when traversing another PG.

Despite using source routing and aggregated routing information, PNNI performs well in hierarchical networks when compared with global routing strategies [21]. This, and its QoS capabilities, makes PNNI a reference model for designing and implementing QoS routing strategies in communication networks.

2.3.5 Limitations of ATM

Despite its ambitious goals as an universal transport mechanism and its QoS capabilities, ATM was not accepted as the universal carrier for the Internet. The main factors that prevented this are [46]:

Deployment cost. Although an ATM switch can provide a comprehensive list of services, it is much more expensive than LAN hardware. ATM network cards for personal computers are also significantly more expensive than alternative Ethernet cards. These cost differentials are maintained as one moves toward the backbone.

Connection setup latency. Connection-oriented technologies such as ATM need to set up virtual paths before sending any data. This works well for long lasting connections. But for sending and receiving small amount of data, connection setup and tear down only introduce additional delay.

Cell overhead. Due to its small cell size the ATM header overhead is around 10%. Most competing protocols introduce less overhead.

QoS requirement specification and complexity. The appropriate choice of QoS parameters for various services is not self-evident. Conservative parameter choices may result in non-transmission of data due to a failed set-up request or wasted bandwidth. Aggressive choices may result in an inadequate QoS being carefully regulated.

Moreover, the fine grain QoS model in ATM increases the routing complexity and raise scalability issues.

Lack of efficient broadcast. Broadcast and multicast are not supported natively by ATM. They must be emulated. This was the main factor that prevented ATM from expanding from the core toward the LANs at the edge of the Internet.

Homogeneity. It would be preferable to have the perfect single type of technology to support all types of communications. However, ATM could not satisfy all networking requirements. The Internet (r)evolution showed us that no single technology can meet all the conflicting demands placed on modern networks.

Although ATM was promised more predictable latency than IP, it failed as an alternative to IP as an access technology, primarily because it was never properly supported by popular OS's such as Windows, UNIX, etc. Also, its QoS capabilities could only be exploited using signalling protocols. The ATM Forum [103] signalling protocols (UNI 3.0 [14] and its successors) were approved rather late, and the ITU standard (Q.2931 [77]) was very complex and more suited to the needs of telcos than the datacomms industry. Moreover, stack vendors charged too much for licences.

ATM failed in the LAN market because:

1. Early attempts to emulate broadcasting (as happens on an Ethernet LAN segment) were very inefficient.

2. The NICs were always more expensive than Ethernet NICs because of the relative complexity of ATM and the absence of economies of scale.

3. Switched Ethernet seemed familiar to LAN administrators, even though it arguably had more in common with ATM than with bridged Ethernet.

ATM didn't fail as an IP carrier for the Ethernet. Actually, ATM is a viable layer 2 technology for IP, and is widely used as such. Some companies are still using ATM as a backbone technology. However, it is increasingly likely to be substituted in the future by cheaper Gigabit Ethernet and by the ability to send IP directly over SDH, allowing IP routing over fibre.

2.4 QoS in IP networks

Due to the simplicity of the IP protocol, IP traffic became the blood that flows through the heart of the Internet. Every host and router in the Internet has an unique IP address. The main deficiency of the Internet is that it was not designed to carry delay sensitive traffic. The Internet protocols have to evolve continuously to accommodate new types of traffic and to be able to distinguish between these types, in order to provide them with differential treatment. Hence some components of the Internet (such as IP's routing concept) must be replaced or modified.

2.4.1 Best effort routing

The devices that perform the network layer forwarding function of the Internet protocol suite are called *routers* by IETF, *intermediate systems* by OSI and were formerly known as *gateways* [23]. An IP router removes the layer 2 header, lightly modifies the layer 3 header, generates a new layer 2 header and retransmits the packet based on the original layer 3 information.

The robustness and simplicity of IP routers were key factors in the Internet being such a successful network. Classic routing protocols such as RIP [71] and OSPF [107] use only a single routing metric, resulting in fast convergence of routing tables, stability and robustness. Applications requiring multiple routing metrics triggered the development of more advanced routing techniques even from the first days of inter-network connectivity as shown in the following sections.

2.4.2 Adaptive routing in ARPANET

A better routing technology was needed since the early ARPANET (the precursor of the Internet). The original routing algorithm in ARPANET used distance vector routing based on a distributed Bellman Ford algorithm [48]. The length (delay) of a link equalled the number of packets queued for transmission plus a

constant. The main issues were caused by large update packets, inconsistencies with distributed route calculation and slow adaptation to congestion and topology changes.

The proposed solution was adaptive routing [100]. In this approach, routing decisions were based on the current state of the network. The idea was to compute path costs based on the total estimated transit time. Each node had to maintain a table of network delays, representing the estimated delay experienced along each possible path. The minimum delay table was periodically sent to neighbours, along with the hop-count shortest path, which provided connectivity information.

The main drawback of this procedure was that traffic adheres to the minimum delay path causing congestion to shift from one path to another. The new congested path will not be optimal at the next election and therefore, a new optimal path will be selected that will attract all the traffic and become congested as well. This will induce oscillations in traffic flows (route flopping) and will result in network instability.

2.4.3 ToS routing

The Type of Service facility (ToS) has been part of the IP specifications since its inception [5, 122, 123]. However, it has been rarely used to data, but it is expected to play an increasing role in the near future. The ToS field is expected to be used to control two aspects of gateway operations: routing and queuing algorithms [34, 35]. In classical ToS the route selection depends upon the ToS field of the IP packet. The value contained in the ToS field may specify one of the following requests:

- minimise delay
- maximise throughput

- maximise reliability
- minimise monetary cost
- provide normal service

Multiple link costs are associated with each link (link costs may be different for different ToS choices). Link computation for a particular ToS takes into consideration one of the link costs. A shortest path tree is computed for each ToS. Therefore, the computation becomes more complex and more expensive while a separate shortest path tree algorithm must be spawned for each ToS.

Shortest path routing used by classical ToS routing may result in localisation of traffic within the network. Thus, effective traffic engineering is difficult to perform in classical ToS routing. Moreover, classical ToS routing became obsolete when a Diffserv field replaced the ToS field in the IP header [116].

2.4.4 Integrated services

The integrated services (Intserv) model was developed by the Internet Engineering Task Force (IETF). The term integrated services (IS) is used for an Internet service model that includes best-effort service, real-time service, and controlled link sharing [36]. The Intserv model proposes to extend the original Internet architecture to support real-time QoS and provide control over end-to-end packet delays. The new components and mechanisms to be added will supplement but not replace the basic IP service. The new components are:

- **a packet classifier** used to identify flows that are to receive a certain level of service;
- **a packet scheduler** to handle the scheduling of service to different packet flows to ensure that QoS commitments are met;
- **admission control** that is used to determine whether a router has the necessary resources to accept a new flow.

The proposed architectural extension comprises an extended service model called the IS model and a reference implementation framework, which provides a set of vocabulary and a generic program organisation to realise the IS model. In order to ensure that the QoS requirements of the traffic flow is satisfied, the model requires resources such as bandwidth and buffers to be reserved *a priori* for a given traffic flow. Therefore, the IETF defines in RFC 2205 [124] a resource *ReSerVation Protocol (RSVP)* designed for an Integrated services Internet. In RFC 2212 [135] and RFC 2211 [162] the IETF defines two models under the integrated services model: **guaranteed service** and **controlled-load service**.

The main drawback with Intserv was scalability in large public IP networks that may potentially have millions of active micro-flows concurrently. This is because Intserv uses per-user flow storage/processing for QoS at routers. This implies hardware complexity for storage, scheduling and monitoring traffic flows. Additional software complexity is also introduced by using the RSVP protocol.

2.4.5 RSVP

RSVP is a *soft-state* signalling protocol [168] for discovering and reserving network resources (soft-state meaning that the reservation at each node needs a periodic refresh). The RSVP protocol is used by a host to request specific qualities of service from the network for particular application data streams or flows. RSVP is also used by routers to deliver quality-of-service (QoS) requests to all nodes along the path(s) and to establish and maintain state information to provide the requested service. RSVP requests will generally result in resources being reserved in each node along the data path [124].

Under RSVP there are two kinds of state information at each intermediate switch, *path* state and *reservation* state. The path state is established or updated by a *Path* message which is periodically sent by data sources with the same source and destination addresses as traffic that the sender will generate. Each receiver

periodically sends a *Reservation(Resv)* message that establishes or update the reservation state.

The path state includes at least the unicast IP address of the previous hop node, which is used to route the Resv messages hop-by-hop in the reverse direction. The Path message contains the following information in addition to the previous hop IP address:

- A sender *Template* specifying the format of the traffic;
- A sender *Tspec* specifying the characteristics of the traffic;
- An optional *Adspec* which is used to support the concept of "one pass with advertising" (OPWA) [124].

Each receiver host sends Resv messages upstream towards the sender. These messages must follow exactly the reverse of the path the data packets will use. Every intermediate router along the path can reject or accept the reservation request of the Resv message. If the request is rejected, the rejecting router will send an error message to the receiver and the signalling process will terminate. If the request is accepted, link bandwidth and buffer space are allocated for the flow and the related flow state information is installed in the router.

The major issue with the original RSVP was scalability because reservations were required for micro-flows, so that the amount of state maintained by network elements tends to increase linearly with the number of micro-flows [27].

2.4.6 Differentiated services

The goal of the Differentiated Services (Diffserv) architecture is to provide scalable service differentiation in the Internet. This architecture achieves scalability by aggregating traffic classification state which is conveyed by means of IP-layer packet marking using the DS field [31, 116]. Packets are classified and marked to receive a particular per-hop forwarding behaviour on nodes along their path.

The IETF Diffserv working group has defined a Differentiated Services field in the IP header (DS field). The DS field consists of eight bits of the IP header, formerly known as the TOS octet. The DS field is used to indicate the forwarding treatment that a packet should receive at each node [116]. The Diffserv working group has also standardised a number of Per-Hop Behaviour (PHB) groups. Using the PHBs, several classes of services can be defined using different classification, policing, shaping and scheduling rules [29].

A Service Level Agreement (SLA) [165] between an end-user and an Internet Service Provider (ISP) may be required to receive Differentiated Services. Rules such as classifier, metering, marking, discarding and shaping can also be defined by a Traffic Conditioning Agreement (TCA) which is explicitly or implicitly specified by a SLA.

Packets are classified, marked and possibly policed and shaped at the ingress to a Diffserv network. When a packet exits the Diffserv cloud, the DS field may be overwritten accordingly to the existing agreements between the administrative domains.

The advantage of Diffserv over Intserv is the scalability issue. Diffserv allows only a finite number of service classes to be defined by the DS field. The resources are allocated on a per-class basis and the amount of state information is proportional to the number of classes rather than the number of traffic flows. However, in Diffserv all the flows within the same class effectively receive best-effort service.

2.4.7 Explicit routing and route pinning

Shortest-path routing algorithms often produce unbalanced traffic distribution and route oscillations [20]. This limitation of the current routing protocols is widely recognised and thus, a load-balancing scheme over equal cost multipath using OSPF was proposed in [154]. Another solution is to use *explicit routing* to

optimise the traffic distribution through the network. Although some level of route control can be provided with IP, if supported by an enhanced routing protocol a much easier approach is to use an overlay network such as MPLS [158]. The algorithm for placing the explicit routes depends on the optimisation objective. One such scheme that redistributes the traffic load based on periodically probing multiple paths is presented in [55] (see also Section 3.4.5).

Route pinning can also be used to fix the path for a loose segment of the path. Therefore in the event of a change in IP routing, that path segment is not rerouted (except for route failures when the path is no longer available). This can prevent unwanted traffic oscillations in the network. Extensions for MPLS signalling protocols include support for both explicit routing and for route pinning [17, 80].

2.5 Internet traffic engineering

According to the Internet Engineering Task Force, "Internet traffic engineering is defined as that aspect of Internet network engineering dealing with the issue of performance evaluation and performance optimisation of operational IP networks" [18].

Hence, Internet traffic engineering (called simply traffic engineering in the following) is a critical component of an end-to-end Internet QoS framework. Its main functions are optimisation and control and will be described in the following sections.

2.5.1 Traffic engineering optimisation functions

2.5.1.1 Enhancing the performance of IP networks

Traffic engineering technologies give service providers a better control over the network in order to enhance the performance of their network. Improvements are to be made at both traffic level and resource levels [20].

Traffic-oriented performance objectives aim to support QoS operations of user traffic; therefore the key traffic oriented performance objectives include:

- Minimising traffic loss;
- Minimising delay;
- Maximising throughput;
- Enforcement of service level agreements (SLAs).

Resource oriented performance objectives deal with issues regarding the optimisation of resource utilisation. In general, the two main aspects of resource management are:

- Resource over-utilisation (congestion);
- Resource under-utilisation.

Both under-utilisation and over-utilisation cause dramatic reduction in the performance and efficiency of a running network. Over-utilisation, also known as congestion, occurs when the offered traffic load exceeds the capacity of a certain resource (i.e. link or router). This will result in delays, jitter and loss of data. On the other hand, resource under-utilisation, at a glance seems just an economic disadvantage where expensive network equipment is used inefficiently. Unfortunately, under-utilisation is more than an economic issue. It is usually a reflection of congestion occurrence in another subset of the network.

There are two main factors that induce congestion:

- Insufficient or inadequate network resources, incapable of accommodating the offered load;
- Traffic flows being inefficiently mapped onto available resources; causing unequal utilisation of network resources (under-utilisation and over-utilisation).

Therefore, one of the central functions of traffic engineering is an efficient management of resources.

2.5.1.2 Facilitating reliable network operations

Another important objective of traffic engineering is to facilitate reliable network operations by providing mechanisms that enhance network integrity and survivability. The provided mechanisms should help to minimise a network's vulnerability to service outages due to errors, faults or failures that have occurred within the infrastructure. A reliable network is more proof to data loss, delays and jitters. Consequently, achieving this objective will substantially improve network performance which is the main objective of Internet traffic engineering. An example of how network reliability can be increased using MPLS protection and restoration techniques is described in Section 3.4.3.

2.5.1.3 Capacity planning and network design

Traffic engineering should continuously monitor the performance of the live network and use the feedback parameters to maintain the network in an optimal state. This process may also include network design and capacity planning in order to create an optimal network topology, more suitable for providing the end-to-end QoS guarantees sought by users.

2.5.2 Traffic engineering control functions

2.5.2.1 Control and optimisation of routing functions

Traffic engineering, as a part of the Internet QoS framework, should be able to control the routing procedure subject to user QoS constraints while maintaining an optimal network performance. Consequently, the Internet research community is developing tools that, by decoupling routing and forwarding, can optimise the forwarding procedure (e.g. MPLS) and allow for more complex routing algorithms to identify and maintain feasible paths for QoS Internet traffic (QoS-routing). MPLS simplifies the forwarding procedure and as a connection-oriented

technology provides better control over the traffic flows. The advantages of using a label-switching technology such MPLS will be presented later in Section 2.9.

QoS-routing techniques can be used to solve the problem of finding optimal paths for QoS traffic requests. Both technologies will be discussed later in this chapter as tools for the traffic engineering process.

2.5.2.2 Enhancing the global network characteristics

The previous subsections described the local objectives of traffic engineering and the parameters to be optimised in order to have an efficient network. However, the user perception of the network is as a single complex entity and not as a sum of its attributes. In the the traffic engineering processes "tools" such as MPLS, Intserv, Diffserv and RSVP can be combined in order to develop a framework for providing QoS in the Internet. This requires perfect correlation and interoperability between the technologies used to perform traffic engineering. Examples of end-to-end QoS architectures are described in Section 3.4.6.

2.5.2.3 Admission control and policy

Unwanted traffic flows can unbalance the equilibrium of a traffic engineered network. Therefore, considerable care should be taken when accepting new incoming flows. Good policies and connection admission control mechanisms should be used to maintain the network in the optimal state. Traffic that does not comply with SLAs should be kept away from the network and sometimes it is appropriate to reject new traffic flows in order to prevent congestion that can dramatically degrade the performance of the network. From the QoS perspective, admission control is a preemptive congestion control mechanism.

Although traffic engineering can be performed using traditional IP routing protocols [63], technologies such as MPLS and QoS routing facilitate the deployment of traffic engineering. Chapter 3 will describe the main characteristics of MPLS and its advantages for traffic engineering.

2.6 Performance of Internet routers

QoS support has traditionally involved the management of links (link bandwidth, jitter in the output buffer feeding the link, etc.). However, a considerable amount of packet processing is performed in modern networks where routers also perform layer-four switching [141], firewalling, deep packet inspection [53], denial of service attack detection [39], etc. This means that a router can be computationally overloaded even when the link metrics are satisfactory. In the absence of QoS parameters to measure such an overload, the router needs to be designed to handle the worst case computational load, and this is particularly a problem in soft routers.

There are two principal solutions to this problem. One is to reduce the packet processing time and the other is to reduce the number of packets to be processed.

For the former approach several solutions have been proposed, including the faster IP hardware and software longest-prefix match algorithms surveyed in [96], and fast mechanisms for layer 4 (and above) switching [53, 141].

There are fewer approaches for reducing the number of packets that need to be processed by a router. These approaches are based on the idea that if the average packet size is increased, the packet rate and the overhead are reduced.

2.6.0.4 Increasing the average packet size

The maximum packet size in an Ethernet network is 1500 bytes. Other layer 2 technologies allow larger MTUs[4]. For example the MTU is 4500 bytes for Fiber-Distributed Data Interface (FDDI), 9000 bytes for ATM and 65280 bytes for High Performance Parallel Interface (HIPPI). Therefore one of the first approaches proposed by IEEE was to increase Ethernet's MTU [119].

However, increasing the MTU in the core of the Internet does not increase the average packet size because the LAN technologies at the edge still use a small

[4]Maximum Transmit Units

MTU. Moreover, the small size of a lot of packets in the Internet is not a consequence of the MTU but of the protocol generating the packets. Hence, one approach to increasing the average packet size is to group multiple packets with similar routing properties (e.g. the same destination network) into a larger frame.

Such an approach is *gathercast* [22], an IP based mechanism for flow aggregation. The main goals are to increase the throughput and to reduce the load by eliminating redundancy and combining small packets. Gathercast uses the concept of *transformer tunnels* [144] over a gathercast sink tree. Various transformation functions such as *reassembly, compression, rate control* and *replication removal* can be attached to the tunnels. The reassembly function assemble small packets to form larger frames.

Gathercast can therefore provide a scalable and efficient aggregation mechanism for a class of applications that need to collect data from a large number of nodes. However, a framework for general Internet traffic has yet to be developed.

One problem with the IP based mechanisms is that one can only group packets with the same destination address. The overhead involved in any more complex scheme cannot be justified. Using tunnelling technologies such as MPLS where various traffic flows are switched along the same virtual circuit (LSP) allows for any packets following the same LSP to be assembled in a larger frame. The complexities involved in identifying the relevant packets is already provided by the tunnelling protocol. Moreover, such longer meta-frames can, in MPLS, be routed along traffic engineered explicit LSPs based on various QoS constraints. A mechanism to provide this facility, that, with the large scale MPLS deployment, can be employed near the edges of the global Internet will be described in Chapter 4.

2.7 Application level QoS

QoS can sometimes be controlled from user-space (at the application level) or by a combination of application and network level support. Two such examples are

the long playout buffer used by multimedia streaming clients to combat jitter or traffic-based adaptive compression to vary the quantity and quality of streamed data based on the available bandwidth [109].

Another example is the use of Forward Error Correction (FEC) codes to provide support for reliable delivery of content (so that lost packets don't need to be resent) in IP multicast [93].

One particular case of application level QoS is increasing Web servers availability. The WWW is the preferred technology used to provide information and e-services over the Internet and busy sites encounter billions of hits per day. At this rate one server is not able to handle all the requests. Some of the application level approaches to overcome this problem are presented below. It is also possible to add features in the network to address such application-level problems. Such solutions to Web servers load balancing and to provide guaranteed level of Web services will be presented later in Chapter 4.

2.7.1 Quality of Web service (QoWs)

2.7.1.1 Web content caching

One of the early approaches to improve the Web services performance was the caching of Web content at the client side, initially on the client local machine (the cache maintained by the Web browsers) then at the corporation level by using proxy servers [1, 2, 32]. Caching mechanisms will deliver the local stored data, if data was previously requested by another client or a previous connection and if the content is up-to-date instead of the content requested from the remote server.

The caching solution was only a temporary attempt to reduce the number of requests by reducing the redundancies in the data transferred over the Internet. This only works with static Web content. With the introduction of new services, a new type of information was processed by Web servers: dynamic data, in which the information is dynamically generated by the server before answering the re-

quest. This kind of information cannot be cached; therefore, the use of caching will not reduce the server workload.

2.7.1.2 Mirroring

The second approach known as mirroring consists of maintaining a number of servers with similar content, but with a different geographic location and a different name and Internet address (e.g. [151]). The client has to choose, among the geographically distributed servers, the one that is best suited to his requests. This approach leaves the decision of choosing the mirror and the responsibility of choosing the right one to the client. Many times the client will initiate more than one request to different servers, in order to determine the "closest" mirror. On the other hand, maintaining a perfect synchronisation between the mirrored servers may not be easy, especially for time critical applications. Moreover, the situation when the clients are not geographically distributed but concentrated within a single geographic area (or even the same WAN or LAN) cannot be solved by spreading the servers around the area. The time spent by the client in deciding which is the most suitable server may be too long for mission-critical applications.

2.7.1.3 Cluster of servers (server farms)

The next approach tries to avoid user involvement in the process of choosing the best server. The technology has to transparently divert the client's request to the optimal server. The technique consists of grouping the servers in so called server clusters and adding a new and transparent service to the network, which is responsible for distributing the requests uniformly among the servers [83].

Successful administration of server clusters or server farms requires the use of queuing theory and load balancing techniques. The most important goal from the Web service provider's point of view is to balance the workload among the servers within the cluster.

The two major methods of building Web clusters are described as follows.

Replicated content servers

The first method is to mirror the content of a Web server and to create two or more servers having identical content. This resembles the geographical distribution of mirrored servers, but has the advantage that the servers are grouped within the same building or room and under the same administration; thus the task of synchronising the Web content between mirrors is much easier. Moreover, the process of choosing the optimal server is no longer the client's responsibility.

Distributed content

The second method is to distribute the Web content among the servers within the farm. Therefore, the decision of choosing the server is based on the client's HTTP request and involves filtering the packets up to the application level. Thus, the problem is no longer a problem of uniformly distributing the requests but of *a priori* distributing the content within the servers in a manner that will result in a balanced workload among the servers.

A cluster of servers is the typical solution to the problem of increasing Web server availability. The main issue with Web clusters is how to balance the load across the servers. Here are the main approaches:

Round-robin Domain Name Service is one approach used to avoid the server congestion by distributing connection loads over a cluster [87]. In a standard scenario, a domain name is associated with an IP address. Since the client uses the domain name to access a Web site, a DNS has to translate the name into the correct IP of the destination server. Therefore, the DNS server is a part of the Web browsing process. Moreover, the server can be modified to answer with different IP addresses for different translation queries. The DNS server will rotate through the list of IP addresses in a round robin fashion in such a way that each server in the cluster will receive an equal share of the incoming requests.

The main advantage of round-robin DNS is its simplicity. On the other hand, there are also major drawbacks for this approach. The caching feature of DNS at the client side prevents an accurate load balancing scheme since not every in-

coming request will get its address directly from the round-robin DNS server. Disabling caching may appear to solve the problem. However, every DNS query must then be resolved by a single server; this is expensive and potentially slower for users. Moreover, a client may use the IP address of the Web server to access it, thereby bypassing the DNS server so that all its requests will be sent to the same server.

The other major disadvantage of this approach is that the DNS server does not have any knowledge about the status of each server in the cluster. The round-robin scheme will continue to send traffic to all servers in turn, even if some of them are overloaded or out of service.

Load balancing switches such as Cisco's LocalDirector [44] and Alteon's ACEdirector [117], are hardware solutions that distribute TCP connections over multiple servers. These Web switches act as a front-end dispatcher between the Internet connection and the Web farm. All the client requests will use the dispatcher IP as a destination address, to make the requests. The switch then forwards the requests to different Web servers based on various load-balancing algorithms implemented in the switch. The decision can be based on the content of the request. Using source IP address alone to create affinities between client and server will not work well since some companies use proxy servers that change the source IP of the request. Therefore, all the requests from behind the proxy will have the same IP and thus the whole network behind the proxy will be treated as a single computer.

Load-balancing Web switches support up to millions of connections simultaneously at high speeds. Moreover, switches will frequently check the status of the servers so they can implement "intelligent" load balancing schemes. Using a Web switch is much better and more scalable than using other approaches but they are quite expensive. In addition, avoiding a single point of failure, may require the use of multiple switches which makes the solution uneconomic.

HTTP redirect can be used by the targeted server if it cannot accept more

connections [8]. This technique will slow down the process since the request is sent back to the client along with another IP to use for the connection. The client will have to initiate another connection to the new IP and use the server to which was redirected.

The above mentioned solutions all have major drawbacks, but they are widely used in today's best-effort Internet. However, the Internet is evolving into a next generation QoS enabled global network and new standards and protocols are now available (e.g MPLS). Therefore, in Section 4.1 an alternative network-level load balancing solution for next generation MPLS-capable networks is proposed. Since MPLS provides better mechanisms to support QoS routing than the legacy IP, it can more elegantly provide QoS functions for Web switching such as *content-based-routing*, *client affinity*, *different classes of service* and *load balancing*, as identified by the authors of [3] and described below:

Content-based-routing is a technique used when the content of the Web site is partitioned between the servers in the cluster. All the requests for the same server will be classified by the ingress nodes into the same MPLS Forwarding Equivalence Class (FEC). This solution has two major advantages. It will reduce the load at the dispatcher since the decisions are taken at ingress nodes. Moreover, the single point of failure can be eliminated at the dispatcher since LSP's can follow different routes toward their destinations within the MPLS network.

Client affinity may be used in the situation when clients have preferences for a certain server. The solution also requires establishing virtual connections between clients and server in a multiple to one fashion (m:1). This is yet another strong advantage of using a label switching technology and building FECs based on the client's source IP. The packets can then be switched to their final destination using MPLS fast switching hardware.

The ISP may wish to provide *different classes of service* to clients, based on service level agreements or other administrative factors. The MPLS approach can provide different FECs for different classes of service. Packets can be labelled

corresponding to the type of the class (e.g. gold, silver or bronze). If servers have different processing performances, the gold-labelled packets can then be switched to the best performing server. Label stacking also can be used to define and identify hierarchical classes of service.

The *load balancing* function performed using MPLS is a key contribution of this thesis and will be described in more detail in section 4.1.2.2.

The first proposal for the use of MPLS for Web routing was presented in a IBM research report [3]. It exploits the deployment of MPLS by mapping application layer information onto layer 2 labels. The technique requires MPLS capable proxys at the client side, which apply appropriate labels to the client requests.

The dispatcher in front of the Web farm (see Fig. 4.5) maintains a table of associations between labels (L_i) and the associated server weight (W_i). The dispatcher will then send a tuple
$< \{L_1, W_1\}, \{L_2, W_2\}...\{L_n, W_n\} >$ to proxy servers situated in front of MPLS ingress nodes using a dedicated signalling protocol.

In Section 4.1 an alternative MPLS approach is proposed and details about its implementation and performance are presented. This approach reduces the load of the dispatcher and the need for a dedicated signalling protocol. It also reduces the complexity of the solution by eliminating the proxy nodes at the client side.

2.7.2 Dimensioning Web clusters

One problem in designing Web clusters is how to dimension the server farm so as to satisfy the customers and to achieve optimal performance. The main trade-off is whether to scale-out (horizontally) by adding new servers to the farm or to scale-up (vertically) by upgrading server capacity [101]. Another trade-off is between cost, performance and reliability. Hence, the main question is how many servers are required to create a cost-effective, reliable architecture that satisfies user requirements.

There are various choices such as to use a large number of low-capacity servers, a lower number of high-capacity servers, or a combination of both. An analytical model using queueing theory, for homogenous Web clusters is presented in [101]. The design considerations for choosing between a large number of low-capacity servers and lower number of high-capacity servers are analyzed based on four criteria: to generate the same average response time, to achieve the same cluster capacity, to have an equal cluster cost and to have the same reliability.

There is an analogy between dimensioning in telephone network and dimensioning Web clusters. Terminology from teletraffic engineering and queuing theory can be used to define the level of service based on requests arrival rate and service time. An ideally load balanced Web cluster can be viewed as a single abstract system consisting of a single queue with an associated *arrival rate (A)* - the average rate at which connections enter the queue. *Service time* (T_S) is the average amount of time that a server needs to process a request. The average *response time* (T) is the sum between the average *service time* and the the average *queuing time* ($T = T_S + T_Q$).

For such a system to be *stable* (so that all jobs will be serviced), the arrival rate must be less than the *service rate* ($1/T_S$). If $A > 1/T_S$ then the system is *unstable* and the queue will grow until eventually, the system will start blocking connections based on the server *utilisation* $U = A \cdot T_S$ which denotes a full server for $U = 1$, an idle server for $U = 0$ and remains between 0 and 1 for any stable system.

The amount of time between phone calls ($1/A$) in a telephone network is random and memoryless and it can characterised by a Poisson process with exponentialy distributed holding times [78]. Such a system is in general modelled using an $M/M/1$ queuing model. Internet traffic however, can be better simulated using heavy-tailed arrival rate distributions such as Weibull or Pareto [58]. But to simplify the analytical model and to exploit the results from teletraffic engineering, often, Web servers are modelled using a Poisson process [101, 139].

A simplified model is also used in Section 4.2 to design a Web cluster that

offers a guaranteed level of service for a class of premium users. The traffic unit *erlang* [78] from teletraffic engineering is used to dimension the server farm for the differentiated service framework proposed in Chapter 4. In this model, a server cannot process more than c requests simultaneously for the required grade of service. The maximum number of concurrent connections for a cluster of n servers is $n \cdot c$.

The requests for the Web cluster can be observed over a busy hour [161] and the total usage time for all servers can be measured (the sum of all service times during that period). Suppose that the interval of time is Δt. Then, the traffic in erlangs is:

$$E = \frac{\sum_{i \in C} T_i}{\Delta t} \tag{2.1}$$

where:

T_i is the service time for connection i and

C is the set of all connections during Δt.

Using Erlang B formulae and an Erlang calculator [148], and knowing the traffic in erlangs, the server can be dimensioned to accept connections with a negotiated blocking probability p or given a server farm, a certain blocking probability can be promised to the clients.

2.8 MPLS

The idea of MPLS originates from two sources: a faster routing mechanism for IP and providing ATM switches with the control and scalability of a *layer3* router.

In the mid 1990s, a few proposals emerged from major networking companies [70] including *Tag Switching, Switching IP Through ATM (SITA), Aggregate Route-based IP Switching (ARIS)*, and the *Cell-Switched Router (CSR)*, details of which are given in Section 3.2.

To develop a standard approach for switching IP, the IETF MPLS working group was established in early 1997. In addition, many Internet-drafts related with the development of MPLS were posted by individual contributors or organisations.

The vendors started to supply MPLS in 1998 and 1999 saw the first MPLS VPN and traffic engineering deployments. However, the first standard tracks appeared only in 2001.

The current status of the MPLS standard as proposed in January 2001 by [133] can be seen at the IETF's *Multiprotocol Label Switching working group* Web site [74].

2.9 Label switching paradigm

The label switching paradigm involves using a short fixed-length label to perform switching decisions. Unlike *longest prefix match* lookup algorithms used by standard IP routing protocols, label switching is based on an exact match and therefore is much faster.

MPLS is a label switching technology. The routers supporting MPLS are referred to as Label Switching Routers (LSRs). Any other router or switch connected to a LSR (ATM switch, IP router) is referred to as non-LSR. An edge router is an LSR connected to a non-LSR. The router by which a packet enters the MPLS cloud is called the ingress LSR, and the one by which it leaves the MPLS cloud is called the egress LSR. The Label Switching Path (LSP) is the route within the cloud followed by a packet, based on its label as seen in Fig. 2.2.

Labels are small locally significant identifiers inserted by the ingress LSR, and removed by the egress LSR. The MPLS label inserted by one router only has significance for the next router which can decide to pop the label, to switch it with another label or to push another label. Labels are used in forwarding decisions to identify not only the destination of the packet but also the specific path for reaching this destination, and to assign a packet to a specific service class.

The advantages of using a label switching architecture are enumerated here. A more complete description can be found in [30].

- **simplicity** -forwarding decisions are based on a short, fixed-length label.
- **speed and delay** -label switching is an efficient solution to the problem of large traffic loads in the Internet by using a faster routing table lookup mechanism. Although fast software and hardware solutions to the longest prefix match problem in IP routers have been found [96], the potential forwarding rate should always be greater using label switching.
- **routing scalability** -label switching offers solutions to the rapid growth of routing tables by allowing a large number of IP addresses to be associated with one or a few labels. The address space in IPv6 is larger than in IPv4, so this advantage will be even more pronounced should the Internet migrate to IPv6.

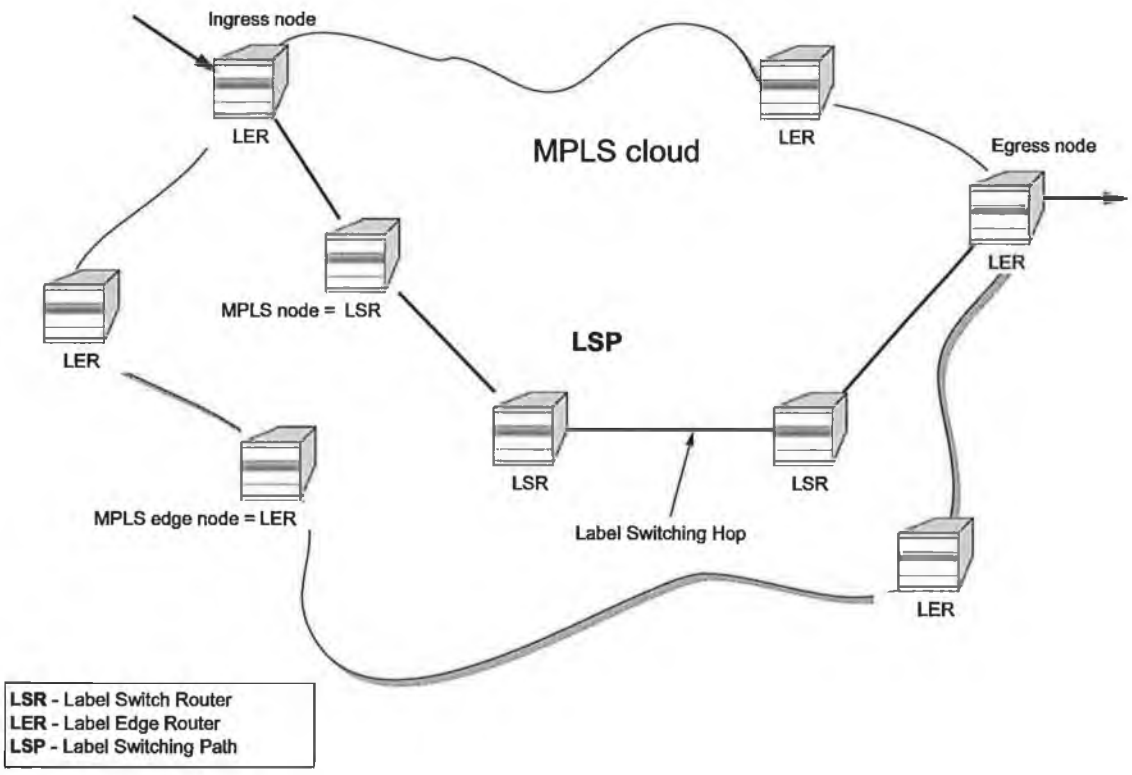


Figure 2.2: *Elements of an MPLS cloud*

- **route control** -label switching allows the router to make routing decisions using various attributes in addition to the IP destination address.

2.10 MPLS and traffic engineering

MPLS plays an important role in engineering the network to provide efficient services to its customers.

RFC 2702 specifies the requirements of traffic engineering over MPLS and describes the basic concepts of MPLS traffic engineering like traffic trunks, traffic flows and LSPs [20]. The advantages of MPLS for traffic engineering include:

- label switches are not limited to conventional IP forwarding by conventional IP-based routing protocols;
- traffic trunks can be mapped onto label switched paths;
- QoS requirements can be mapped to flows and traffic trunks;
- MPLS permits address aggregation and disaggregation (unlike IP forwarding which permits only aggregation);
- constraint-based routing is easy to implement;
- MPLS hardware offers QoS capabilities resembling those of ATM, but at lower cost.

The comparison between ATM and MPLS comes from the analogy between ATM's virtual circuits and MPLS's LSPs. Both use an overlay model for switching connectionless IP packets in a connection-oriented manner.

Other the reasons why MPLS is preferred as an underlying technology for traffic engineering include:

- MPLS is more scalable than IP over ATM and there is no need for per-flow state in the core;

- There is no need for special packets for connection setup;
- The forwarding procedure is simple enough to allow a straightforward hardware implementation [127];
- The MPLS label stack facilitate fast rerouting and restoration mechanisms (some of these mechanisms will be discussed in Section 3.4.3).

However, the main advantage of using MPLS for traffic engineering is the ability to use paths, other than the shortest one selected by the routing protocol to achieve optimal network utilisation. In an MPLS environment, this can be achieved by moving the traffic away from the over-congested shortest paths using explicit LSP tunnels.

Moreover, the simple forwarding procedure is decoupled from the control component. Thus, new routing functions can readily be deployed without disturbing the forwarding paradigm. This means that it is not necessary to re-optimise forwarding performance (by modifying either hardware or software) as new routing functionality is added.

The authors of [157] show another advantage of MPLS for traffic engineering: the choice of label granularity. Some of the choices are enumerated as follows:

- *Egress router* - coarsest granularity
- *IP Prefix* - medium granularity
- *Application flow* - finest granularity

In [33] Boyle *et al.* enumerate some of the scenarios where MPLS-based traffic engineering capabilities are applicable in service provider environments:

Avoidance of Congested Resources. One of the objectives of Internet traffic engineering is to prevent network congestion. This may occur as a result of many factors such as equipment failure, traffic bursts or inefficient bandwidth management. MPLS can be used to overcome these problems by setting up explicit LSP tunnels (which need not follow the paths determined by datagram

routing protocols) to route a subset of the traffic to less congested paths. This approach is available when parts of the network are congested while other parts are under-utilised. If congestion occurs due to equipment failure MPLS protection and restoration (see Section 3.4.3) can be used to map the LSPs to alternative paths with spare capacity.

Resource utilisation in network topologies with parallel links. MPLS traffic engineering methods can be used to distribute the traffic across parallel links such as NxOC-n (N parallel OC-3/12/48 links). LSP bandwidth parameters can be used to control the proportion of traffic traversing each parallel link. Moreover, LSP tunnels can be mapped to physical links based on affinities (administrative or users' preferences).

Implementing routing policies using affinities [33]. In practice there are scenarios when subclasses of traffic have to be restricted to a subset of the network. This can be used to achieve network engineering objectives or business policies.

Virtual Private Networks (VPNs) are a concrete example of policing the traffic using affinities. The traffic within the VPN should be restricted to a certain network subset, whereas the external traffic has to be kept out of the VPN. Another example is to force some types of traffic to traverse only links with given capacity. This is to reserve the high capacity links for mission-critical application and restrict the less important traffic to lower capacity resources.

Protection, restoration and re-optimisation. Hardware failures occur within the live network. Preemptive measures such as as the association of a primary LSP to a set of secondary hot-standby LSPs can reduce packet loss during the outage. Fast-reroute mechanisms should also be used to reduce the amount of packets lost during restoration. Additionally, it may be desirable to calculate a new set of paths for LSPs to optimise the performance over the residual topology [88].

The applicability of MPLS for traffic engineering includes but is not limited to the issues mentioned above. Some other MPLS based approaches to traffic

engineering will be described in the following sections.

2.11 MPLS in the global QoS picture

QoS is not a single layer issue; it affects all layers. Not only does this mean that QoS is to be performed at multiple layers but it follows that many QoS technologies stretch across multiple OSI layers. An example is Intserv which includes a packet classifier, a packet scheduler and admission control plus a reservation protocol such as RSVP. Another example is the use of active networks to manage QoS, a solution which stretches from the data link to the application layer. Other technologies such as MPLS (layer 2.5) insert new layers in the stack to supplement the layers of the OSI model.

MPLS not only modifies the protocol stack but its deployment will trigger modifications in other protocols and technologies from all the OSI layers. A global picture is presented in Fig. 2.3 with more details being given in Chapter 3. Some elements of this architecture are original to this thesis and will be discussed in Chapter 4.

The numerous proposed mechanisms for QoS developed for all OSI layers, show that if a global end-to-end QoS scheme is to be deployed in the Internet, the protocol stack will be modified at least in the core of the Internet. However, the end user will probably still be using IP since this is the foundation of the Internet, although it may in the future be an updated protocol version (IPv6).

2.12 Concluding remarks

QoS and traffic engineering were once topics of interest only in telecommunication networks. Gradually QoS features were introduced into the Internet as new applications with more demanding requirements were deployed.

Various IP based QoS schemes were developed including (but not limited to)

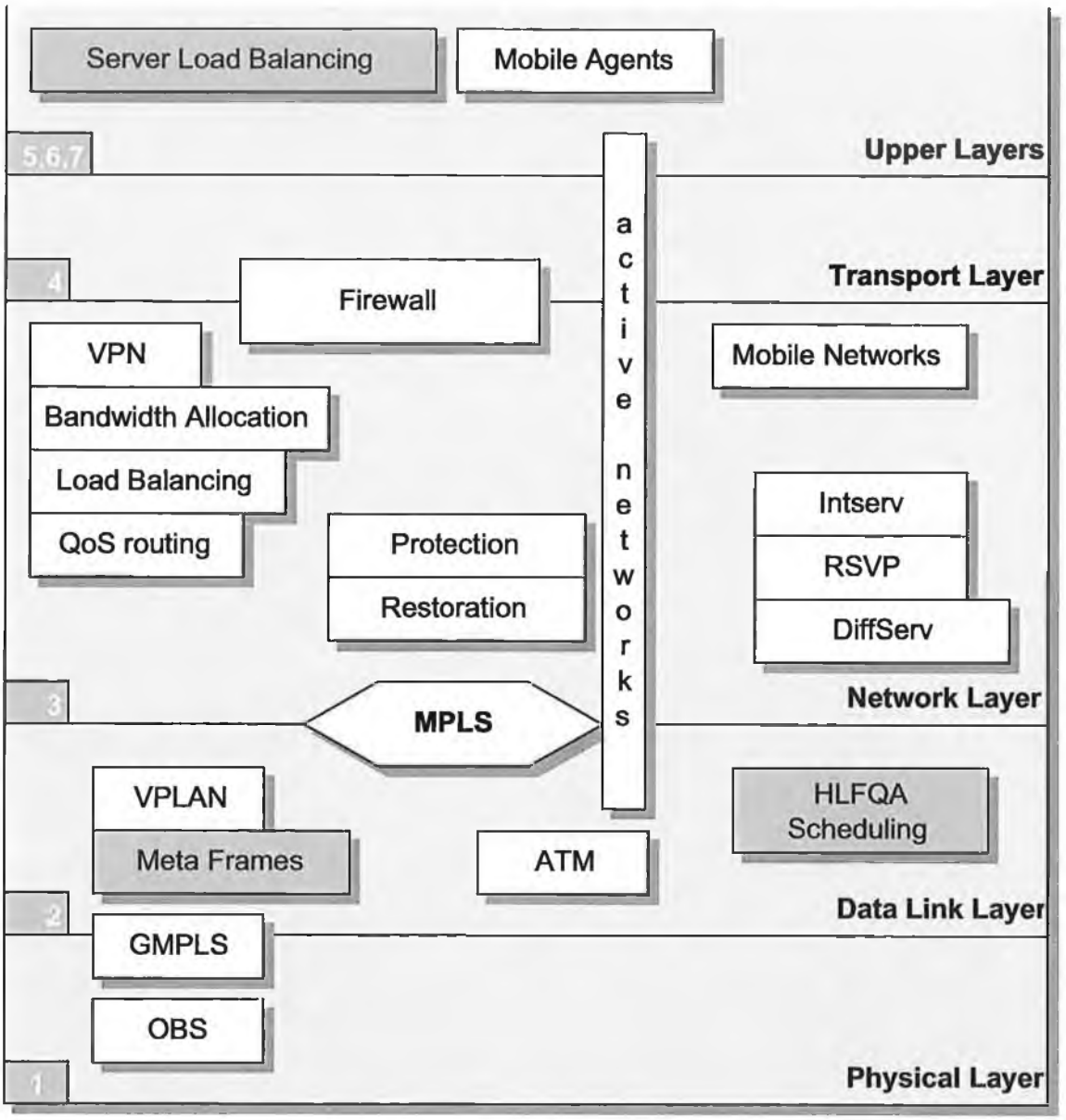


Figure 2.3: *The big QoS picture*

Intserv, Diffserv, RSVP and MPLS. The Internet community also developed a set of requirements for Internet traffic engineering. There are many institutions involved in developing QoS and traffic engineering mechanisms, some of which are already deployed in the Internet. An overview of the main components of Internet QoS support was presented in this chapter.

Despite these advances, the main goal of implementing a universally accepted end-to-end QoS scheme in the Internet has yet to be achieved. Nevertheless, a giant step forward has been the deployment of MPLS on a large scale. MPLS can provide a homogenous forwarding plane for the Internet and allows complex QoS routing and signalling protocols to be implemented. This should allow the next generation of Internet services to be built around MPLS just as current services are built around the IPv4 protocol.

Wider deployment of MPLS (as to the edges of the Internet) will simplify the task of providing end-to-end QoS but will bring its own challenges. The remainder of this thesis comprises an evaluation of the MPLS protocol as a mechanism for providing QoS and traffic engineering capabilities in the Internet, and a description of some new mechanisms to enhance the user experience in an Internet dominated by MPLS.

CHAPTER 3

MPLS

The success of ATM in deploying QoS in IP networks showed the advantages of a label switching technology for the Internet. There were numerous efforts to develop a label switching technology better suited to the needs of the Internet than is ATM, some of which are presented in Section 3.2. These efforts converged in the development of a Multiprotocol Label Switching (MPLS) protocol. MPLS was standardised by IETF and the architecture is presented in RFC 3031 [133]. Despite its name, MPLS is primarily intended to inter-operate with IP.

This chapter will discuss the main characteristics of MPLS that give this protocol such potential for traffic engineering and Internet QoS. Then, I will give an overview of the role of MPLS in the global QoS picture and how it interacts with other tools used to provide QoS in the Internet.

3.1 The MPLS label switching paradigm

Label switching enabled routers determine the next hop in the routing process using small locally significant identifiers that are encoded within the packets. Identifiers are often referred to as labels, tags, circuit identifiers, etc. Labels have

local significance because each router along the path makes the routing decision based on the current (local) label, then switches the label with another value that only has significance in the next router.

Packets tagged with the same label belong to the same Forwarding Equivalence Class (FEC) and will all follow the same path through the label switching network. The procedure of establishing FECs and tagging the packets is not the responsibility of label switching. This is because in label switching networks control and forwarding planes are separated. Therefore, label switching networks have some interesting properties especially suited for traffic engineering.

The main characteristics of MPLS as a label switching paradigm are:

- Forwarding can be done by switches that are not capable of analyzing layer 3 headers at adequate speed (such as software based routers) or not capable of analyzing layer 3 headers at all.
- MPLS integrates with existing layer 2 switching technologies such as ATM and Frame Relay.
- Label based forwarding can take into account more information than is stored in the layer 3 header such as:
  - Incoming interface
  - Ingress router
  - Upper layer information
- Routing and forwarding separation allows for more complex routing strategies to be used, while keeping forwarding unchanged. Forwarding can be considered as a separate layer that does not have to change when routing algorithms are changed or upgraded. Moreover, some level of portability is provided since label switching does not depend on higher level routing and signalling protocols.

- In label switching, as opposed to datagram routing, it is possible to explicitly specify some or all the nodes along the path which is especially useful for:
  - Traffic engineering in order to divert traffic from congested links.
  - Policing in order to deny some traffic access to particular parts of the network.
- Additional services such as VPN are easier to provide when using label switching.

3.2 Other label switching technologies

MPLS is not the only label switching technology available. Layer 2 technologies such as ATM and Frame Relay may also be regarded as label switching mechanisms.

Before MPLS standardisation, various label switching based approaches were proposed to improve forwarding performance [157]. These proposals are called *IP switching techniques*. Their common characteristic is a multi-layer label-swapping mechanism implemented by:

- providing semantics to bind labels to specific streams of packets;
- using a protocol to distribute binding information among routers;
- forwarding packets from the incoming interface to the outgoing interface based solely on the label information, rather than the destination IP address.

Forwarding can be performed in hardware by the switch fabric of the router, or it can be performed in software by indexing the label of the incoming packet into a label forwarding information base to find out the corresponding outgoing interface. The result is a router with the speed of a link-layer (layer-2) switch and the flexibility of a network-layer (layer-3) router.

The main IP Switching protocols are:

1. *Toshiba's Cell Switch Router (CSR)*
2. *Ipsilon's IP Switching*
3. *IBM's Aggregate Route-Based IP Switching (ARIS)*
4. *Cisco's Tag Switching*
5. *Multiprotocol Label Switching*

Cell Switch Router (CSR). The CSR proposal by Toshiba [82] is one of the first attempts to implement IP switching. Essential to the proposal is the notion of a *"cell switch router"* (CSR), which is a device that interconnects logical IP subnetworks (LISs) and is capable of both IP forwarding and ATM cell switching. Within an LIS, layer 3 connectivity between nodes is provided by either LANE or classical IP over ATM. The address resolution is performed by ATMARP [89] and InATMARP [37] servers. Connectivity that spans multiple LISs is provided via CSRs that interconnect them. The CSR identifies individual traffic flows and binds each flow to a virtual circuit (VC). When both an incoming VC and an outgoing VC (or VCs) are dedicated to the same IP flow(s), those VCs can be concatenated at the CSR (ATM cut-through) to constitute a Bypass-pipe.

A signalling protocol is needed to establish new VPI/VCI values for specific flows of IP packets arriving at an input interface. Then, these special values could be bound to the corresponding VPI/VCI values at an output interface. In this way a cell arriving with one VPI/VCI value could be switched at the ATM layer to the appropriate output interface and could be assigned the correct VPI/VCI for forwarding to the next hop router or end station.

Label binding can be driven by either RSVP messages or data traffic. Distribution and maintenance of label binding information is performed via a separate protocol: the flow attribute notification protocol (FANP) [110].

Ipsilon's IP Switching IP Switching is a technology proposed by Ipsilon [115] and became popular in the mid 1990s. It is very similar in many respects to Toshiba's *Cell Switch Router*.

In Ipsilon's IP Switching proposal, the main element is the IP Switch. An IP Switch is made by taking the hardware of an ATM switch and removing the software resident in the control processor above AAL-5. Therefore, signalling, existing routing protocols, LAN emulation servers and address resolution servers are removed. A simple low-level control protocol, called the *general switch management protocol* (GSMP) [114], replaces the ATM software. The IP switch controller is a processor running standard IP router software with GSMP extensions that allow it to make use of the switching hardware. You can see the structure of the IP switch, as well as an example of an IP Switching network, in Fig. 3.1.

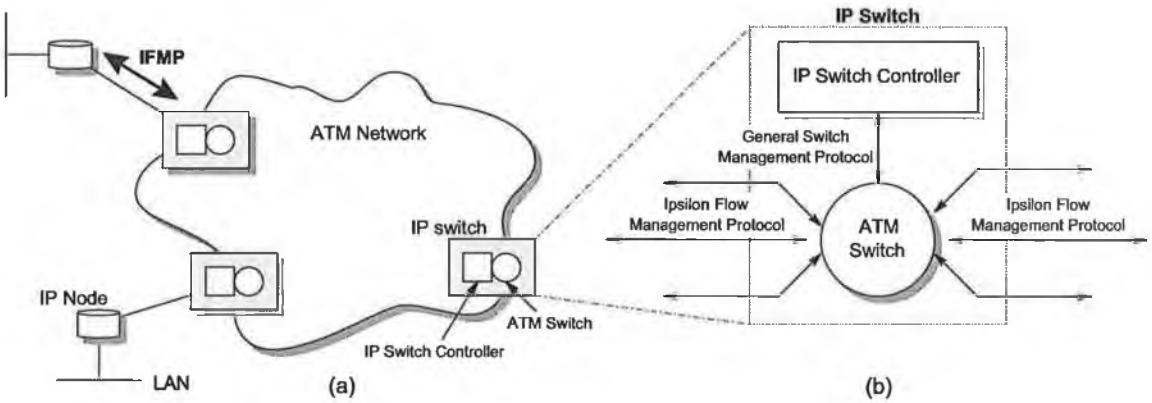


Figure 3.1: *(a) An IP Switching Network (b) The structure of an IP Switch*

Previous switching proposals relied on the use of native ATM signalling to establish at least default ATM virtual circuits. Ipsilon Networks abandoned the standard ATM signalling and introduced a new signalling protocol, which associates IP flows with ATM virtual channels. This protocol was called the Ipsilon Flow Management Protocol (IFMP) [113].

The Ipsilon approach had the advantage over Toshiba's CSR proposal, of being able to reduce the default-forwarding load. Unlike CSR, however, IFMP depends to a large degree on flow detection at each IP routing node in a network

composed of IFMP-participating IP routers. This could significantly increase IP packet processing overhead in the default-forwarding mode.

Aggregate Route-Based IP Switching (ARIS). ARIS was introduced by IBM, though it was also under development as an open IETF standard [156]. It was intended for use with switched network technologies, whether ATM, frame relay, or LAN switches and permits layer 2 switching to be used for IP datagram forwarding.

The goal of ARIS is to improve the aggregate throughput of IP and other Network Layer protocols by switching datagrams at wire speed. Thus, it proposes *VC merging*, meaning that packets arriving with different VPI/VCI values can be forwarded with the same VPI/VCI value (merged).

ARIS also proposes the *route-based* paradigm for assigning the labels. A route in this sense is rather like a multicast distribution tree, rooted at the egress point, and traversed in reverse. The egress point is specified by an *"egress identifier"*, which may be one of:

- IP destination prefix
- egress router IP address
- OSPF Router ID
- multicast (source, group) pair
- multicast (ingress-of-source, group) pair

The main element in an ARIS network is the *Integrated Switched Router* (ISR). An ARIS network (a network comprised of ARIS capable ISRs) establishes switched paths to egress points. The egress points are established using the standard layer 3 routing protocols such as OSPF and/or BGP. It is the responsibility of the egress ISR to initiate the path setup by sending messages (*establish messages*) to upstream

neighbours, as can be seen in Fig. 3.2. These neighbours forward *establish messages* upstream in reverse path multicast style, so eventually all ARIS ISRs have switched paths to every egress ISR.

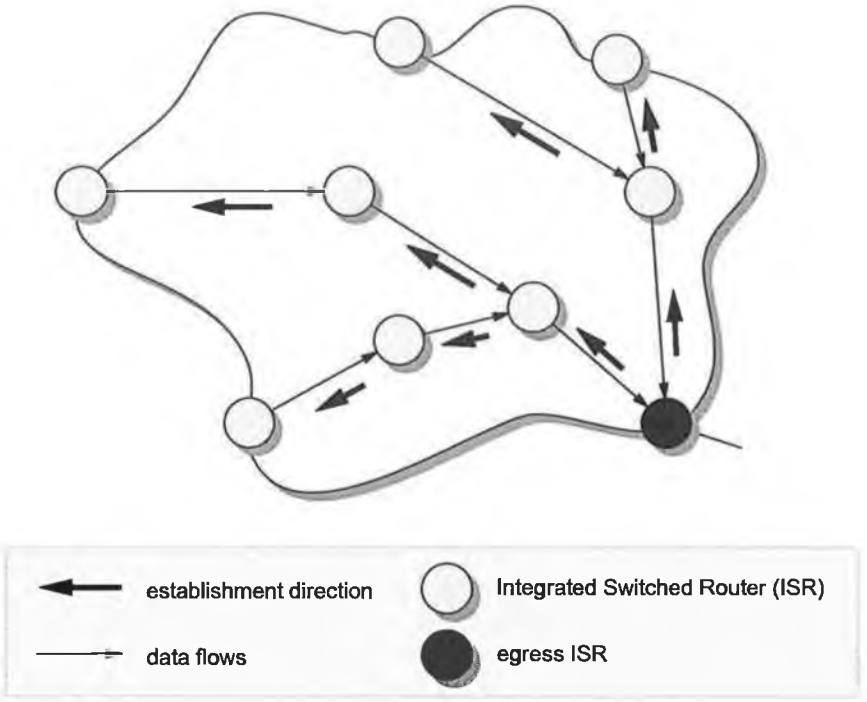


Figure 3.2: *IBM ARIS Switched Paths*

One important feature of ARIS is that switched paths are guaranteed to be loop-free, despite using standard IP routing protocols. Each ISR appends its own ISR ID to the *establish messages* it forwards, so it can then determine whether an *establish message* has passed its way before. If so, it means that there is a loop and it refuses to continue the path.

Another aspect of ARIS is that path information is soft state, meaning that it is maintained only for as long as ARIS messages are sent within a time frame. *KeepAlive* messages are used to maintain state in the absence of other ARIS messages.

ARIS is also the first technology to introduce the term *label* and the concept of a *stack of labels*.

Cisco's Tag Switching. With its Tag Switching architecture, Cisco Systems also wished to address a key performance issue of IP routers, i.e. the *longest-prefix-match* lookup of a packet's destination address. This architecture was intended to be applicable across all nodes in a heterogeneous network, whether layer 3 routers or layer 2 switches. The architecture is outlined in [127].

When a packet enters a tag switching capable "cloud", a short *tag* is attached to it. This identifier is an index into a *Tag Information Base* (TIB) residing in each Tag-Switching capable router. *Tags* are used much like ATM VPI/VCI fields. An interior tag switching router can implement a very fast, hardware-based, layer 2-like switching capability for those packets that carry these tags. However, a software upgrade to the router's operating system confers some of the benefits of the quicker lookup, without modifying the switching hardware.

In ATM switches the tag is likely to be mapped directly to cell VPI/VCI fields. For conventional routers, the *tag* is embedded as an additional protocol header, either between the Network and Data Link Layer headers, or within the Data Link Layer header. TIB associates each incoming *tag* to an outgoing *tag*, an outgoing interface and layer 2 information. *Tags* are swapped at each switch point, as in native ATM. Routing information resides in a Forwarding Information Base (FIB), which is constructed using standard routing protocols (e.g., OSPF, BGP). Tag-Switching capable devices exchange FIB information using the Tag Distribution Protocol (TDP).

Tag enabled devices perform fast layer 2 switching rather than slower Network Layer forwarding as routers do. The tags may be somewhat more complex than ATM VPI/VCI headers as there can be a stack of tags. This allows tunnelling through enclosed domains; by using tag switches as ingress/egress routers, only the border switch-routers need to maintain exterior routing information. Switches within the domain need only to know about interior routing. Packets tunnelled through the domain will have exterior routing information pushed onto the tag stack at the ingress switch and popped off at the egress

switch.

Tag Switching is similar to ARIS in the sense that both approaches include proposals for signalling the values to be used by peers in implementing the switching paradigm. Both rely on the use of topology information from routing protocols to establish the paths to be used in packet switching and both have the concept of a *stack of tags*. In addition to this, the tag-switching proposal provides alternatives in the distribution of switching information, unlike the CSR and IP-Switching proposals.

While there had been an earlier attempt to establish a tag-switching forum, with the advent of Tag Switching, ARIS and other proposals, it was clear that the possibility of developing a standard packet switching approach needed to be considered. Hence an IETF working group was formed for what would later come to be called Multiprotocol Label Switching (MPLS).

The convergence of IP switching technologies into MPLS. In 1996 IETF[1] started to develop an IP switching technology which should contain the best features from the four previous proposals. In December 1996 the IETF MPLS Working Group[2] was formed. Since then it has been responsible for standardising a base technology for using label switching and for the implementation of label-switched paths over various link-level technologies, such as Packet-over-Sonet, Frame Relay, ATM, and LAN technologies (e.g., all forms of Ethernet, and Token Ring). Subsequently, it has produced a number of *Requests for Comments* (RFCs) that define the basic MPLS architecture [133] and encapsulations [132], the Label Distribution Protocol (LDP) [7, 152], the definitions for how MPLS runs over ATM [50] and Frame Relay [47], and the requirements for traffic engineering over MPLS [20]. The original motivation for MPLS, and its IP switching precursors, was to improve forwarding speed by reducing the number of IP table lookups. Hardware techniques for fast longest-prefix-match lookups [40, 54] have since addressed

[1]See *http://www.ietf.org*
[2]See *http://www.ietf.org/html.charters/mpls-charter.html*

this bottleneck in IP packet processing, but MPLS is now favoured for its traffic engineering capabilities.

3.3 The MPLS architecture

3.3.1 Label encapsulation

The key element of the MPLS architecture is the MPLS label. There are two types of label encoding: *native layer two encoding* for technologies such as ATM or Frame Relay and *generic MPLS encapsulation* for Ethernet and packet over SONET networks.

The IETF standard for generic MPLS encapsulation [132] requires that labels must be inserted as a "shim header" between the link layer and network layer headers as depicted in Fig. 3.3.

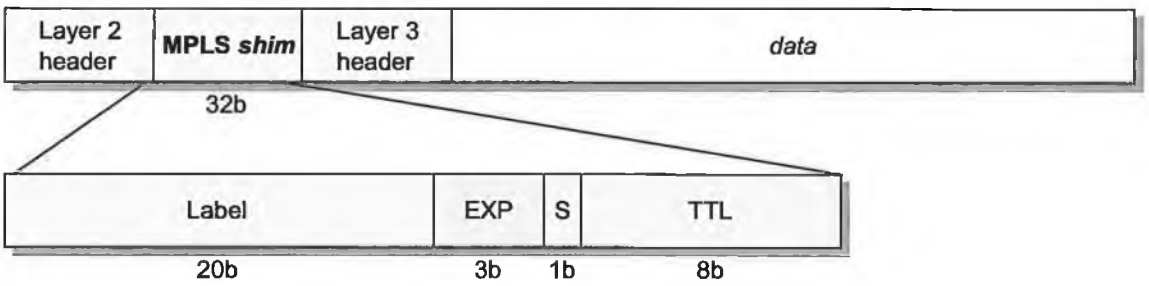


Figure 3.3: *The MPLS "shim header"*

The generic MPLS encapsulation allows for multiple labels to be encoded as a stack of MPLS shims. In the 32 bit MPLS shim entry, the 20 bit label is followed by 3 experimental bits, a bottom of stack bit and the 8 bit Time to Live (TTL) entry. The label value is used in the forwarding process and it is switched with another value at each hop. The experimental bits could be used for example to map Differentiated Services Code Point (DSCP) entries into MPLS labels. The TTL field is used for loop prevention and to allow the TTL field of the upper layer protocols to be updated to reflect the transition through the MPLS cloud.

Native layer 2 label swapping technologies such as ATM and Frame Relay cannot accommodate the MPLS stack and therefore the labels must be encoded in the link layer information. These layer two protocols are label switching based technologies. Therefore, MPLS can use the circuit identifier space to encode the label. This is the virtual path identifier/virtual circuit identifier (VPI/VCI) pair for ATM and the Data Link Connection Identifier (DLCI) for Frame Relay.

3.3.2 MPLS label stack

In generic MPLS label encapsulation, multiple label shims can be inserted between the layer 2 and layer 3 headers as shown in Fig. 3.4. Labels are processed in a last-in first-out stack order. Hence, the packet is always processed based on the label at the top of the stack.

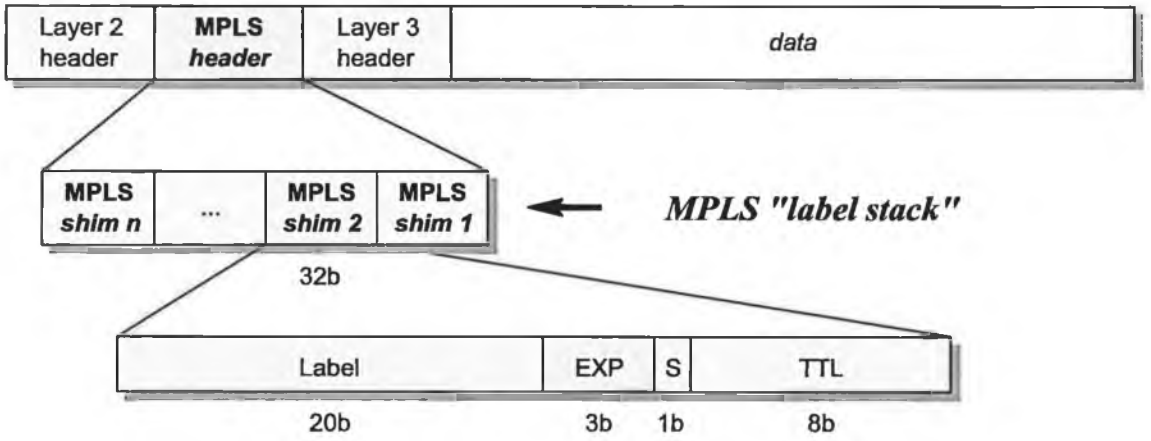


Figure 3.4: *MPLS label stack entry*

The MPLS label stack allows an arbitrary number of nested LSP tunnels to be created [133]. This is especially useful in hierarchical MPLS networks for greater routing scalability. The MPLS stack is also useful for traffic aggregation to reduce the number of micro-flows and the number of entries in routing/forwarding tables in the core of the network.

ISPs can use the MPLS label stack to provide layer2 and layer3 VPNs [131]. Therefore, the routing equipment industry added MPLS VPN [45] and Virtual

private LAN service (VPLS) [118] to their arsenal. The benefits of providing MPLS/BGP VPN services include (but are not limited to) [131]:

- layer 2 independent VPNs;
- different customers can have overlapping address spaces;
- less management for both customers and providers since there is no need for *virtual backbones* for each customer.

Another area of use for the MPLS label stack is in path protection. In the event of a failure the protected paths can be tunnelled through the bypass LSP by pushing another label onto the stack [72]. This approach has the advantage of increasing the scalability and reducing resource utilisation by using a single LSP (called the bypass tunnel) to backup an entire set of protected LSPs. Further details about MPLS protection will be presented in Section 3.4.3.

Our MPLS based solutions to Web switching and to increase the average packet size in the Internet (presented in Chapter 4) also take advantage of the MPLS stacking capabilities.

The main disadvantage of using a label stack is the traffic overhead introduced by large MPLS stacks. Furthermore, this traffic overhead might also give rise to additional fragmentation and reassembly operations [132].

3.3.3 Forwarding tables

The main principle of label forwarding is to use a short label in the routing decision in order to simplify the process of choosing a next hop and output interface. The label is used as an index in the forwarding tables. However, routers at the boundary with IP networks have to use a traditional longest prefix match to classify incoming IP packets (at the MPLS ingress) and route them to the next hop (at the egress).

The tables involved in routing and forwarding decisions inside the MPLS cloud are described below.

When an unlabelled packet arrives at an ingress node its FEC is determined and an output label is inserted. Packet assignment to FECs and label distribution for that particular FEC is the responsibility of MPLS control plane. *FEC-to-NHLFE (FTN)* maps each FEC to a set of *Next Hop Label Forwarding Entries (NHLFE).*

NHLFE are used when a LSR forwards a labelled packet. Each NHLFE contains:

- the next hop for that entry
- the MPLS operation to be performed on the packet's label stack such as:
  - switch the current label with a new specified label
  - pop the top label from the stack
  - switch the current label with another label then push one or more labels onto the label stack.
- data link encapsulation, the way to encode the label stack and any other information that is needed to send out the MPLS packet.

Inside the MPLS network, when labelled packets arrive at an LSR, the *Incoming Label Map (ILM)* is used to map each incoming label to a set of NHLFEs.

3.3.4 MPLS routing and signalling

Another feature of MPLS is its decoupling of the forwarding plane from the control plane. This allows for complex, QoS aware routing and signalling technologies to be deployed without any change in the forwarding plane. Hence, there are three abstract layers in the MPLS framework as depicted in Fig 3.5.

Routing and signalling are not always clearly separated. This is because label information can be piggybacked by routing protocols such as the Border Gateway

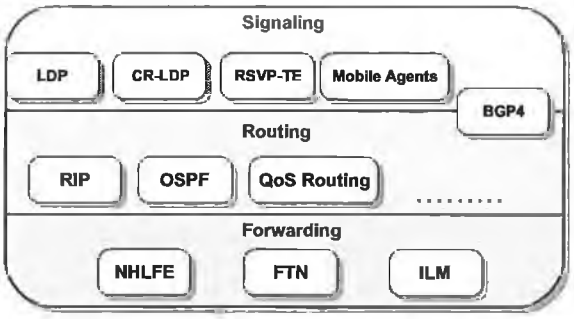


Figure 3.5: *MPLS planes*

Protocol (BGP) [130]. BGP is capable of distributing label mapping information for a particular route in the same BGP update message used to carry the route information for it.

Routing in MPLS networks can be performed either by traditional routing protocols or by some QoS aware routing scheme. Since routing is decoupled from forwarding, routing information can be carried by virtually any routing protocol. However, the establishment of label switched paths requires label allocation which is the responsibility of signalling protocols. Label distribution protocols for MPLS are discussed below.

Signalling is a very important component of the MPLS framework since it is responsible for label distribution among LSRs. The LSPs for a small MPLS domain can be configured manually by network administrators. However, this is not desirable for large networks with dynamic traffic behaviour.

Signalling protocols are also responsible for creating and maintaining LSPs for VPNs, explicit traffic engineered LSPs and constraint-based LSPs. Hence, new protocols have been proposed for MPLS signalling and existing protocols have been modified to be able to carry label information.

The Label Distribution Protocol (LDP) [7, 152] emerged as a basic signalling protocol for MPLS. LDP capable routers use a discovery mechanism to identify

potential LDP peers. Then LSRs may exchange label bindings for FECs maintained by the underlying IP routing protocols. LDP uses TCP to ensure reliable delivery of LDP session messages. To protect against spoofed TCP segments LDP defines the optional use of the TCP MD5 Signature Option.

LDP supports both downstream unsolicited and downstream on-demand label bindings in order to create LSPs throughout the network. The major shortcoming of LDP is that LSPs created using LDP will follow the shortest paths (determined by the existing IP routing protocols) from the MPLS ingress node to the egress node for that FEC. Extensions to the standard LDP must be developed in order to allow constraint-based or traffic engineered explicit paths and resource reservation.

Mechanisms and TLVs (Type/Length/Value) for constraint-Based LSP Setup using LDP (CR-LDP) are presented in the IETF document RFC 3212 [80]. The specification proposes an end-to-end setup mechanism to support CR-LSPs (constraint-based routed Label Switched Paths) initiated by the ingress LSR. Mechanisms are also specified to provide means for the reservation of resources and to change the reservation parameters using LDP.

CR-LDP is a hard-state signalling protocol delivering messages reliably using TCP. Its suggested applications are presented in RFC 3213 [13]. However, IETF has decided that CR-LDP (RFC 3212) will never be progressed beyond its current Proposed Standard status, that is, it will never become a full standard and new work on CR-LDP is strongly discouraged by the IETF. This is mainly because another MPLS signalling protocol was adopted by major switch vendors and that is RSVP-TE.

IETF's RFC 3209 [17] describes RSVP-TE: Extensions to RSVP for LSP Tunnels. Resource ReSerVation Protocol (RSVP) is a pioneer of the receiver initiated-reservation paradigm. The advantages of this paradigm are especially apparent for multicast applications especially for video/audio conferences [168].

RSVP was initially developed to perform resource reservation using Intserv

parameters to describe data flows. A basic introduction to RSVP was presented in Section 2.4.5. However this use of RSVP was unsuccessful because of Intserv's scalability issue [160].

RSVP was revived as RSVP-TE following the emergence of MPLS and the need for a constraint based signalling protocol. The key application of RSVP-TE with MPLS is traffic engineering. RSVP-TE is useful for establishing and maintaining explicit routed LSPs in order to force the traffic through other routes than those given by routing protocols. LSP tunnels can also be instantiated for measurement purposes (to monitor traffic statistics between two LSPs) and for routing control purposes (explicit routing and load balancing) [19].

The RSVP-TE specification solves the Intserv scaling issue with the original RSVP protocol. This is because state is not required for each micro-flow since flows are aggregated into LSPs. It seems therefore, that RSVP-TE is emerging as the universal signalling protocol for MPLS traffic engineering. Moreover, extensions to RSVP-TE are now being developed in order to provide signalling in GMPLS networks [26, 92].

Border Gateway Protocol (BGP-4) is the fourth version of an inter-autonomous system routing protocol specified by IETF in [128, 129]. BGP carries sufficient network information to provide reachability loop detection between autonomous systems. Moreover when BGP is used to distribute a particular route, it can also be used to distribute a Multiprotocol Label Switching (MPLS) label which is mapped to that route. [130].

BGP is especially useful as a means to increase the scalability of MPLS and to reduce the label distribution complexity for LSRs that are also BGP peers. Therefore no additional label distribution protocol is needed to distribute label bindings for BGP routes.

3.3.5 Service differentiation in MPLS networks

One of the key concepts in MPLS is the tunnelling of higher layer protocols through LSPs. This makes the information above MPLS forwarding layer invisible for LSRs. However, for traffic engineering, routers need a way to differentiate particular sets of traffic in order to provide different treatment and priorities.

In MPLS this can be done at two granularity levels. At one level, packets can be differentiated by the label they are carrying (i.e. the LSP to which they belong). Since an LSP is usually an aggregate of traffic flows the label only provides coarse granularity in distinguishing flows.

A finer granularity can be achieved by taking advantage of the three experimental bits in the MPLS shim. When a label is pushed onto the MPLS stack the experimental bits can be used to carry priority information provided the routers along the path can interpret the values in the experimental field and are able to treat the flows appropriately.

The most probable use of MPLS service differentiation is in conjunction with Diffserv where the differentiated services field in the IP header might be mapped to the experimental bits field. More details about MPLS and Diffserv are presented in Section 3.4.4.

3.4 QoS and traffic engineering topics

An increasing number of QoS technologies are available that make use of the simplicity and flexibility of the MPLS forwarding plane. Although some of them were invented before MPLS and can be used within traditional IP networks, they are particular suited for use in MPLS networks.

Such approaches and their connection with MPLS will be presented in this section.

3.4.1 The MPLS traffic engineering problem

Analytical models for traffic engineering problems in MPLS networks are presented in [67]. The authors identified four distinct traffic engineering problems:

The connection admission control problem involves determining whether a connection or demand request can be admitted or not.

The constraint based routing problem determines the optimal placement of the demands through a given network given a set of demands or connections.

The rerouting problem occurs due to the failure of network elements.

The capacity planning problem deals, on a less frequent basis, with the determination of the optimal network topology to cater for a given set of demands.

However, all the above mentioned traffic engineering problems are NP-complete and cannot be solved by any known polynomial time algorithm [67]. Therefore, the development of heuristics, approximation algorithms and exact solutions for simplified versions are required.

3.4.2 Generalised MPLS (GMPLS)

MPLS is a protocol that uses labels to switch IP packets (or ATM cells) in IP based networks. IETF has developed an extended scheme that generalises the switching domain to time (TDM/SONET), wavelength (DWDM) or space (OCX). The objective is to develop forwarding and control planes to dynamically provision resources and to provide network survivability using protection and restoration techniques for future terabit networks [95].

The MPLS framework was extended to include LSRs whose forwarding plane recognise time slots, wavelengths and physical ports. The control plane was extended as well so that GMPLS can provide the same traffic engineering capabilities as MPLS does.

3.4.3 Protection and recovery

A traffic engineered network must be able to protect itself and the information it carries in the event of node or link failures. The traffic must be rerouted over alternative paths using the residual bandwidth (bandwidth not used by current traffic flows), in a way that will not overload the new paths and impair the performance of already existing traffic flows. The current routing algorithms have the advantage of being robust and survivable. However, they can require a significant amount of time to recover from a failure [155].

The MPLS approach allows significant improvement in the protection switching time compared to legacy IP networks. This is mainly because of the MPLS capability to preestablish explicit LSPs as well as backup LSPs. The goal is to achieve fast recovery times comparable with SONET's 50 ms recovery time [24].

RFC 3469 [137] specifies a framework for MPLS based recovery as a set of requirements for LSRs to support fault detection, fault notification, and fault recovery mechanisms.

Based on how the LSP is repaired, two types of repairs can occur: *local repair* activated by the LSR that has detected the failure and *global repair* which is activated on an end-to-end basis by the ingress and egress LSRs, regardless of where the failure occurred. The MPLS recovery models can also be classified in two categories, *rerouting* and *protection switching* [4, 62].

Rerouting is the process that occurs after a failure and reroutes the LSPs away from the problem. It uses up-to-date information to temporarily switch the traffic until the fault is repaired. However, it introduces delay since it takes time to compute the new paths for rerouted LSPs. Because of this problem, most of the schemes consider local repair and the intent is to protect against a single link failure, since this is faster than full path recovery.

Protection is the process of provisioning backup LSPs that can be used in the event of failure. Protection switching is the preferred approach for global repair

due to the setup overhead and delay that it is not tolerable in MPLS rerouting.

Resources may also be reserved for the backup LSPs but this will make the network under-utilised since bandwidth that is reserved for backup LSPs cannot be used by active LSPs. An alternative is to create backup LSPs without reserving resources. Hence, in the event of a failure the backup LSPs cannot receive the same guarantees as the protected LSPs. Moreover, the LSPs that now share the link with the backup LSPs will be affected by the new traffic flows.

One algorithm that deals with the problem of resource reservation for protected LSPs was presented in [105]. The algorithm enables very efficient bandwidth reservation for single fault protection. Another economical alternative is presented in [72]. In the event of a failure the protected paths can be tunnelled through the bypass LSP by pushing another label onto the stack. This approach has the advantage of increasing the scalability and reducing resource utilisation by using a single LSP (called the bypass tunnel) to backup an entire set of protected LSPs.

3.4.4 MPLS and differentiated services

The idea of combining MPLS and Diffserv to perform traffic engineering in the Internet, appeared first in the early stages of MPLS standardisation [9, 125]

The initial approaches proposed to use MPLS and Diffserv over ATM networks. Consequently, an additional signalling protocol to distribute the correlation between MPLS label values and the DSCP field was needed [9]. This also meant that for each egress router a separate LSP would be needed for each DSCP value used. In this approach when LSP priorities are inferred from the label value the LSP is called *L-LSP*.

An alternative was to use the VPI (Virtual Path Identifier) and part of the VCI (Virtual Circuit Identifier) of the ATM cell header to encode the label and the remaining eight least significant bits to map the DSCP field.

In [125], the authors propose using a Centralised Resource Manager (CRM) to keep track of available network resources and to accept connection requests by setting up explicit LSPs. The CRM is the primary contact with the customer in order to create and keep track of existing TCS (Traffic Conditioning Specifications). In this approach MPLS is used to pin a particular route for a flow, while Diffserv specifies the treatment for data packets.

A dynamic link-colouring algorithm is proposed in [169] to engineer QoS paths within a Diffserv aware MPLS domain. This algorithm applies a set of rules across the domain to allocate LSP's to traffic trunks based on the Diffserv classes of service and dynamic link metrics.

RFC 3270 [57] defines a flexible solution to support Diffserv over MPLS. This solution make use of both L-LSP (as described previously) and *E-LSP*. E-LSPs are LSPs for which the three experimental bits in the MPLS shim are used to *colour* the traffic flows. Hence, the Per Hop Behaviour (PHB) that determines the scheduling treatment is inferred from the EXP field.

Some other end-to-end QoS architectures based upon MPLS and Diffserv will be presented in Section 3.4.6

3.4.5 Bandwidth allocation, reallocation and load balancing

The main goal of traffic engineering is to optimise network resource utilisation. Best effort routing uses the shortest paths through the network, leading to congestion on some links while leaving other links empty. MPLS LSPs can be explicitly routed over under-utilised subsets of the network. Moreover, each LSP can be load balanced over multiple paths toward the egress LSR.

In [55], the authors propose a multipath adaptive traffic engineering mechanism (MATE) for MPLS networks. Its main goal is to avoid network congestion by adaptively balancing the load among multiple paths based on the measurement and analysis of path congestion. The algorithm is intended for best effort

traffic which does not require bandwidth reservation.

Another approach is taken by the Minimum Interference Routing Algorithm (MIRA) [86]. As the name suggests, the algorithm routes the new bandwidth guaranteed LSPs so that they will not interfere "too much" with a route that may be critical to satisfy future demands. MIRA considers all possible pairs of ingress egress routers and uses graph theory to calculate the maximum flow (maxflow) between each pair. For each new LSP initiated between that pair the value of the maximum flow decreases. The value may also decrease for LSPs between other ingress-egress pairs (LSP interference). An LSP has minimum interference with other LSPs if it is explicit routed so that it maximises the minimum maxflow between all other ingress-egress pairs. As the problem is NP-hard, MIRA proposes a path selection heuristic, based on the idea of deferred loading of certain critical links.

Profile-based routing [145] improves MIRA by using network traffic profiles to predict the future traffic distribution. This can be used both to guide the online path selection algorithm and to impose admission control. Also, the framework is quite general and can be extended in numerous ways to accommodate a variety of traffic management priorities in the network.

Another strategy is to balance the network load by using re-routing techniques and bandwidth reallocation on a medium term scale. One possible scenario is to combine different technologies such as MPLS and Diffserv and is presented in [97].

3.4.6 MPLS-based end-to-end QoS architectures

MPLS by itself is not able to provide end-to-end QoS services. It must interoperate with various other QoS tools. In this section some MPLS-based frameworks for providing end-to-end QoS guarantees in the Internet are presented.

In 1998 a survey of Internet QoS architectures [16] underlined the fact that

QoS related work has been within the context of individual architectural layers such as the distributed system platform, operating system, transport subsystem and network. In this context the authors of [16] propose a generalised QoS framework based on five design principles (i.e. the principles of integration, separation, transparency, multiple timescales and performance).

Since 1999, MPLS was proposed as a main component in QoS frameworks [165], together with constraint based routing, Intserv, RSVP and Diffserv. In this framework the main role of MPLS is to reduce scalability issues by flow aggregation. In 2000, the same authors [163] discuss the importance of MPLS to achieve other traffic engineering objectives such as establishing explicit routes for load distribution and load balancing and secondary LSPs for backup and re-optimisation.

An end-to-end QoS scheme based on MPLS was proposed in 2002 by Fineberg [60]. The architecture combines current and developing QoS technologies from different areas such as IP, LAN and VoIP, usually considered separately. The author emphasises on the importance of inter-operability between LAN QoS support (e.g. IEEE 802.1D) and WAN QoS support such as Diffserv and MPLS.

Other approach use bandwidth brokers with various heuristics integrating game theory, utility theory and pricing mechanisms [41]. These heuristics would aim for fair resource allocation, while at the same time provide maximum profit to service provider and yet achieve maximum value (or benefit) to applications from the customers.

3.4.7 MPLS implementations and deployment

MPLS implementation began even before it became standardised. This was mainly because it emerged from a CISCO project called *Tag Switching* [127] and therefore CISCO was continuously modifying its hardware and software in parallel with MPLS evolution.

The research community rushed to keep up with hardware providers and started to develop an open source MPLS implementation for the Linux operating system. MPLS for Linux[140] started in 1999 as a tool for testing and analysing the LDP protocol. Later on it branched into an implementation of the MPLS forwarding plane and an implementation of LDP. The forwarding plane is available for the Linux 2.6.x kernel.

Another research group [64] developed a Linux MPLS emulator along with a Diffserv-capable MPLS forwarding engine and a Linux based multi-threaded implementation of LDP.

Today, there are multiple vendors that provide MPLS capable hardware. A comprehensive list of MPLS providers is maintained at the MPLS resource centre [108].

The deployment of MPLS began in late 1999. In 2000 Xiao *et al* proposed [163] a generic procedure for deploying MPLS in a live network. Today it is very difficult to keep track of all ISPs that are using MPLS in their core networks. Every year MPLS vendors meet to demonstrate inter-operable converged MPLS services. The results of the 2006 event are summarised in [102].

3.5 Concluding remarks

The MPLS architecture was designed for QoS, based on an already existing QoS-capable technology (i.e. ATM). Key elements such as label-based virtual paths switching were imported from circuit switched telecommunications networks into connectionless networks blending the QoS features of a connection-oriented network and the flexibility of datagram routing.

MPLS implements a forwarding plane situated between the layer 2 and the layer 3 of OSI's protocols stack, allowing MPLS to function on top of any layer 2 technology and making a clear separation between the forwarding and control planes. This, in turn, allows for complex routing and signalling procedures to be

implemented on top of MPLS while keeping the forwarding untouched.

Therefore, MPLS can work as a common framework for traffic engineering and for deploying Internet services such as VPNs, local and global protection schemes and ultimately for deploying end-to-end QoS for Internet applications.

Although many ISPs have already deployed MPLS in their networks it hasn't become yet the universal forwarding plane for the Internet. MPLS is also challenged by GMPLS which promises to simplify further the protocols stack and to become the universal backbone technology for both data and telecommunication networks. The reminder of this thesis addresses how to build upon the capabilities of MPLS to deliver QoS to the user.

CHAPTER 4

Exploiting the large scale deployment of MPLS

Where "The MPLS Resource Center" [108] once kept records of MPLS deployments now simply states that: "It used to be easy to maintain a list of worldwide MPLS deployments, these days it would be easier to maintain a list of networks that haven't deployed MPLS in one fashion or another. Nearly every global service provider now offers MPLS-based VPN services and many are using MPLS internally for traffic engineering. Maintaining an accurate list of actual service deployments would be nearly impossible."

The large scale deployment of MPLS shows that it is a mature standard ready for wider use in the Internet. The Internet must also be ready for MPLS so that ISPs can fully exploit to the capabilities of MPLS for traffic engineering. This chapter describes a number of novel techniques to exploit MPLS capabilities in support of user applications.

4.1 Web server load balancing

Web service remains a key application in today's Internet. The traffic demands at popular Web sites and the requirements of redundancy and reliability can only be met by using multiple Web servers.

In Section 2.7.1 an overview of the solutions to overcome the problem of overloaded Web servers was presented. Among them, the Web clustering approach is the only one that could satisfy the today's high demand for computational intensive Web requests. This approach requires an expensive dispatcher in front of the server farm. There are scalability issues with layer 4 (or up) dispatchers that need to perform layer 4 (or above) lookups, TCP connection tracking and tear-down.

One alternative solution would be to distribute a dispatcher's load across multiple network equipments. MPLS could help such an approach since it maps application-layer information to MPLS labels so that only the MPLS ingress nodes need to perform layer 4 (or above) look-ups. TCP connection tracking could also be performed by the LERs. The load is thus distributed across MPLS ingress nodes.

A new solution to Web server load balancing based on MPLS is presented here. This solution relies on a novel Web switching architecture featuring switching at layer two. It has been implemented in a soft MPLS router using the Linux operating system.

4.1.1 Overloading a Web server

A Web server is considered overloaded when the number of incoming requests exceeds the server's capacity. The maximum capacity of a Web server is limited by a soft or hard threshold.

A soft threshold is a limit in the number of simultaneous accepted requests. Beyond this limit the server will not be able to process the requests in a timely manner.

A hard threshold is the maximum number of simultaneous connections that the system can accommodate (e.g. 150 clients [150]). If this limit is ever reached, subsequent clients will be rejected. In e-business, an overloaded server is a critical problem for companies providing Web based services since they can lose clients and revenue. Therefore, the Web server has to be always available and reliable.

An overloaded server can be avoided using a Web farm, provided that the peak demand is known, thereby allowing the minimum number of servers required to be estimated.

Consider the situation where packets arrive at the server with an arrival rate uniformly distributed over the interval [0,20] seconds so that the average rate λ is 10 connections/second. The connection duration is assumed to to be uniformly distributed over the interval [0,60] seconds. This is an average of $l = 30$ seconds. After "switch-on" transient time the system will reach a steady state and the number of active connections will vary around an average value of $c_1 = \lambda \cdot l = 300$ connections as shown by simulation results of Fig. 4.1.

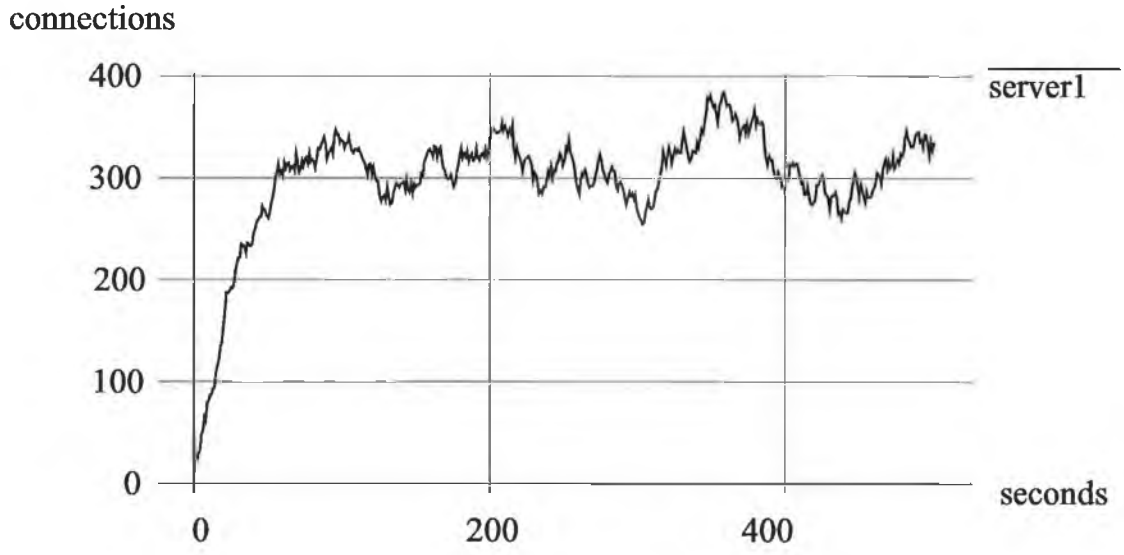


Figure 4.1: *The number of active connection for 1 server*

An Apache [149] Web server in a standard configuration accepts a maximum number of $max = 150$ simultaneous connections. The above situation exceeds

the maximum server capacity and consequently not all the requests will be processed. More than one server is needed to deal with such a large number of connections.

In an ideal situation the average number of connections per server using n load-balanced servers, is $c_n = \lambda \cdot l/n = 300/n$ and, two servers seem to be enough since $max = 150 = 300/2 = c_2$ connections/server. But in the real world, λ and l vary in time and c_n will take values greater than max (Fig. 4.2).

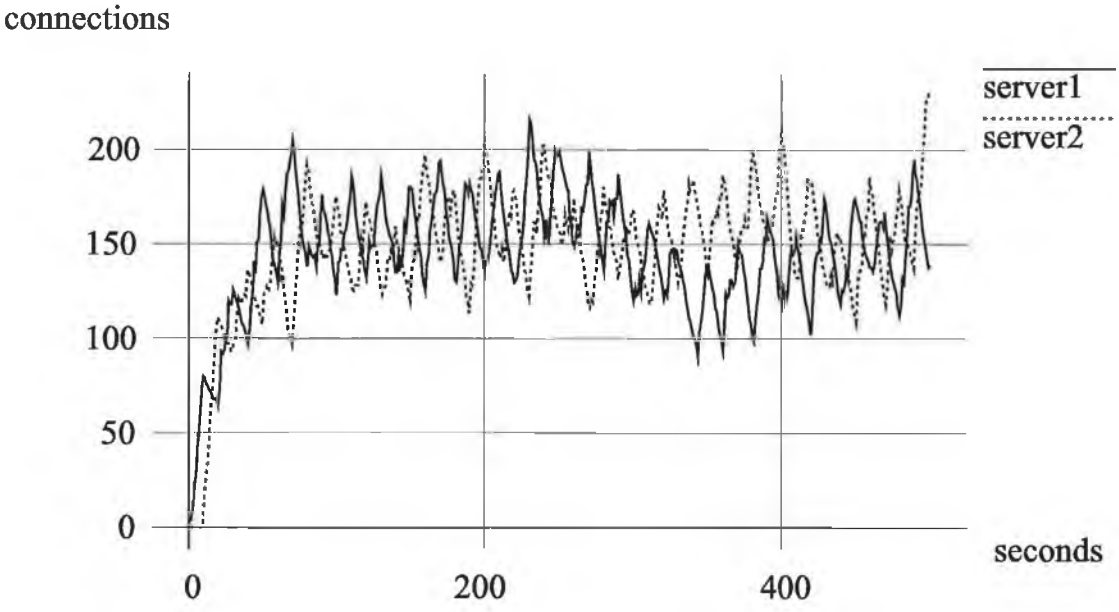


Figure 4.2: *The number of active connection for 2 servers*

Using the same simulation, it can be seen that acceptable results are obtained for 3 servers, $n = 3$ and the obtained average load per server of $c_3 = \lambda \cdot l/n = 100$ connections/server (Fig. 4.3).

In conclusion, more than one server is required for a high number of simultaneous requests. The number of servers can be estimated if the arrival rate λ and the average connection length l can be predicted using rules originally devised for calculating grades of service in telephone network.

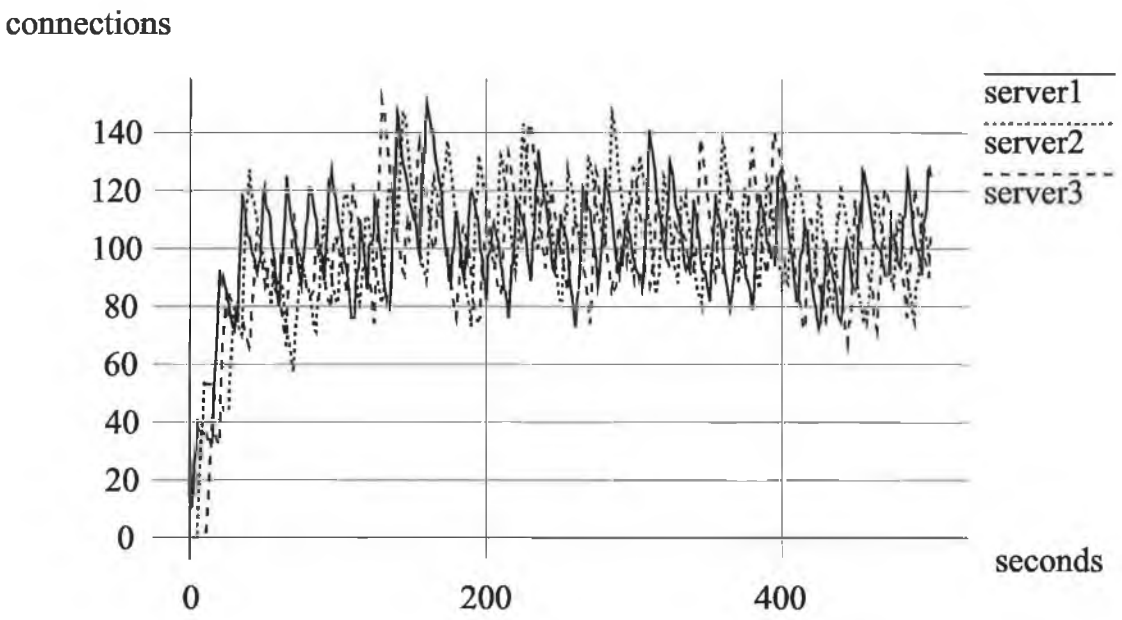


Figure 4.3: *The number of active connections for 3 servers*

4.1.1.1 The TCP continuity problem

Another major issue with HTTP traffic is that it uses the TCP protocol to establish and maintain the connection between the Web browser and Web server. TCP is a connection-oriented protocol. This causes a major problem for load balancing techniques. Imagine the situation when the first request from a certain client is sent to the optimal server from the cluster. The connection will be established between the peers and then during the connection, the load balancing algorithm will choose another optimal server and send the remaining packets of the TCP session to the second one. This will result in the connection being broken and the flow will be interrupted.

The *TCP continuity* problem must be avoided and consequently the load balancing technology has to implement a mechanism for maintaining the TCP connections alive. Generally, this is done by applying the algorithm only for the first packet of the connection (marked with the SYN TCP flag). Then, all the subsequent packets of the session will be directed toward the same destination. To address this problem one has to maintain state information for the active TCP

connection in order to avoid breakouts of long HTTP transfers and inconsistency of e-commerce transactions. The available solutions require the examination of the TCP or HTTP headers. Information such as the socket port, TCP flags, SSL session timeout or cookies can be used to identify the packets belonging to the same session and thereby maintain the session uninterrupted [3].

A natural approach to solve the *TCP continuity* problem in Web switching is to map the TCP flows into MPLS LSPs. The idea is to use different labels to specify the flows for each server across the cluster. The following section depicts the framework for an MPLS based approach to Web switching.

4.1.2 MPLS approach to Web server load balancing

The Internet is a connectionless network. Nevertheless, the WWW architecture, uses the HTTP application layer protocol to deliver information. HTTP relies on TCP layer 4 protocol which is a connection-oriented protocol. MPLS is also a connection-oriented protocol which can be used to solve the above mentioned *TCP continuity* problem by mapping the TCP flows into layer 2 LSP.

Another reason to use MPLS for Web switching is to reduce the load at the front-end dispatcher and to distribute it across the MPLS ingress nodes which can perform layer 4 and layer 7 look-ups. The dispatcher can thus rely on the faster layer 2 MPLS forwarding to distribute the request across the Web server farm.

Moreover, since MPLS provides better mechanisms to support QoS routing than the legacy IP, it can more elegantly provide QoS functions for Web switching such as *content-based-routing*, *client affinity*, *different classes of service* and *load balancing* (see Section 2.7.1).

4.1.2.1 Framework

A first approach to Web servers load balancing was introduced in Section 2.7.1. A better solution would reduce the load of the dispatcher and the need for a dedicated signalling protocol. The implementation complexity can be reduced by eliminating the proxy nodes used in [3] at the client side. This solution is presented as follows and the performance tests will be described in Section 4.1.2.3.

The approach in this thesis presumes that the ISP providing the Web service already uses an MPLS enabled network. All the ISP's devices are MPLS capable. The clients for the Web service do not have to implement MPLS since the ingress of the ISP's administrative domain will be the ingress of an autonomous MPLS domain as well. The solution involves the use of a front-end dispatcher and a Web server farm as in Fig. 4.5.

The main problem with using MPLS is that it is preferable to access layer 4 or layer 7 (TCP, HTTP) headers at the dispatcher. This is because MPLS, being a fast switching technology used at a lower level (between layer 2 and layer 3), and accessing higher level headers can dramatically slow it down. The access to the TCP or HTTP headers has to be performed at the ingress nodes.

Each server has a unique MPLS label (e.g. L_1 for the first server, etc.) associated with it, which can either be configured manually or by a label distribution protocol (if the number of servers in the cluster changes periodically). A layer 4 filter placed at the ingress nodes classifies the SYN packets (used to initiate the TCP session) and labels the packets with a dedicated label (L_{SYN}) marking the beginning of a new session. Therefore, the SYN packets marking the beginning of every new connection will be tagged with the same label(L_{SYN}). Another label is then pushed into the stack and used to forward the packet through the MPLS network. This label is swapped at each LSR with another label based on the MPLS ILM to NHLFE associations, although for clarity of illustration, in Fig. 4.4, the top label is shown as maintaining the same value (L_b).

The dispatcher need only to determine which is the most lightly loaded server (to which label L_1 will be associated), and then replace the incoming L_{SYN} label with the label L_1 and forward the packet to the server. The optimal server can be decided based on the processor load, the number of active connections, the traffic generated through its network interface, or in a round-robin fashion.

Once the packet reaches its destination, the MPLS label is removed and the packet can then be treated as a standard HTTP request. The server generates the usual reply, labels the packet with its personal label (L_1) and sends it back to the dispatcher.

The packets originated from the server are relabelled at the dispatcher using an MPLS label stack. Another label is pushed on top of the stack and used to switch the packet along the MPLS cloud, back to the ingress node. Again the simplified model in Fig. 4.4 presents the top label unchanged (L_b) although it might be changed by LSRs. The label added initially by the server (L_1) remains in the stack unchanged and will be used later to identify the server.

At the edge router, the top label is removed and the second label (L_1) is used to maintain a table of active sessions for that server. The table is mandatory in order to keep the TCP sessions alive by forwarding all the subsequent packets of the session to the same server. This will slightly increase the storage overhead but the computational overhead will not increase because the LER will have to perform the same table look-up as a traditional MPLS LER but using another table.

The remaining packets of the connection are routed to server L_1 using the table at the edge routers to identify it. The edge router uses a two layer stack to label the packets. First, the label associated with the current connection and its corresponding server is pushed accordingly to the associations in the table. Another label is then pushed on top of the stack and used by the next hop to forward the packet to the dispatcher.

Here, the top label is removed, and the second label is used to switch the

packet to its server. The server receives the packet, removes the label, and then processes the request. The cycle is completed and the HTTP connections remains uninterrupted during the TCP session.

The main advantage of this approach is that the edge routers share the label association function. Consequently, the dispatcher will perform as an ordinary MPLS switch with an added load-balancing function. However, all it has to do is to apply the function for the first packet of each connection. The rest of the packets will arrive already classified and will be switched to their destination. Nevertheless, the connection tracking process is now distributed along the edge routers and not centralised in a single box.

The above mentioned mechanism is pictured in Fig. 4.4.

4.1.2.2 Implementation

Linux was chosen as a platform for implementing the MPLS based Web switching architecture. Linux is a free, open-source, POSIX compliant, UNIX clone operating system. Its true preemptive multitasking, multi-user support, memory protection and symmetric multiprocessing support characteristics together with its networking, graphical user interface, speed and stability make Linux a preferred tool for research and development. Although the platform is open source and thus it is possible to modify the operating system internals, the architecture can be implemented without kernel modifications. Fig. 4.5 depicts the overview of the system.

Operating System (OS)

The free Linux distribution from RedHat [126] was used as a platform. The only add-ons to the standard distribution were:

- adding MPLS support to the Linux kernel
- adding MPLS support to the Linux standard firewall

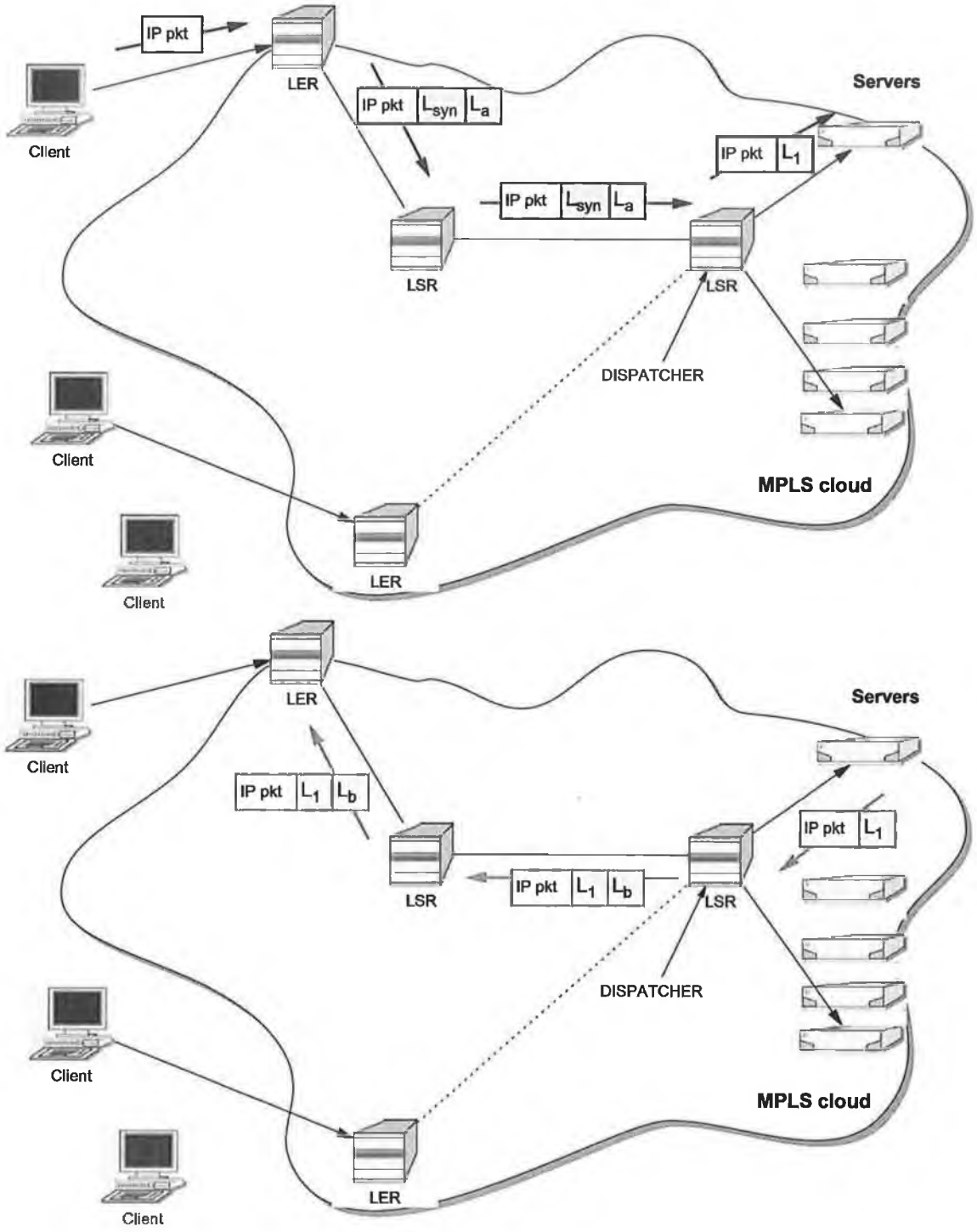


Figure 4.4: *A framework for MPLS Web switching*

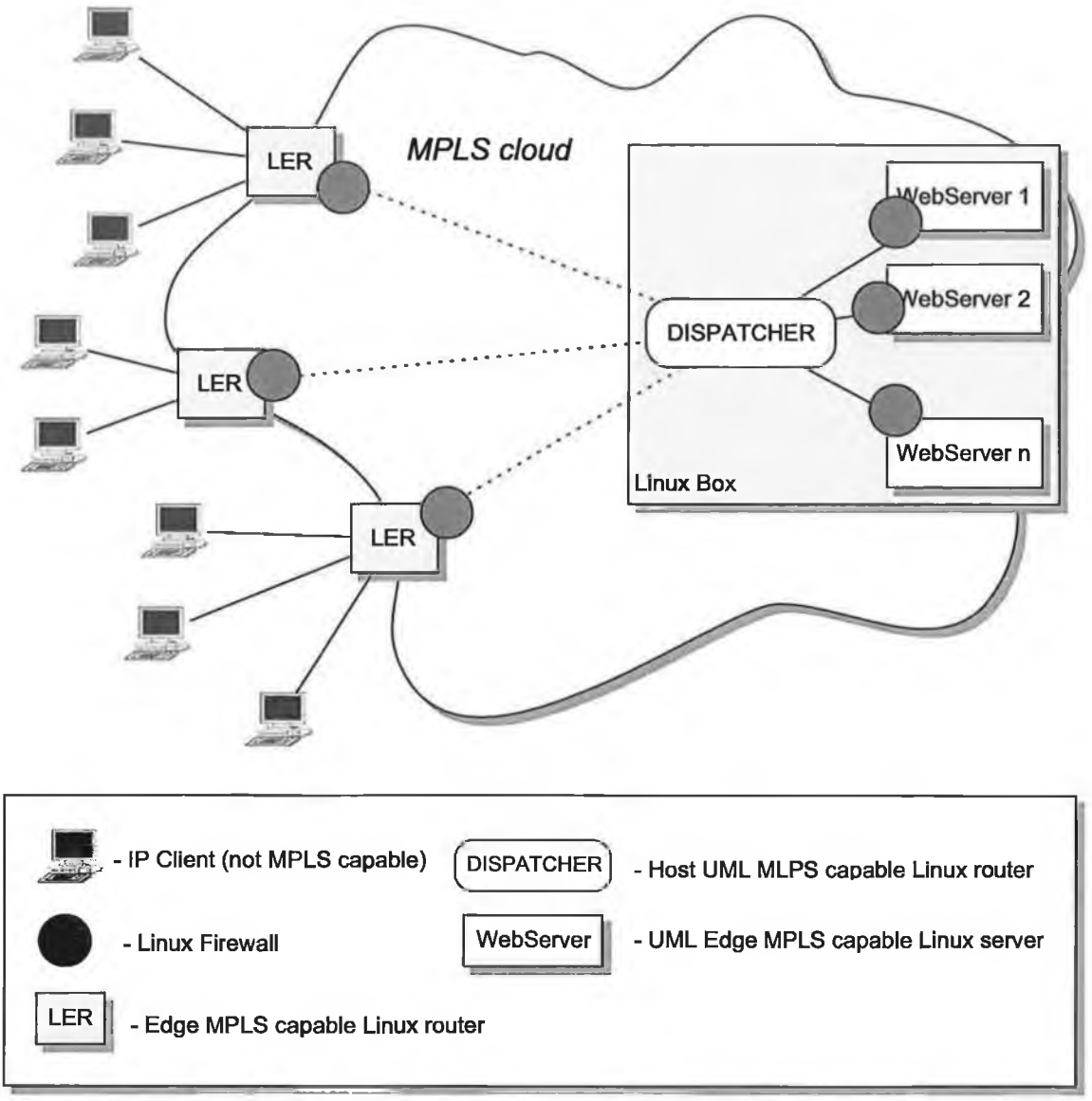


Figure 4.5: *Elements of the MPLS based Web switching implementation*

MPLS

MPLS for Linux is an open source effort to create a set of MPLS signalling protocols and an MPLS forwarding plane for the Linux OS [140]. The current version is not yet a stable version and does not offer the high performance of hardware based MPLS switches, but makes it possible to test and develop MPLS in an open source environment.

Connection Tracking

Netfilter is a firewall, Network Address Translator (NAT) and packet mangling tool for Linux [112] and is a standard component of RedHat Linux. The only modification to the standard distribution was the support for MPLS filtering.

The dispatcher

The machine hosting the UML Web servers was used as the dispatcher and also ran MPLS. A shell script was used to decide the optimal server, based on a round-robin algorithm and a C program was used to implement the load balancing function.

The main challenges of the implementation were the code for the load-balancing function at the dispatcher and for maintaining the active sessions table at the edge routers.

- **The load-balancing function** was implemented at the dispatcher side using C code and shell scripts. For a round-robin algorithm, a script was used to associate the FEC of the incoming requests to the LSP for the optimal server. For more complex algorithms, C programs are used to retrieve information about the load of each server in the cluster. Simple Network Management Protocol (SNMP) can be used to gather information like CPU usage, bandwidth usage or the number of active connections, and hence to decide the

best server for the incoming requests. If the traffic is predictable, static algorithms (e.g. every 10 seconds) can be used to elect the least loaded server. If fluctuations in the number and type of requests make the traffic unpredictable, alternative methods are needed to dynamically divert the traffic to the optimal server. By example, the dispatcher can maintain a sorted list of server loads, which can be updated whenever a new request is initiated or terminated.

- **An active sessions table** is maintained at the edge routers and used to solve the *TCP continuity problem*, keeping the state of active connections (all packets from the same TCP flow are sent to the same server). In our implementation the queuing to user-space capabilities of the Linux firewall [112] was used to perform this function. C programs were used to filter the SYN, ACK responses from the Web server and then use the label at the bottom of the MPLS stack to identify the server and maintain the table of the active TCP sessions.

Using a dispatcher for Web switching requires all the requests to be sent to the dispatcher's IP address. The IP addresses of all servers are transparent to the client. Therefore, the dispatcher needs to use techniques such as DNAT[1] to change the destination IP in each packet before transmitting them to the servers. Using MPLS, this function can be performed by the ingress nodes because the packets are tunnelled inside LSPs and their true IP is not required in the forwarding process inside the MPLS cloud. Moreover, the LSPs can be engineered to follow explicit paths across the network (for network load balancing purposes). Therefore, only the first packet of each connection needs to pass through the dispatcher. Both these MPLS advantages can further reduce the load at the dispatcher.

[1]Destination Network Address Translator

4.1.2.3 Performance evaluation

The performance was evaluated empirically in a live test.

Apache is the standard Web server shipped along with the OS. The standard configuration of the HTTP server was used. The OS running Apache constituted the target for the load balancing scheme. User Mode Linux (UML) is a simple and secure way to run and test multiple Linux kernels on a single PC. It can be used to run multiple identical Linux Web servers using a single PC-based computer.

A computer using IP but not MPLS was used to generate HTTP requests for the cluster. A simple round-robin load balancing scheme was used to verify the scheme for redirecting HTTP Traffic. A two server Web farm was sufficient to test our implementation. The client requested a large file (with a download time greater than 3 seconds) every 3 seconds. The dispatcher rotated through the server list every 2 seconds. The files were downloaded from the server according to the scheme depicted in Fig. 4.6.

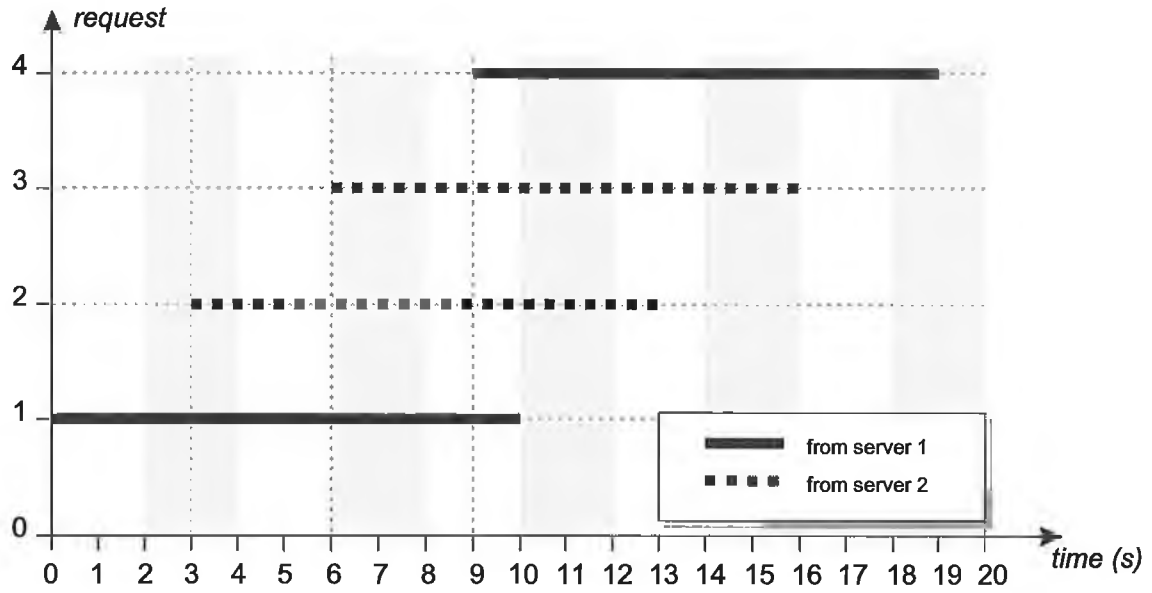


Figure 4.6: *Distributed requests*

Fig. 4.6 shows in primitive form the behaviour of the requests at the cluster side. For a more complex scheme, 3 servers were considered and more concurrent requests were generated.

Web servers are particulary stressed when acting as multimedia stream servers and/or file servers which must deal with long connections and big files being requested simultaneously. Therefore, for the first performance test relatively large files (4.2MB) were considered. An average arrival rate $\lambda = 0.4$ connections/second was chosen. This corresponds is almost 35000 requests per day. Three tests were performed generating 20, 30 and 50 connections respectively. A round-robin algorithm was used to distribute the requests among the cluster. Table 4.1 shows the number of connection processed by each server and the percentage of the total number of connections. The results show as expected, that servers share the workload almost equally (the load varying around the value of 0.33). The load never exceeded 0.4 for any server.

	Server 1		**Server 2**		**Server3**	
number of connections	connections served	share of load	connections served	share of load	connections served	share of load
20	7	0.35	5	0.25	8	0.40
30	11	0.36	12	0.40	7	0.23
50	16	0.32	16	0.32	18	0.36

Table 4.1: *Round-robin load balancing for large files*

The second test suite was intended to study the behaviour of the load-balanced cluster when a higher number of requests were present but for smaller files. Files with sizes uniformly distributed over the interval $[100KB, 1024KB]$ were used and the arrival rate was varied from $\lambda = 3$ to $\lambda = 12$ connections/second, which corresponds to over 1 million hits per day. The results for three different arrival rates are shown in Table 4.2 and reveal a more uniform distribution for shorter connection at a high arrival rates than for the previous test with longer connections.

The results presented in these tables shows that the architecture provides good results when using a static load-balancing algorithm such as round-robin.

	Server 1		Server 2		Server3	
time, connections/ second	connections served	share of load	connections served	share of load	connections served	share of load
100s, 3	102	0.34	99	0.33	99	0.33
100s, 9	295	0.32	303	0.33	302	0.33
25s, 12	94	0.31	96	0.32	110	0.36

Table 4.2: *Round-robin load balancing for small files*

4.1.3 Summary

Users requirements of high Web service availability, redundancy and reliability can only be met by using multiple Web servers. The current approaches to distribute the load across a Web cluster cannot satisfy these requirements or are expensive.

The proposal documented above is a working, cost-effective architecture, for small institutions or corporations, in an open source (Linux) environment. The performance tests showed that the MPLS based solution performs well, even for highly loaded Web sites (12 connections per seconds corresponds over 1 million hits per day). A hardware implementation should have the performance and reliability needed for large-scale Web switching.

The performance results were obtained empirically on a laboratory network using a simple round-robin load-balancing algorithm. The Web switching architecture was implemented in soft MPLS routers using the Linux operating system. A mechanism to increasing the forward rate of such soft MPLS routes will be presented in Section 4.3.

4.2 An MPLS framework to provide differentiated Web services

The above architecture may be regarded as providing a best-effort service model to users. Thus it strives to provide a uniform QoS to all users. In the same way that LSPs in MPLS allow QoS differentials to be supported. The question arises as to whether it can be used to support QoS differentials at the application level.

Consider the problem of providing two classes of differentiated Web service. The two classes comprise privileged users and best-effort users. Some possible scenarios for distributing users among the classes are:

- Paying customers versus non-paying customers;
- Intranet users versus Internet users;
- Professors versus students.

In the following sections, the two classes, will be referred to as the premium services class and the basic services class. The total server capacity required may be reduced if the system is dimensioned only to provide guaranteed levels of service to premium users, thus reducing the total cost of service provision.

4.2.1 Dynamic weighted load balancing

The Internet evolves from a network providing best-effort service toward a QoS network that can offer service differential for different classes of customers and applications. Several new technologies to provide network level QoS have been proposed and some of them already implemented. However, for a viable end-to-end QoS scheme the Internet servers (such as Web servers) must also be able to provide service differentiations and guaranteed level of service for premium customers. In this section an architecture for providing differentiated Web services in an MPLS aware network is presented.

4.2.1.1 Traffic classification

The requests must be mapped into two classes of services The architecture here uses labels to classify the traffic. At the edge of the MPLS cloud, the requests are labelled according to their class of service. The label bindings for the two classes could be configured statically by the network administrator or distributed by the dispatcher using separate label spaces for each class. The dispatcher can then identify the class of a request, based on the MPLS label value.

Alternatively, mapping the classes of service to different FECs can be performed using the standardised MPLS signalling protocol RSVP-TE. The attributes required by the dispatcher to differentiate between the two classes of service could be carried by the same RSVP Path message that establishes the explicitly routed LSPs. The attributes can be encoded either in the SESSION_ATTRIBUTE object of the Path message [17] or as a TLV[2] in the RSVP object defined by [56].

4.2.1.2 Traffic estimation and load distribution

To simplify the analytic model, it is assumed that the server could generate the response in a constant time t_1 (e.g. 10 seconds) for each request, if only a single request were to be processed at any one time. Empirical tests have established that the execution time of a CGI script increases linearly with the number of concurrent executions on an Apache Web server running on a single processor PC. Therefore, it can be considered that $t(x) = a \cdot x + b$, where x is the number of simultaneous processes and a and b are parameters that depend on the CPU speed and process complexity but can be previously estimated for a particular system and a specific type of request.

The agreement between the service provider and the client might specify that the requests must be served in a time less than T and a blocking probability below p. Writing $T = t(x_{max})$, this means that no more than $x_{max} = \frac{t(x_{max})-b}{a}$ requests

[2]Type Length Value

should arrive simultaneously at one server. The total number of simultaneous requests for a farm with n servers is $n \cdot x_{max}$. If the observed traffic load at a busy hour is E, than using the *Erlang B* formula (as described in Section(2.7.2), the blocking probability can be calculated. The cluster can therefore be dimensioned to guarantee both the maximum execution time t_{max} and the blocking probability p.

It is assumed that, for the grade of service promised to premium clients, a server must be able to accommodate c active clients simultaneously. In a Web cluster with n servers, with an ideal load balancing algorithm, the requests are equally distributed among servers. Therefore, each server will encounter at most c/n requests. It follows that:

$$\frac{c}{n} \leq x_{max} \Rightarrow n \geq \frac{c}{x_{max}} \Rightarrow n \geq \frac{c}{\frac{t(x_{max})-b}{a}} \tag{4.1}$$

The required number of servers (n) can then be estimated using formula 4.1. There are various scenarios for distributing the premium and basic requests across the n servers.

The first approach is to use the load balancing mechanism presented in Section 4.1 to distribute both premium and basic requests across the entire cluster. In this case, the servers need preemption capabilities to prioritise the premium requests. Another disadvantage is that an unbalanced distribution of premium requests per server may create considerable differences in the execution times for premium requests.

To overcome the above problem, another approach is to balance the premium requests across the servers and use the available CPU resources to accommodate basic requests balanced as well across the servers. However, the servers need to be preemptive. A similar approach, but for servers with no preemption is to use operating systems that can handle the requests with different priorities.

A common problem with all these approaches is that servers handle both pre-

mium and basic requests. This makes the execution time for premium (and basic) requests difficult to predict and guarantee.

A fourth approach would be to separate the servers into two groups, one for each request class. The membership of each group is adjusted dynamically based on the number of existent premium requests. In this approach the servers need no preemption or operating system priority support and it is the choice for the differentiated Web services implementation.

One disadvantage of this approach is that when a server is moved from one group to another, the available capacity changes by a large quantum. However, this only acts to the detriment of basic requests, and premium users will never suffer. The transition of a server from one group to another is detailed below along with other aspects of this solution.

Let $S = \{s_1, s_2, ...s_n\}$ be the set of n Web servers. My proposal uses subsets of S for the two classes of requests: $S_p = \{s_1, s_2, ...s_i\}$ for premium requests and $S_b = \{s_i + 1, s_i + 2, ..., s_n\}$ for basic requests, where $0 < i \leq n$. Clearly $S_p \cap S_b = \emptyset$ and $S_p \cup S_b = S$. The role of the dispatcher is to balance the load among the subsets and to map premium traffic and basic traffic to S_p and S_b respectively. The initial state for n servers in the cluster is: $\mid S_p \mid = 0$ and $\mid S_b \mid = n$.

In order to keep the premium customers satisfied, in this approach to the provision of differentiated Web services, the execution time for any premium request should be lower than the agreed value of T and also at any given time, the execution time for any basic request should be greater than for any premium request. The first condition can be achieved by good provisioning of servers using (4.1). The second condition can be achieved if the number of connections in any server from the premium class is lower than the number of connections in any server from the basic class. This can be explained mathematically as follows.

The following function is defined: $con(s)$ = the number of requests being served by server s. Then, for any $s_i \in S_p$ and $s_j \in S_b$ the following is true:

$$con(s_i) \leq con(s_j) \tag{4.2}$$

The number of active premium requests changes over time. When the number decreases, available resources can be used by basic requests and if it increases, the resources must be reclaimed. Therefore the number of servers in both subclasses S_p and S_b changes triggered by the following events:

A new premium connection arrives and if the number of existing connections is such that by accepting a newly arrived premium connection will mean that there exist $s_i \in S_p$ and $s_j \in S_b$ so that $con(s_i) > con(s_j)$ (i.e. the condition (4.2) is no longer satisfied). Therefore, a server $s \in S_b$ moves into S_p (premium users get an extra server).

The second event that triggers a server to move from one class to another can not be precisely defined mathematically but it is rather an administrative decision to give more servers to the basic class when the class of premium servers is under-utilised. Therefore, a server $s \in S_p$ can be moved into S_b (basic users get an extra server) only if by doing this, condition (4.2) is still satisfied and for any $s_i \in S_p, con(s_i) \leq x_{max}$. The issues involved in moving a server from one subclass to the other are described below.

Transition from S_b to S_p. When a server $s \in S_b$ has to be moved into S_p it means that it will start accepting premium requests and stop accepting basic requests. However, at the transition time, s is processing a higher number of connections than any other server in S_p. So, by accepting premium requests the load will further increase and the load and the execution time for these premium requests might be greater than T.

The simplest solution to this problem is to instantly drop all basic connections and start accepting premium requests. Then, the premium connections can be balanced across the enlarged subset S_p of premium servers. This is the approach implemented and simulated in the following section.

An alternative is to drop only some of the basic connections in order to satisfy the condition $con(s_i) \leq x_{max}$), the number dropped being chosen so that the execution time for the newly arrived premium requests is less than T. Since no further basic connections will be accepted, once the active ones terminate the server will be processing only premium requests.

Another alternative is possible only if the server supports task preemption. Thus, when the server starts accepting higher priority premium requests there is no need to drop the basic connections but to run them with lower priority. However, it my take a long time until all the basic requests are cleaned up from the system since they run with low priority. During this transition time the servers in S_p cannot be properly load balanced.

Transition from S_b to S_p. When the number of premium connections decreases and there is a high number of basic requests, a server $s \in S_p$ can be moved into S_b if for any $s_i \in S_p, con(s_i) \leq x_{max}$ and (4.2) remains true. However, if at the transition time the server s is still executing premium requests, by accepting basic requests the execution time for these premium requests may be greater than T.

The simplest solution is to briefly stop accepting any new connections for server s until all premium connections finish. This will be inefficient if transitions are frequent.

A more sophisticated approach is to gradually accept new basic requests as long as $con(s) \leq x_{max}$, thereby keeping the execution time for premium requests below T. During the transition period, basic customers will briefly receive premium levels of service, but the overall efficiency is higher.

4.2.2 Simulation results

4.2.2.1 Relation between execution times and the number of concurrent requests

The first experiment evaluated the behaviour of an overloaded Web server. For this experiment a system with an AMD K-6 cpu (233MHz) and 64MB RAM running the Linux operating system and the Apache[149] Web server was used. A CPU intensive CGI script that executes at the server side in approximately $t_1 = 10$ seconds was written. The number of concurrent requests was incremented gradually and the request execution times were noted. The results in Fig. 4.7 show that the execution time increases linearly with the number of simultaneous requests x. The slopes depend on the system characteristics and the CGI script's computational complexity. When the number of simultaneous connections causes server overload, the excess incoming connections will be dropped.

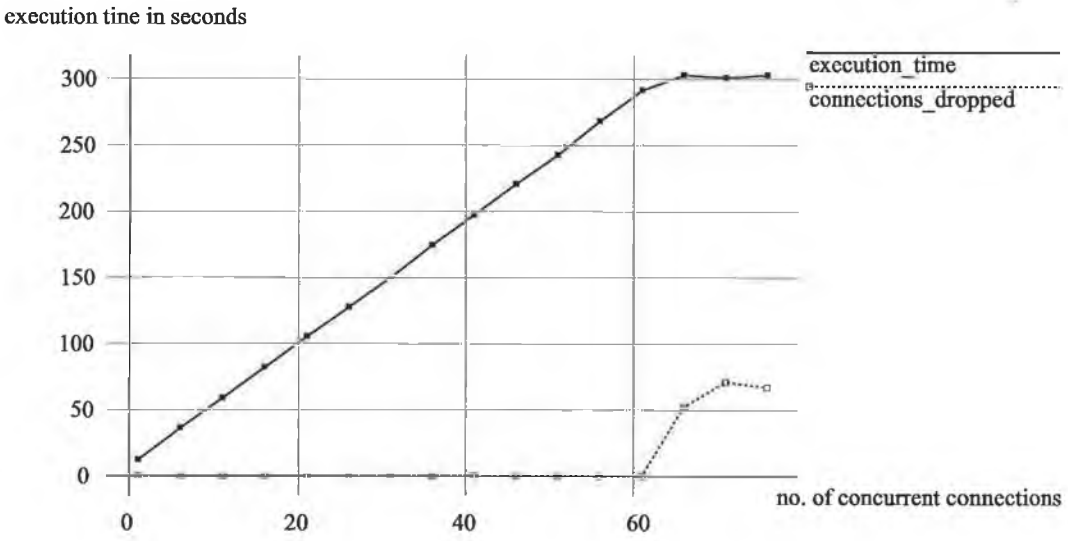


Figure 4.7: *Execution times for concurrent connections*

4.2.2.2 Adaptive load balancing distribution

The model for providing differentiated Web services is simulated in the second experiment. Two sources of traffic were considered: premium traffic and basic

traffic. The requests arrival is modelled by a heavy tailed probabilistic distribution of inter-arrival times (Pareto). Execution times are given by the results from the previous experiment: $t = ax + b(a = 5, b = 5)$.

Premium traffic requests arrive with inter-arrival times given by a Pareto distribution with a mean arrival rate $\lambda = 1$ connections/second and basic requests with a mean of $\lambda = 2$ connections/second. The load is dynamically distributed among the 8 servers within the cluster. The subsets of servers dedicated for the two classes of requests had initially the cardinality $\mid S_p \mid = 1$ and $\mid S_b \mid = 7$.

In order to satisfy the promised grade of service for premium users (execution time below $t(x_{max})$), the number of servers in S_b was increased or decreased, based on the algorithm described in Section 4.2.1. In this particular case, the system had to maintain the premium services response times below a value of 50 seconds and the basic services response times above the times for the premium requests. Part of the basic requests were dropped at peak times, but more basic connections were accepted when the servers were lightly loaded as shown in Fig. 4.8.

A higher arrival rate for basic requests did not affect the premium services as long as the condition 4.2 was satisfied. The simulation proved that two classes of services can be delivered using a load balancing architecture with different weights for the two classes. Moreover, the promised grade of service for premium service was satisfied since the cluster was dimensioned to serve the maximum possible number of requests.

4.2.3 Summary

In this section a challenge faced by today's Web service providers was described. The traffic through the Web sites increases along with the number of clients and the number of services offered. In this context, separating the clients in classes of priority can improve the performance of a Web content hosting site. Economical

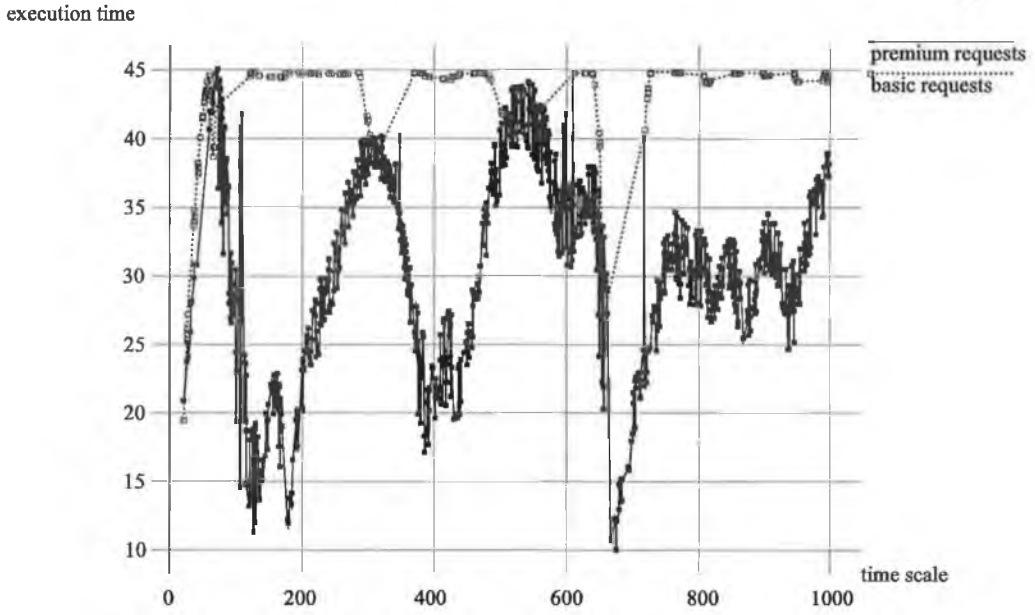


Figure 4.8: *Execution times for premium and basic requests*

parameters may also impose a differentiation between potential classes of clients. By providing a solution which uses MPLS, it is assumed efficient interaction with the favoured protocol for high-speed QoS aware networking in today's Internet.

4.3 Increasing router performance using MPLS meta-frames

The solutions presented in the previous sections were implemented using cost-effective soft routers. As MPLS deployment extends from the Internet core to the access network, such routers will have a role to play in providing affordable access to MPLS functionality.

A move from MPLS to the edges of the Internet puts the most demanding LSR (the edge router) in exactly the position in the network where customers are most cost-sensitive and where the hardware capabilities are likely to be most limited. This section addresses this issue by looking at one of the most significant bottlenecks in a soft router - the packet processing time. An overview of other ap-

proaches to increase router performance by reducing the packet processing time or the number of packets to process was presented in Section 2.6.

4.3.1 The average packet size in the Internet

An empirical study was undertaken to determine the average packet size in an IP network. The Anritsu MD1230A IP/Ethernet/POS Quality Analyzer [10] was used both as a traffic generator and packet analyzer. It will be referred throughout this section as the DQA (Data Quality Analyzer). The DQA is capable of sending wire speed IP or MPLS traffic through one or more of its multiple 10Base-T(10Mb/s), 100Base-T(100Mb/s) and Gigabit Ethernet (1000Mb/s) interfaces at up to 62.5 million packets per second. It can also compute the throughput and latency with a resolution down to 10^{-9} seconds. The DQA was used to gather statistics about the traffic transmitted and received during one hour by 15 workstations in the Switching and Systems Laboratory in Dublin City University.

The results show that over 85% of the frames were small frames of between 64 and 128 bytes in length. The mean packet length was 110.2 bytes and the median was 64 bytes.

Another sample of Internet traffic was traced at the input/output interface of the router connecting the School of Electronic Engineering's network to the main Dublin City University router. Statistics show an average packet size of 222 bytes for the output flow and 340 bytes for the input flow resulting in an overall average of 281 bytes.

Other statistics collected from the Internet backbone [104] show that almost 60% of packets comprise 44 bytes or less. Also, the packet length distribution seen at NASA Ames Internet Exchange (AIX) [98] shows a mean packet length value of around 400 bytes and a median value below 100 bytes.

The average packet size varies from one network to another due to the various patterns of traffic types. Large file transfers or multimedia streaming use large

size packets while Web browsing or emails use small size packets. However, the results clearly show that the Internet is traversed by many packets much smaller than the allowed Maximum Transmission Unit (MTU).

4.3.2 The effects of small packet size on router performance

Traffic consisting of small frames is considered harmful due to its encapsulation overhead and the higher palletisation cost. Here some measurements undertaken to reveal the drawbacks of traffic consisting of small frames are presented.

4.3.2.1 Encapsulation overhead

The throughput of a flow cannot reach the maximum bandwidth provided by a link due to factors such as protocol overhead and inter-packet gap. An overview of the mathematical calculations and formulas which can be used to determine network throughput and performance can be found in [38].

Consider TCP/IP traffic traversing an MPLS network with Ethernet links. The minimal per packet overhead added by the Ethernet encapsulation and inter packet gap, by TCP/IP and MPLS is 38 bytes, 40 bytes and 4 bytes respectively. This is a total of 82 bytes per packet.

The percentage of data payload for data traffic sent using IP over MPLS encapsulated in Ethernet frames is represented in Fig. 4.9. It can be observed that the efficiency increases rapidly with the frame size. For layer 2 protocols, the IP and TCP headers are considered to be payload as well. Therefore, the maximum throughput that can be achieved considering only the ethernet header and trailer as overhead is given by the following formula:

$$theoretical\_value = \frac{(packet\_size - header) \cdot 100}{packet\_size}$$

The term "theoretical throughput value" is used by the DQA for the graphical representation of the above formula. Clearly the overhead is considerable

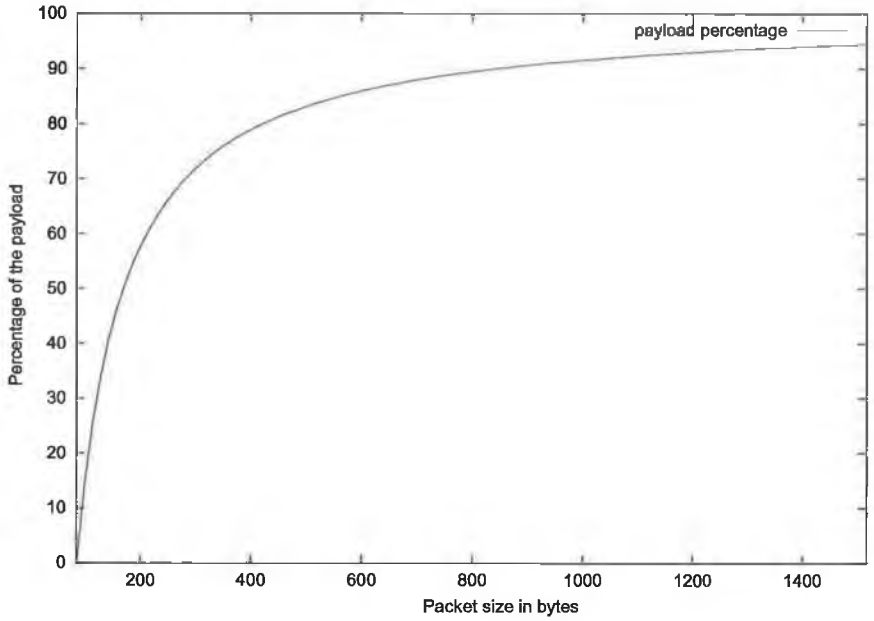


Figure 4.9: *The variation of the payload with the packet size*

for packet sizes below 50 bytes, which, as noted above, represents a significant proportion of Internet traffic.

4.3.2.2 Computational overhead

Much of the cost of packet-switched communication is per-packet rather than per-byte. In order to send a certain amount of data in a time interval, the number of frames is inversely proportional to the frame size. Hence, more headers per second are processed by the router and more hardware interrupts are generated when packet sizes are small. This can drastically overload soft routers situated at the edge of the network and therefore, the throughput will suffer degradation. This is a QoS issue because bandwidth reservations on links are based on the assumption that router interfaces operate at wire speed and can thus occupy 100% of the link bandwidth. Fig. 4.10 shows the throughput that was achieved by a soft Linux router for various packet sizes and for 100Mb/s Fast Ethernet links. The results are plotted next to the graphical representation of the maximum "theoretical value" function. The throughput achieved using large frames traffic is higher

because a smaller number of frames had to be processed.

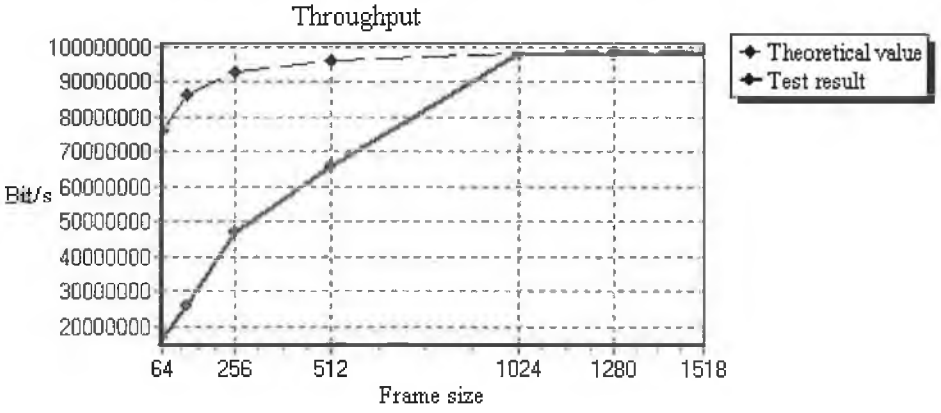


Figure 4.10: *Throughput bit/s rate on 100Mb/s links*

The results are similar for larger MTUs as well. Related measurements [6] show that jumbo frames [119] (9000 bytes) can provide 50% more throughput with 50% less CPU load than 1500 byte frames. Other layer 2 technologies allow larger MTUs. For example MTU is 4500 bytes for Fiber-Distributed Data Interface (FDDI), 9000 bytes for ATM and 65280 bytes for High Performance Parallel Interface (HIPPI). However, as long as the large majority of LANs are Ethernet in an increasing Ethernet market and reaching speeds up to 10Gbps [73], with no mechanism for increasing the packet size in the core, the MTU across the Internet remains 1500 bytes.

4.3.3 Target MTU for meta-frame

The main goal of the meta-frame solution is to increase the average packet size in the Internet as close as possible to the MTU which is 1500 bytes due to the wide use of ethernet LANs. However, the meta-frame approach can also take advantage of large MTU networks.

Although most Internet packets originate in Ethernet LANs with MTU of 1500 bytes, in the core of the Internet the MTU could be larger. Hence, the meta-frame MTU is not limited to 1500 bytes. If a path MTU discovery protocol [106] determines that the the MTU across the meta-frame network is higher than 1500 bytes,

then meta-frames larger than Ethernet's MTU can be assembled since they will be disassembled before reentering a lower value MTU network.

4.3.4 Meta-frames overview

The discussion above illustrates the negative impact on network performance of having a predominance of short packets in the network. An obvious response is to change the packet mix so as to increase the average packet length. However, most traffic carried on the Internet originates as IP packets encapsulated in Ethernet frames and generated by higher level protocols such as TCP and UDP which take no particular care (other than using piggybacking to carry flow control messages) to avoid injecting short packets into the network. Modifying the transport layer protocols to improve this situation would be a major undertaking. Persuading the community of the Internet users to install the revised protocols would be well-high impossible, given the variety operating system types and versions in use.

Thus, a network-level response is required, and the most beneficial location for this response is in access networks at the edge of the Internet. The only available response at the network level is to merge short packets so as to increase the average packet size. This is a simple concept, but implementing it is a challenge. A solution using MPLS, called "meta-frames", is documented below.

In the approach an ingress node buffers and assembles multiple consecutive IP packets into larger frames called meta-frames and a meta-frame header is added. A meta-frame is then forwarded based on the information contained in this header. An egress router will disassemble the meta-frame and forward the packets based on their own headers.

Since packets are forwarded as a group toward a common point, they must have similar properties, such as the same destination address. This is because between their assembly and disassembly point, packets are encapsulated inside the

meta-frame and therefore, the forwarding is performed based on the meta-frame header. This is a real problem for protocols such as IP, which require routing decisions to be performed at each hop. Therefore, in one meta-frame can be assembled packets having the same destination IP address. In MPLS, the flows can be aggregated (say all flows that have the same MPLS ingress and egress node) reducing the number of FECs and hence increasing the number of packets that can be assembled in the same meta-frame.

The function required of the router in assembling meta-frames is to identifying packets with similar transport requirements. In MPLS the IP traffic is tunnelled through LSPs, and the task performed by the ingress node is precisely to map incoming IP packets into FECs. Therefore, if the meta-frame function is undertaken by the ingress router, negligible additional computational overhead will be incurred.

If this technique is to be deployed, a number of implementation issues must be considered as well. One such issue is the delay introduced by the meta-frame packetization. The delay occurs when packets wait for other packets to be assembled in the meta-frame. This delay should be limited and if the time limit is reached a meta-frame will be generated and forwarded even if its size is much lower than the MTU. This limit can be dynamically configured based on the packet rate if the packet rate does not fluctuate too much. However the meta-frame concatenation is applied for the core of the Internet where aggregated traffic trunks encounter less variations of packet rate.

Another issue is that routing information must exist in the meta-frame header in order to allow routing to be performed along the path. In an MPLS environment, the header is relatively small and since the same label is used for all the packets belonging to the same FEC, a single MPLS header per meta-frame is sufficient to forward all the component packets toward the egress LSR.

In order to accommodate meta-frame traffic, modifications to the protocol stack must be performed. The number of routers that have to be modified must

be minimised to allow the deployment of a low cost meta-frame approach. In MPLS networks the changes are performed to edge routers only (i.e. ingress and egress LSRs). The complexity of meta-frame encapsulation and decapsulation must be as low as possible to avoid CPU overloading. Assembly and disassembly algorithms that are too complex can add unacceptable processing delays.

4.3.5 Frame format

Almost all data on the Internet is carried using IP packets. Therefore, our concern is for encapsulating IP traffic into MPLS meta-frames. Since meta-frames traverse MPLS networks, they should be encapsulated as MPLS packets. The generic MPLS encapsulation places an MPLS shim between the layer 2 and layer 3 headers as depicted in Table 4.3.

Table 4.3: *Generic MPLS encapsulation*

L2 header	MPLS shim	IP header	IP payload

In an MPLS meta-frame the IP header is the header of the first encoded packet that is followed by its payload and a succession of IP headers and payloads as seen in Table 4.4

Table 4.4: *MPLS meta-frame encapsulation*

L2 head	MPLS shim	IP header 1	IP payload 1	...
		...	IP head n	IP payload n

Such a packet will be treated as an ordinary MPLS packet. Therefore, the meta-frame encapsulation is transparent for ordinary LSRs. However, egress routers must be able to identify the meta-frames in order to be able to decapsulate them. There are various approaches to make this possible, such as:

- One simple approach it to use the experimental bits in the MPLS label. However, this can not be done if Diffserv's DSCP is encoded in the EXP field of the MPLS header.

- Dedicated MPLS label ranges for meta-frames can be used. This requires either dedicated label spaces for meta-frames or modifications to the signalling protocol both of which increases the complexity of network administration.

- Probably the most suited approach is to register a new protocol type. This will leave the existing MPLS protocol stack intact an allow the meta-frame MPLS to coexist in the same router.

4.3.5.1 Encapsulation

At the MPLS ingress, packets are classified based on ILM (Incoming Label Map) or FTN (FEC-to-Next Hop Label Forwarding Entry). Packets corresponding to the same NHLFE entry can be encapsulated into meta-frames. Hence, for each NHLFE entry, a buffer must be reserved. This buffer will accumulate the partial content of a meta-frame during the assembly process as long as the size of the buffer is less than the MTU. The content is then labelled and sent out as an MPLS meta-frame.

The first buffered packets have to wait in the buffer until there are enough packets to assemble a meta-frame. Timers can be used to limit this delay to a tolerable value when packet rate is low and there are not enough packets to fill the MTU. In this case (light traffic) frames of small size might have to be sent. But since the traffic is not heavy (locally), small frames are acceptable for transmission. If small frames passing that router arrive at a busy router downstream, it may chose to encapsulate them in a meta-frame itself.

4.3.5.2 Decapsulation

Upon receiving an MPLS meta-frame an egress router (or an "Penultimate Hop Popping" router) must be able to decapsulate the original IP packets. First the MPLS label (or labels) is popped. Then, the IP header is examined (i.e. the IP

header of the first IP packet) and based on the total length field, the payload of the first IP packet is identified and the whole IP packet is restored. The procedure continues for the remaining IP packets. Once an IP packet is restored, it is forwarded using its own restored header.

4.3.6 Performance results

The performance results below were obtained using soft routers. As mentioned in the introduction of Section 4.3, such devices are typical of the technology deployed at the edge of the Internet, and the benefits of applying the meta-frames architecture are most pronounced if it is employed at the network edges. The assumption here is that the interface between IP and MPLS will occur in the access network. It might be argued that a software router could never achieve wire speed when configured as an MPLS LER. Even if this is so, it is reasonable to assume that a high software overhead in a software router maps into a complex hardware implementation in a high-end router. Thus, the results obtained here may be used to extrapolate the cost, if not the performance, of hardware routers.

Another reason why software routers were used in the prototype implementation below was expediency. It would not have been possible to configure a hardware router to implement the meta-frame protocol. Another advantage of using software routers is that the implementation of routing can be modified to achieve better performance. Such a modification is considered in this section.

4.3.6.1 Empirical results

As a proof of concept the meta-frame framework was implemented using loadable Linux kernel modules. One module was written for the ingress node to assemble the meta-frames and another for an egress router to disassemble them.

In the performance test packets of length 128 Bytes were sent over a network of Linux routers. A simple test network was used where one Linux router was

an MPLS ingress and meta-frame assembly point and the second one was an MPLS egress and meta-frame disassembly point. The traffic was sent at 30% of the 100Mb/s link capacity. Because of the high frame rate ($\simeq$ 30000 fps) the Linux routers were overloaded and could only forward packets at 14% of link capacity.

Fig. 4.11 shows the relation between the incoming traffic rate (Traffic 1 in the legend) and the achieved throughput rate (Traffic 2). The measured experiment has 3 stages. In the first stage, packets are sent unmodified. In the second stage every 2 consecutive frames are assembled and the throughput rate is increased to around 24%. In the third part, every 3 consecutive frames are assembled in a meta-frame and the routers are able to forward all the incoming traffic. In the last stage the throughput rate equals the input traffic rate.

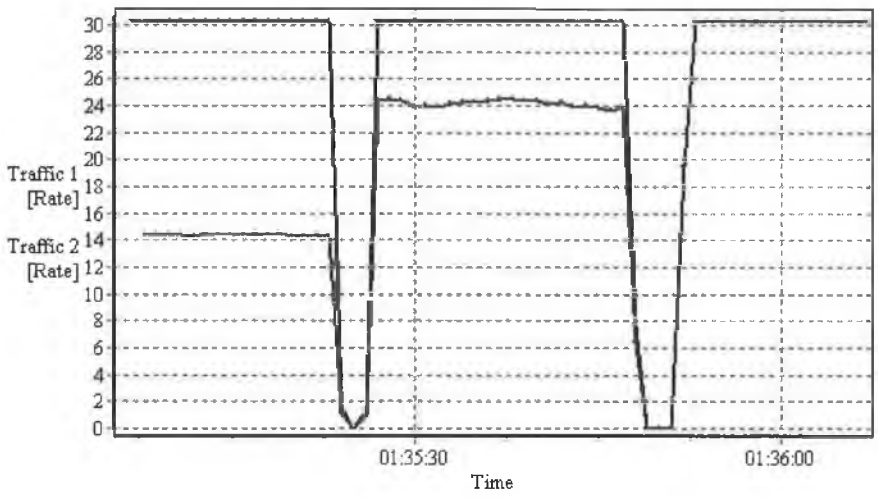


Figure 4.11: *The throughput performance for no assembly, 2 packet meta-frame and respectively 3 packet meta-frame.*

The meta-frame encapsulation/decapsulation shows an important increase in throughput rate while introducing a negligible 50μs delay. This verifies the meta-frames concept, but does not assess its performance with real traffic. The later was evaluated by simulation.

4.3.6.2 Simulation results

Simulations were performed using traces of real traffic measured at NASA Ames Internet Exchange (AIX) [99]. An algorithm was written that allowed the simulation of MTUs larger than 1500 bytes. The input traffic had a mean packet size of 400 bytes and a median of 100 bytes. The simulator was coded using the C programming language. Three simulations were performed for assembling meta-frames in networks with MTUs of 1500 bytes, 4500 bytes and 9000 bytes. The mean meta-frame size, the "meta-framisation" delay and the median number of packets per meta-frame were measured and summarised in Table 4.5.

MTU in bytes	mean meta-frame size in bytes	delay in milliseconds	median number of packets in a meta-frame
1500	1250	0.08	11
4500	3500	0.1	32
9000	6000	0.2	55

Table 4.5: *The average meta-frame size and the packetisation delay for MTUs of 1500, 4500 and 9000 bytes*

The overhead decreases using the meta-frame encapsulation and was calculated in the simulation. Table 4.6 shows the encapsulation overhead before and after meta-frame encapsulation for the three MTUs.

MTU in bytes	mean overhead before meta-frame encapsulation	mean overhead before meta-frame encapsulation
1500	0.2	0.06
4500	0.2	0.02
9000	0.2	0.01

Table 4.6: *Mean overhead before and after meta-frame encapsulation for MTUs of 1500, 4500 and 9000 bytes*

The delay can be further controlled using timers. However, if this delay is not acceptable, delay sensitive classes of traffic may simply not be assembled into meta-frames. This is simple to achieve in an MPLS enabled environment. A class based forwarding behaviour can be implemented for various delay requirements

in order to satisfy customer needs.

4.3.7 Summary

In this section the problem of small packet lengths in Internet traffic was discussed. A novel technique to increase the average packet size in an MPLS environment is presented. This meta-frame technique can increase the overall traffic throughput.

The MPLS meta-frame implementation requires modification only at the edge of the network and it is transparent for core routers. A prototype implementation showed us that even low-end software routers can easily make use of this framework and increase their performance with little software modifications and without adding a significant delay.

The motivation for looking at packet size here was to address performance issues in access networks. However, the solution presented also benefits from the Internet core, by reducing the amount of packet processing to be performed there. In particular, such a framework can help future Tera-Bit speed GMPLS optical core switches that need larger frames to make the most of (for example) optical burst switching technology.

By increasing the average packet length in the core of the network, this protocol ensures that switching speeds need not improve in step with increases in transmission rate, which may be problematic in the optical network core, in the absence of a practical high-speed optical packet switching technology.

4.4 Heart-like fair queuing algorithms (HLFQA)

This thesis has described how MPLS can be used to provide traffic engineering capabilities in the Internet. Its facility to map traffic flows into label-switched paths allows traffic to receive differential treatment in the network and can thus provide differential QoS levels to various classes of traffic. However, although the

QoS required for a particular stream of traffic can be determined from its label, differently labelled packets must receive differential service at the link level if the QoS differentials are to be realised. This is the function of a traffic scheduling algorithm.

Traffic scheduling as a component of Internet QoS was reviewed in Section 2.2.3. One issue with current scheduling algorithms is that they are either not fair enough or are difficult to implement in hardware.

In this section, a new work conserving traffic scheduling algorithm is presented, that is inspired by the principles of the human heart. First the main concepts of our scheduling algorithm are explained as is its similarity to the atrium-ventricle model in the human heart. Then, the fairness and complexity of the algorithm is evaluated using an analytical model and computer simulations. An extended algorithm for weighted fair queuing is presented in the end of this section.

A lighter version of this algorithm is then presented, that is easier to implement and has better storage complexity. This simplified version is suitable for implementing weighted fair queuing.

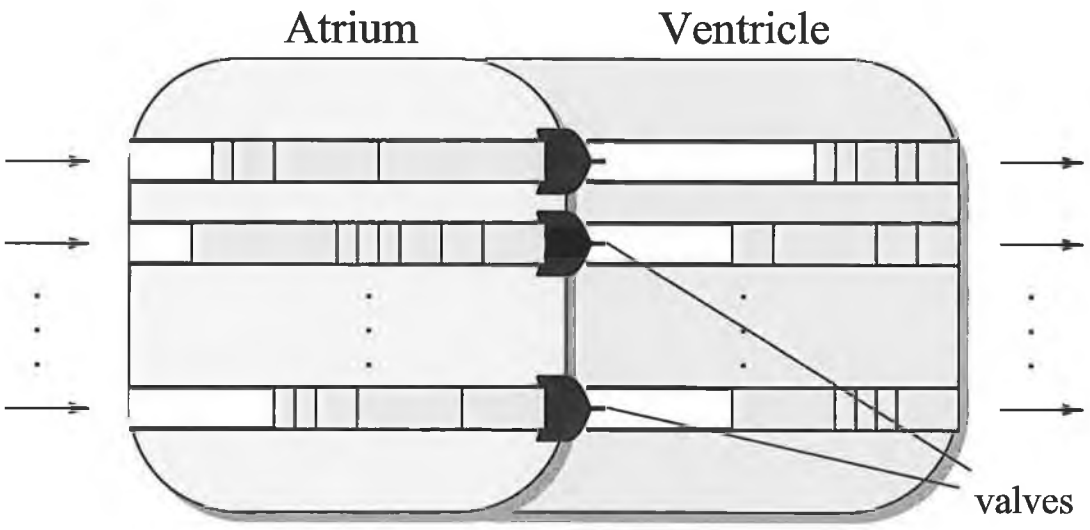


Figure 4.12: *Atrium-ventricle model*

4.4.1 The atrium-ventricle model

The new scheduling algorithm is based on the atrium-ventricle model of the heart in the cardiovascular system. The output queue of an interface is divided into an atrium section where packets are buffered and a ventricle section where packets are sent out by applying pressure to the ventricle.

Atrioventricular valves allow packets to move from the atrium to the ventricle during the atrial systole and prevent packets from running back from the ventricle to the atrium during the ventricular systole.

Whereas the human heart is quadric-cameral, in this model there are two chambers for each FEC: one atrial and one ventricular as depicted in Fig. 4.12. The atrial and ventricular chambers for each FEC will be referred to as the holding queue and submit queue respectively.

When the ventricle is contracted, packets are sent out through the output interface (aorta). The first packet to get out is the one from the submit queue with the highest *pressure*[3]. After one packet (or more) is released from that submit queue, the pressure decreases enough so that another submit queue will have the highest pressure and the next packet will be sent from this queue.

When one or more submit queues are empty, the packets are moved from holding queues into submit queues through atrioventricular valves.

4.4.1.1 Ventricular systole

Pressure in submit queues is a positive rational value. Before the first ventricular systole, the pressure is equalised by being set to unity for all submit queues so that each FEC starts with an equal chance of transmission. That is:

[3]The interpretation of the term *pressure* in this context will be described later.

$$P_0^k \leftarrow 1; \text{for } 0 < k \leq N$$

where

P_0^k is the initial pressure for FEC k (4.3)

N is the number of FECs;

Q is the maximum submit queue size.

At step i, a packet is selected from the queue with the highest pressure ($\max(P_i^k)$). When the i^{th} packet of size S_i^k is released from queue k, the pressure becomes:

$$P_i^k \leftarrow P_{i-1}^k - \frac{S_i^k}{Q}. \tag{4.4}$$

4.4.1.2 Ventricular diastole and atrial systole

These two phases are simultaneous. This happens when one or more submit queues are empty and the ventricle needs to relax so that the packets from the atrium can be pushed into the ventricle through the atrioventricular valves.

The counter is reset to 0 and the pressure in all submit queues is reset to $P_0^k \leftarrow 1 + P_i^k$, where P_i^k is the pressure for FEC k before the ventricular diastole.

4.4.1.3 Atrial diastole

The atrium must be able to receive packets continuously. Therefore, the atrium will be in a permanent diastole. The short systolic contractions will take place during the atrial diastole phase.

The hold and submit queues have limited buffer capacity as it is the space in the human heart. In the cardiovascular system if the rate of blood from the veins increases, so will the heart rate and the amount of blood entering the atrium equals the amount that leaves the ventricle. In a similar way in a network switch, the input traffic rate almost equals the output rate (small variations may be accepted, depending on the size of the holding queue). Consequently, HLFQA will

not accept packets if the holding queue is full and must be able to decide which packets to drop before entering the atrium.

4.4.1.4 Aorta

The output interface resembles the aorta in the cardiovascular system. However, the packets that leave the submit queues could be pre-buffered before sending them out through the interface. This is to avoid the idle times when the ventricle is in diastole and does not push out packets. Therefore, the output interface will always have packets to process in the buffer. In this model the shared output buffer is now the aorta as seen in Fig 4.13.

4.4.2 Evaluating the algorithm

The fairness of this scheduling mechanism derives from the fluid model used in its design. Compressing the ventricle equalises the pressure in all the submit queues (although the equalisation is never exact, given that the packet is the smallest unit that can be transmitted).

Let k and l be two FECs. T_i^k and T_i^l are the total amount of data sent for FECs k and l up to (and including) step i.

$$T_i^k = \sum_{j=1,i} S_j^k; \quad T_i^l = \sum_{j=1,i} S_j^l \tag{4.5}$$

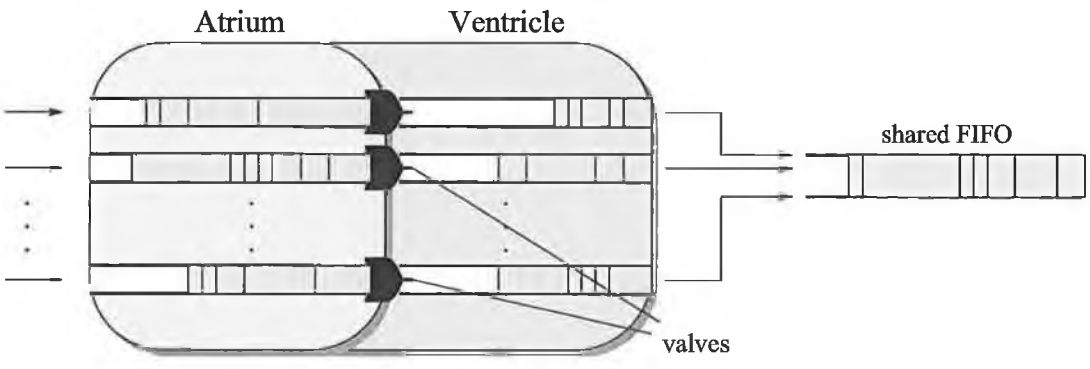


Figure 4.13: *Using a shared output FIFO as aorta*

At the beginning of each ventricular systole I have $T_0^k = T_0^l = 0$. At each step a single packet is sent out through the aorta. Therefore:

$$|T_1^k - T_1^l| \leq MAX \tag{4.6}$$

where MAX is the maximum packet size. Now, assuming that:

$$|T_{i-1}^k - T_{i-1}^l| \leq MAX \tag{4.7}$$

I want to prove that $|T_i^k - T_i^l| \leq MAX$.

If at step i no packet is sent out either from queue k or l, I have $T_{i-1}^k = T_i^k$ and $T_{i-1}^l = T_i^l$ and therefore $|T_i^k - T_i^l| = |T_{i-1}^k - T_{i-1}^l| \leq MAX$.

If at step i a packet is sent from one of the two queues, for instance queue l, it means that the pressure in queue l is greater than in queue k. That is:

$P_{i-1}^k \leq P_{i-1}^l$ and therefore,

$T_{i-1}^k \geq T_{i-1}^l$ and $|T_{i-1}^k - T_{i-1}^l| = T_{i-1}^k - T_{i-1}^l$

Because a packet is sent from queue l and no packet is sent from queue k it results that: $T_i^k = T_{i-1}^k$ and $T_i^l \leftarrow T_{i-1}^l + S_i^l$

Hence:

$$|T_i^k - T_i^l| = |T_{i-1}^k - T_{i-1}^l - S_i^l| \leq |MAX - S_i^l| \leq MAX \tag{4.8}$$

It has been proved (4.6) and from assumption (4.7) it can be derived (4.8) to be true. Hence, using mathematical induction, it has been proved that for any two queues, at any step i,

$$|T_i^k - T_i^l| \leq MAX \tag{4.9}$$

Now, if it is considered the total service provided until the moment i to be T then, the service of the ideal GPS discipline for each FEC will be $\frac{T}{N}$. But the total

amount of service is also the sum of service of all FECs:

$$T = \sum_{j=1,N} T_i^j \tag{4.10}$$

In the worst case (and using (4.9)) there is a FEC k so that

$$\sum_{j=1,N} T_i^j = N \cdot T_i^k \pm N \cdot MAX \tag{4.11}$$

From (4.10) and (4.11) it can be deduced that:

$$T = N \cdot T_i^k \pm N \cdot MAX \tag{4.12}$$

Hence, in the worst case, the difference between the service of GPS and the service of FEC k is:

$$\begin{aligned} &\frac{T}{N} - T_i^k = \frac{N \cdot T_i^k \pm N \cdot MAX}{N} - T_i^k = \\ &= T_i^k \pm MAX - T_i^k = \pm MAX \end{aligned} \tag{4.13}$$

Therefore, the fairness of the algorithm is MAX. Also, the algorithm does not keep state of previous events and the FECs are not penalised for using excess bandwidth when other FECs were idle, unlike VirtualClock [167].

4.4.2.1 Complexity of the algorithm

The complexity of scheduling is given by the number of operations required to send one packet. In the approach used by HLFQA the pressure for each submit queue is stored in a sorted array. The head of the array is the highest pressure value. A packet is sent from the submit queue with the highest pressure, and then the pressure is recalculated for that queue. This requires (based on (4.4)) only two basic operations having constant complexity. The complexity of the algorithm derives from the operation of inserting the new pressure value in a sorted array which is of $O(\log n)$ complexity. An overview of the complexity and

fairness of other scheduling algorithms was presented in Section 2.2.3.

Although there are scheduling algorithms such as Emulated Weighted Fair Queueing (EWFQ) [85] or Self-Clocked Fair Queuing (SCFQ) [69] having lower complexity ($O(1)$), the price paid is the reduced level of isolation among the sessions, causing the end-to-end delay bounds to grow linearly with the number of FECs [143].

4.4.3 Weighted scheduling

The algorithm can be modified to provide weighted fair scheduling. If there are N concurrent FECs requesting a proportion p^k of available bandwidth, where $\sum_{k=1,N} p^k = 1$, The initial pressures in submit queues can be set to:

$P_0^k \leftarrow \frac{p^k \cdot Q}{p^k \cdot Q}$ and $P_i^k \leftarrow P_{i-1}^k - \frac{S_i^k}{p^k \cdot Q}$.

This will provide service for FECs proportional with their requested p^k percent of the available bandwidth.

4.4.4 Implementing the algorithm

4.4.4.1 Storing packets

For each FEC two queues (hold and submit) are needed. Single linked lists can be used to implement the FIFO queues. Although, in the human heart, blood cells in the atrium are separated from those in the ventricle, in this implementation the linked lists of packets from atrium are linked with those in ventricle. Therefore, moving packets through the atrioventricular valve is seamless. Pointers are used to identify the first and the last packet in both hold and submit queues.

A doubly linked list stores the values of pressure in each submit queue. This is an ordered list; the values are stored in descending order.

4.4.4.2 Atrial diastole (receiving packets)

The algorithm is in a permanent atrial diastole phase because the system must be able to receive packets continuously.

It was made the assumption that packets are already classified into FECs. Therefore, buffering packets means linking every new packet at the end of its corresponding linked list.

4.4.4.3 Atrial systole and ventricular diastole (moving packets from atrium to ventricle)

When one or more submit queues are empty, packets from the atrium will flow into the ventricle. In the actual implementation, only a few pointers are changed.

The pressure must be recomputed for each queue. The complexity of this operation is $O(N)$. However this operation is performed only when a ventricular queue is empty and it is not required for the simplified version of the algorithm.

4.4.4.4 Ventricular systole

The head of queue of the sorted doubly linked list off pressure values represents the submit queue with the highest pressure. The head of queue packet is selected from that submit queue and sent out the network interface.

The pressure is recomputed only for that particular queue and the value inserted in the sorted list of pressure values. The complexity of this operation is $O(\log N)$.

4.4.5 Simulation results

Two simulations were performed . In the first one, three FECs share a link equally. Their average rate will stabilise at one third (0.33) of the bandwidth as shown in Fig. 4.14.

In the second test the flows were weighted with weights 1, 2, and 3, so they take 0.16, 0.33 and 0.5 respectively of the bandwidth. After a while the third flow stops sending packets, then the second. In Fig.4.15 it can be seen that after FEC3 stops sending packets, the remaining flows share the bandwidth with weights 1 and 2 representing now 0.33 respectively 0.66 percent of the bandwidth. When FEC1 remains alone it will use the entire bandwidth. The second simulation showed that if one (or more) FEC is idle, the unused bandwidth is evenly (proportional) distributed among remaining FECs.

4.4.6 Simplified HLFQA (s-HLFQA)

The analogy of the HLFQA algorithm with the operation of the human heart is attractive, but brings the disadvantage that two queues must be maintained per FEC. It is possible to reduce this to a single queue by appropriately modifying HLFQA to obtain a simplified (s-HLFQA) algorithm.

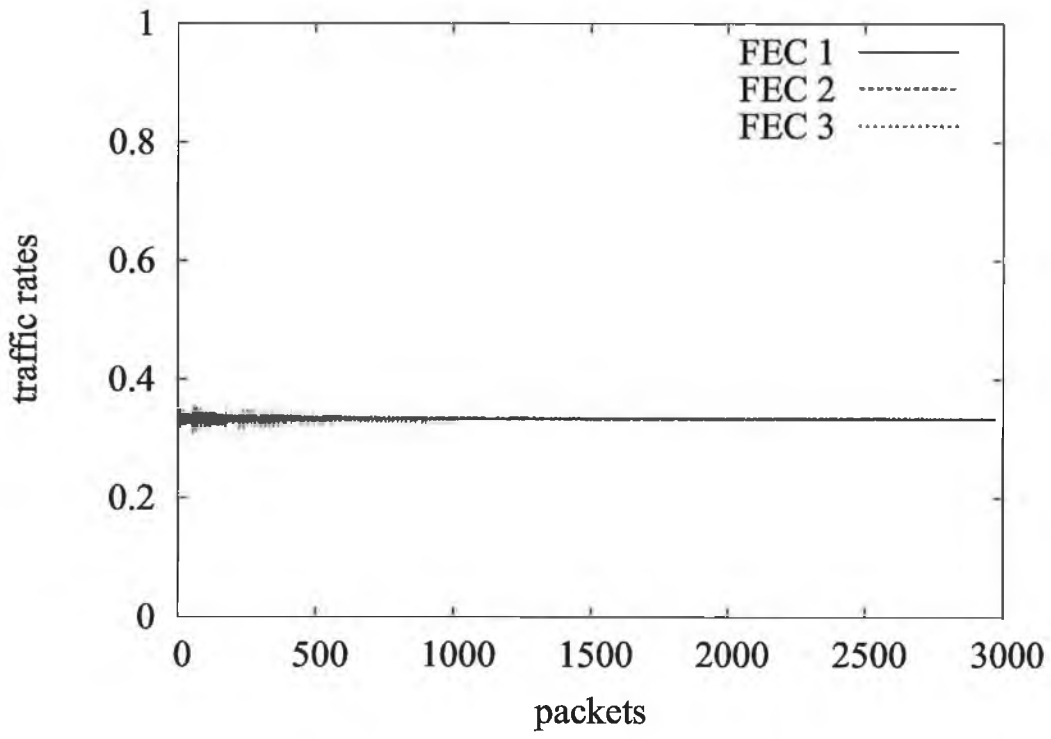


Figure 4.14: *3 FECs sharing equally 0.33 of the link*

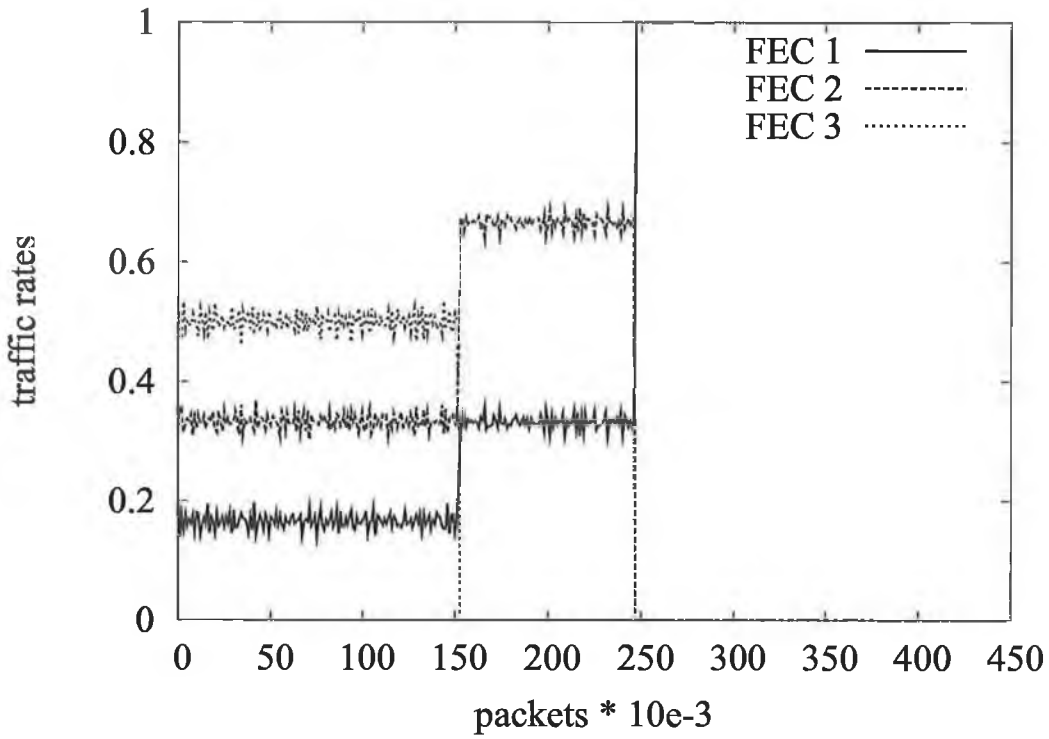


Figure 4.15: *3 weighted FECs sharing respectively 0.16/0.33/0.5 of the link then 2 flows 0.33/0.66 then 1 flow all the bandwidth*

In the simplified version, there is only one queue per FEC, a unified hold and submit queue. In this context, since packets enter arbitrarily into the queue (the heart is always open) queue pressures cannot be used in making the scheduling decision. Hence, another measure is used to decide which packet will be sent next and from which queue. While in HLFQA a packet is sent from the queue with the highest pressure, in s-HLFQA the packet is sent from the queue which has received the least amount of service.

Therefore at step i, a packet will be sent from queue k if and only if:

$$T_i^k = min(T_i^j), \quad j = 1, 2...N; \tag{4.14}$$

where T_i^k is the total amount of data sent for FEC k until step i as described in 4.5.

However, the T_i^k are continuously increasing values and therefore they can be normalised or reset to lower values when they reach an upper bound and each

time the value of N changes. The procedure is explained below: **Resetting T to lower values:** When the values T are considered too big, they can be simply reset to 0. However, to maintain the perfect fairness of the algorithm an alternative is to reset them based on the following function:

For each $k = 1, 2...N$;

$$T_i^k \leftarrow T_i^k - min(T_i^j), \quad j = 1, 2...N; \tag{4.15}$$

The second alternative does not affect the fairness of the algorithm since it maintains the differences between the values of T for any queue.

4.4.6.1 Fairness of s-HLFQA

Let k and l be two FECs. T_i^k and T_i^l are the total amount of data sent for FECs k and l until step i.

$$T_i^k = \sum_{j=1,i} S_j^k; \quad T_i^l = \sum_{j=1,i} S_j^l \tag{4.16}$$

Initially there was $T_0^k = T_0^l = 0$. At each step a single packet is sent out. Therefore:

$$|T_1^k - T_1^l| \leq MAX \tag{4.17}$$

where MAX is the maximum packet size. Now, assuming that:

$$|T_{i-1}^k - T_{i-1}^l| \leq MAX \tag{4.18}$$

I want to prove that $|T_i^k - T_i^l| \leq MAX$.

If at step i no packet is sent out either from queue k or l, then $T_{i-1}^k = T_i^k$ and $T_{i-1}^l = T_i^l$ and therefore $|T_i^k - T_i^l| = |T_{i-1}^k - T_{i-1}^l| \leq MAX$.

If at step i a packet is sent from one of the two queues, for instance queue l, it means that $T_{i-1}^k \geq T_{i-1}^l$ and $|T_{i-1}^k - T_{i-1}^l| = T_{i-1}^k - T_{i-1}^l$

Because a packet is sent from queue l and no packet is sent from queue k then: $T_i^k = T_{i-1}^k$ and $T_i^l \leftarrow T_{i-1}^l + S_i^l$

Hence:

$$|T_i^k - T_i^l| = |T_{i-1}^k - T_{i-1}^l - S_i^l| \leq |MAX - S_i^l| \leq MAX \tag{4.19}$$

It has been proven (4.17) and from assumption (4.18) it was derived (4.19) to be true. Hence, using mathematical induction, it has been proved that for any two queues, at any step i,

$$|T_i^k - T_i^l| \leq MAX \tag{4.20}$$

This is the same result as in (4.9) for HLFQA. Therefore, it can be again deduced (4.13) and so the fairness of s-HLFQA is MAX.

4.4.7 Complexity of s-HLFQA

The time complexity of s-HLFQA is given by the number of operations performed in order to send one packet from N queues.

The values T are stored in a sorted array (or list). Selecting the min from that array requires one basic operation. Another basic operation is required to increase T: $T_{i+1}^k \leftarrow T_i^k + S_i^k$. The new value of T must be inserted in the sorted array. This operation has $O(\log N)$ complexity. Therefore s-HLFQA belongs to the $O(\log N)$ class of complexity.

4.4.8 Weighted s-HLFQA

s-HLFQA can be used to provide weighted fair scheduling as well. If there are N concurrent FECs requesting a proportion of p^k of the available bandwidth, where $\sum_{k=1,N} p^k = 1$, the total service for FEC k will be recorded as follows:

$$T_i^k = T_{i-1}^k + \frac{S_i^k}{p^k} = \frac{\sum_{j=1,i} S_j^k}{p^k}. \tag{4.21}$$

From (4.20) I have $|T_i^k - T_i^l| \leq MAX$ for any $l, k \in 1, 2, ..., N$. Since $T_i^k \gg MAX$ it can be considered that

$$T_i^k \approx T_i^l. \tag{4.22}$$

Let S_i be the total amount of packets processed for all the queues until step i. That is:

$$S_i = \sum_{j=1,N} S_i^j \tag{4.23}$$

From (4.22) results: $T_i^1 \approx T_i^2 \approx ...T_i^N$ and using (4.21) it results that: $\frac{\sum_{j=1,i} S_j^1}{p^1} \approx \frac{\sum_{j=1,i} S_j^2}{p^2} \approx ... \frac{\sum_{j=1,i} S_j^N}{p^N} \approx \frac{S_i}{\sum_{j=1,N} p^j} = S_i$. Consequently, for any k I have: $\frac{\sum_{j=1,i} S_j^k}{p^k} \approx S_i$ and therefore:

$$\sum_{j=1,i} S_j^k \approx p^k \cdot S_i \tag{4.24}$$

Hence, the proportion of packets sent for FEC k is approximately equal with p^k.

4.4.9 A comparison of HLFQA and WFQ

The HLFQA and s-HLFQA scheduling algorithms achieve MAX fairness and have $O(\log N)$ computational complexity. This property is shared by the *Weighted Fair Queueing (WFQ)* algorithm described in [52]. However, unlike HLFQA, WFQ, although it does not fall behind GPS by more than MAX, can go far ahead of it [25]. HLFQA's superior performance in this regard is illustrated in the following example.

Consider a scheduler with 2 active queues. In the first queue (Q_1), there are 3 packets of size 1 (p_1^1, p_1^2 and p_1^3) and in the second queue (Q_2) there is only a single packet of size 4 (p_2^1). In WFQ, because all the packets in the first queue would complete service in GPS earlier than the packet in the second queue, all of them will be sent before any bit from the second queue is sent. The order in which the packets are sent is thus: p_1^1, p_1^2, p_1^3, p_2^1.

In HLFQA, once the packet p_1^1 completes the service, the pressure in the first

queue decreases, so the next packet will be sent from the second queue. The packet order in HLFQA will thus be: p_1^1, p_2^1, p_1^2, p_1^3, and therefore, HLFQA does not go far ahead of GPS as does WFQ. This example illustrates the property of HLFQA established in subsection 4.4.2, that it will not advance the the scheduling of packets compared to GPS by more than MAX.

4.4.10 Summary

A new scheduling algorithm, suitable for deployment in MPLS networks, has been proposed, based on an analogy with the workings of the human heart. It has been shown that the HLFQA class of algorithms achieve MAX fairness and $O(\log N)$ complexity. This is the optimal fairness that can be achieved with packet based schedulers. Scheduling algorithms such as EWFQ and SCFQ have lower computational complexity ($O(1)$). However, they do not achieve the same optimal fairness and the price paid is the reduced level of isolation among the sessions, causing the end-to-end delay bounds to grow linearly with the number of FECs. WFQ has similar properties to HLFQA in term of fairness and complexity. However, the calculations to be performed are simpler for HLFQA. A simplified implementation (having the same fairness and complexity) called s-HLFQA has also been proposed. Both algorithms are simple enough to be implemented in hardware so that wire-speed operation is possible at high bit rates.

The complexity of HLFQAs increases with the number of users[4] to be scheduled. In MPLS networks, traffic flows with similar forwarding characteristics are aggregated into FECs and thus reducing the number of users and the complexity of the scheduling algorithm. Moreover, since the packets are already classified into FECs at the MPLS ingress node, the scheduling algorithm does not need an additional packet classifier. HLFQAs can therefore take full advantage from the large scale deployment of MPLS.

[4]See Section 2.2.3.1.

4.5 Conclusions

MPLS proved itself as a scalable, flexible and robust framework on top of which QoS can be provided in the Internet core. However, in order to be able to offer end-to-end QoS guarantees, the edge network also has to be QoS aware and moreover, to be capable of exploiting the traffic engineering capabilities of the MPLS framework.

MPLS based Web switching techniques like that presented in this chapter can become the future scalable mechanisms to balance the Web requests across a cluster of servers. Such mechanisms can also be used to provide differentiated classes of Web service in an Internet that no longer provides only a best-effort service class. The grade of service negotiated can be supported even when using low cost software routers and servers.

This kind of router is particulary affected by the growing packet transmission rates in the Internet and by the per-packet processing time. Therefore, a scheme to reduce the packet rate by increasing the average frame size by aggregating traffic into MPLS meta-frames was also proposed. In conjunction with other approaches to enlarge the MTU in the Internet core, this MPLS based framework can help to achieve higher throughput by aggregating the small frames that originate in LANs with small MTUs.

Along with the large number of QoS provisioning schemes that can be deployed on top of MPLS, traffic control mechanisms such as traffic schedulers are needed to enforce the grade of service for each QoS class. A new class of traffic scheduling mechanisms that is inspired by the human heart was proposed. The fluid model of this approach achieves the maximum fairness (for a packet based scheduler) and its simplicity lends itself to hardware implementation. This solution takes full advantage of the MPLS class-based virtual circuit model, and its queuing model can help the previous mentioned framework to increase the frame size and traffic rate in the Internet.

CHAPTER 5

Conclusions

Although not designed for such uses, the Internet already transports voice, video and data in an integrated framework. Networking equipment manufacturers are struggling to cope with the ever increasing demand for fast and reliable Internet services. They are fighting on two fronts, the first of which is to provide wider bandwidth and high-speed capability in routers. But since the traffic steady growth can quickly flood any amount of bandwidth, they are also fighting to keep the traffic under control by providing means for traffic engineering.

The major companies are converging their efforts in developing an universal framework to help the deployment of Internet QoS schemes. From this work emerged MPLS, a scalable and flexible traffic engineering mechanism for datagram networks, inheriting the QoS capabilities of the virtual circuit switched ATM. MPLS deployment started in 2000 and since then, it has been used as a foundation for traffic engineering, QoS routing, VPNs, protection and restoration mechanisms, etc.

Initial efforts in deploying such schemes were concentrated on the core of the Internet. This left a gap at the edge where the traffic engineering capabilities of

MPLS are not currently exploited.

5.1 Contributions

A survey of Internet QoS strategies reveals the place of MPLS in the overall QoS picture. Architectures to increase the grade of service in the Internet are building around an MPLS framework. Although QoS can be provided in the Internet with traditional IP, MPLS provides an unified frame to integrate these architectures. Additionally, with MPLS some application level QoS mechanisms can now be offered at network level.

An example of this is the novel MPLS Web switching architecture presented in Section 4.1. The advantages of such an approach is that it can be distributed along the ingress nodes of an MPLS network, thus allowing it to be deployed using cost-effective soft routers. This architecture has been implemented and evaluated using Linux based routers and servers.

The same architecture was used to develop a framework for differentiated Web services. Using results from teletraffic engineering and queuing theory to dimension server farms, cost-effective scalable solutions can be provided to guarantee the grade of service promised to customers. The main advantage of the proposed approach is that over-provisioned resources do not remain idle (like in telecommunications) but can be used to provide best-effort Web services. This solution does not require servers to support preemption because it uses a dedicated set of servers for each class of service. Servers can migrate from one set to another when required based on a predefined set of conditions, in order to provide the guaranteed level of service while still accommodating best effort requests.

Another advantage of using MPLS based Web switching techniques is that the QoS support over MPLS can be extended from the Internet to become an end-to-end QoS scheme. MPLS capabilities for traffic engineering (such as establishing explicit LSP paths) can be exploited to differentiate both the level of service and

the path through the network for the various classes of service.

Public transportation is a solution to reducing congestion in large metropolitan areas. Hence, using subways or trains or other mass transit vehicles and dedicated lines one can travel faster from one location to another. In a similar way, to prevent congestion in Internet routers, an MPLS based framework to reduce the number of packets that need processing was proposed in Section 4.3. The MPLS meta-frame approach not only reduces the frame rate and increases the throughput but also reduces the overhead per packet.

There is a large number of QoS provisioning mechanisms available. However, they must be supported by QoS control mechanisms such as traffic schedulers. The main tradeoff in designing a traffic scheduler is between complexity and fairness. The algorithms that achieve perfect fairness (for packet based traffic) are more complex to implement. The less complex algorithms have a reduced level of isolation among the sessions, causing the end-to-end delay bounds to grow linearly with the number of traffic flows. A new class of scheduling algorithms is described in Section 4.4, intended for deployment in MPLS networks. Their operation is based on an analogy with the workings of the human heart. This class of algorithms achieves the optimal fairness for packet based schedulers and has low hardware complexity. It can be combined with the packet aggregation mechanism above to provide an effective interface between the edges of tomorrow's Internet and its high-speed core.

5.2 Future work

The Web switching architecture to provide differentiated services presented here, was based on a model for homogenous server types and requests. Future work will explore adaptive load-balance algorithms for heterogenous web clusters, and the development of a queuing model for such a Web server system. This will allow the most economic hardware to be deployed to meet the growing demand

for diverse Web services.

The meta-frame mechanism is not appropriate in all situations and the delay it introduces make it inappropriate for some real time and network control applications. Thus the decision as to whether to invoke meta-frame generation for a particular stream is a QoS issue, as is the choice of parameters (target meta-frame size and timeout). Further study will be required to see how this scheme interacts with other QoS mechanisms, and how (in an MPLS context) its use should affect how flows are aggregated into FECs.

An optimisation will be to combine the meta-frame process with HLFQA scheduling, whereby the packets queued for scheduling can be assembled into larger MPLS frames. Thus the packets need only to be queued once not separately for meta-frames and at the scheduler.

I am currently looking at ways to parallelise the algorithm. A parallel implementation should enable line rates of 40 Gb/s to be accommodated. At such rates, the scheduler will typically interface to a high-speed optical network core, where GMPLS is used to manage the combined MPLS/optical network. We are looking at how to combine the pre-buffering in HLFQAs holding queues to allow packets of the same FEC to be aggregated in larger frames (see Section 4.3) in order to increase the average frame size in the Internet core. This will result in less stringent switching requirements in the Internet core. However, packet aggregation increases the value of MAX (the maximum packet size) and thus adversely affects scheduler fairness. Selective aggregation (where packets are merged only when it is fair to do so) can address this difficulty and is a topic for future research.

5.3 Concluding remarks

The Internet required a simple but powerful traffic engineering tool. Therefore, the companies rushed to deploy MPLS even before it was completely standardised. New QoS mechanisms were quickly deployed over the MPLS framework.

The ATM and Frame-Relay forums soon realised the weight of such a label switching technology, and joined their effort with the MPLS forum. However, the QoS picture puzzle is not yet complete.

The QoS mechanisms that exploit the large scale MPLS deployment presented in this thesis complement existing QoS mechanisms being deployed in the Internet core, thus contributing to the development of an end-to-end Internet QoS scheme. As new services and new technologies appear, the main concern will be to continuously adapt current QoS mechanisms to the new environment or to discover new and more powerful tools in order to transform the Internet into a secure and robust multiservice network.

BIBLIOGRAPHY

[1] G. ABDULLA, *WWW proxy traffic characterization with application to caching*. Technical Report, Computer Science Department Virginia Technology, (CS-97-03):1–20, March 1998.

[2] M. ABRAMS, C. STANDRIDGE, G. ABDULLA, S. WILLIAMS, AND E. FOX, *Caching Proxies: Limitations and Potentials*. In Proceedings of the Fourth International World Wide Web Conference, pages 119–133, Boston, USA, December 1995.

[3] A. ACHARYA, A. SHAIKH, R. TEWARI, AND D. VERMA, *Scalable Web Request Routing with MPLS*, Tech. Report RC 22275, IBM Research Report, December 2001.

[4] G. AHN, J. JANG, AND W. CHUN, *An Efficient Rerouting Scheme for MPLS-Based Recovery and Its Performance Evaluation*, Telecommunication Systems, 3 (2002), pp. 481–495.

[5] P. ALMQUIST, *Type of Service in the Internet Protocol Suite*, RFC 1349, IETF, July 1992. Status: PROPOSED STANDARD.

[6] ALTEON NETWORKS, *Extended Frame Sizes for Next Generation Ethernets*. Whitepaper, 1998. San Jose, USA.

[7] L. ANDERSSON, P. DOOLAN, N. FELDMAN, A. FREDETTE, AND B. THOMAS, *LDP Specification*, RFC 3036, IETF, January 2001. Status: STANDARDS TRACK.

[8] D. ANDRESEN, T. YANG, V. HOLMEDAHL, AND O. H. IBARRA, *SWEB: Towards a Scalable World Wide Web Server on Multicomputers*, Tech. Report TRCS95-17, 1995.

[9] I. ANDRIKOPOULOS AND G. PAVLOU, *Supporting Differentiated Services in MPLS Networks*, in Seventh International Workshop on Quality of Service. IWQoS '99, 1999, pp. 207–215.

[10] ANRITSU CORPORATION, *Anritsu - europe.* http://www.eu.anritsu.com /products/data_communications.asp.

[11] G. APOSTOLOPOULOS, R. GUERIN, S. KAMAT, A. ORDA, AND S. K. TRIPATHI, *Intra-Domain QoS Routing in IP Networks: A Feasibility and Cost/Benefit Analysis*, IEEE Network, 13 (1999), pp. 42–54.

[12] G. APOSTOLOPOULOS, R. GUERIN, S. KAMAT, AND S. K. TRIPATHI, *Quality of Service Based Routing: A Performance Perspective*, in SIGCOMM, 1998, pp. 17–28.

[13] J. ASH, M. GIRISH, E. GRAY, B. JAMOUSSI, AND G. WRIGHT, *Applicability Statement for CR-LDP*, RFC 3213, IETF, January 2002. Status: Informational.

[14] ATM FORUM, *ATM User-Network Interface Specification , version 3.0*, Tech. Report af-uni-0010.001, ATM Forum, September 1993.

[15] ——, *Private network-network interface specification , version 1.0*, Tech. Report af-pnni-0055.000, ATM Forum, March 1996.

[16] C. AURRECOECHEA, A. T. CAMPBELL, AND L. HAUW, *A Survey of QoS Architectures*, Multimedia Systems, 6 (1998), pp. 138–151.

[17] D. AWDUCHE, L. BERGER, D. GAN, T. LI, V. SRINIVASAN, AND G. SWALLOW, *RSVP-TE: Extensions to RSVP for LSP Tunnels*, RFC 3209, IETF, December 2001. Status: Standards Track.

[18] D. AWDUCHE, A. CHIU, A. ELWALID, I. WIDJAJA, AND X. XIAO, *Overview and Principles of Internet Traffic Engineering*, RFC 3272, IETF, May 2002.

[19] D. AWDUCHE, A. HANNAN, AND X. XIAO, *Applicability Statement for Extensions to RSVP for LSP-Tunnels*, RFC 3210, IETF, December 2001. Status: Informational.

[20] D. AWDUCHE, J. MALCOLM, J. AGOGBUA, M. O'DELL, AND J. MCMANUS, *Requirements for Traffic Engineering Over MPLS*, RFC 2702, IETF, September 1999.

[21] B. AWERBUCH, Y. DU, AND Y. SHAVITT, *The Effect of Network Hierarchy Structure on Performance of ATM PNNI Hierarchical Routing*, Seventh International Conference on Computer Communications and Networks (ICCCN '98), (1998), pp. 73–80.

[22] B. BADRINATH AND P. SUDAME, *Gathercast: An efficient multi-point to point aggregation mechanism in IP networks*, 1998.

[23] F. BAKER, *Requirements for IP Version 4 Routers*, Tech. Report 1812, IETF, June 1995. Status: Standards Track.

[24] BELLCORE, *SONET common generic criteria.* GR 253 Core, December 1995.

[25] J. C. R. BENNETT AND H. ZHANG, *WF2Q: Worst-Case Fair Weighted Fair Queueing*, in proceedings IEEE INFOCOM, San Francisco, March, 1996, pp. 120–128.

[26] L. BERGER, *Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Extensions*, RFC 3473, IETF, January 2002. Status: STANDARDS TRACK.

[27] L. BERGER, D. GAN, G. SWALLOW, P. PAN, F. TOMMASI, AND S. MOLENDINI, *RSVP Refresh Overhead Reduction Extensions*, Tech. Report RFC2961, IETF, April 2001.

[28] Y. BERNET, *Networking Quality of Service and Windows Operating Systems*, Sams, 1st ed., 2000.

[29] D. BLACK, S. BRIM, B. CARPENTER, AND F. L. FAUCHEUR, *Per Hop Behavior Identification Codes*, RFC 3140, IETF, June 2001. Status: INFORMATIONAL.

[30] U. BLACK, *MPLS and Label Switching Networks*, Prentice Hall, 2001.

[31] S. BLAKE, D. BLACK, M. CARLSON, E. DAVIES, Z. WANG, AND W. WEISS, *An Architecture for Differentiated Service*, RFC 2475, IETF, December 1998. Status: INFORMATIONAL.

[32] J.-C. BOLOT AND P. HOSCHKA, *Performance Engineering of the World Wide Web: Application to Dimensioning and Cache Design*, Computer Networks and ISDN Systems, 28 (1996), pp. 1397–1405.

[33] J. BOYLE, V. GILL, A. HANNAN, D. COOPER, AND D. AWDUCHE, *Applicability Statement for Traffic Engineering with MPLS*, RFC 3346, IETF, August 2002.

[34] R. BRADEN, *Requirements for Internet Hosts – Application and Support*, Tech. Report RFC1123, IETF, October 1989.

[35] ——, *Requirements for Internet Hosts – Communication Layers*, Tech. Report RFC1122, IETF, October 1989.

[36] R. BRADEN, D. CLARK, AND S. SHENKE, *Integrated Services in the Internet Architecture: an Overview*, Tech. Report RFC1633, IETF, June 1994.

[37] T. BRADLEY, C. BROWN, AND A. MALIS, *Inverse Address Resolution Protocol*, RFC 2390, IETF, September 1998. Status: STANDARDS TRACK.

[38] L. BREIT, *Network Throughput and Performance Calculations*, July 2000. draft-breit-network-perf-throughput-00.txt.

[39] G. CARL, G. KESIDIS, R. R. BROOKS, AND S. RAI, *Denial-of-Service Attack-Detection Techniques*, Internet Computing, IEEE, 10 (2006), pp. 82–89.

[40] R. V. CHAKARAVARTHY, *Ip routing lookup: Hardware and software approach*, master's thesis, Texas A&M University, 2004.

[41] S. CHATTERJEE AND J. BYUN, *Quest for the End-to-End Network QoS*, Eighth Americas Conference on Information Systems, (2002), pp. 1919–1925.

[42] S. CHEN, *Routing Support for Providing Guaranteed End-to-End Quality-of-Service*, Ph.D. Thesis, UIUC, http://cairo.cs.uiuc.edu/papers/SCthesis.ps, May 1999.

[43] S. CHEN AND K. NAHRSTEDT, *An Overview of Quality-of-Service Routing for the Next Generation HighSpeed Networks: Problems and Solutions*, IEEE Network, 12 (1998), pp. 64–79.

[44] CISCO, *Load Balancing. A Multifaceted Solution for Improving Server Availability*. http://www.cisco.com/en/US/products/hw/contnetw/ps1894/prod_white_papers_list.html.

[45] CISCO SYSTEMS, *MPLS Virtual Private Networks*. Cisco IOS Release 12.0(5)T New Features.

[46] D. E. COMER, *Computer Networks and Internets*, Prentice Hall, 3rd ed., 2001.

[47] A. CONTA, P. DOOLAN, AND A. MALIS, *Use of Label Switching on Frame Relay Networks Specification*, RFC 3034, IETF, January 2001. Status: STANDARDS TRACK.

[48] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to Algorithms*, MIT Press and McGraw-Hill, second ed., 2001, pp. 588–592. ISBN: 0262032937.

[49] E. CRAWLEY, R. NAIR, B. RAJAGOPALAN, AND H. SANDICK, *A Framework for QoS-based Routing in the Internet*, RFC 2386, IETF, August 1998. Status: INFORMATIONAL.

[50] B. DAVIE, J. LAWRENCE, K. MCCLOGHRIE, E. ROSEN, G. SWALLOW, Y. REKHTER, AND P. DOOLAN, *MPLS using LDP and ATM VC Switching*, RFC 3035, IETF, January 2001. Status: STANDARDS TRACK.

[51] S. DEERING AND R. HINDEN, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 1883, IETF, December 1995. Status: PROPOSED STANDARD (Obsoleted by RFC2460.

[52] A. DEMERS, S. KESHAV, AND S. SHENKER, *Analysis and simulation of a fair queueing algorithm.*, SIGCOMM Symposium on Communications Architectures and Protocols, (1989), pp. 1–12.

[53] S. DHARMAPURIKAR, P. KRISHNAMURTHY, T. SPROULL, AND J. LOCKWOOD, *Deep packet inspection using parallel Bloom filters*, 2003.

[54] W. N. EATHERTON, *Hardware-based internet protocol prefix lookups*, master's thesis, Washington University Electrical Engineering Department, 1999.

[55] A. ELWALID, C. JIN, S. H. LOW, AND I. WIDJAJA, *MATE: MPLS Adaptive Traffic Engineering*, in INFOCOM, 2001, pp. 1300–1309.

[56] A. FARREL, D. PAPADIMITRIOU, J.-P. VASSEUR, AND A. AYYANGAR, *Encoding of Attributes for Multiprotocol Label Switching (MPLS) Label Switched Path (LSP) Establishment Using Resource ReserVation Protocol-Traffic Engineering (RSVP-TE)*, RFC 4420, IETF, February 2006. Status: STANDARDS TRACK.

[57] F. L. FAUCHEUR, L. WU, B. DAVIE, S. DAVARI, P. VAANANEN, R. KRISHNAN, P. CHEVAL, AND J. HEINANEN, *Multi-Protocol Label Switching (MPLS) Support of Differentiated Services*, RFC 3270, IETF, May 2002. Status: Standards Track.

[58] A. FELDMANN, *Characteristics of TCP Connection Arrivals*, 1998.

[59] D. FERRARI AND D. C. VERMA, *A Scheme for Real-Time Channel Establishment in Wide-Area Networks*, IEEE Journal on Selected Areas in Communications, 8 (1990), pp. 368–379.

[60] V. FINEBERG, *A practical architecture for implementing end-to-end QoS in an IP network*, IEEE Communications Magazine, 40 (2002), pp. 122–130.

[61] S. FLOYD AND V. JACOBSON, *Random early detection gateways for congestion avoidance*, IEEE/ACM Transactions on Networking, 1 (1993), pp. 397–413.

[62] J. FOO, *A Survey of Service Restoration Techniques in MPLS Networks*, In Proceedings of Australian Telecommunications, Networks and Applications Conference (ATNAC), Australia, Melbourne, 8-10 December 2003.

[63] B. FORTZ, J. REXFORD, AND M. THORUP, *Traffic engineering with traditional IP routing protocols*, IEEE Communications Magazine, Volume 40, Issue 10, Oct. 2002 Pages:118-124.

[64] A. GADGIL AND A. KARANDIKAR, *LiME: A Linux based MPLS Emulator*, In proceedings of The Eighth National Conference on Communications, Mumbai, India, Jan. 26 - 27, 2002.

[65] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman & Company, 1979.

[66] D. GHOSH, V. SARANGAN, AND R. ACHARYA, *Quality-of-Service Routing in IP Networks*, IEEE Transactions on Multimedia, 3 (2001), pp. 200–208.

[67] M. K. GIRISH, B. ZHOU, AND J.-Q. HU, *Formulation of the Traffic Engineering Problems in MPLS Based IP Networks*, Fifth IEEE Symposium on Computers and Communications (ISCC 2000), (2000).

[68] S. J. GOLESTANI, *A Framing Strategy for Congestion Management*, in INFOCOM, vol. 9, 1991, pp. 1064–1077.

[69] ——, *A Self-Clocked Fair Queueing Scheme for Broadband Applications*, in IEEE Journal on Selected Areas in Communications, 1994, pp. 636–646.

[70] E. W. GRAY, *MPLS. Implementing the technology*, Addison-Wesley, 2001.

[71] C. HEDRICK, *Routing Information Protocol*, Tech. Report RFC1058, IETF, June 1988.

[72] C. HUANG AND D. MESSIER, *A Fast and Scalable Inter-Domain MPLS Protection Mechanism*, Journal of Ccommunications and Networks, 6 (2004), pp. 60–67.

[73] IEEE, *IEEE P802.3ae 10Gb/s Ethernet Task Force.* `http://grouper.ieee.org/groups/802/3/ae/public/index.html`.

[74] IETF, *Multiprotocol label switching (mpls) working group.* URL: http://ietf.org/html.charters/mpls-charter.html.

[75] IETF, *QoS Routing Working Group(qosr).* http://www.ietf.org/html.charters/OLD/qosr-charter.html.

[76] ——, *The Next Steps in Signaling Working Group (NSIS).* URL: http://www.ietf.org/html.charters/nsis-charter.html.

[77] ITU, *Q.2931.* Digital Subscriber Signalling System No. 2 - User-Network Interface (UNI) layer 3 specification for basic call/connection control, February, 1995, http://www.itu.int/rec/T-REC-Q.2931.

[78] ——, *TELETRAFFIC ENGINEERING Handbook.* Study Group 2, Question 16/2, Geneva, January 2005.

[79] R. JAIN AND W. SUN, *QoS/Policy/Constraint-based routing. In Carrier IP Telephony*, Comprehensive Report, International Engineering Consortium, Heidelberg, Germany, 2000. ISBN:0933217-75-7.

[80] B. JAMOUSSI, L. ANDERSSON, R. CALLON, R. DANTU, AND L. WU, *Constraint-Based LSP Setup using LDP*, RFC 3212, IETF, January 2002. Status: STANDARDS TRACK.

[81] C. R. KALMANEK AND H. KANAKIA, *Rate Controlled Servers for Very High-Speed Networks*, Proceedings of the Conference on Global Communications (GLOBECOM), (1990), pp. 12–20.

[82] Y. KATSUBE, K. NAGAMI, AND H. ESAKI, *Toshiba's Router Architecture Extensions for ATM : Overview*, RFC 2097, IETF, February 1997. Status: INFORMATIONAL.

[83] E. D. KATZ, M. BUTLER, AND R. MCGRATH, *A Scalable HTTP Server: The NCSA prototype*, Computer Networks and ISDN Systems, 27 (1994), pp. 155–164.

[84] S. KESHAV, *An Engineering Approach to Computer Networking: ATM Networks, the Internet and the Telephone Network*, Addison-Wesley, 1997.

[85] N.-S. KO AND H.-S. PARK, *Emulated Weighted Fair Queuing Algorithm for High-Speed Packet-Switched Networks.*, in ICOIN, 2001, pp. 52–60.

[86] M. S. KODIALAM AND T. V. LAKSHMAN, *Minimum Interference Routing with Applications to MPLS Traffic Engineering*, in INFOCOM (2), 2000, pp. 884–893.

[87] T. T. KWAN, R. MCCRATH, AND D. A. REED, *NCSA's World Wide Web Server: Design and Performance*, IEEE Computer, 28 (1995), pp. 68–74.

[88] W. LAI AND D. MCDYSAN, *Network Hierarchy and Multilayer Survivability*, RFC 3386, IETF, November 2002.

[89] M. LAUBACH AND J. HALPERN, *Classical IP and ARP over ATM*, RFC 2225, IETF, April 1998. Status: STANDARDS TRACK.

[90] T. LI, *MPLS and the Evolving Internet Architecture*, IEEE Communication Magazine, 37 (1999), pp. 38–41.

[91] D. LIN AND R. MORRIS, *Dynamics of Random Early Detection*, in SIGCOMM '97, Cannes, France, September 1997, pp. 127–137.

[92] Z. LIN AND D. PENDARAKIS, *Documentation of IANA assignments for Generalized MultiProtocol Label Switching (GMPLS) Resource Reservation Protocol - Traffic Engineering (RSVP-TE) Usage and Extensions for Automatically Switched Optical Network (ASON)* , RFC 3474, IETF, March 2002. Status: INFORMATIONAL.

[93] M. LUBY, L. VICISANO, J. GEMMELL, L. RIZZO, M. HANDLEY, AND J. CROWCROFT, *Forward Error Correction (FEC) Building Block*, RFC 3452, IETF, December 2002.

[94] K.-S. LUI, K. NAHRSTEDT, AND S. CHEN, *Hierarchical QoS Routing in Delay-Bandwidth Sensitive Networks*, in Proceedings of the 25th Annual IEEE Conference on Local Computer Networks, Tampa, Florida, November 2000, pp. 579–588.

[95] E. MANNIE, *Generalized Multi-Protocol Label Switching (GMPLS) Architecture*, RFC 3945, IETF, October 2004. Status: STANDARDS TRACK.

[96] MARCEL WALDVOGEL, *Fast Longest Prefix Matching: Algorithms, Analysis, and Applications*, Ph.D Thesis, SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH, 2000.

[97] J. MARZO, P. MARYNI, AND P. VIL, *Towards QoS in IP-based core networks. A survey on performance management, MPLS case*, Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems, SPECTS'2001, July 15-19, 2001, Orlando, Florida, USA.

[98] S. MCCREARY, *Packet Length Distributions.* http://www.caida.org/analysis/AIX/plen_hist/index.xml.

[99] ——, *Packet Length Distributions.* URL:http://www.caida.org/analysis/AIX/plen_hist/index.xml.

[100] J. MCQUILLAN, I. RICHER, AND E. ROSEN, *The New Routing Algorithm for the ARPANET*, IEEE Transactions on Communications, 28 (1980), pp. 711–719.

[101] D. A. MENASCE, *Trade-offs in Designing Web Clusters*, IEEE Internet Computing, (2002), pp. 2–6.

[102] MFA FORUM, *Converged Network Services using MPLS.* URL: http://www.mfaforum.org/tech/MPLSWorldCongress2006-WhitePaper.pdf.

[103] ——, *The MFA Forum Web Page.* http://www.mfaforum.org.

[104] G. MILLER AND K. THOMPSON, *the nature of the beast: recent traffic measurements from an Internet backbone.* URL:http://www.caida.org/outreach/papers/1998/Inet98/index.xml.

[105] L. MO, *General Considerations for Bandwidth Reservation in Protection* , draft, IETF, July 2000.

[106] J. MOGUL AND S. DEERING, *Path MTU Discovery*, Tech. Report RFC1191, IETF, November 1990.

[107] J. MOY, *OSPF Version 2*, RFC 2328, IETF, April 1998. Status: STANDARDS TRACK.

[108] MPLS-RC, *The MPLS Resource Center.* http://www.mplsrc.com.

[109] G.-M. MUNTEAN AND L. MURPHY, *An Adaptive Mechanism For Pre-recorded Multimedia Streaming Based On Traffic Conditions*, W3C International World Wide Web Conference, (2002). Honolulu, Hawaii, USA, May 7-11, 2002, ISBN 1-880672-20-0.

[110] K. NAGAMI, Y. KATSUBE, Y. SHOBATAKE, A. MOGI, S. MATSUZAWA, T. JINMEI, AND H. ESAKI, *Toshiba's Flow Attribute Notification Protocol (FANP) Specification*, RFC 2129, IETF, April 1997. Status: INFORMATIONAL.

[111] J. NAGLE, *On Packet Switches With Infinite Storage*, RFC 0970, IETF, December 1985.

[112] T. NETFILTER.ORG PROJECT", *Netfilter: firewalling, NAT and packet mangling for Lunix 4.2*. http://www.netfilter.org/.

[113] P. NEWMAN, W. L. EDWARDS, R. HINDEN, E. HOFFMAN, F. C. LIAW, T. LYON, AND G. MINSHALL, *Ipsilon Flow Management Protocol Specification for IPv4 Version 1.0*, RFC 1953, IETF, May 1996. Informational.

[114] ——, *Ipsilon's General Switch Management Protocol Specification Version 1.1*, RFC 1987, IETF, August 1996. Status: INFORMATIONAL.

[115] P. NEWMAN, G. MINSHALL, AND T. LYON, *IP Switching: ATM Under IP*, IEEE/ACM Transactions on Networking, 6 (1998), pp. 117–129.

[116] K. NICHOLS, S. BLAKE, F. BAKER, AND D. BLACK, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, Tech. Report RFC2474, IETF, December 1998.

[117] NORTEL NETWORKS, *Alteon Web Switching Portfolio*. `http://www.nortelnetworks.com/products/01/alteon/acedir/index.html`.

[118] ——, *Virtual Private LAN Service(VPLS) using Distributed MPLS*. Positioning Paper, March, 2003.

[119] M. O'DELL, J. KAPLAN, J. HAYES, T. SCHROEDER, P. SINGH, D. MORRELL, AND J. HSU, *Extended Ethernet Frame Size Support*. draft-kaplan-isis-ext-eth-02.txt, May, 2002.

[120] A. ORDA AND A. SPRINTSON, *QoS Routing: The Precomputation Perspective*, in INFOCOM, vol. 1, 2000, pp. 128–136.

[121] A. PAREKH AND R. GALLAGER, *A generalized procesor sharing approach to flow control - the single node case*, Proceedings of INFOCOM'92, 2 (1992), pp. 915–924.

[122] J. POSTEL, *DoD standard Internet Protocol*, RFC 0760, IETF, January 1980. Obsoleted by RFC0791.

[123] ——, *Internet Protocol*, RFC 0791, IETF, September 1981. Status: STANDARD.

[124] S. B. R. BRADEN, L. ZHANG, *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*, RFC 2205, IETF, September 1997. Status: PROPOSED STANDARD.

[125] R. RABBAT, K. P. LABERTEAUX, N. MODI, AND J. KENNEY, *Traffic Engineering Algorithms Using MPLS for Service Differentiation*, in ICC (2), 2000, pp. 791–795.

[126] REDHAT LINUX, *Red Hat – Linux, Embedded Linux and Open Source Solutions*. URL:http://www.redhat.com.

[127] Y. REKHTER, B. DAVIE, D. KATZ, E. ROSEN, AND G. SWALLOW, *Cisco Systems' Tag Switching Architecture Overview*, RFC 2105, IETF, February 1997. Status: INFORMATIONAL.

[128] Y. REKHTER AND P. GROSS, *Application of the Border Gateway Protocol in the Internet*, Tech. Report 1772, IETF, March 1995. Status: Standards Track.

[129] Y. REKHTER AND T. LI, *A Border Gateway Protocol 4 (BGP-4)*, Tech. Report 1771, IETF, March 1995. Status: Standards Track.

[130] Y. REKHTER AND E. ROSEN, *Carrying Label Information in BGP-4*, RFC 3107, IETF, May 2001. Status: STANDARDS TRACK.

[131] E. ROSE AND Y. REKHTER, *BGP/MPLS VPNs*, RFC 2547, IETF, March 1999. Status: INFORMATIONAL.

[132] E. ROSEN, D. TAPPAN, G. FEDORKOW, Y. R. D. FARINACCI, T. LI, AND A. CONTA, *MPLS Label Stack Encoding*, RFC 3032, IETF, January 2001. Status: STANDARDS TRACK.

[133] E. ROSEN, A. VISWANATHAN, AND R. CALLON, *Multiprotocol Label Switching Architecture*, RFC 3031, IETF, January 2001. Status: STANDARDS TRACK.

[134] R. H. S. DEERING, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 2460, IETF, December 1998. Status: DRAFT STANDARD.

[135] R. G. S. SHENKER, C. PARTRIDGE, *Specification of Guaranteed Quality of Service*, RFC 2212, IETF, September 1997. Status: PROPOSED STANDARD.

[136] A. SEN, I. MOHAMMED, R. SAMPRATHI, AND S. BANDYOPADHYAY, *Fair Queuing with Round Robin: A New Packet Scheduling Algorithm for Routers*, in "Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02)", 2002.

[137] V. SHARMA AND F. HELLSTRAND, *Framework for Multi-Protocol Label Switching (MPLS)-based Recovery*, RFC 3469, IETF, February 2003. Status: INFORMATIONAL.

[138] M. SHREEDHAR AND G. VARGHESE, *Efficient Fair Queueing Using Déficit Round Robin*, in SIGCOMM, 1995, pp. 231–242.

[139] L. SLOTHOUBER, *A Model of Web Server Performance*, Proc. of 5 th Conference On WWW, (1996).

[140] SOURCE FORGE, *MPLS for Linux - Project*. URL: http://sourceforge.net /projects/mpls-linux.

[141] V. SRINIVASAN, G. VARGHESE, S. SURI, AND M. WALDVOGEL, *Fast and Scalable Layer Four Switching*, in Proceedings of ACM SIGCOMM '98, September 1998, pp. 191–202.

[142] W. STALLINGS, *ISDN and Broadband ISDN with Frame Relay and ATM*, Prentice Hall, 3rd ed., 1995.

[143] D. STILIADIS, *Traffic Scheduling in Packet-Switched Networks: Analysis*, 1996.

[144] P. SUDAME AND B. R. BADRINATH, *Transformer Tunnels: A Framework for Providing Route-Specific Adaptations*, pp. 191–200.

[145] S. SURI, M. WALDVOGEL, AND P. R. WARKHEDE, *Profile-based routing: A new framework for MPLS traffic engineering*, in Quality of future Internet Services, F. Boavida, ed., no. 2156 in Lecture Notes in Computer Science, Berlin, Sept. 2001, Springer Verlag, pp. 138–157.

[146] C. SYSTEMS, *Policy-based routing*, white paper, Cisco Systems, August 2002.

[147] A. S. TANENBAUM, *Computer Networks*, Prentice Hall, 4th ed., 2003.

[148] TELECOM TRAFFIC ONLINE, *What is an Erlang*. `http://www.erlang.com/whatis.html`. Online; accessed 23-July-2006.

[149] THE APACHE SOFTWARE FOUNDATION, *Apache HTTP Server Project*. URL: http://httpd.apache.org.

[150] ——, *httpd.conf: Standard Apache Configuration File for Linux HTTP Server*. URL: http://httpd.apache.org.

[151] THE FREE SOFTWARE FOUNDATION, *Mirrors of www.gnu.org*. URL: http://www.gnu.org/server/list-mirrors.html.

[152] B. THOMAS AND E. GRAY, *LDP Applicability*, RFC 3037, IETF, January 2001. Status: INFORMATIONAL.

[153] Y. TIAN, *A Survey on Connection Admission Control in ATM Networks*, In Proceedings of DePaul CTI Research Symposium, Chicago, IL, November 1999.

[154] C. VILLAMIZAR, *OSPF Optimized Multipath (OSPF-OMP)*, February 1999. draft-ietf-ospf-omp-02.

[155] A. VIRK AND R. BOUTABA, *Economical Protection in MPLS Networks*, Journal of Computer Communications, Elsevier, Volume 29, Issue 3, pp. 402-408, February 2006.

[156] A. VISWANATHAN, N. FELDMAN, R. BOIVIE, AND R. WOUNDY, *ARIS: Aggregate Route-Based IP Switching*, Tech. Report draft-viswanathan-aris-overview-00, IETF, March 1997. Status: expired.

[157] A. VISWANATHAN, N. FELDMAN, Z. WANG, AND R. CALLON, *Evolution of Multi-Protocol Label Switching*, IEEE Communication Magazine, 36 (1998), pp. 165–173.

[158] Y. WANG AND Z. WANG, *Explicit Routing Algorithms for Internet Traffic Engineering*, Proc. of IEEE ICCCN, New York, USA, (1999).

[159] Z. WANG AND J. CROWCROFT, *Quality-of-Service Routing for Supporting Multimedia Applications*, IEEE Journal of Selected Areas in Communications, 14 (1996), pp. 1228–1234.

[160] M. WELZL, L. FRANZENS, AND M. MÜHLHÄUSER", *Scalability and Quality of Service: A trade-off?*, IEEE Communications Magazine, 41 (2003).

[161] WIKIPEDIA, *Busy hour – Wikipedia, The Free Encyclopedia.* `http://en.wikipedia.org/w/index.php?title=Busy\_hour&oldid=57992813`, 2006. Online; accessed 23-July-2006.

[162] J. WROCLAWSKI, *Specification of the Controlled-Load Network Element Service*, RFC 2211, IETF, September 1997. Status: PROPOSED STANDARD.

[163] X. XIAO, A. HANNAN, B. BAILEY, AND L. NI, *Traffic Engineering with MPLS in the Internet*, IEEE Network, (2000).

[164] X. XIAO, L. NI, AND V. VUPPALA, *A Comprehensive Comparison of IP Switching and Tag Switching*, in IEEE Intl. Conf. on Parallel and Distributed Systems (ICPADS '97), Seul, december 1997, pp. 669–675.

[165] X. XIAO AND L. M. NI, *Internet QoS: A Big Picture*, IEEE Network, 13 (1999), pp. 8–18.

[166] X. XIAO, T. TELKAMP, V. FINEBERG, C. CHEN, AND L. M. NI, *A practical approach for providing QoS in the Internet backbone*, IEEE Communications Magazine, 40 (2002), pp. 56–62.

[167] L. ZHANG, *"virtualclock: A new traffic control algorithm for packet switching networks"*, "ACM Transactions on Computer Networks", 9 (1991), pp. 101–124.

[168] L. ZHANG, S. DEERING, D. ESTRIN, S. SHENKER, AND D. ZAPPALA, *RSVP: A New Resource ReSerVation Protocol*, IEEE Network Magazine), 7 (1993), pp. 8–18.

[169] J. A. ZUBAIRI, *An Automated Traffic Engineering Algorithm for MPLS-Diffserv Domain*, Proc. Applied Telecommunication Symposium, pp43-48, ASTC'02 Conference, San Diego, April 2002.