SCALABLE QOS ROUTING IN MPLS NETWORKS USING MOBILE CODE

By

Sanda-Maria 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: 50161180

Date: 26/07/2006

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 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 Muntean and Dr. Cristina Muntean who offered us (me and my husband) unconditional friendship and support.

Support and encouragements from both my colleagues and my students at *"Babeş-Bolyai" University* is gratefully acknowledged. I specially want to thank Prof. Florian Mircea Boian and Prof. Leon Ţâmbulea for believing in me.

I would like to acknowledge the debt I owe to Sergio González-Valenzuela for teaching/helping me how to program in WAVE and offering moral support at critical and opportune times. Many thanks!

I thank Dr. Peter Sapaty for creating WAVE. Coming across with this technology and the collaboration with Dr. Sapaty has had a decisive influence on the direction of this research. He has also given useful comments on my texts as my external Viva examiner. Other valuable comments and suggestions have come from my internal Viva examiner Prof. Barry McMullin.

The most important support I have received from my beloved husband, Radu Dragoş. He 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 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 sister Liana and my brother Silviu 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 *Sanda-Maria Dragoş*

Abstract

In a continually evolving Internet, tools such as *Quality of Service routing* must be used in order to accommodate user demands. However, deploying and developing QoS routing in the legacy Internet is difficult. Multiprotocol Label Switching (MPLS) facilitates the deployment of QoS routing, due to its separation of functions between the control and forwarding plane. Developing QoS routing raises scalability issues within very large networks. I propose overcoming these issues by using topology aggregation and distributed routing based on modern techniques such as active networks and mobile agents. However, topology aggregation introduces inaccuracy, which has a negative impact on QoS routing performance. To avoid such problems I propose a hierarchical routing protocol, called *Macro-routing*, which by using distributed route computation is able to process more detailed information and thus to use the most accurate aggregation technique, i.e. Full-Mesh. Therefore, the protocol is more likely to find the best path between source and destination, and can also find more than one available path.

QoS routing, which is used for finding feasible paths that simultaneously satisfy multiple constraints, is also called multiple-constrained routing and is an NP-complete problem. The difficulty of solving such problems increases in a hierarchical context, where aggregation techniques influence the path computation process. I propose a new aggregation technique which allows the selection of multiple paths that satisfy multiple QoS constraints. This reduces the probability of a false negative, i.e., of the routing algorithm incorrectly reporting that no path satisfying the constraints exists. This aggregation technique is called extended full-mesh (EFM) and is intended for use with the Macro-routing protocol. Deploying these protocols in the Internet will allow multi-constrained routing to be practically implemented on large networks.

List of Publications

- **S. DRAGOŞ** AND M. COLLIER, *Multi-constraint Macro-routing by using the Extended Full-Mesh aggregation technique*, in 2nd International Conference on Intelligent Computer Communication and Processing (ICCP), Cluj-Napoca, Romania, September 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.
- **S. DRAGOŞ** AND M. COLLIER, *Macro-routing: a new hierarchical routing protocol*, in IEEE Global Telecommunications Conference (GLOBECOM), vol. 3, Dallas, Texas, USA, November/December 2004, pp. 1510–1514.
- **S. DRAGOŞ** AND M. COLLIER, *A proposed architecture for integrating Active Networks and MPLS*, in 10th International Conference on Software, Telecommunications & Computer Networks, Croatia-Italy, October 2002, pp. 386–390.
- 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 FIGURES

LIST OF TABLES

PSEUDO-CODE LISTINGS

CHAPTER 1

Introduction

> *"...communication is both a human need and an indispensable part of modern society."* [83]

From the beginning of mankind people started to work on their communications skills. First it was *language*, followed by *writing*. Then there were *messengers*, followed by the *fire signals* and then there was the *morse code* and the *telegraph*. This evolution reached its apogee in the twentieth century also known as the *"information era"*, where technologies such as telephony, radio, television, computers and computer networks all deal with collecting, processing and distributing information.

The trend for all these technologies is now to converge. This convergence started by the mid 1980s, when there were three worldwide communication networks. These were the telephony network which carried voice, the television network which carried video and the Internet which carried data. At that point the telecom community decided to develop a new network which would transport all three types of traffic. The result of their work was the *Asynchronous Transfer Mode* (ATM) network which was an important step forward but proved not to be

a total success, because the *Internet Protocol* proved more popular for computer communications. ATM, however, brought a very new principle in computer networks: the idea of *Quality of Service* (QoS) provision.

By the mid 1990s, both datacom and telecom researchers competed in developing one network protocol which would have the advantages of both worlds: the simplicity and flexibility of IP networks and the provision of Quality of Service guarantees of ATM networks. This competition led to the emergence of MPLS which brought the separation of routing and forwarding and developed a totally new hierarchical forwarding mechanism.

1.1 Motivation

Although the youngest (1960s) technology of the *information era*, the computer networks had experienced an unexpected success. This success led to a multitude of networks being developed, which was the point from where the *Internet* emerged. From a research prototype, the Internet expanded, in a very short time, into a global communication system that nowadays reaches all countries of the world.

The main challenges the Internet currently faces are all related to its increasing size on different levels. First there is the exponentially increasing number of hosts which evolves at surprising rates (i.e. in July 1993 it included around 1.7 million hosts, in July 1998 it already comprised more than 36 million hosts, while the latest statistics show that in January 2006 it had more than 394 million hosts[1]). Second there is its increased complexity and importance. That is because the Internet serves as the common ground for the convergence of all communication types (i.e. voice, video and data).

Thus, numerous solutions and technologies were proposed as a response to the challenge of Internet's rapid growth in number of users and increasing re-

[1]Internet Domain Survey, http://www.isc.org/index.pl?/ops/ds/reports/2006-01/

quirements for service quality, reliability, and efficiency.

The research described in this thesis encompasses five of such technologies which are major areas of current research in network communications: network routing, Quality-of-Service (QoS), Multi-Protocol Label Switching (MPLS), active networks and mobile software agents.

1.2 Thesis contributions

1.2.1 Problem statement

The focus of the present thesis is to find the optimal solution for implementing efficient QoS routing strategies for large networks. This work has two aims:

- to minimise the overhead introduced by QoS in the routing process;
- to identify improved approaches to the multi-constrained routing problem.

1.2.2 Solutions

Deploying QoS routing schemes in the legacy Internet can be very difficult. MPLS facilitates this by its separation between the control and forwarding planes. Moreover, traditional QoS routing schemes which use source routing generate considerable overhead, i.e.:

- *communication overhead*, as routing state information has to be disseminated more frequently and the state information is more consistent (e.g., more metrics can be considered) than in legacy/nonadaptive routing;
- *computational overhead*, as the entire path is centrally computed at the source node using the "collected" global state;
- *storage overhead*, as all the state in the network has to be stored on every node that participates in the routing process.

The approaches presented here will use distribute routing implemented by using modern techniques such as active networks and mobile agents. This will alleviate the storage and the computational overhead requirements. Also, such distributed routing algorithms use only local states, thereby eliminating the communication overhead generated by dispatching routing information. However, by using mobile agents, the mobile agent population generated to search multiple feasible paths can itself generate a significant communication overhead. The active network approach, although it does not generate as great a communication overhead, can find only a single path, which might not be the optimal one. The preferred choice for implementing QoS routing protocols is, in this author's view, to use mobile agents. However, there are situations when active networks prove to be a very useful tool. To exploit the potential of active networks I propose the integration of MPLS and active networks in order to overcome the inability of MPLS to perform switching above layer two in access areas, where such operations are needed.

QoS routing for large networks is still a research area. I propose a new hierarchical QoS routing protocol, called *Macro-routing*, which is able to find multiple end-to-end paths. The use of local states allows the Full-Mesh aggregation method to be used. This is recognised by the research community as being the most accurate method available but is considered to be impractical because it generates too much overhead in the form of state advertisements. Using mobile agents, such advertisements are not needed as the routing information is consulted "in-situ". I also propose to use mobile agents not only for the routing purposes, but also for resource reservation and path setup.

Finding paths based on multiple constraint is an NP-complete problem. Many approximations and heuristical solutions have been proposed by the research community. For hierarchical multi-constraint routing the problem is even more complex as it is required to select a single path between any pair of border nodes for building the aggregate representation. Therefore, I propose a new aggrega-

tion method, called *Extended Full Mesh (EFM)*, which is a compromise between Full-Mesh aggregation and no aggregation at all. This technique stores the cost of more than one path between the two border nodes when performing aggregation, which considerable increases the chances of finding a final multi-constraint path.

1.2.3 Summary of Contributions

The main contributions presented in this thesis are:

- A comparison between active networks and mobile agents as techniques to implement QoS routing. I present the advantages and disadvantages of both technologies. I conclude that mobile agents are the preferred option for implementing efficient QoS routing strategies. The applicability of active networks for routing purposes is limited because their actions occur only as the traffic flows through the network and not beforehand as in most QoS routing schemes.

- The integration of active networks and MPLS in order to overcome the inability of MPLS to perform switching above layer two. I prove that such integration is possible and give some examples to illustrate its usefulness.

- A hierarchical QoS routing protocol, called *Macro-routing* which reduces the computational and storage overhead, introduced by QoS routing, by using distributed routing. This protocol also finds multiple feasible paths with no added cost.

- An aggregation method, called *Extended Full-Mesh (EFM)*, which performs better that Full-Mesh aggregation for finding multi-constrained paths and increases the chances of finding a multi-constrained path. I also propose different path selection methods which can be used by EFM.

1.3 Thesis outline

The remainder of this thesis is organised as follows.

Chapter 2 introduces the five main topics that underpin the research presented in this thesis, i.e. routing, Quality of Service (QoS), MultiProtocol Label Switching (MPLS), mobile agents and active networks.

Chapter 3 presents the main QoS routing challenges and existing solutions. Topology aggregation and efficient distribute routing mechanisms are the preferred approaches to overcome the main QoS routing problems by improving the scalability and reducing the computational overhead respectively.

Chapter 4 compares active networks and mobile agents as techniques to implement QoS routing and presents some application examples for which active networks prove suitable. This chapter concludes by arguing that mobile agents are the preferred choice for implementing efficient QoS routing mechanisms.

Chapter 5 proposes a new hierarchical routing protocol, called *Macro-routing*, which reduces the computational and storage overhead introduced by QoS routing, while also finding multiple feasible paths. This comes at the price of a potentially large communication overhead. Thus, techniques for limiting this communication overhead are developed.

Chapter 6 proposes a new aggregation method, called *Extended Full Mesh (EFM)*, which is able to obtain better results while searching for multi-constrained paths by using Macro-routing. Path selection mechanisms are also presented and compared through simulations.

Chapter 7 concludes the thesis with a summary of this work along with some suggestions for future research.

CHAPTER 2

Routing primitives in data networks

"Before building the network of the future, we must first understand what exists." S. Keshav [91]

The work presented in this thesis encompasses five major areas of study in network communications: routing, Quality of Service, Multi-Protocol Label Switching, Active Networks and Mobile Software Agents. To understand how these technologies can interact together to reach the goal of providing better QoS routing mechanisms, it is first necessary to understand them, their evolution over the last few years, as well as their strengths and relative weaknesses.

2.1 Routing

Routing and forwarding is what the Internet is all about: How can an IP packet from one host be delivered to the destination host? says Adrian Farrel in [65].

The formal definition of routing describes it as being the process of determining an end-to-end path between a source and a destination machine [67, 152]. This process is supported by *routing protocols*, which allow routers to communi-

cate with each other, exchanging information such as connectivity and link states so that they build up a picture of the whole network and can choose the best way to forward a packet. Based on this information *routing algorithms* determine the best path along which to forward a packet. The results of routing algorithms are used to create and update the *routing tables* that are used later in the *forwarding* process.

Routing evolved with advancing technology and changing networking concepts, providing us with a choice of routing strategies. Some of them are presented in the remainder of this section.

2.1.1 Static vs. Dynamic Routing

In *static* routing, routing tables are usually manually configured and the routes are not modified unless an error is detected. This type of routing was used in the early stages of the Internet, when routing tables were relatively small. Now, it is used only at the edge of the Internet, in small networks. The early telephone network also used static (hierarchical) routing [13], but because routing did not depend on the state of the network or the time of day, the network had to be overprovisioned so that it could carry user traffic during a busy hour of an average day. Static routing is simple, easy to specify and it does not generate network overhead as no routing information needs to be disseminated. However, it is relatively inflexible as static routes require manual actions in the case of network failures or changes in topology.

Most networks, however, use *dynamic* routing because it allows the network to handle problems automatically. A routing protocol is used to build and maintain dynamic routing tables. Moreover, the network traffic can be monitored as well as the status of the network hardware and routes can then be modified accordingly.

2.1.2 Nonadaptive vs. Adaptive Routing

The first routing protocols did not regard the state of the network while building routing tables. Thus, they are considered *nonadaptive* routing protocols. Because there are usually multiple paths between a source and a destination, only one has to be selected. The selection mechanism was initially based on the number of hops, designing the shortest path to be the winner of a path contest. This generated the family of *shortest path algorithms*. Two of the best known algorithms belong to this family: the Dijkstra algorithm [60] and the Bellman-Ford algorithm [18].

The early ARPANET was the first to recognise the importance of *adaptive routing* where routing decisions were based on the current state of the network. Therefore, each node maintained a table of network delays, representing estimated delays that packets would experience along different paths toward their destinations. The minimum delay table was periodically transmitted by each node to its neighbours. The shortest path, in terms of hop count, was also propagated to give the connectivity information. One drawback to this approach is that congestion shifts from one region of a network to another, resulting in oscillation and network instability.

This can be considered the opening attempt to provide QoS by routing mechanisms. It was not however typical of QoS routing methods because it was not connection-oriented and it was not supported by an appropriate resource reservation mechanism.

2.1.3 Hop-by-hop vs. Source Routing

In *hop-by-hop* routing, each router makes its routing decisions independently, based on the information residing in its routing tables. Usually, the next hop routing decision does not depend on the packet's original source or on the path the packet has taken before it arrives to a particular network node. Instead, the next hop to

which a packet is sent depends only on the packet's destination. This concept is called *source independence* and allows a compact and efficient forwarding mechanism. However, hop-by-hop routing as with most distributed routing mechanisms may generate looping paths, and thus other solutions like source routing are sometimes preferred.

In *source* routing each router selects a complete path from the source to the destination based on the global state[1] information. Such information has to be distributed among all nodes of the network and it may contain topology information and the state of every link. The centralised route computation (at the source node) will eliminate the possibility of a looping path. However, it implies a higher network overhead than in hop-by-hop routing as all the routing information has to be distributed to all network nodes.

2.1.4 Distance-vector vs. Link-state Routing

The *distance-vector* mechanism is the simplest way to distribute network connectivity information. Its name was generated by the form of update messages sent from one network node to another, which contain pairs of values specifying a destination and a distance to that destination. The *distance* to a destination is defined to be the sum of weights assigned to network links along the path to that destination. Each network node periodically sends update messages across the network to neighbours. Each network node that receives update messages examines each item in the message, and changes its routing table if the neighbour has a shorter path to some destination than the path the current node has been using. Distance-vector routing is based on the Ford-Fulkerson [68] algorithm and it was first used within ARPANET and then in the Internet under the name of *Routing Information Protocol* (RIP) [118]. Distance-vector routing protocols converge very slowly following changes in the network due to their count-to-infinity[2] problem.

[1] See Section 2.2.2.1

[2] The routing information is passed in a circular manner through multiple network nodes. Each traversed node increments the metric appropriately and passes it on. As the metric is passed

An alternative for distributed route computation is *link-state* routing, which is informally known as *Shortest Path First* or SPF routing because it is based on Dijkstra's algorithm [60]. The main difference between these two approaches is that link-state messages do not contain information from routing tables as in distance-vector routing; instead each message carries the status of a link between two network nodes. Moreover, these status messages are broadcast to all network nodes. Each network node collects incoming status messages and uses them to build a graph of the network which is then used to build a routing table with itself as source. Link-state routing is widely used in the Internet. The most widely used routing protocol within the Internet, called *Open Shortest Path First* (OSPF) [122] is a link-state routing protocol. Another link-state routing protocol used within the Internet is the *Intermediate System-Intermediate System* (IS-IS) [36] protocol.

2.1.5 Flat vs. Hierarchical Routing

Flat routing is the process of distributing routing information and finding paths between a source and a destination, while no other network organisation is required (i.e. all routers may be visualised as sitting on a flat geometric plane). It can be used within relatively small networks. As networks grow in size, the routing table grows proportionally. This results in memory consumed by large routing tables, more CPU time required to scan them and more bandwidth needed to send status reports about them. The network may grow to the point where it is no longer feasible for every router to have an entry for every other router. A solution to this scalability problem is *hierarchical* routing, where the entire network is divided into *domains* and a network node will have detailed information only about the nodes from its domain and aggregated information about nodes outside its domain. Hierarchical routing protocols are already used in the Internet (e.g., IP networks, ATM networks).

around the loop, it increments to ever increasing values until it reaches the maximum for the routing protocol being used, which typically denotes a link outage.

2.1.5.1 Internet Routing

The Internet uses a two level routing hierarchy. Routers and networks in the Internet are grouped into *Autonomous Systems* (ASs). An Autonomous System (AS) is described in [49] as a contiguous set of networks under the control of one administrative authority, while [53] defines an AS as a routing domain which has a common administrative authority and a consistent internal routing policy. Routers within an AS exchange routing information, which is then summarised before being passed to another group. Routing protocols used within ASs are called *Interior Routing Protocols* (IGPs), while routing protocols used between different ASs are called *Exterior Routing Protocols* (EGPs) as seen in Figure 2.1.

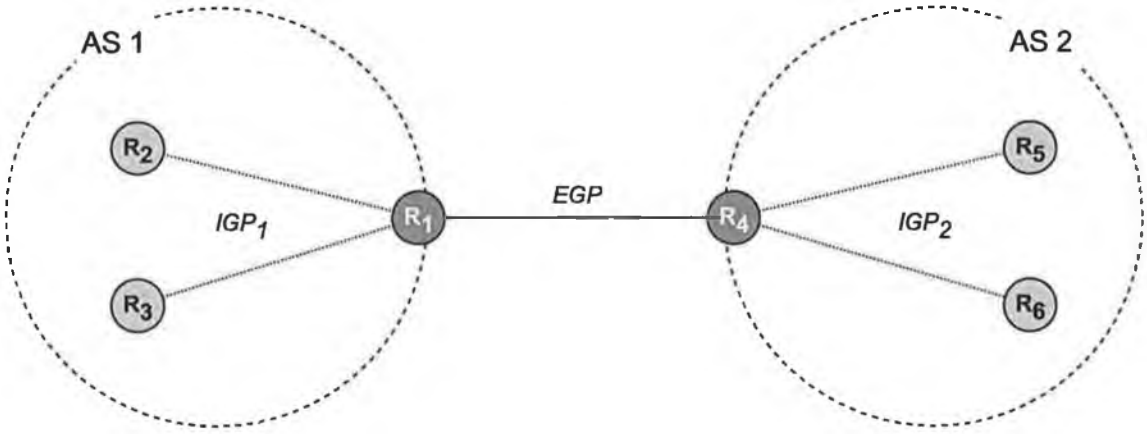


Figure 2.1: *The Internet routing architecture*

Interior Routing Protocols (IGPs). Routers within an AS use an IGP to exchange routing information. Then, on each router, the IGP's routing algorithm uses this information to choose an optimal path based on a specific metric[3]. Each autonomous system is free to choose which IGP to use. Two of the most popular IGPs are the *Routing Information Protocol* (RIP) and the *Open Shortest Path First* (OSPF).

RIP [118] was the original IGP written for the Internet. Thus, it is a very simple protocol that uses a *distance-vector* algorithm to propagate routing information

[3]Typical Internet routing uses a combination of two metrics: *administrative cost* and *hop count* [53]

and the Bellman-Ford [18] algorithm to determine the shortest path. Although it is still used in today's Internet, RIP performs poorly in large and complex networks as it inherits some of the disadvantages of distance-vector protocols. Each message contains a complete list of destinations and their distances, and therefore, messages are large. Moreover, the receiving router has to compare each entry in the incoming message with entries in its routing table. Thus, processing a message consumes CPU cycles and introduces delay. Such delay means that route changes propagate slowly [49]. Therefore, many Internet Service Providers prefer more elaborate protocols from the *link-state* family.

OSPF [122] is a link-state routing protocol and RIP's successor. Developed by the OSPF working group of the Internet Engineering Task Force, it became a standard in 1990. It is so-called because it is an *open*[4] protocol that uses the "Shortest Path First" algorithm developed by E.W. Dijkstra [60]. OSPF has several features that make it a more complex and powerful routing protocol than its predecessor, RIP, or other routing protocols. Some of them are mentioned here.

- OSPF supports a variety of distance metrics like *physical distance* or *delay*, while RIP always measures distance in network *hops*;
- OSPF adapts quickly to changes in the topology [122], while RIP has the *count-to-infinity* problem which slows its convergence following changes in the network;
- OSPF uses an authenticated message exchange to ensure that messages come from a trusted source, while RIP uses unreliable transport (i.e. UDP) for all message transmissions;
- OSPF performs hierarchical routing by allowing a manager to partition the routers within an AS into multiple *areas*, and therefore it can scale much better than other IGPs [49].

[4]OSPF is published in the *open* literature, i.e. any person can be a member of the OSPF working group and be allowed to see the code source, to discuss it with other members and make contributions to the development of the protocol.

Exterior Routing Protocols (EGPs). EGPs are used for inter-AS routing. Although usually more complex to install and operate than IGPs, EGPs offer more flexibility and lower overhead (i.e. less traffic). The main role of an EGP is twofold:

- to summarise routing information from ASs before passing it to other ASs;
- to implement *policy constraints* that allow the traffic that crosses certain ASs to be controlled.

Unlike IGPs, which choose an optimal path based on a routing metric, EGPs cannot find an optimal path. That is because different autonomous systems may use different routing metrics. Thus, an EGP can report only the existence of a path and not its cost.

The most popular EGP in the Internet is the *Border Gateway Protocol* (BGP) [133]. Now at its fourth version, BGP is very important in the global Internet because all major ISPs are using it to exchange routing information. BGP is fundamentally a distance-vector protocol, but quite different from most others such as RIP. Instead of maintaining just the cost to each destination, each BGP router keeps track of the path used. And instead of periodically giving each neighbour its estimated cost to each possible destination, each BGP router tells its neighbours the exact path it is using. By using this strategy, BGP solves the count-to-infinity problem that plagues other distance-vector routing algorithms.

2.1.5.2 Routing in ATM Networks

The Private Network-to-Network Protocol (PNNI) [9] is a hierarchical link-state routing protocol used in Asynchronous Transfer Mode (ATM) networks. It allows up to 104 levels in the hierarchy, thus being able to scale to very large networks. Nodes at a specific level are grouped into Peer Groups (PGs). A single node elected among the nodes of the PG, called the Peer Group Leader (PGL), represents each PG into the next level of the hierarchy as depicted in Figure 2.2.

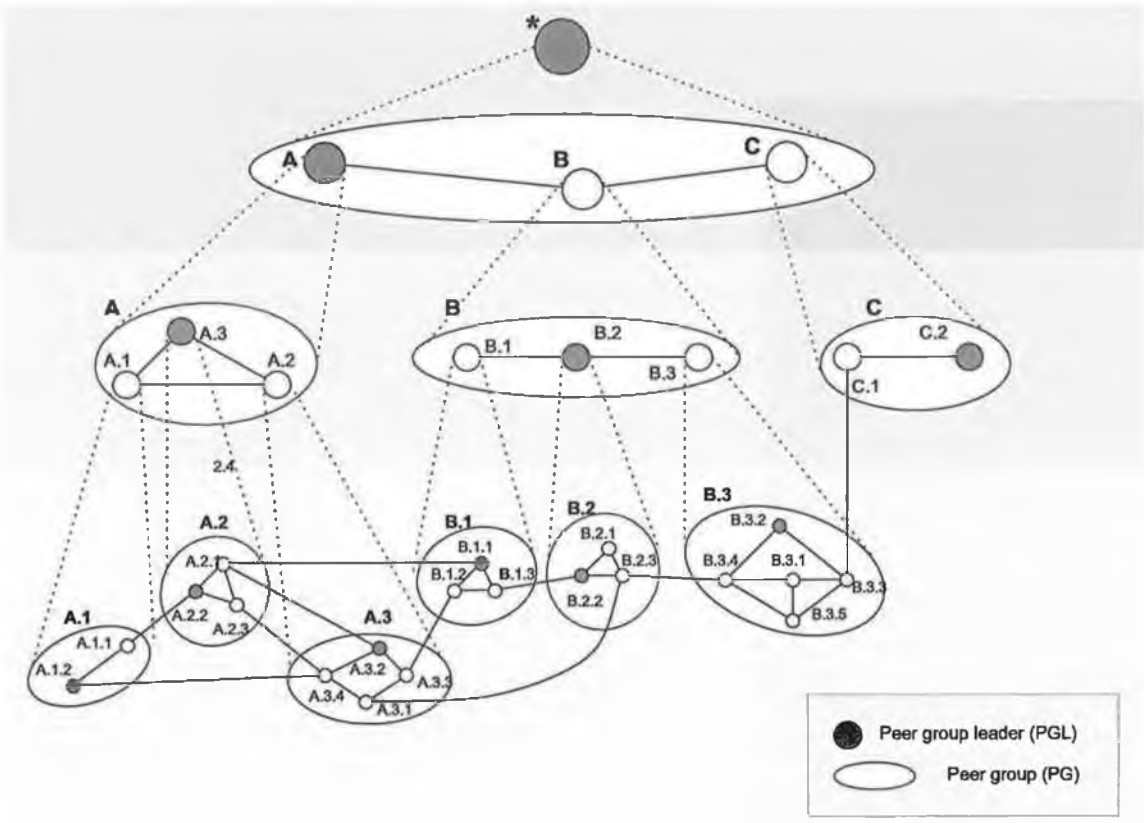


Figure 2.2: *An example of a PNNI routing hierarchy*

The PNNI path selection is based on the *source routing* model. It results in loop free paths and the path selection algorithm needs to be executed only once at the source point. Therefore, when a path request is issued, the source node selects a path that appears to be able to support the QoS requirements specified by the request. This selection is made based on the currently available network state information. After the path is selected, the connection setup process starts. During this process each node along the path has to confirm that the resources requested are available. If the resources are not available, *crankback* occurs, which causes a new path to be computed if possible. Thus, the final outcome of a setup request is either the establishment of a path satisfying the request, or refusal of the connection.

Further details about PNNI are given in Section 2.2.2.1, where the aggregated format in which PNNI keeps its state information is presented, Section 3.2.1, where different types of aggregation methods including the PNNI default aggre-

gation method are introduced, and in Section 3.2.2.1, where the main advantages and disadvantages of PNNI are discussed.

2.1.6 Best-effort vs. QoS

The traditional network service on the Internet is *best-effort* datagram transmission. That means that packets from a source are sent to a destination with no guarantee of delivery. Thus, applications which require guaranteed delivery use the Transport Control Protocol (TCP), which assures correct reception by retransmitting those packets that fail to reach the destination. This, however, comes at the cost of packet delay. For traditional computer-communication applications such as FTP and Telnet in which correct delivery is more important than a timely delivery, this service is satisfactory. For new applications which use voice, video and data (e.g., video teleconferencing and video-on-demand) trading packet delay for correct reception is not an acceptable trade-off.

An alternative to the current service model is the *Quality of Service* (QoS) framework proposed in RFC 2386 [53]. This seeks to augment the current *best-effort* service with predictable service that is unaffected by external conditions such as the number of concurrent traffic flows, or their generated traffic load [82]. A detailed discussion about how this can be accomplished is presented in the next section.

2.2 Quality of Service

The fundamental design consideration of the Internet was simplicity. Thus, it was not conceived and it is not prepared to deal with real-time exchange of data for applications sensitive to the quality and timeliness of the delivery.

Quality of Service (QoS) is a concept familiar in the telecommunication industry as their networks were developed to carry voice traffic. Thus, they are sensitive to the aspects of noise, distortion, loss, delay, and jitter. However, ef-

forts have been made and research is still carried out to introduce and develop QoS mechanisms within the Internet. Therefore, the Internet community defines QoS as "a set of service requirements to be met by the network while transporting a flow" [53].

2.2.1 The roots of QoS

Starting in the 1960s there has been an ongoing discussion about how to support Quality of Service (QoS) controls in packet switched networks. That is why the IP datagram header includes a *Type of Service* (ToS) field intended to be used as a prioritization token of each datagram. However, techniques for using this field are still under development.

By the mid 1980s, three types of communication networks had evolved, each carrying information worldwide. The telephone network carried voice calls; television networks carried video transmissions; and newly emerging computer network carried data. At that point, the telecommunication industry decided to expand its business by developing networks to carry all types of traffic. Their goals were extremely ambitious. Their efforts resulted in the protocol called Asynchronous Transfer Mode (ATM). They faced a difficult challenge because the three intended uses (voice, video, and data) have different sets of requirements.

Recent advances in networking applications have created new requirements for QoS. For example, in grid computing where distributed resources are shared between different heterogeneous applications, the need for guaranteeing Quality of Service may be of paramount importance [120, 157].

2.2.1.1 QoS constraints and metrics

The requirements of different traffic types are specified using *QoS constraints*. There are three different ways of implementing the QoS constraints [9]: link constraints, node constraints and path constraints. For link and node constraints,

non-additive link-state and node-state metrics respectively are used. These constraints are used to prune the network graph during path selection. For path constraints, additive link-state metrics are needed.

Three different types of metrics are defined in [161] so that a metric $m(i,j)$ for the link (i,j), for a path $p=(i,j,k,...,l,n)$, can be:

$$\begin{array}{ll} \text{Additive} & \text{if } m(p)=m(i,j)+m(j,k)+...+n(l,n) \\ \text{Multiplicative} & \text{if } m(p)=m(i,j)\times m(j,k)\times ...\times n(l,n) \\ \text{Concave} & \text{if } m(p)=min\{m(i,j),m(j,k),...,n(l,n)\} \end{array}$$

According to this definition, metrics such as: delay, jitter, cost, and hop count are additive; packet loss probability is multiplicative, and bandwidth is concave. However, any multiplicative metric can be reduced to an additive metric [129], and thus the metrics can also be divided into additive (additive and multiplicative) metrics and non-additive (concave or min/max) metrics.

Tanenbaum presents several applications in [152] with their requirements regarding reliability, delay, jitter, and bandwidth. They are described in Table 2.1. Some applications require reliability (the first four applications) while others are sensitive to delay or jitter (telephony and videoconferencing) and some need considerable bandwidth (video on demand and videoconferencing).

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

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

Selecting a feasible path with a single QoS constraint can be done by using any shortest-path algorithm. However, it has been proven in [162] that finding a path

subject to two or more independent additive and/or multiplicative constraints is an NP-complete[5] problem [69]. This would be the case with telephony and videoconferencing, both of which have high demands on delay and jitter (see Table 2.1). Various approaches have been proposed to "solve" such NP-complete problems (see Appendix D). Solutions applying to special cases are presented in Section 3.1.3, while the more general approximation and heuristical solutions are presented in Section 3.3.

2.2.1.2 The QoS debate

It is still a debated issue whether and how Quality of Service should be provisioned. Therefore, two schools of thoughts have been distinguished regarding the provision of QoS guarantees. Some researchers believe that in the future there will be enough resources that QoS schemes will not be necessary. Others, including this author, argue that no matter how much resources exist in the future, new applications will be developed to use them. History has already proved that links with higher capacities are rapidly consumed by new generations of more demanding applications.

2.2.2 QoS and routing

The term "QoS routing" (Quality of Service routing) can be attributed to a set of protocols and algorithms that find paths in the network that satisfy given requirements, while achieving global efficiency in resource maximisation [42, 53].

Routing deployed in today's Internet is focused on connectivity and typically supports only one type of datagram service called *best effort* [161]. Current Internet routing protocols, e.g., OSPF, RIP, use *shortest path routing*, i.e. routing that is optimised for a single arbitrary metric, e.g., administrative weight or hop count. Therefore, alternative paths with acceptable but non-optimal cost can not be used

[5]Optimal solutions cannot be found by any known polynomial algorithm and hence the computational complexity increases exponentially with the size of the problem (see Appendix D).

to route traffic even if they have more available resources. This strategy may generate over-maximisation problems for shortest paths and under-maximisation for other acceptable paths. QoS routing can be used to avoid such situations.

QoS routing consist of two basic tasks [42, 53]:

- to collect the state information and keep it up-to-date;
- to find a feasible path based on the collected information.

The performance of any QoS routing algorithm greatly depends on how well these tasks are solved.

Both of these two basic routing tasks are described in the next sections as follows: Section 2.2.2.1 presents strategies for maintaining the state information, while Section 2.2.2.2 describes methods for finding feasible paths.

2.2.2.1 The maintenance of state information

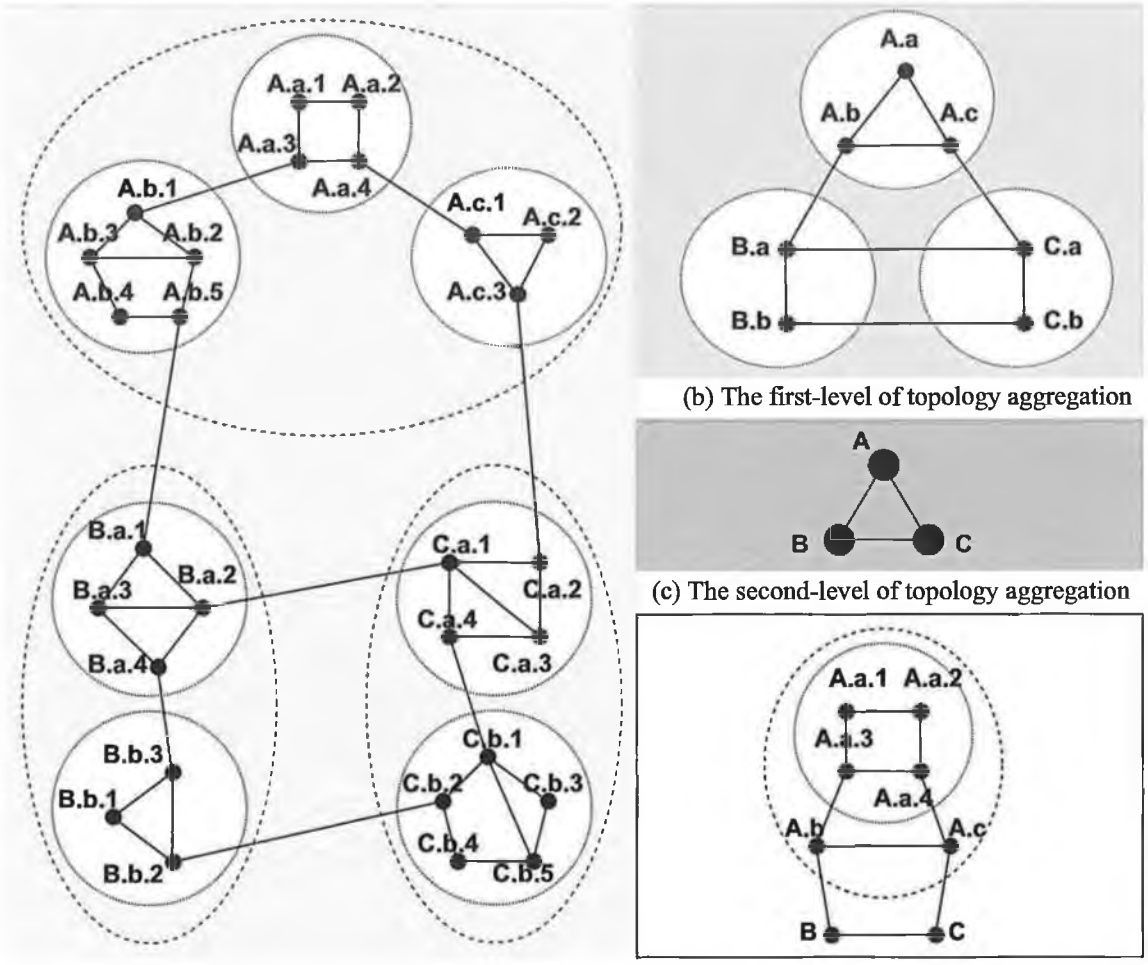
There are three ways in which the state information can be maintained [41, 42].

Local state. Each node maintains its own up-to-date local state which may include queueing and propagation delay, and residual bandwidth of its outgoing links and the availability of other resources (e.g., memory buffers, CPU cycles).

Global state. The combination of the local states of all nodes is called global state. Every node is able to maintain the global state by either a *link-state* protocol (e.g., OSPF) or a *distance-vector* protocol (e.g., RIP). The difference between the two types of protocols consists in the way they distribute and maintain the routing information. In the link-state protocol each node broadcasts the local state to every other node so that each node knows the topology of the network and the state of every link, while in the distance-vector adjacent nodes periodically exchange distance vectors[6].

[6]A distance vector has an entry for every possible destination, consisting of the cost of the best path and the next node on the best path.

Aggregated (partial) state. Maintaining global state in large networks raises scalability issues. A common approach to achieve scalability is to reduce the amount of global state by aggregating information according to the hierarchial structure of large networks.



(a) The physical network

(b) The first-level of topology aggregation

(c) The second-level of topology aggregation

(d) The network image viewed by node A.a.1

Figure 2.3: *A hierarchical network model*

A hierarchical model used by ATM PNNI is presented in Figure 2.3. Here, the physical network presented in Figure 2.3 (a) contains groups of nodes which are represented by a single *logical node* at the next level of aggregation (see Figure 2.3 (b)). The links connecting logical nodes are *logical links*. This aggregation process starts from the physical network (i.e. Figure 2.3 (a)) and continues until there is a single group of logical nodes as in Figure 2.3 (c). At each hierarchical level, the nodes in a group are called the *children* of the logical node representing

the group, while the logical node is called the *parent*. Nodes with at least one link crossing two groups are called *border nodes*.

Within ATM PNNI, each physical node maintains an *aggregated network image*. For instance, the image maintained by the node A.a.1 is shown in Figure 2.3 (d) - it contains detailed information about the A.a group and aggregated information about the rest of the network (e.g., A.b, A.c, B and C). Thus, as the network topology is aggregated, the state information is aggregated as well. Section 3.2.1 describes different aggregation techniques and their advantages and disadvantages.

2.2.2.2 The search for a feasible path

The search for feasible paths greatly depends on how the state information is collected and where the information is stored. Therefore, there are three routing strategies [41, 42].

Source routing. A link-state protocol is used for each node to maintain a global state which includes the network topology and the state information of every link. Based on the global state, a feasible path is locally computed at the source node. A control message is then sent along the selected path to inform the intermediate nodes about their preceding and successor nodes.

Many source routing algorithms are conceptually simple and easy to implement, evaluate, debug and upgrade. The simplicity of source routing is a result of transforming a distributed problem into a centralised one. By doing things this way, it avoids some of the problems of distributed computing. Moreover, it guarantees loop-free routes.

The main weaknesses of source routing are also tightly connected with the centralised way of solving a distributed problem. The global state maintained at every node has to be updated frequently enough to cope with the dynamics of network parameters such as bandwidth and delay. For large networks, the communication overhead will become excessively high. Moreover, the link-state

algorithm can only provide *approximate* global state as the distribution of state messages involve non-negligible propagation delay [42]. As a consequence, the QoS routing algorithm may fail in finding an existing feasible path due to imprecision in the global state. The computational burden is another big disadvantage of source routing. Therefore, source routing has scalability issues [42]. Moreover, it is impractical and insecure for a single node to have access to all detailed state information about all other nodes [78].

Distributed routing. The path computation is distributed among the intermediate nodes between the source and the destination. Therefore, the routing response time can be shorter and the algorithm is more scalable compared with source routing. Moreover, the chances of success in finding a feasible path are greater as it is possible to search multiple paths in parallel.

Some QoS distributed routing algorithms [161] need a distance-vector protocol to maintain a global state in the form of distance vectors at every node. Using distance-vectors, the routing is done on hop-by-hop basis. Other QoS distributed routing algorithms use flooding-based algorithms [41, 43]. They do not require any global state to be maintained and their routing decision and optimisation is done based entirely on local states.

Distributed routing algorithms that depend on the global state suffer from largely the same problems as source routing algorithms. Moreover, when the global state at different nodes is inconsistent, loops may occur. Distributed algorithms that do not need any global state tend to send too many messages [42].

Hierarchical routing. Nodes are clustered into groups which are clustered further up in higher-level groups and so on until a multi-level hierarchy is created. Within the same group, the nodes contain detailed state information while outside the group its state information is advertised in aggregated form.

Hierarchical routing has long been used to cope with the scalability problem

of source routing in large internetworks [9, 42, 80, 105]. Hierarchical routing scales well because each node maintains only a partial global state where groups of nodes are aggregated into logical nodes. The size of such aggregated state is logarithmic in the size of the complete global state [42].

The aggregation of the state information, although lowering routing overhead, introduces additional imprecision, which has a significant negative impact on QoS routing [78].

2.2.3 QoS related projects within the IETF

The Internet Engineering Task Force (IETF)[7] has proposed many service models and mechanisms to meet the demand for QoS.

2.2.3.1 Integrated Services (IntServ)

The Integrated Services (IntServ) [32] model requires resources, such as bandwidth and buffers, to be reserved *a priori* for a given traffic flow to ensure that the Quality of Service requested by the traffic flow can be achieved. The IntServ model includes components beyond those used in the best-effort model such as:

packet classifiers - used to distinguish flows that are to receive a certain level of service;

packet schedulers - which handle the scheduling of service to different packet flows to ensure that QoS commitments are met;

admission control component - used to determine whether a router has the necessary resources to accept a new flow.

[7] IETF is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. The actual technical work of the IETF is done in its working groups, which are organised by topic into several areas (e.g., routing, transport, security, etc.). Much of the work is handled via mailing lists.

Two services have been defined under the IntServ model: *guaranteed service* and *controlled-load service*.

The **guaranteed service** [149] can be used for applications requiring bounded packet delivery time. For this type of application, data that is delivered to the application after a pre-defined amount of time has elapsed is usually considered worthless. Therefore, *guaranteed service* was intended to provide a firm quantitative bound on the end-to-end packet delay for a flow. This is accomplished by controlling the queuing delay on network elements along the data flow path. The guaranteed service model does not, however, provide bounds on jitter (inter-arrival times between consecutive packets).

The **controlled-load service** [170] can be used for adaptive applications that can tolerate some delay but are sensitive to traffic overload conditions. This type of application typically functions satisfactorily when the network is lightly loaded but its performance degrades significantly when the network is heavily loaded. *Controlled-load service*, therefore, has been designed to provide approximately the same service as best-effort service in a lightly loaded network regardless of actual network conditions.

The main issue with the IntServ model has been scalability, especially in large public IP networks which may potentially have millions of active micro-flows in transit concurrently.

A notable feature of the IntServ model is that it requires explicit signalling of QoS requirements from end systems to routers. The Resource Reservation Protocol (RSVP) was developed to perform this signalling function and is a critical component of the IntServ model. The RSVP protocol is described next.

2.2.3.2 Resource ReserVation Protocol (RSVP)

RSVP is a *soft-state*[8] signalling protocol [33, 176]. It was originally developed as a signalling protocol within the IntServ framework for applications to communicate QoS requirements to the network and for the network to reserve relevant resources to satisfy the QoS requirements.

RSVP functioning principles are that the sender or source node sends a PATH message to the receiver with the same source and destination addresses as the traffic which the sender will generate. Every intermediate router along the path forwards the PATH message to the next hop determined by the routing protocol. After receiving a PATH message, the receiver responds with a RESV message which includes a flow descriptor used to request resource reservations. The RESV message travels to the sender or source node in the opposite direction along the path that the PATH message traversed. 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.

One of the issues with the original RSVP specification was scalability. This is 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 [21].

Recently, RSVP has been modified and extended in several ways to mitigate the scaling problems. As a result, it is becoming a versatile signalling protocol for the Internet. For example, RSVP has been extended to reserve resources for aggregation of flows, to set up MPLS explicit label switched paths, and to perform

[8] A *soft-state* protocol is one in which the failure to receive an update or refresh of state information causes the information to become out of date and be discarded. It can operate fairly well in an environment in which delivery of update or refresh message events is not reliable.

other signalling functions within the Internet. There are also a number of proposals to reduce the amount of refresh messages required to maintain established RSVP sessions [21].

2.2.3.3 Differentiated Services (DiffServ)

The goal of the Differentiated Services (DiffServ) effort within the IETF is to devise scalable mechanisms for categorisation of traffic into behaviour aggregates, which ultimately allows each behaviour aggregate to be treated differently, especially when there is a shortage of resources such as link bandwidth and buffer space [28]. One of the primary motivations for the DiffServ effort was to devise alternative mechanisms for service differentiation in the Internet that mitigate the scalability issues encountered with the IntServ model.

The IETF DiffServ working group has defined a Differentiated Services field in the IP header (DS field). The DS field consists of six bits of the part 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 a node [127]. 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.

For an end-user of network services to receive Differentiated Services from its Internet Service Provider (ISP), it may be necessary for the user to have a Service Level Agreement (SLA) with the ISP. An SLA may explicitly or implicitly specify a Traffic Conditioning Agreement (TCA) which defines classifier rules as well as metering, marking, discarding, and shaping rules.

Packets are classified, and possibly policed and shaped at the ingress to a DiffServ network. When a packet traverses the boundary between different DiffServ domains, the DS field of the packet may be re-marked according to existing agreements between the domains.

Differentiated Services allows only a finite number of service classes to be

indicated by the DS field. The main advantage of the DiffServ approach over the IntServ model is scalability. Resources are allocated on a per-class basis and the amount of state information is proportional to the number of classes rather than to the number of application flows.

2.2.3.4 Internet Traffic Engineering (TE)

Internet traffic engineering (or simply traffic engineering) is defined in [13] as "that aspect of Internet network engineering dealing with the issue of performance evaluation and performance optimisation of operational IP networks". Therefore, traffic engineering is a critical component of an end-to-end Internet QoS framework, which give service providers better control over the network in order to enhance its performance. This can be done by improvements made at both traffic level and resource level. The authors of [14] define two main classes of traffic engineering performance objectives:

Traffic oriented performance objectives concern the enhancement of the QoS of traffic streams (i.e., minimisation of packet loss, minimisation of delay, maximisation of throughput, and enforcement of service level agreements);

Resource oriented performance objectives concern the optimisation of resource utilisation.

The goal is to maximise the use of the resources. Both under-maximisation and over-maximisation cause dramatic reduction in the performance and efficiency of a running network. Over-maximisation, 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. Under-maximisation has two negative consequences. The obvious one is economic - underused resources cost money, and either are generating less revenue than they could, or cost more than the optimal amount of the resource. More subtly, local

under-maximisation may be symptomatic of a regional or global traffic imbalance, which implies congestion elsewhere in the network.

There are two main factors inducing congestion:

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

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

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 network vulnerability to service outages due to errors, faults or failures occurring 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.

Initially, most vendors were sceptical about the merits of traffic engineering, despite the large variety of methods that have been proposed. That is because the offered technologies were considered too complicated and too risky to be deployed in the legacy Internet. Therefore, commercially available solutions were rather simple [96]. Recently, however, new solutions for MPLS traffic engineering have emerged, which have the potential to provide efficient control over the traffic [8, 11, 14, 27, 76, 108].

2.2.3.5 Multi-Protocol Label Switching (MPLS)

MPLS [27, 76, 135] is a very powerful technology for Internet traffic engineering and QoS routing, because it supports explicit routing, aggregation and disaggre-

gation and source routing. These abilities make MPLS a tool that can be used to optimise resource maximisation [11, 14, 27, 76].

The key MPLS features supporting traffic engineering are:

- MPLS has the ability to build *explicit* non-shortest *paths* that can be used to shift traffic from congested routes to alternative ones.
- MPLS allows both traffic *aggregation* and traffic *disaggregation*, while traditional IP forwarding allows only aggregation. This can be used in traffic engineering by aggregating traffic flows of the same class which are placed inside a Label Switched Path into *traffic trunks* [109], which are routable objects similar to ATM VPs.
- A *set of attributes* can be explicitly assigned to traffic trunks. They are parameters that influence their behavioural characteristics [14]. Some examples would be attributes that designate the *priority*, *preemption* or *resilience* of a traffic trunk.

A detailed MPLS overview is presented in Section 2.3.

2.2.3.6 Constrained-based Routing (CR)

Constraint-based routing [14] refers to a class of routing systems that compute routes through a network subject to the satisfaction of a set of constraints and requirements.

Unlike QoS routing which generally addresses the issue of routing individual traffic flows to satisfy prescribed flow-based QoS requirements subject to network resource availability, constraint-based routing is applicable to traffic aggregates as well as flows and may be subject to a wider variety of constraints which may also include policy restrictions (e.g., security and administrative regulations) [53].

2.2.4 Enabling technologies

QoS routing can be implemented by suitable extending today's link-state routing protocols. However, some other techniques are available to provide new architectures with QoS routing support.

2.2.4.1 Active Networks

Active Networks [153–155] is a relatively new concept, where a network is not just a passive carrier of data but also a more general model capable of performing customised computations on carried data. Since it is able to provide various operations on network flows, such technology can be used as a platform for the gradual deployment of QoS aware routing algorithms. Section 2.4 will present a detailed description of Active Networks, while arguments for using it in order to achieve QoS routing will be addressed again in Chapter 4.1.

2.2.4.2 Mobile Agents

Mobile agents [45] are related to the Active Networks concept in the sense that the code is *mobile*. The main difference between the two technologies, as the author sees it, consists in how are they using the code and what impact the execution of such code has on current traffic. From this perspective, active networks have a direct impact on the existing/current traffic as the code either travels along with the (traditional) packets or waits for them on intermediate nodes in order to perform computations ranging from redirecting the packets to modifying their content. Mobile agents are more "passive" from this point of view as they travel independently of data traffic with the primary aim to "discover" the network and collect information scattered into the network rather than to change data traffic.

Mobile agents, however, can indirectly modify the way data traffic flows if used for routing purposes. Moreover, they are well suited to the task of searching for feasible paths in the network as they are able to perform the search in a highly

distributed and parallel manner. The motivation for their use is even stronger when QoS routing is the final objective as consulting information for multiple constraints is more efficient if mobile agents go to the source of routing information rather than bringing all this information to a central point to be computed. This and other advantage of mobile agents for routing will be discussed in Sections 2.5 and 3.4. Chapters 5 and 6 present proposals for using mobile agents for hierarchical QoS routing and for multi-constrained hierarchical QoS routing respectively in an MPLS context.

2.3 Multi-Protocol Label Switching

MultiProtocol Label Switching (MPLS) is intended to combine the principles of the *Internet Protocol* (IP) which enables a relatively inexpensive, robust and scalable network with those of *Asynchronous Transfer Mode* (ATM) that offers Quality of Service guarantees.

For a better understanding of MPLS I present, in the following section, a brief history of the genesis of MPLS.

2.3.1 From IP and ATM to MPLS

The Internet Protocol (IP) is a layer 3 (network layer)[9] protocol used within the Internet to send information by breaking up messages into packets. The packets are then sent from the source to a destination host via routers which determine the best path for each packet. The router uses a routing protocol to decide a best path locally. In this way packets travel from router to router until they reach the router of the destination host. This method whereby routers retrieve the whole

[9]To reduce the design complexity of networking protocols, they are organised as a stack of layers or levels, each one built upon the one below it. The number of layers, the name of each layer, the contents of each layer, and the function of each layer differ from network to network [152]. The International Standards Organisation (ISO) proposed a 7-layer model called Open Systems Interconnection (OSI). Within this model, the IP is placed on the 3rd layer. More about the OSI reference model can be found in [152].

packet, determine its next hop, and then forward it is often referred to as *store-and-forward* or *hop-by-hop* routing. Therefore, each packet follows its own route toward the destination. Consequently, IP is a connectionless protocol. Such a model cannot guarantee performance requirements such as delay and bandwidth. This makes IP a *best effort* service model and it does not provide any Quality of Service guarantees.

Developed for telecommunication networks, the Asynchronous Transfer Mode (ATM) uses short, fixed length packets, called *cells*, that allow for fast switching and low latency communication. ATM is a connection-oriented technology meaning that a point-to-point connection is set up to allow any two nodes to communicate. The connection persists for the duration of the message and is then torn down. Because the connection, or virtual circuit, is set up prior to the message transfer, guaranteed bandwidth and buffer space can be negotiated and provisioned between the sender and the network. Therefore, an ATM network can provide QoS guarantees.

The main advantage of ATM is that by using a fixed length label for forwarding and relatively small fixed length cells, it allows the forwarding scheme to be implemented in hardware. However, each 48-byte cell carries a 5-byte header implying approximately a 10% overhead. Also, whenever a packet is not an exact fit into n cells, some capacity is wasted in the last cell. By using virtual circuits (VCs), ATM is a good candidate for traffic engineering (TE), whereas IP *hop-by-hop* routing has the tendency to concentrate all traffic onto the shortest path. VCs may also be used to meet the requirement for QoS routing that the required resources be reserved for specific flows of user data. In some cases, the shortest path may already be fully reserved and unable to support a particular additional flow of data. For those cases, it is necessary to route around congestion.

Despite its appealing features, ATM has not been widely accepted. Although many voice and data communication providers use it in their backbone network, the expense, complexity and lack of inter-operability with other technologies

have prevented ATM becoming more prevalent.

Many protocols tried to integrate IP and ATM in order to get the best from each technology. These proposals can be grouped into two classes: *IP over ATM protocols* and *IP switching protocols*.

2.3.1.1 IP over ATM Techniques

One way of integrating IP and ATM was that IP would treat ATM as a link-level technology (like Ethernet or FDDI), ignoring the routing and the quality-of-service aspects of ATM. This approach was called IP over ATM.

Fundamental differences between IP and ATM made the integration difficult:

- connectionless (IP) versus connection oriented (ATM);
- packets (IP) versus cells (ATM);

The main IP over ATM protocols are:

1. *Classical IP over ATM*
2. *ATM Address Resolution Protocol (ATM ARP)*
3. *Next Hop Resolution Protocol (NHRP)*
4. *LAN emulation (LANE)*
5. *Multiprotocol over ATM (MPOA)*

The first three protocols were developed by the Internet Engineering Task Force (IETF), while the last two are standards proposed by the ATM Forum[10].

Classical IP over ATM. Classical IP over ATM [101, 102] is also known as the *overlay model* because IP is "layered over" ATM similarly to how IP is "layered" over Ethernet, Frame Relay, and other link-layer technologies.

[10]The ATM Forum is an industrial consortium that issues recommendations regarding ATM protocols.

In order to implement this protocol, the ATM network is divided into one or more Logical IP Subnetworks (LISs) where two hosts within the same LIS communicate through a direct VC while two hosts from different LISs intercommunicate via a router as illustrated in Figure 2.4.

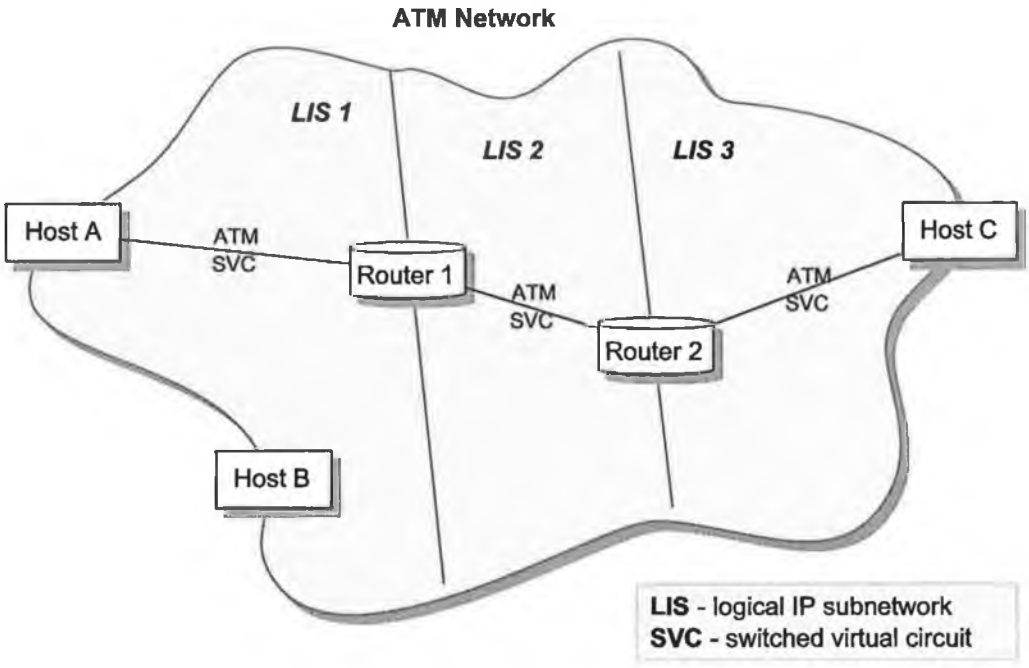


Figure 2.4: *Multiple IP LIS's on one ATM network*

This approach has two advantages. It is the closest possible adaptation for this attempt (i.e., to overlay IP on top of ATM) and it does not require any changes to ATM switches and protocols because all the work is confined in the IP layer.

The disadvantages of this approach are that it does not solve the delay problem present in the store-and-forward network because packets sent to a host outside of the source's LIS need to be routed (i.e., processed by a router). Moreover, because IP and ATM use separate address spaces, an address resolution protocol is required to map between them, similar to the address resolution protocol (ARP) used on broadcast LANs.

Address Resolution protocol (ATM-ARP). Classical IP over ATM defines an ATMARP protocol [101, 102] for resolving IP addresses to ATM addresses. This

approach is different from the technique used when IP is run over Ethernet where broadcast messages are used to transmit ARP requests to all of the other stations on the LAN. Because ATM does not have a broadcast mechanism, a server must be used instead. Each LIS requires an ATMARP server to translate from IP to ATM addresses.

Every host within a LIS has to register its ATM and IP addresses with a configured ATMARP server. Then, it may query the ATMARP server for the ATM addresses corresponding to IP addresses of hosts within the same LIS. Such ATM addresses are then used to open VCs between the relevant host and the host that originated the query.

Next Hop Resolution Protocol (NHRP). The architecture of classical IP over ATM specifies that hosts which are in separate LISs must communicate via a router, even when it means that packets must leave the ATM network, only to reenter another ATM network. This process is inefficient because the cells must be reassembled into packets as they enter the router and then re-segmented into cells as they enter again an ATM network. In addition, any QoS provided by ATM is lost at each router hop.

To address this inefficiency, IETF developed the Next Hop Resolution Protocol (NHRP) [113], which allows IP hosts in different LISs, but on the same ATM network, to be able to open direct VCs between each other.

NHRP can be viewed as an enhancement of ATMARP where ATMARP clients (end stations) became NHRP clients, and NHRP servers (NHS's) replace ATMARP servers. Like ATMARP, NHRP is based on a query/response model: an NHRP client that wishes to forward IP packets destined to other hosts sends an NHRP query to its local NHS. If the NHS knows the ATM address of the destination host, an NHRP response is sent back to the originating client indicating the ATM address to use to reach the specific destination. Otherwise, it forwards the query to another NHS. The client that originated the query opens a direct point-

to-point VC to either the destination host or the router that will forward the IP packet off the ATM network.

A drawback of this method is that the response returned is based on the state of the IP routing at the time the query is sent. If the route that should be followed changes, the originating NHRP client will not to be notified. In such cases the packets are misrouted.

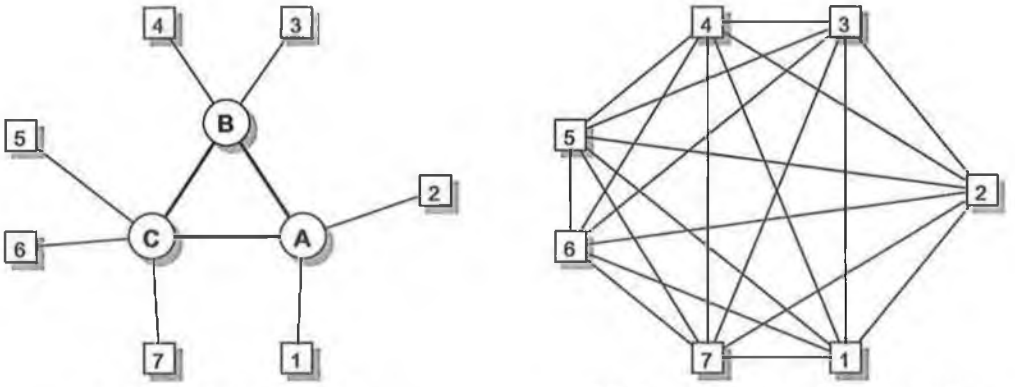
LAN Emulation (LANE). In order to be expanded toward the edge of communication networks, the ATM technology had to replicate the services of existing LANs. Thus, the ATM Forum defined an ATM service called "LAN emulation" that emulates the services of LANs across an ATM network. The services provided by LANE are: connectionless and multicast services, MAC driver interfaces in ATM stations, emulated LANs and interconnection with existing LANs.

The LANE solution is preferable to IP over ATM because of all the additional facilities it offers. However, the additional layer of overhead, protocol processing, and interactions with a second set of servers adds to the processing delay in the network and is a more expensive solution for the end user.

Multiprotocol over ATM (MPOA). MPOA [10] is a combination of LANE and NHRP, combining the virtual networking ability of LANE with the address resolution ability of NHRP. An MPOA client has both LANE and NHRP client capabilities, and MPOA servers are extensions to NHSs.

Limitations of IP over ATM approaches. The main problem with the techniques described previously is scalability.

IP routing protocols scale on the order of the number of router-to-router connection in the routing topology, be it physical or virtual. In the overlay model, as you can see in Figure 2.5, the total number of logical links that are advertised between the n ATM-attached routers equals $\frac{n\,(n-1)}{2}$ links.



(a) Physical topology (b) Resulting logical connectivity

Figure 2.5: *IP over ATM logical connectivity*

Breaking up the overlay into multiple IP subnetworks solved the scaling problem with the overlay model. While reducing the number of route connections, this creates another problem: inefficient paths for data traffic. This second problem has led to the addition of protocols such as NHRP to the overlay model. The additional protocols increase the complexity of the overlay model and make its performance depend on traffic-sensitive caching techniques.

The overlay method of running IP over ATM also suffers from other problems such as [163]:

1. excessive management complexity and overhead associated with running two separate routing protocols;

2. the difficulty of representing the ATM subnetwork in such a way that IP routing can deal with it yet that accurately reflects the capabilities of the ATM subnetwork;

3. the resulting difficulty in extending the approach to QoS routing, traffic engineering, and similar advanced features.

2.3.1.2 IP Switching Techniques

An alternative to supporting the overlay model is to let the ATM switches participate in IP routing. Such 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 [89] 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 sub-networks (LISs) and is capable of both IP forwarding and ATM cell switching. Within a LIS, layer 3 connectivity between nodes is provided by either LANE or classical IP over ATM. The address resolution is performed by ATMARP [102] and InATMARP [34] 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 can 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) [123].

Ipsilon's IP Switching IP Switching is a technology proposed by Ipsilon [126] 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) [125], 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 Figure 2.6.

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

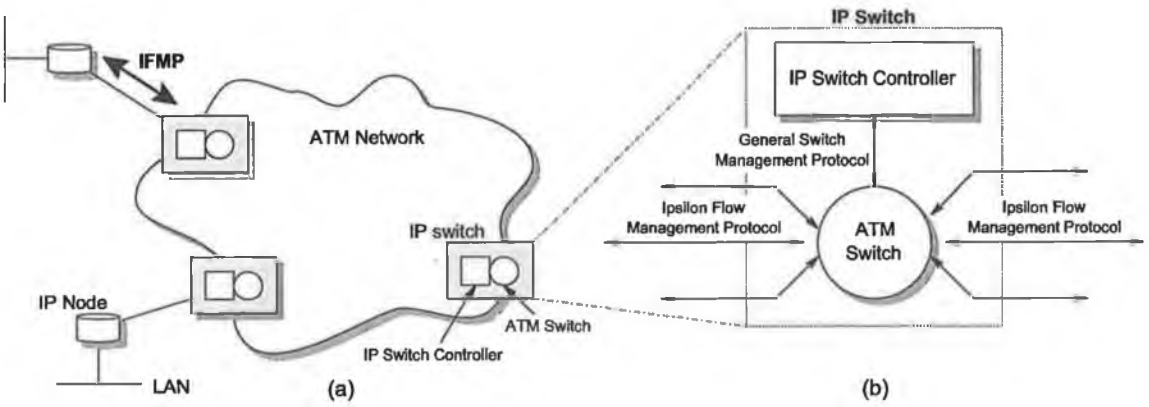


Figure 2.6: *(a) An IP Switching Network (b) The structure of an IP Switch*

Flow Management Protocol (IFMP) [124].

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 [159]. 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 Figure 2.7. These neighbours forward *establish messages* upstream in reverse path multicast style, so eventually all ARIS ISRs have switched paths to every egress ISR.

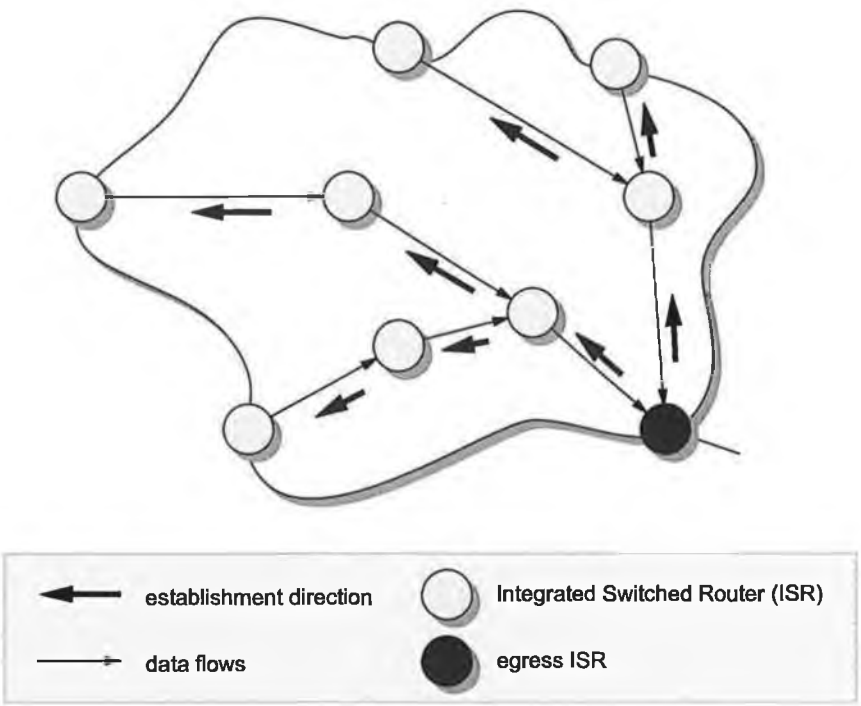


Figure 2.7: *IBM ARIS Switched Paths*

One important ARIS particularity 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 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 [132].

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. 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 a 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[11] 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

[11]See *http://www.ietf.org*

Group[12] 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)[13] that define the basic MPLS architecture [135] and encapsulations [134], the Label Distribution Protocol (LDP) [3, 156], the definitions for how MPLS runs over ATM [54] and Frame Relay [50], and the requirements for traffic engineering over MPLS [14]. 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 [39, 63] have since addressed this bottleneck in IP packet processing, but MPLS is now favoured for its traffic engineering capabilities.

2.3.2 MPLS architecture

The MPLS IETF working group defines in [135] the main elements of an MPLS system, which are also presented in Figure 2.8, as follows:

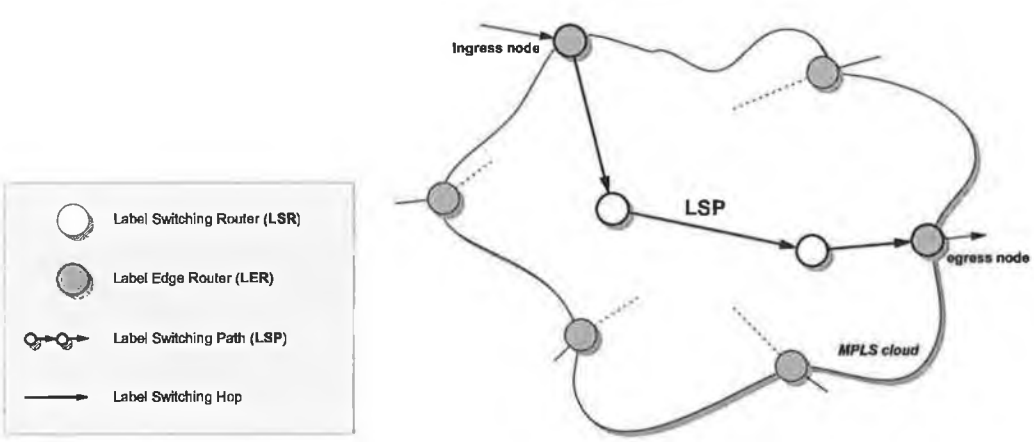


Figure 2.8: *The MPLS cloud and its main components*

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

[13] *RFCs* are IETF documents that contain proposals and standards related to the Internet technology.

- **an MPLS domain** is a contiguous set of MPLS capable nodes which operate MPLS routing and forwarding and which are also in one routing or administrative domain.
- **a Label Switching Router (LSR)** is an MPLS capable node, which will be aware of MPLS control protocols, will operate one or more L3 routing protocols, and will be capable of forwarding packets based on labels, and may optionally also be capable of forwarding native L3 packets.
- **a Label Edge Router (LER)** is an MPLS capable node that connects an MPLS domain with a node which is outside of the domain, because it does not run MPLS, and/or because it is in a different administrative domain.
- **a Label Switched Path (LSP)** is a path through one or more LSRs at a single level of the hierarchy followed by packets in a particular Forwarding Equivalence Class[14].
- **a label switched hop** is the hop between two MPLS nodes, on which forwarding is done using labels.
- **an ingress node** is an MPLS capable edge node which handles traffic that enters an MPLS domain.
- **an egress node** is an MPLS capable edge node which handles traffic that leaves an MPLS domain.

2.3.2.1 MPLS forwarding

MPLS simplifies the forwarding mechanism by using the *label switching* paradigm. That means that instead of forwarding each packet based on the network layer address and information distributed by routing protocols, the nodes in the

[14] A *Forwarding Equivalence Class (FEC)* is a group of IP packets which are forwarded in the same manner [135].

network can use labels carried on the packets and label switching information distributed by new protocols or extensions to the existing protocols.

Figure 2.9 illustrates the MPLS forwarding mechanism. Here, a packet en-

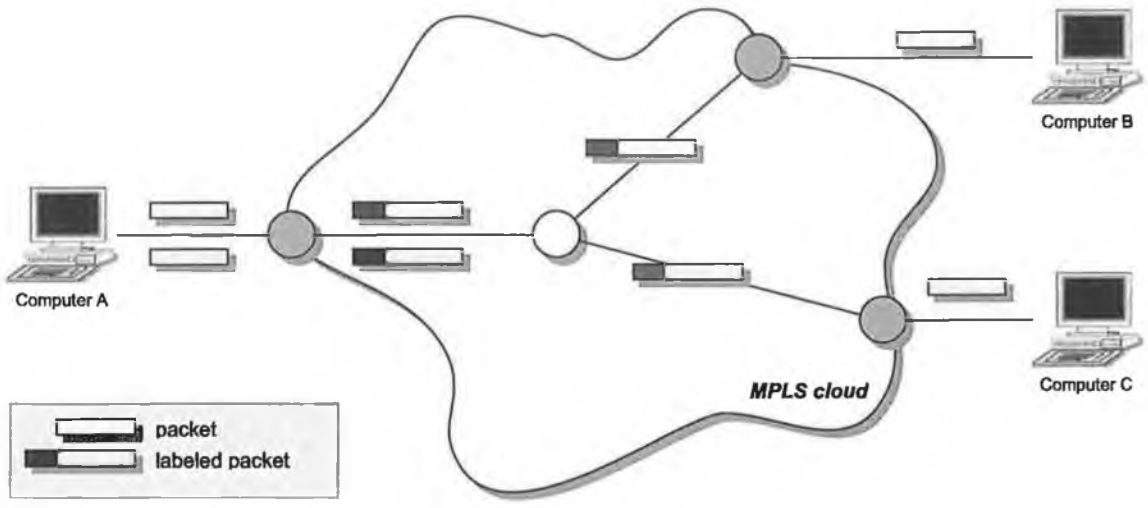


Figure 2.9: *MPLS: Basic principles*

ters an MPLS cloud through a *Label Edge Router* (LER). The *LER* labels the packet and forwards it to the next *Label Switching Router* (LSR). In the MPLS cloud, the labelled packet is forwarded, from one LSR to the next, with its next hop determined by a label lookup. No lookup of the IP routing table is performed. The packet leaves the MPLS cloud by passing through a *LER* which pops the label and forwards the packet to another legacy router or to the destination, in the traditional way.

The *Forwarding Equivalence Class.* As packets enter the MPLS cloud, the *Label Edge Router* (LER) analyses their network-layer headers and, based on the information gathered, assigns them to *Forwarding Equivalence Classes*. Packets which belong to the same *Forwarding Equivalence Class* are labelled with the same label and are treated in the same way by the network.

The *MPLS shim.* When a LER labels a packet it places a *"shim"* between the header of layer 2 and the header of layer 3, as shown in Figure 2.10. This *shim* has

32 bits that are distributed amongst 4 fields as follows:

- **20 bits**- *Label*;
- **3 bits** - *EXP* (*EXPerimental use*);
- **1 bit** - *S* (*bottom of Stack*)= 1 if the label is in the bottom of the stack, 0 otherwise;
- **8 bits** - *TTL* (*Time To Live*).

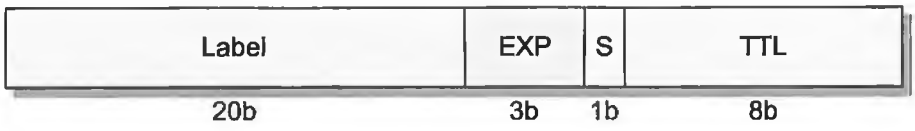


Figure 2.10: *MPLS Label. MPLS shim*

The label width of 20 bits implies that there are $2^{20} = 1048576$ distinct label values that can be used of which the values from 0 to 15 are reserved. The label value is used in the forwarding process and it is replaced with a new value after each hop. Thus the label associated with an FEC is specific to an outgoing interface and does not have global significance.

There were multiple proposals for the use of the experimental bits. The most important use is to prioritise the MPLS traffic [65].

The *TTL* field is copied from the IP header when a packet is labelled and is treated the same way as in IP (i.e. decremented on a hop-by-hop basis; used for loop prevention and to reflect the transition through the network).

The *label stack*. Unlike Frame Relay which has a single label, the data link connection identifier (DLCI), or ATM which has two, the virtual path identifier/virtual channel identifier (VPI/VCI), MPLS allows an arbitrary number of labels as illustrated in Figure 2.11.

The power of the label stack is that it allows multiple control components to act on a packet. This can occur with little or no coordination between the control

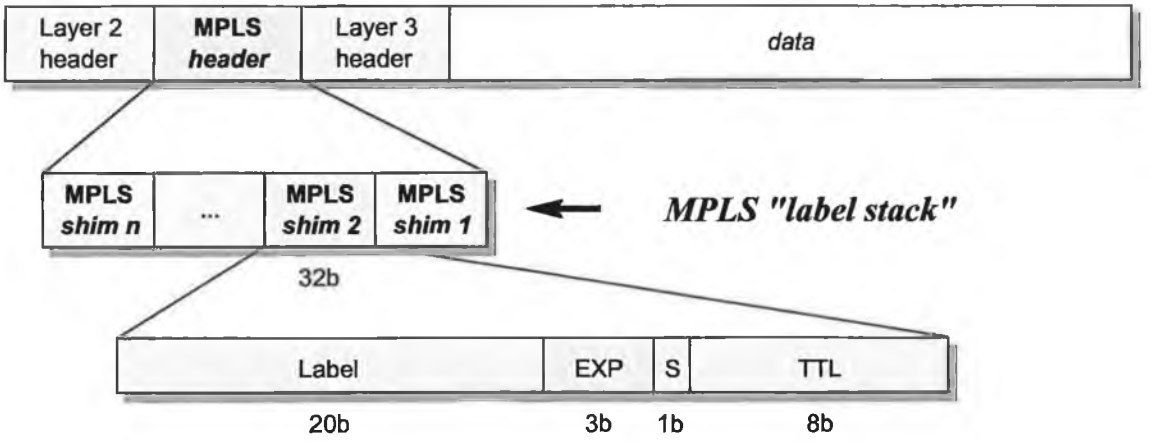


Figure 2.11: *MPLS Label Stack*

planes, making it simple to combine services in useful ways. For example a second label can be introduced to balance the load on alternative paths; or to set a "loosely" explicitly routed LSP[15]; or for restoration purposes. More obvious uses of the label stack are the implementation of *aggregation/disaggregation* mechanisms or setting *Virtual Private Networks* (VPNs).

Packet transitions. A packet encounters three types of modification when it traverses an MPLS cloud:

1. In the *ingress node*, when the packet should be labelled, an MPLS shim is created and added to the packet.

2. Traversing the *Label Switching Path*:

   - one or more labels can be popped off the MPLS label stack;
   - one or more labels can be pushed onto the MPLS label stack;
   - the current label can be replaced with another one.

3. In the *egress node*, when the MPLS shim must be removed from the packet, and the forwarding decision is instead made by analysing the packet's network header.

[15]The *"loosely" explicitly routed LSP* implies that some of the LSRs traversed by it are explicitly specified (usually by the ingress LSR).

The MPLS Data Structures. These are tables which contain the information needed by a router in order to be able to:

- receive an unlabelled packet, to label it and to forward it;
- receive a labelled packet and, using that label to index the relevant structures, to modify it and to forward it.

There are three types of such data structures:

1. The **Next Hop Label Forwarding Entry** (NHLFE) - is required when adding an outgoing label;
2. The **Incoming Label Map** (ILM) - is needed when interpreting incoming labels;
3. The **"Forward Equivalence Class" to NHLFE** (FTN) - is needed in deciding what label to add to an unlabelled packet.

As you can see in Figure 2.12, when an unlabelled packet arrives to an *Label Edge Router* (LER), its network layer header is analysed and the packet is assigned to a *Forwarding Equivalence Class* (FEC). The *FEC-to-NHLFE* (FTN) maps each FEC to a set of NHLFEs. If the set of NHLFEs mapped by the FTN contains more than one element, only one NHLFE element must be chosen before the packet is forwarded. The NHLFE entry contains the packet's next hop and the operation to perform on the packet's label stack (*push*, in this case). Then, the labelled packet is forwarded to the next hop (LSR).

Figure 2.13 depicts the situation when a labelled packet arrives at a *Label Switching Router* (LSR). Using ILM, the incoming label is mapped to a set of NHLFEs. If more than one NHLFE entry corresponds to the same label, only one should be chosen. The NHLFE entry contains the packet's next hop and the operation to perform on the packet's label stack. This operation may be one of the following:

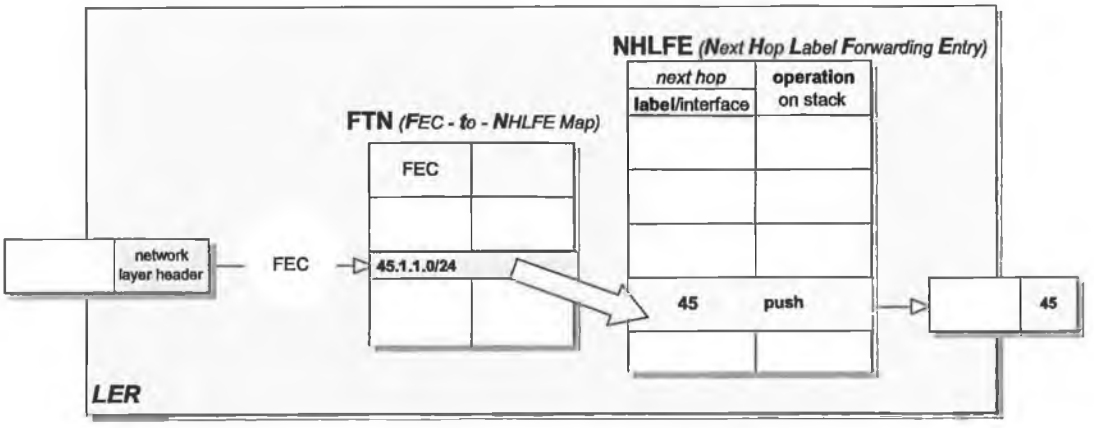


Figure 2.12: *MPLS Data structures needed for an unlabelled packet*

- replace the label at the top of the label stack with a specified new label;
- pop the label stack;
- replace the label at the top of the label stack with a specified new label, and then push one or more specified new labels onto the label stack.

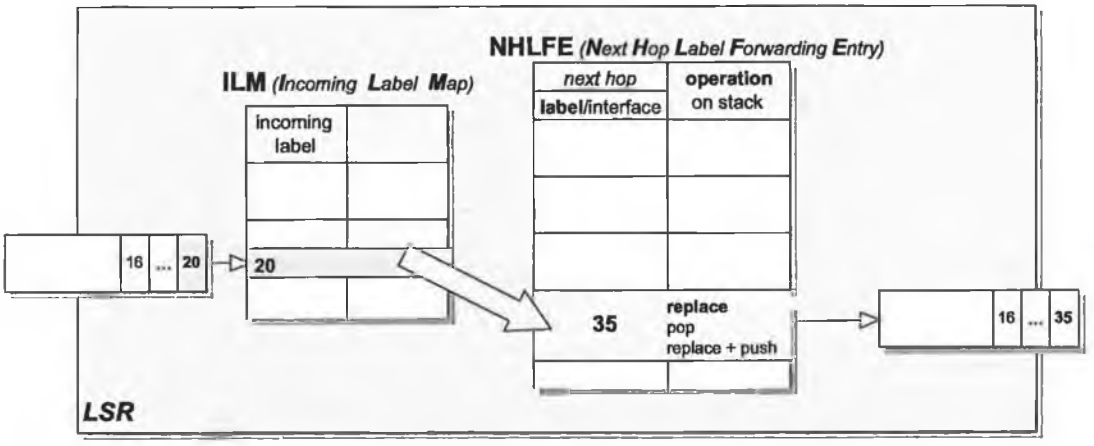


Figure 2.13: *MPLS Data structures needed for an labelled packet*

The MPLS label stack of the packet is modified by the operation indicated in the NHLFE entry and the packet is forwarded to the next hop (LSR).

2.3.2.2 MPLS signalling

A fundamental concept in MPLS is that two *Label Switching Routers* (LSRs) must agree on the meaning of the labels used to forward traffic between and through them. This common understanding is achieved by using a set of procedures, called a *label distribution protocol*, by which one LSR informs another of label bindings it has made. RFC 3031 [135] defines a set of such procedures by which LSRs assign and distribute labels to support MPLS forwarding along normally routed paths.

The standard defines some principles for this protocol:

1. The labels distribution is done *downstream* and it can be:
   - "Downstream-on-Demand", meaning that an explicit request has been made for a label binding;
   - "Unsolicited Downstream", which means that the bindings are distributed without any request.
2. The *Label Retention Mode* can be:
   - "Liberal Label Retention Mode", which maintains the bindings between a label and a FEC which are received from LSRs which are not its next hop for that FEC;
   - "Conservative Label Retention Mode", which discards such bindings.
3. The *Scope and Uniqueness of Labels*:
   - "Per-Interface Label Space", when labels are unique for one interface;
   - "Per-Platform Label Space", when labels are unique for the same LSR.
4. The *LSP control* can be:
   - "Independent", when each LSR makes its own decision to bind a label and to distribute it to its label distribution peers;

- "Ordered", when only the LSR which is the egress node or which has already received a binding label, binds a label.

Both label distribution techniques (i.e. Downstream-on-Demand and Unsolicited Downstream) may be used in the same network at the same time. However, for any given LDP session, each LSR must be aware of the label distribution method used by its peer in order to avoid situations where one peer using Downstream Unsolicited label distribution assumes its peer is also. In this case, a livelock situation may occur, where no labels are distributed. (For example, a downstream LSR X using Downstream on Demand advertisement mode may assume that the upstream LSR Y using Downstream Unsolicited mode will request a label mapping when it wants one and Y may assume that X will advertise a label if it wants Y to use one.) Thus, LSRs exchange advertisement modes during the initialisation phase of the distribution protocol. The major difference between Downstream Unsolicited and Downstream on Demand modes is in which LSR takes responsibility for initiating mapping requests and mapping advertisements. Thus, in general, the upstream LSR is responsible for requesting label mappings when operating in Downstream on Demand mode, while the downstream LSR is responsible for advertising a label mapping when it wants an upstream LSR to use the label.

The MPLS architecture [135] introduces the notion of label retention mode which specifies whether an LSR maintains a label binding for a FEC learned from a neighbour that is not its next hop for the FEC. The main advantage of the conservative mode is that only the labels that are required for the forwarding of data are allocated and maintained. This is particularly important in LSRs where the label space is inherently limited, such as in an ATM switch. A disadvantage of the conservative mode is that if routing changes the next hop for a given destination, a new label must be obtained from the new next hop before labelled packets can be forwarded.

The main advantage of the liberal label retention mode is that reaction to routing changes can be quick because labels already exist. The main disadvantage of the liberal mode is that unneeded label mappings are distributed and maintained.

The notion of *label space* is useful for discussing the assignment and distribution of labels. Platform-wide incoming labels are used for interfaces that can share the same labels. In other cases, interface-specific incoming labels are used. However, the use of a per-interface label space only makes sense when the LDP peers are directly connected over an interface, and the label is only going to be used for traffic sent over that interface. A situation where an LSR would need to advertise more than one label space to a peer and hence use more than one LDP Identifier occurs when the LSR has two links to the peer and both are ATM (and use per-interface labels). Another situation would be where the LSR had two links to the peer, one of which is ethernet (and uses per-platform labels) and the other of which is ATM.

The behaviour of the initial setup of LSPs is determined by whether the LSR is operating with independent or ordered LSP control. An LSR may support both types of control as a configurable option. In the case of ordered label distribution, the label mappings are propagated from egress toward ingress. In the case of independent label distribution, an LSR may map a label for an FEC before receiving a label mapping from its downstream peer for that FEC. In this case, the subsequent label mapping for the FEC received from the downstream peer is treated as an update to LSP attributes, and the label mapping must be propagated upstream. Thus, it is recommended that loop detection be configured in conjunction with ordered label distribution, to minimise the number of Label Mapping update messages.

The implementations of an MPLS label distribution protocol are many and different. I discuss here three of them: **LDP**, **CR-LDP** and **RSVP-TE**.

The *Label Distribution Protocol* (LDP).

LDP [3, 156] is based on the original Tag Distribution Protocol (TDP) and the Aggregate Route-based IP Switching (ARIS) signalling proposal.

As the first step of the LDP process, LDP capable routers use a discovery mechanism to identify potential LDP peers. Then, peer LSRs may exchange or withdraw label bindings.

LDP uses both models of label distribution: downstream-on-demand and unsolicited downstream, of which the former is preferred. As a transport protocol, LDP uses the Transfer Control Protocol (TCP) for sending its messages and thus it is considered to be a *hard-state*[16] protocol. The exceptions are the discovery messages which are sent using the User Datagram Protocol (UDP).

The major shortcoming of LDP is that LSPs created using LDP will follow normally-routed (hop-by-hop) paths, as determined by destination-based routing protocols [3]. Extensions to the standard LDP must be developed in order to allow constraint based or traffic engineered explicit paths and resource reservation.

The *Constraint-based Routing Label Distribution Protocol* (CR-LDP).

CR-LDP [85] provides extensions to the basic LDP standar for the support of explicitly routing LSP setup requests and (potentially) reserving resources along the resulting LSP.

CR-LDP, like LDP, is a *hard-state* protocol because it uses the Transmission Control Protocol (TCP) for all session, advertisement, notification, and LDP messages, while the User Datagram Protocol (UDP) is used only for peer discovery.

CR-LDP supports the creation of both control-driven LSPs (also called hop-by-hop LSPs) and explicitly routed LSPs (also referred to as constraint-based routed LSPs or CR-LSPs). The difference between the two is that while control-driven

[16] A *hard-state* protocol is one in which the state information remains valid until explicitly changed. Proper operation of a hard-state protocol requires absolute reliability in the delivery of message events because it must not be possible for events to be missed. Most protocols that are considered hard-state are based on TCP.

paths are setup solely based on information in routing tables or from a management system, the constraint-based route is calculated at one point at the edge of network based on criteria, including but not limited to routing information. The intention is that this functionality shall give desired special characteristics to the LSP in order to better support the traffic sent over the LSP [85].

To set up an CR-LSP, a *Label Request* message is sent along the constraint-based route. The *Label Request* message contains the list of nodes to be traversed. Once the signalling message has travelled all the way to the destination, and if the requested path can satisfy the resources required, labels are allocated and distributed by *Label Mapping* messages which start from the destination and propagate in the reverse direction back to the source.

The *Resource Reservation Protocol for Traffic Engineering* (RSVP-TE).

RSVP-TE [12] provides extensions to the original Resource ReSerVation Protocol (RSVP) to support explicitly routed LSP setup requests.

RSVP was initially developed as a resource reservation protocol within the IntServ framework. However, the success of RSVP was limited because of IntServ's scalability issue [165] as mentioned in Section 2.2.3.2. The emergence of MPLS and the need for a constraint based signalling protocol revived RSVP as a new protocol: RSVP-TE. The key application of RSVP-TE with MPLS is traffic engineering. RSVP-TE is useful for establishing and maintaining explicitly routed LSPs in order to force the traffic through other routes than those given by standard routing protocols. Additionally, RSVP itself defines mechanisms to support the allocation of network resources to the paths defined by protocol activity.

Choosing between CR-LDP and RSVP-TE.

Both CR-LDP and RSVP-TE proved to be suitable for MPLS traffic engineering. Also, in terms of supporting quality of service, building VPNs or implementing Voice over IP, each offer similar capabilities. Moreover, both were implemented

and supported by different groups of major equipment vendors. However, the IETF decided (see [4]) that CR-LDP (RFC 3212) will never be progressed beyond its current Proposed Standard status, that is, it will never become a full standard. This makes RSVP-TE the preferred option for signalling within MPLS networks. Moreover, extensions to RSVP-TE are now being developed in order to provide signalling in GMPLS networks [19].

2.3.3 MPLS and QoS routing

Whatever signalling protocol is used, it only distributes the labels that LSRs have assigned for different *Forwarding Equivalence Classes (FEC)*. Therefore, the label distribution protocol builds ingress-egress paths for each FEC by using the available routing information. The route selection for different LSPs could be done either by using *hop-by-hop* routing or by *explicit* routing [135].

Hop-by-hop routing allows each node to independently choose the next hop for each FEC. This is the usual mode in existing IP networks. That means that the labelled paths are constrained by the same shortest path route selection strategy that applies in a conventional IP network.

In an explicitly routed LSP each LSR does not independently choose the next hop; rather, a single LSR, generally the LSP ingress or the LSP egress, specifies several (or all) of the LSRs in the LSP. If all the LSRs in the LSP are specified, the LSP is *strictly* explicitly routed. If only some of the LSRs traversing the LSP are specified, the LSP is *loosely* explicitly routed. The sequence of LSRs followed by an explicitly routed LSP may be chosen by configuration, or selected dynamically by using a routing algorithm. This algorithm can support QoS.

As can be seen from this route selection mechanism, MPLS is a very useful tool for emulating a connection-oriented network on a pure datagram network [108]. The MPLS connection is the LSP and it carries datagram traffic. By using LSPs in

a manner similar to a connection-oriented network, MPLS can provide many of the same advantages of a connection-oriented network (i.e. QoS support) while still retaining the underlying efficiency of a datagram network.

2.4 Active Networks

Traditionally, network nodes would not perform any kind of computations on the packets passing them, except connectivity-related computations (e.g., routing, congestion control). This kind of network can be regarded as "passive". The limitations of *passive* networks are [130, 155]:

- the difficulty of integrating new technologies and standards into the shared network infrastructure;
- the reduced performance due to redundant operations at several protocol layers;
- the difficulty to accommodate new services in the existing architectural model.

Over time, as computing power became cheaper, more and more functionality is deployed inside the network, in an effort to provide better service to users. Examples of such functionality include: admission control, explicit congestion notification, and packet filtering. In 1994, *active networking* emerged from a DARPA-funded initiative as a next step in this evolution by allowing users to program the network [130, 154, 155].

Active networks are active in two ways [155]:

1. routers and switches within the network can perform computations on user data flowing through them;
2. users can "program" the network, by supplying their own programs to perform these computations.

2.4.1 Active Networks Overview

The concept of active networking emerged from discussions within the DARPA[17] research community in 1994 and 1995 on the future directions of networking systems [154]. The idea of messages carrying procedures and data came as a natural step beyond the traditional circuit and packet switching, and can be used to rapidly adapt the network to changing requirements.

A formal definition of active networks is given in [130]:

> *An active network is a network that allows intermediate routers to perform computations up to the application layer. In addition, users can program the network by injecting their programs into it. These programs travel inside network packets and are executed in intermediate nodes resulting in the modification of their state and behaviour.*

Such computations performed on network packets may vary from redirecting the packet to modifying its content. Figure 2.14 illustrates how routers of an IP network could be augmented to perform such customised processing on the datagrams traversing them. These active routers could also inter-operate with legacy routers, which transparently forward datagrams in the traditional manner.

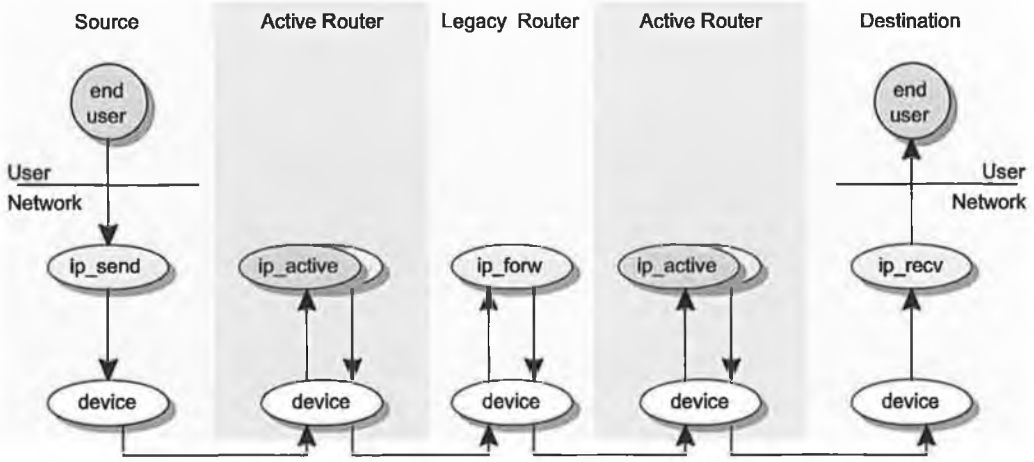


Figure 2.14: *Application-specific processing within the nodes of an Active Network*

[17] See `http://www.darpa.mil/ato/programs/AN/index.htm`

2.4.1.1 Architectures

There are three architecture approaches for implementing active networks. They differ by the placement of the code in the network (i.e. code residing in nodes, in packets or in nodes and packets). Table 2.2 presents some examples for the three approaches.

Table 2.2: *Active networks architectures*

Active nodes	Active packets	Active packets and nodes
aGIT	Smart Packets	Switchware
DAN	Active IP Option	NetScript
ANTS	M0	

The *active nodes* approach. The authors of [154, 155] refer to this as the *discrete model* or *Programmable switches* approach because the programs and data are carried separately (i.e. are discrete), while [130] refers to it as the *active node approach.*

In this approach, the packets carry some identifiers or references to predefined functions that reside in the active nodes. The packets are active in the sense that they decide which functions are going to be executed on their data, but the actual code resides in the active node.

Examples of "active nodes" architectures are the following.

- An **architecture for Active Networking** proposed at Georgia Institute of Technology [23] (referred to in Table 2.2 as aGIT). In this architecture users control the invocation of predefined network-based functions through control information in packet headers . Users can select from the available set of functions to be computed on their data and can supply parameters as input to those computations. The available functions are chosen and implemented by the network service provider, and support specific services. Thus, users are able to influence the computation of a selected function but cannot define arbitrary functions to be computed.

- **DAN Architecture.** The packets in the *Distributed Code Caching for Active Networks (DAN)* architecture [56] contain a finite sequence of function identifiers, and parameters for the functions. The functions are daisy-chained in the sense that one function calls the next according to the order of the identifiers in the packet. However, only a subset of the functions may be called. That depends on the type of node that the packet is processed upon and on the packet's content. If the node is unable to locate a function, it temporarily suspends the processing of the packet and calls a "code server" for the implementation of the function. The code server is a well known node in the network which provides a library of functions for different types of operating systems from various developers. Once the module is downloaded, its code is cached locally on the node in order to prevent more downloads of the same module. The option of downloading modules differentiates this technology from the previous one.

- **ANTS.** The *Active Network Transport System* (ANTS) [166], is a Java-based active network toolkit where different applications are able to introduce new protocols into the network by specifying the routines to be executed at network nodes that forward their messages.

  The active node approach was selected to implement ANTS so that the implementation of the new protocols would have limited access to the shared resources. Thus, there were defined a set of primitives which can be used by applications to define their processing routines. The main primitives were either operations which involved the cooperation of the *active node* (e.g., queries on environment information like node location, state of links, routing tables, local time, or the possibility to store application-defined objects for a short period of time) or operations performed on the *"active" packet* (e.g., access to both header fields and payload, duplication of the "active" packet, creation and forwarding of new "active" packets, or discarding of

the "active" packet).

The active packets approach. The Active packets approach, referred to as *the integrated approach* or *capsules* in [154, 155] is characterised by the fact that the code is carried by the packet. The nodes are also active because they allow computations up to the application layer to take place, but no active code resides in them. This is the reason why [130] refers to this approach as *active packets*.

Being carried by the packet the code within an active node cannot be very large, but only composed of "primitive" instructions, that perform basic computations on the packet's content.

Some "active packets" architectures are described below.

- The **Smart Packets** project was proposed at BBN Technologies [147]. In an attempt to provide a rich and flexible programmable environment without overloading the computer power of the managed node and without making an environment so complex that it is difficult to be secure, two decisions were made:

  1. programs must be self contained and must fit entirely into one packet (which cannot be more that 1 Kbyte long);
  2. the operating environment must provide safety and security because packets containing executable code are extremely dangerous.

- **Active IP Option [167]** describes an extension to the IP options mechanism that supports the embedding of program fragments in datagrams and the evaluation of these fragments as they traverse the network. Legacy passive packets are replaced by active capsules, which are miniature programs that are executed as they travel. These capsules can invoke predefined primitives that interact with the local node environment, and leave information behind in a node that they have visited.

- **M0 Architecture [16]** The messenger in the M0 system is similar to the capsule or the smart packet. Messengers are programs exchanged between M0 nodes. There are four elements inside the M0 node:

  1. concurrent messenger threads;
  2. a shared memory area;
  3. a simple synchronisation mechanism;
  4. channels toward neighbouring nodes.

  Messenger code is written in the M0 language. M0 is a high level language that inherits from PostScript the main concepts of operand, dictionary, execution stack, and the main data manipulation and flow control operators. The M0 interpreter is written in C.

The active packets and nodes approach. The third approach is a combination of the previous two, where active packets carry the actual code and other more complex code resides in active nodes. Using this approach, users can choose either the active packets approach or the active nodes approach according to the nature of their application.

Examples of "active nodes and packets" architectures are the following.

- The **SwitchWare Architecture** [2] uses a layered architecture to provide a range of different flexibility, safety and security, performance, and usability tradeoffs. The three layers defined in SwitchWare are:

  1. *Active Packets* - to realise the active packets approach;
  2. Active extensions called *Switchlets* - to realise the active nodes approach;
  3. The *Active Router Infrastructure*.

  Active packets are used within the first layer. They are relatively short programs with limited functionality written using the programming language

PLAN (*Programming Language for Active Networks*) [81]. Active packets can access a lower layer of software components with greater functionality called *switchlets*, which reside in the active nodes and can be dynamically loaded. While the top two layers support several forms of dynamic flexibility, the lower layer is primarily static [2]. The *active router infrastructure* layer represents a base layer of the architecture with the goal of providing a secure foundation upon which the other two layers build.

- **NetScript Architecture.** The NetScript Project [172] seeks to program networks efficiently. Therefore, it provides:
  - an architecture for programming networks;
  - an architecture of a dynamically programmable network device/node;
  - a language called *NetScript* for building network software on a programable network.

  NetScript uses delegated agents to program and control the functions of intermediate network devices/nodes.

A brief comparison between the three approaches. The active nodes approach has good performance because security issues are of less concern than in the active packets approach. However, the flexibility of the relevant architectures is limited. In an effort to increase flexibility, the DAN and ANTS architectures have adopted a scheme where code is downloaded on demand and is cached for future use. As a result, these two technologies can easily deploy any new arbitrary protocol. Nevertheless, downloading code on demand causes some delay that reduces the overall performance.

The active packets approach suffers from performance related problems because of the significant safety and security requirements. In an effort to reduce the security burden and thus increase performance, some researchers have decided to restrict the functionality of the programs carried by the active packets.

The combination of both approaches seems to be very appealing. The SwitchWare architecture realises this idea by the use of a layered architecture, and manages to provide a range of different flexibility, safety and security, performance, and usability tradeoffs. Finally, the NetScript architecture proposes a novel viewpoint where the network is treated as a single programmable abstraction.

2.4.1.2 Applications

The most important applications of active networks stems directly for their ability to program the network. New protocols and innovative cost-effective technologies can be easily employed at intermediate nodes. The main active network specific applications can be grouped, as stated in [107], into four classes:

Fusion. The active node forwards fewer packets than it receives. Examples of such filter-type active applications are mixing sensor data [154], scaling multimedia content within the network [15].

Fission. The active node forwards more packets than it receives. Multicast video distribution [37] is a good example of fission applications.

Caching. Active nodes which lie directly in the paths followed by the requests travelling from clients to servers. It has more efficiency in the case of intra-network caching of rapidly changing data as in [107].

Delegation. The active node performs tasks delegated to it by another node. Examples of such applications include online auctions [107] and congestion control [164, 167].

Specific examples of active networking applications span a wide variety of areas. Amongst the most important areas are:

Network Management. Actives nodes provide a natural answer to the network management problem as they are able to move the management centres to the in-

termediate nodes within the network. Thus, the responses delays and the bandwidth maximisation for management purposes are both reduced. Moreover, the problems are discovered quickly and reported automatically. Also, "first aid" code can be injected into packets [130]. This code can be used in case a problematic node is encountered. Other packets can act as "patrols", constantly looking for anomalies as they trace the network.

Several projects considered the use of active networking to improve network management. The Smart Packets project [147] at BBN developed architecture, languages, and protocols for making managed nodes programmable. The Netscript project [172] at Columbia developed techniques to automatically create systematic management instrumentation and the corresponding MIBs, using the structure of active elements.

Congestion Control. Active networking is a suitable tool for dealing with congestion. That is because congestion is an intra-network event and it often takes a considerably long time for notification information to propagate. Thus, by using conventional[18] methods of dealing with congestion, in the time which elapses between the occurrence of congestion and the moment when the notification reaches the applications contributing to traffic congestion, either the congestion can increase or it may have dissipated so that self-regulation is no longer needed. Therefore, schemes that require the sender to adapt to network conditions have well known limitations, including the time for the sender to detect the condition, react to it, and transmit the adapted data to the receiver.

Active network efforts in this area have included transparent inline protocol "boosters" to adapt to network conditions (e.g., by adding forward error correction over errorprone links) and intelligent discard strategies for preserving the quality of MPEG video in the face of network congestion [24].

[18]Usually applications reduce their traffic as they get notified about the congestion (e.g., the TCP window sliding mechanism).

Multicasting. The "traditional" IP multicast service hides the details of the routing topology and the number and location of receivers from its users. For unreliable multicast, this approach makes sense and allows scaling to large applications; however, there are inherent problems in using this model when it is desired to deliver data to all receivers reliably. The difficult arises in recovering from losses, which typically affect all receivers downstream in the multicast tree from the point of a loss. A common approach that can be implemented using active networks is to spread the responsibility for multicast retransmissions among the receivers, to avoid overburdening the sender. For good performance, this requires that receivers be aware of nearby receivers, that are above the loss point, to provide retransmission.

By including state and processing (i.e. active networks) in the network, retransmissions can be directed to nearby receivers or can come from caches at network nodes [107]. Both approaches reduce the delay and transmission resources required for retransmission.

Caching. A substantial fraction of all network traffic today comes from applications in which clients retrieve objects from servers (e.g., the WorldWide Web). The caching of objects in locations "close" to clients is an important technique for reducing both the network traffic and the response time for such applications. Caching schemes require decisions about where to locate objects and how to forward requests between caches. Current wide area caches are manually configured into a static hierarchy, thus incurring an administrative burden and limiting the ability to react to dynamic conditions.

Efforts to use active networking for caching include using network mechanisms to route cache requests to preconfigured cache locations [107], and combining small caches with information about the contents of nearby caches, at each network node [25].

2.4.2 Active Networks advantages for routing

Active networks present the following advantages compared with the traditional routing mechanisms:

Fast deployment. A key reason for using *active networks* for routing purposes is that they allow users to implement their own way of packet routing without waiting for new standards to be deployed. Moreover, support for customised protocol mechanisms can be introduced in a running network.

Implementation at various OSI layers. Using *active networks*, the code written by users can be implemented at any OSI layer from the network up to the application layer. Therefore, the behaviour of various layers can be tailored to the needs of the specific application.

More flexible routing. Changes in routing protocols require the approval of most of the competing carriers and manufacturers. By using Active Networks, the protocol can be adjusted by the end-users or applications according to their needs.

Interconnection of autonomous systems. In large networks, traffic may cross multiple autonomous systems. In the Internet, BGP is the main protocol used for inter-AS routing. Unfortunately, BGP cannot make routing decisions based on a QoS constraint but only on connectivity information. However, in Active Networks, active packets could carry rules or constraints to select a feasible path. In this way the source network would have control over the packet's path even while it traverses another autonomous system without needing a standardised protocol or an inter-AS agreement.

Diversity and Optimality. Many active networks-based routing techniques may co-exist in the network at one given time. Different end-users and applications may use different techniques. This means that best routing practice is determined from a larger set of protocols that can be deployed in current net-

works, so that the network may be expected to converge more rapidly on the best protocols. Also, applications with specialist requirements can be more rapidly accommodated than in conventional networks.

The co-existence of a number of routing protocols in the network would require standards for the collection of, and setting of, network state, so that each protocol may respond to network state changes caused by others (e.g., reservation of bandwidth) but otherwise the current need for routing protocol standardisation would not apply.

2.5 Mobile Software Agents

Current routing algorithms are no longer adequate to face the increasing complexity of today's networks because:

> *"centralised algorithms have scalability problems; static algorithms have trouble keeping up-to-date with network changes; and other distributed and dynamic algorithms have oscillations and stability problems"* [22].

Some researchers, including this author, believe that mobile agents are a promising solution for future routing as they are autonomous entities, both proactive and reactive, and have the capability to adapt, cooperate and move intelligently from one location to the other in the communication network [26]. These are properties that allow a route to be searched for in the network, rather than being passively requested from it.

2.5.1 The Mobile Agent Paradigm

The mobile agent technology originally emerged from advances made in distributed systems research [38]. All started by sending executable programs between clients and servers, followed by methods of remote dispatch of script programs

or batch jobs. Mobile agents are considered to be an extension of such techniques [45].

The word *agent* has a universal meaning and is defined by Green and Somers as [77]:

> *"...a computational entity which: acts on behalf of other entities in an autonomous fashion; performs its actions with some level of proactivity and/or reactiveness; and exhibits some level of the key attributes of learning, cooperation and mobility."*

Green and Somers explain this definition by considering a travel or an estate agent, which *acts on behalf of others* (e.g., the estate agent sells houses he does not own and the travel agent sells flight tickets or reserves rooms in different hotels). While doing so, they have *autonomy* (e.g., the estate agent can make viewing appointments for unoccupied properties without reference to the owners). In order to reach its goal, an agent can behave *proactively* (e.g., an estate agent which advertises the property in the local press) or *reactively* (e.g., an estate agent who simply places a *"For Sale"* sign outside a property for sale and waits for purchasers to come into his shop).

A mobile (intelligent) agent is defined as [45]:

> *"...a software entity which exists in a software environment. It inherits some of the characteristics of an Agent (as defined earlier). A mobile agent must contain all of the following models: an agent model, a life-cycle model, a computational model, a security model, a communication model and finally a navigation model."*

The *software environment* in which the agent exists is also known as the *mobile agent environment* and is defined as [77]:

> *"...a software system which is distributed over a network of heterogeneous computers. Its primary task is to provide an environment in which mobile*

agents can execute. The mobile agent environment implements the majority of the models which appear in the mobile agent definition. It may also provide: support services which relate to the mobile agent environment itself, support services pertaining to the environments on which the mobile agent environment is built, services to support access to other mobile agent systems, and finally support for openness when accessing non-agent-based software environments."

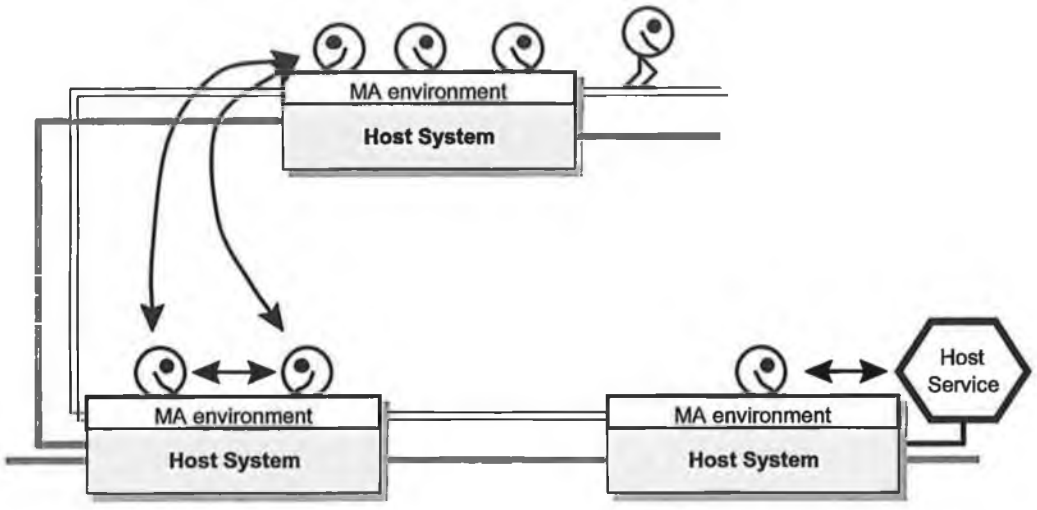


Figure 2.15: *Basic mobile agent architecture*

Figure 2.15[19] explains the place and role of a mobile agent environment. Here, the smiling faces are mobile agents that travel between mobile agent environments, which are built on top of host systems. Communication between mobile agents (local and remote) is represented by bi-directional arrows. Communication can also take place between a mobile agent and a host service.

In addition to the basic model (i.e. the *agent model*) any software agent defines a *life-cycle model*, a *computational model*, a *security model*, a *communication model* and a *navigational model* [45, 77].

the agent model defines the intelligent part of a mobile agent (i.e. the autonomy, learning and co-operative characteristics of an agent plus the reactive and

[19] A similar image is also presented in [77].

proactive behaviour);

the life-cycle model defines the different execution states (e.g., start, running[20], frozen, task[21], death)[22] of a mobile agent and the events that cause the movement from one state to another;

the computational model defines how a mobile agent executes when it is in a *running* state;

the security model is concerned with three main classes of problems [45, 121]: *protecting host nodes from destructive mobile agents, protecting mobile agents from destructive hosts*, and *protecting mobile agents from each other*;

the communication model defines communication protocols with other entities (e.g., users, other static or mobile agents, the host mobile agent environment or other systems);

the navigation model involves all aspects of agent mobility from the discovery and resolution of destination hosts to the manner in which a mobile agent is transported.

2.5.1.1 Mobile Agent Technologies

A number of mobile agent technologies have been created. Some of them rely on their own programming foundations, while others make use of existing programming languages such as Java or Tcl/tk languages, both designed by Sun Microsystems.

Table 2.3 lists the best known mobile agent technologies. Three of them are reviewed here: *Aglets*, *AgentTcl* and *Telescript*. The *WAVE* system is described in Sections 2.5.4, 3.4.2 and in Appendix A.

[20]The *running* and *frozen* execution states are defined in the life cycle models adopted by Telescript and AgentTCL, called *persistent process model*.
[21]The *task* based model is specific to Aglets.
[22]See more about the life-cycle states in [77]

Table 2.3: *Mobile agent systems*

System	Based-on	Organisation
Aglets	Java	IBM Japan
AgentTcl	Tcl/tk	Dartmouth College
Telescript	custom	General Magic, Inc.
Mole	Java	University of Stuggart
Ara	Tcl	University of Kaiserslautern
Tacoma	Tcl	Cornell University and University of Tromso
Wave	Wave language	University of Karlsruhe and University of Surrey

Aglets [40]: This mobile agent system introduces program code that can be transported along with state information, but not the current execution context. Aglets are Java objects that can move from one host to another in the Internet. While executing on one host, an aglet can suddenly halt execution, dispatch to a remote host, and resume execution there. Being based upon a programming extension made to the Java language it provides an enhanced level of migration. However, aglets only support communication by means of message passing. Moreover, the security support is weak.

AgentTcl [95]: It is built on top of the Tcl/tk scripting system and lacks the power and flexibility of a full computer language such as Java. However, its mobile agents can communicate by direct streaming connections or by using a messaging scheme. This system provides security enhancements to limit unauthorised access of key resources.

Telescript [168]: It is an object-oriented mobile agent language. In contrast with the previous two mobile agent technologies which are free, Telescript is a very expensive piece of software which has limited its popularity. Telescript has three main concepts: *agents*, *places* and the go instruction. Agents travel from place to place using the go instruction, being able to maintain their execution state during travel. The places in Telescript are equivalent to the concept of a mobile agent environment where static services are located at a

host. Moreover, Telescript agents can interact with each other and with any services located at the places.

2.5.2 Mobile Agents advantages for QoS routing

A number of research groups have pioneered efforts to introduce new mobile agent based routing mechanisms [48, 59, 71, 88, 121]. They were motivated by the following mobile agent advantages over traditional routing techniques.

"Strong mobility" [17]. Mobile agents encapsulate code (intelligence) and data (states). Each agent runs independently of all others and by navigating the network they build solutions and modify the problem by using the collected information.

"Stigmergy" [59, 88, 121]: Mobile agents have the ability of indirect communication by leaving information on the nodes they have visited. Other agents visiting those nodes can access such information. It is argued that stigmergy is a robust and adaptive mechanism for information sharing. The term was first defined by Grass in [75] and describes how ants find shortest paths by using the pheromone trail deposited by other ants.

"Emergent behaviour" [88]: A complex behaviour may be accomplished using simple interactions of autonomous agents, with simple primitives.

Robustness and fault tolerance [26, 77, 88, 100, 111]: Robust and fault tolerant distributed systems are easier to build by using mobile agents as they have the ability to dynamically react to unfavourable situations. Therefore, the loss of network nodes or links will not affect the overall distributed system, leading to a graceful, scalable degradation by dispatching mobile agents from those specific nodes or links somewhere else or by terminating them.

Adaptability [77, 88, 100, 111]: Mobile agents have the ability to autonomously react to changes in their environment. They can change, die or reproduce

according to the network changes. The population of mobile agents can increase/decrease according to the size of the network; they react to the loss of network nodes or links by terminating or dispatching themselves somewhere else in the network; they can extend their capabilities on-the-fly, by downloading required code off the network; they can change over time, so that new usage contexts and models can be accommodated. Moreover, propagation of changes in the network is sometimes faster than by using traditional methods (e.g., the Bellman-Ford algorithm) [22].

Efficiency [26, 45, 77, 121]: Mobile agents consume fewer network resources as they move the computation to the data rather than the data to the computation. Code is often smaller than the data it processes. In such situations, transferring mobile agents to the source of data creates *less traffic* than transferring the data. Moreover, computing data in a distributed manner is less computational intensive for an individual device than computing it centrally. *Space saving* is another efficiency issue where mobile agents have an advantage because a mobile agent resides on only one node at a time. Moreover, mobile agents are able to duplicate and distribute themselves in the network, and therefore each agent can execute on different machines in parallel. This means that mobile agents are also *time efficient*.

Modularity [88, 111]: Different mobile agents solve different parts of the problem obtaining intermediate results which are used to obtain the final, global solution.

Autonomy [26, 77, 88, 111]: Once released into the network, mobile agents can operate asynchronously and independently of the sending program. They will autonomously decide which locations they will visit and what instructions they will execute.

Flexibility [45, 121]: Mobile agents can change over time, new usage contexts

and models can be accommodated. For instance, multiple network management strategies can coexist and co-evolve.

Distributivity [77]: Mobile agents are inherently distributed in nature and hence offer a natural view of a distributed system. Moreover, a more realistic and up-to-date network image can be obtained by applying the computational operations to data directly where it is physically located.

Parallelism [88]: Mobile agents are able to duplicate and dispatch themselves within the network and therefore agents will execute on different machines in parallel. For a large amount of data a single computer might not be able to process all data within a given time frame.

Extensibility: Mobile agents can extend their capabilities on-the-fly, by downloading required code off the network. They can initially be equipped with minimal functionality, which can be enhanced depending on future requirements [26].

The main disadvantage of using mobile agents in the routing process is the communication overhead. Methods for controlling such overhead are investigated in Chapters 5 and 6.

2.5.3 General problems associated with mobile agents

In general, each mobile agent runs independently of other mobile agents, it moves itself to another node and it can preserve state when it moves from one node to the next. To achieve this, a mobile agent has to encapsulate and bring with it its entire thread of execution (e.g. when and where it moves, how and whether it communicates to other mobile agents). Therefore, the majority of mobile agent implementations generate "fat" and complex mobile agents.

To minimise this overhead, a related paradigm called WAVE[23] was used for

[23]See Section 2.5.4 and Appendix A.

the work presented in this thesis. This technology differs from other mobile agent technologies as the code/language used by it describes semantics of *"what to be done in a distributed space"* rather than implementing code for *agents*. The tasks described by WAVE programs are however performed by the small mobile-agent-like entities called *waves*, which are dynamically generated by one or more WAVE interpreters at the WAVE code injection[24] time. Moreover, this technology has a recursive spatial control mechanism, which coordinates societies of small mobile agents. These control mechanisms are implemented via interpreters distributed on the computer network. Thus the size and the complexity of mobile-agent-like *waves* used by this paradigm are greatly reduced.

Another important Wave feature is that it dynamically creates distributed virtual networks on top of the physical ones as described in Section A.1. This allows WAVE, unlike other mobile agent technologies, to assign code and data to both nodes and links in these virtual networks which can be navigated and processed/modified by *waves*. Hence, WAVE can be used to implement active networks as well. A more detailed description of WAVE technology is presented in the following subsection.

2.5.4 The WAVE Technology

The Wave technology [139, 142, 144, 146] is based on installing multiple copies of intelligent agents throughout the network which can do *local data processing, exchange information with other subsystems and between themselves*, and *interpret a special navigational Wave language* [145]. A recursive code written in this language is dynamically self spreading in the network space in a parallel and cooperative mode. The code (*wave strings* or *waves*) may be injected from any node within the network and propagate together with intermediate data. The main advantages of Wave over other mobile agent implementations are:

[24]See details in Appendix A.

Minimal and specialised code. Mobile agent implementations are usually based on object oriented technologies (e.g., Java, C++). These technologies are intended more for local data processing and communications with other programs on a client-server basis. Unlike them, Wave is based on self-navigation and a pattern matching philosophy. Therefore, many network navigational processing tasks for which Wave is well suited, written in other languages, will inevitably have to explicitly include a variety of special functions which are hidden in the Wave interpreter and shared by many *waves*. This is the reason why Wave code is very compact, typically 20 to 50 times shorter that equivalent programs written in C/C++ or Java [145]. A comparison between Wave and Java (which is observed to be the most widely used platform for the implementation of mobile agents) is presented in [160].

Powerful recursive spatial control: There are two dynamic service layers operating between mobile application programs (waves) and the computer networks they propagate through [144, 146]:

1. The *Knowledge Network (KN) layer* is arbitrarily distributed between computers of the actual (physical) computer network, where each computer may have 0^* (none, one or more) nodes allocated to it. Links of KN may therefore connect nodes within the same or between different computers. Within KN, any information (declarative or procedural) can be associated with both nodes and links. Nodes within KN have absolute addresses in space.
2. The *Dynamic tracks layer* consists of tracks which accompany and support the spreading and coordination of *waves*. Starting in different nodes of KN they grow as spanning trees and serve as a special dynamic control infrastructure.

This layering organisation of Wave is depicted in Figure 2.16 and described

in much more detail in [143]. The use of these layers allows the application

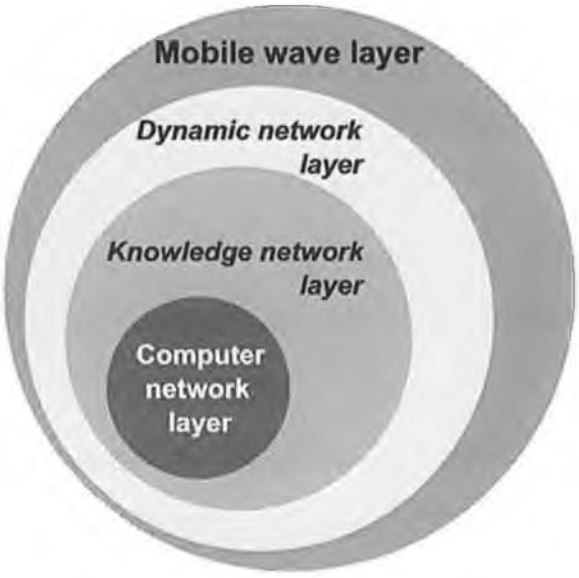


Figure 2.16: *Layered Wave Model*

programming to be relieved of most routine tasks (such as synchronisation, message passing, garbage collection, etc.) which have to be explicitly managed within traditional distributed systems. Therefore, the Wave model can operate without any centralised control and can support robust distributed algorithms working efficiently in a rapid changing environment while maintaining integrity of the societies of mobile cooperative processes and full control over them from any point in space [143].

Different types of internode communication: There are three type of communication between KN nodes [139]:

1. *Surface* communication refers to bidirectional channels between two adjacent nodes.
2. *Tunnel* communications are direct transmission of messages to remote nodes by using their unique address.
3. *Loops* is an abstraction symbolising *time-propagation* through KN with some computational processing in nodes.

Surface and *tunnel* links, representing *space-propagation*, may be used in both *broadcasting* or *selective* modes.

2.5.4.1 The Wave Language

Wave is an interpreted language, oriented towards either software or hardware execution [143]. A Wave program consists of sequences of spatial actions called *moves*, which propagate and process data across a KN. The sequence of moves can be either executed sequentially (when the moves are separated by periods), or become independent pieces of executing code which are executed in parallel but inherit all the functional characteristics of the original wave program (when the moves are separated by commas). An example is depicted in Figure 2.17.

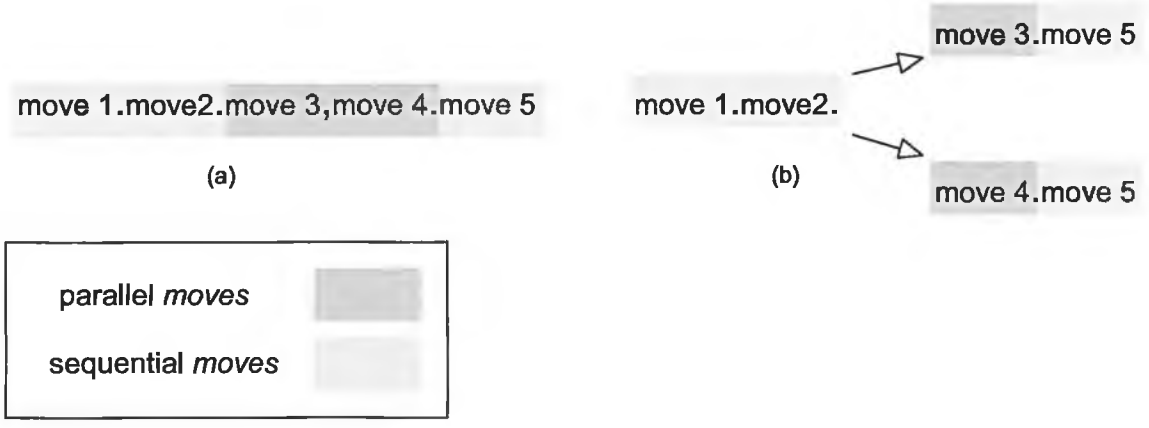


Figure 2.17: *(a) a Wave structure (b) seen by the Wave interpreter*

The execution of individual moves can return four different values: TRUE, DONE, FALSE, and ABORT. Failure to properly execute a move or wave causes the interpreter to generate a FALSE value, which halts further execution of the wave. A DONE value indicates the completion of the current wave being executed in the KN, and further wave executions may or may not continue, depending on the design structure of the wave. A TRUE value returned indicates full success of the specific move, and further development of additional waves is consequently performed. An ABORT state causes an emergency halt of the whole Wave program before being cloned by using a comma, or rule, as described later.

The moves of a wave may be composed of different types of operations known as acts. An act may work on two different operands. Moves may also be composed of other operations, such as assignments to variables, filter operations applied to variables, and rules.

A more detailed description of the WAVE language is presented in Appendix A.

2.5.4.2 Wave Navigational Techniques

The space navigation mechanisms used by *waves* to traverse the KN in their search are called *spreads*. The basic Wave *spreads* are depicted in Figure 2.18.

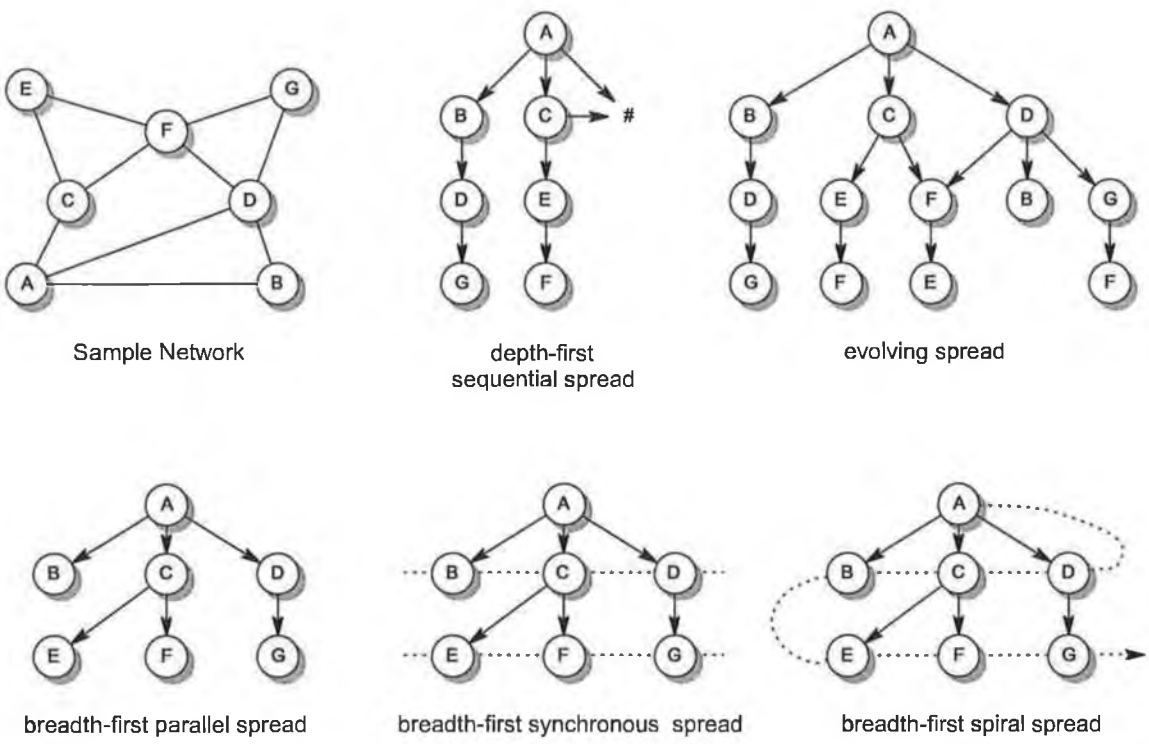


Figure 2.18: *Basic Space Navigation Mechanisms in Wave*

Depth-first sequential spread (DFSS). This technique is extensively used in solving graph theory problems. Here, waves are generated to conduct a search with as much depth as possible on each incident edge of the starting node in a network. When the searching process in a given branch halts, the search is put

back in action with the next incident edge of the last node with an alternative route in the same branch (if applicable) - otherwise the search is tracked back to the next available branch in the starting node. This procedure is repeated until no further progress is possible.

Evolving spread (ES). This navigation method permits the propagation of waves throughout the network as long as the nodes reached have not yet been visited by the same waves. To verify this condition, traversing waves carry with them a list of nodes previously visited, rather than leaving a record of their presence at visited nodes. Inherently, this navigation method prevents the formation of loops; however, in large networks, waves following this navigation technique may become large if the depth of the spread is extensive. Also, the fact that waves travel across the network without mutual cooperation implies that higher traffic is generated, as individual waves may traverse paths already traversed by other waves.

Breadth-first parallel spread (BFPS). In this navigation mechanism, a wave starts in a given network node, it clones itself, and it spreads in a fully asynchronous and parallel way to all neighbouring nodes, creating a breadth-first tree. Specifically, a wave following this navigation method verifies that the current network node has not been previously visited by other waves. In such a case, the node is marked, and the spreading algorithm continues; otherwise, the wave halts. This method guarantees finding a spanning tree that includes all the network nodes. In addition, Wave also allows the implementation of this method using a synchronous scheme. In doing so, the waves propagating in the breadth-first navigation technique not only share information, but also signal each other to synchronise their rate of progress while they traverse the network.

Breadth-first synchronous spread (BFSS). The Wave language allows us to describe completely asynchronous distributed processes [143]. However, it may also be used to create any (logical) synchronisation, such as this technique. It re-

sembles *BFPS*, but the network is traversed in a stepwise manner. That means that further spreading through links by one parallel step may start only after full completion of the preceding step.

Spiral spread (SS). There might be cases where only one node at a time is active when implementing a synchronous breadth-first navigation technique. In such cases, a spiral spread method may be followed, in which all the direct nodes involved in a breadth-first navigation scenario are sequentially made active. This allows for a gradual navigation of the network, while keeping traffic overhead to a minimum.

2.6 Summary

This chapter introduces five main topics, i.e. routing, QoS, MPLS, active networks and mobile agents, that underpin the research presented in this thesis. They all offer solutions but also raise issues which are described in this chapter. The main drawback of current routing is that most strategies used in the Internet select paths based only on a single metric (i.e. *hop count* or *administrative cost*). However, a single (static) metric does not suffice for demanding (e.g., time-sensitive and/or bandwidth-intensive) traffic, which has specific requirements. To satisfy such requirements *QoS routing* algorithms are needed to compute end-to-end paths based on multiple constraints such as: bandwidth, delay, and jitter. Deploying QoS routing is very difficult in the legacy Internet, but is facilitated in MPLS networks by the separation between the control and the forwarding planes. Developing a feasible QoS routing mechanism is still a topic of active research. Some researchers, including this author, believe that efficient QoS routing strategies can best be implemented using modern technologies such as *active networks* and/or *mobile agents*.

This chapter reviewed the main routing approaches currently used in the In-

ternet and outlined the concepts of *Quality of Service* (QoS) and QoS routing. It also presented the historical evolution of MPLS along with its main components. *Active networks* and *mobile agents* concepts were introduced along with their advantages over traditional techniques for performing routing. The next chapter discusses how *active networks* and *mobile agents* can help QoS routing to achieve end-to-end QoS goals in the Internet and why they are optimal solutions for an efficient QoS routing scheme.

CHAPTER 3

QoS routing challenges and solutions

"That's how it is with people. Nobody cares how it works as long as it works." Councillor Harmann, The Matrix Reloaded

"There are programs running all over the place. The ones doing their job, doing what they were meant to do, are invisible. You'd never even know they were here. But the other ones, well, we hear about them all the time." The Oracle, The Matrix Reloaded

Quality of Service routing has generated much debate in the research community exactly because there are still very many unsolved issues. Some of these will be addressed in this chapter. Its main focus is on reducing the communication, computation and storage overheads of traditional routing solutions by the use of hierarchical routing and/or techniques such as active networks and mobile agents. Moreover, multi-constrained routing is also a hotly debated and still unsolved QoS routing issue due its high complexity. Thus, this is also a subject of concern in this chapter.

3.1 QoS routing tradeoffs

QoS routing requires topology and resource availability information in order to compute QoS routes. If policy constraints are considered while computing the final path, the process of route calculation is referred to as *constraint-based routing*. It is used in QoS routing to select routes that can meet certain QoS requirements, but can also be used to, for example, increase maximisation of the network. In this case, constraint-based routing may find a longer but lightly loaded path instead of the heavily loaded shortest path. In this way the network traffic is distributed more evenly.

3.1.1 Meeting the Requirements vs. Additional Overhead

The very existence of QoS routing is a tradeoff. QoS routing finds paths that meet the needs for QoS requirements of flows. However, it also introduces communication and computation overhead, increases the routing table size, and consumes more resources when selecting longer paths. It can also create routing instability. Some of these issues are separately addressed in subsequent sections. Questions to be addressed include:

- *Is the increase in performance worth the added cost?*
- *Can QoS routing achieve its goals ... in terms of:*
  - *finding paths satisfying multiple QoS constraints?*
  - *offering improved services to its users?*
  - *using efficiently the resources of the overall network?*

3.1.2 Accuracy vs. Scalability

The primary issue for QoS routing solutions in very large networks is *scalability* [5, 80, 103]. Unlike routing protocols that are based on relatively static information

(e.g. path hop count), QoS routing requires the frequent updating of dynamic network state information. The update messages consume significant network bandwidth. Storing such an amount of state information needs increased storage space, while processing it requires much more computational power than best-effort routing. Thus, a large scale deployment of QoS routing generates three main types of overhead: *communication*, *computation* and *storage*.

The authors of [79] present the major techniques to reduce protocol overhead and enhance scalability of QoS routing. A possible classification of such techniques is depicted in Figure 3.1.

I first classify the main scalability solutions corresponding to the three major types of overhead. Solutions for reducing the computational overhead include pre-computation and path caching, which are both approaches which tend to amortise the cost of on-demand computation. In the same class I also consider the routing computation itself, because centralised routing generates much more computational overhead compared with distributed and/or hierarchical routing.

The communication overhead is usually generated by the routing updates. Thus, there are two main approaches to reducing such overhead: considering what state information to distribute and where (quantity reduction), and considering adjusting the frequency of routing updates. Both approaches offer a reduction in the size and/or number of routing messages while preserving the routing performance.

There are no specific techniques to reduce the storage overhead. However, some of the approaches used for the other two classes have a great impact on the storage overhead as well. The most significant are the route computation (distributed and/or hierarchical routing) and the quantity reduction.

3.1.2.1 Accuracy vs. Computational Overhead

On-demand path computations are invoked for every connection request. Thus, route computation may become a bottleneck especially when using QoS routing

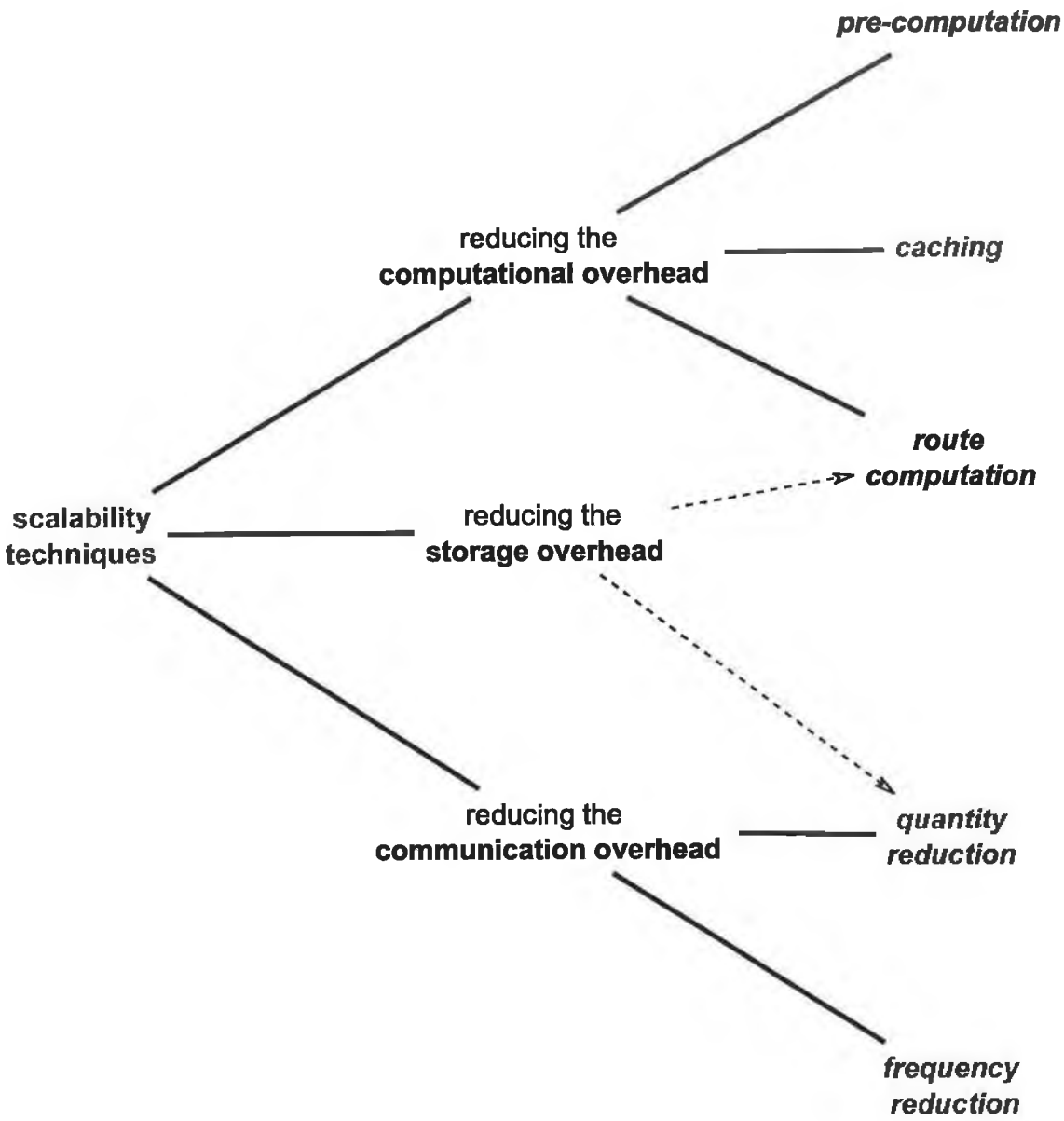


Figure 3.1: *Techniques to enhance the QoS routing scalability*

algorithms. Reducing the computational overhead (Figure 3.1) means reducing the computation involved in calculating the requested paths. This can be done either by "preparing" some paths before they are requested (i.e. pre-computation and caching) or using computation techniques that require less computational overhead than others.

Pre-computation. Hao and Zegura [79] give an overview of two main approaches to reduce the computational overhead. The first one is *route pre computation* [116, 148], which involves computing and storing the paths to all destinations before any connection request. Thus, on-demand processing of requests is reduced to path lookup instead of path computation. However, pre-computed paths may become invalid when the network state changes. This means that those paths have to be updated from time to time to be consistent with the real network state. Moreover, pre-computing paths for different constraints for all destinations can also affect the storage overhead as well as the complexity of path pre-computation. Thus, in [116] the bandwidth values are partitioned into a few classes, and the pre-computations are done for each class. In [148] the computing complexity is reduced by considering only a coarse-grained link cost metric while computing multiple paths to the same destination. Also, to reduce the additional required storage space, the pre-computed paths are placed into a compact data structure, from where they are "extracted" when needed.

Path caching. The second approach to path computation reduction is *path caching* [6], which involves caching all the paths that were computed on-demand. Thus, a cached path can be used for future requests with the same requirements. The main issue with caching schemes is that it might occur that no cached path can satisfy the request. Another issue is selecting the best policy for administrating the cache, i.e. updating or removing the paths in the cache.

Route computation. There is also a noticeable difference in computational overhead of different routing strategies that are used while computing the path (e.g., source, distributed and hierarchical route computation). Source routing needs frequent updates so that they can cope with the dynamics of the network parameters (e.g., bandwidth, delay). For large networks this may give rise to a high communication overhead. Moreover, using global state also introduces imprecision due to the non-negligible propagation delay of state messages. Also, gathering the global state centrally (at the source node) requires, for large networks, considerable memory in which to store it (i.e. storage overhead) and significant processing power to process it (i.e. computational overhead). Moreover, it is impractical for any single node to have access to detailed state information about all nodes and all links in a large network [78].

In distributed routing, the path computation is distributed among the intermediate nodes between source and destination. There are two main categories of distributed routing algorithms: hop-by-hop (requiring global state information) and flooding-based (requiring local state information). For those algorithms which require the maintenance of global state (distance vectors) within each node [158, 161], the routing decision is made on a hop-by-hop basis. Due to the global state maintenance, these routing algorithms suffer from similar overhead issues to the source routing algorithms. Moreover, if there are any inconsistencies between the global state as recorded by different nodes, loops may occur [42]. There are also flooding-based algorithms which do not require any global state to be maintained. Routing decision and optimisation is performed entirely based on local state information. Some of these use selective probing [43], while others use new approaches like active networks [66] or mobile agents [48, 59, 71, 88, 121] (see Section 3.4). Compared to conventional methods, their communication overhead is considerable reduced, since the flooding of requests avoids the need for flooding of state information.

Hierarchical routing scales best among the three routing strategies. That is

because it reduces all three overheads (i.e. communication overhead by sending detailed information only within a domain and aggregated information across domain boundaries, computational overhead by "hierarchically" computing the path and storage overhead by requiring an aggregated state). However, aggregating the network state may introduce imprecision which may have a negative impact on the QoS routing performance [78].

3.1.2.2 Accuracy vs. Communication Overhead

Reducing the communication overhead (Figure 3.1) requires a reduction of routing update information. The trade-off which has to be made is between the need for accurate information and the need to avoid frequent flooding of link state advertisements.

Quantity reduction. The goal of *quantity reduction* techniques is to reduce the number as well as the size of the routing messages while preserving routing performance [79]. To do that, the detailed and accurate state information is distributed only to a part of the network. One commonly used quantity reduction technique is *topology aggregation*, which is also described in Section 3.2.1. Another quantity reduction technique is *limited update distribution*. The main difference between the two is that in topology aggregation, nodes that do not receive detailed information receive aggregated information, while in limited update distribution, nodes either receive detailed information or no information at all.

Topology aggregation is maybe the most important technique for achieving scalability in QoS routing. That is because it can reduce the amount of routing information and routing table size by orders of magnitude. Thus, it is used in both the current Internet, as well as in almost all of the proposed large-scale QoS networks. The accuracy-scalability tradeoff of topology aggregation techniques, as presented in detail in Section 3.2.1, differs among the topology aggregation methods used (e.g., Full-Mesh, Spanning Tree, Complex Node, Star). This means that

different aggregation methods, although more scalable than others, may have a negative impact on routing performance [78]. However, the authors of [79] show that the routing performance when applying different aggregation methods is also influenced by the selection of routing update intervals.

According to [79], limited update distribution uses two main methods in order to achieve scalability by reducing the quantity of routing updates. The first method, *hop count limited flooding*, reduces the updates by limiting the number of hops they are allowed to travel. The reason to do this is that there are considerably more interactions within a local area. The other limited update distribution method is called *reverse path update* and it delivers routing updates only to the nodes that are actively using the links. Thus, each node keeps a record of active connections and sends the updates only to these connections. In both schemes, only the distribution of dynamic QoS information is limited, while static information (e.g., connectivity, reachability) is maintained within each network node. However, such strategies may have difficulty adjusting to rapidly changing traffic patterns. This is mainly because they introduce a dependence between routing updates and connections, which may negatively affect routing performance [79].

Frequency reduction. *Frequency reduction* techniques have as the final goal reducing the communication overhead by increasing the update intervals as much as possible without compromising the routing performance. There are two different approaches to achieving this goal. The first approach is to use appropriate update triggering mechanisms to trigger an update at the right time. Two types of routing update policies may be triggered upon certain conditions: a *change-based trigger* [128] (i.e. an update is triggered by a significant[1] change in link state), and a *time-based trigger* [115] (i.e. a timer can be used to either trigger an update at fixed intervals (*clamp-down* timer) or to enforce a minimum interval between two consecutive updates (*hold-down* timer)).

[1]The change can be considered significant when it passes a certain *threshold* or changes to a different *class*.

The second frequency reduction approach is to use appropriate routing algorithms to tolerate less frequent updates (e.g., widest-shortest routing which is the most insensitive to changes in routing tables, and has the best performance when the update interval is long).

3.1.3 Complexity vs. Computational Overhead

Computing optimal routes subject to constraints comprising two or more additive and/or multiplicative metrics is an NP-complete problem [161, 162]. The proof of NP-completeness is based on the assumption that all of the metrics are independent.

There are a number of different approaches used for NP-complete problems. They are described in Section D.3. Some approximation and heuristical solutions are presented in Section 3.3. Other solutions consider only special cases:

- Algorithms that consider any combination between bandwidth and one of the additive metrics are much simpler [7]. Most such proposals consider:
  - **bandwidth** and **hop count** constraints arguing that [171]:
    * There are few applications that cannot tolerate occasional violation of delay and jitter. Also, delay and jitter constraints can be mapped to bandwidth and hop-count constraints if needed.
    * Many real-time applications will require a certain amount of bandwidth. The hop count metric of a route is also important because the more hops a flow traverses, the more resources it consumes.
  - **bandwidth** and **delay**, or, more precisely the residual bandwidth and the propagation[2] delay, which are also referred to as the *width* and the *length* of a path. Arguments for using only these two constraints are [161]:

[2]The other delay component, i.e. queuing delay, is determined by the residual bandwidth and traffic characteristics [161].

  * For most applications, particularly real-time ones, the end-to-end delay is one of the most important QoS requirements. Thus, the residual bandwidth and the propagation delay are the key elements to be considered to assure the end-to-end delay.
  * After finding candidate paths satisfying the bandwidth/delay requirement, other requirements (e.g., loss probability, jitter and cost) can still be considered in the admission control and resource setup protocols.

- Consider that the additive metrics are not independent. For instance, if WFQ (Weighted Fair Queueing) scheduling algorithms are used, metrics like delay, jitter, and loss probability are no longer independent and can be expressed as a function of bandwidth [116].

3.1.4 Granularity vs. Computational and Storage Overhead

Coarser granularity, e.g., destination-based routing, has lower storage and computational overheads but is only suitable for best-effort traffic. However, fine granularity, e.g., flow-based routing, provides lower blocking probability for bandwidth requests, but requires a huge number of states and has high computational cost [110].

The QoS routing table size depends directly on the routing granularity and the number of metrics. Thus, the size of the QoS routing table is generally much larger than the size of a normal[3] routing table. This involves significant storage overhead, which may also slow down the routing table lookup. Some approaches to reduce the routing table size in QoS routing propose either to use coarser routing granularity or to keep routing tables only for the *best-effort* traffic while QoS routes are computed on-demand [5]. The latter approach trades computation time for smaller storage requirements.

[3]One which considers a single metric during the routing process.

3.1.5 Resource Reservation vs. Load Balancing

As QoS routing may find longer paths which are lightly loaded, instead of heavy loaded shorter paths, a tradeoff is generated between reserving the resources (using shorter paths) or load balancing the traffic (using longer but lightly loaded paths). Thus, there are three main possibilities of action:

1. to preserve network resources by choosing the shortest path, also known as the *widest-shortest* path (i.e. the path with minimum hop count and, if there are multiple such paths, the one with the largest available bandwidth);

2. to load balance the traffic by choosing the widest path, also known as the *shortest-widest* path (i.e. the path with the largest available bandwidth and, if there are multiple such paths, the one with the minimum hop count);

3. to make a tradeoff between the two extremes. Examples of such an approach are the *shortest-distance* path [116], which chooses the shortest path when the network load is heavy and the widest path when there is medium network load, and the *dynamic-alternative* path [70], which puts an upper bound on the *widest-shortest* path.

Kowalik shows [96] that the network topology and especially the network connectivity is a key network feature, that dictates which of these approaches offers the better performance. Thus, for networks with low connectivity those approaches that distribute/balance the load perform well, while in networks with a high level of connectivity, where there are multiple alternative paths between a source and a destination, traffic oscillations are more likely to occur. Therefore, the approaches that act to conserve resources should be preferred in such cases.

3.2 Hierarchical routing

In order to make routing scalable, large networks are organised hierarchically. The overall network is divided into domains and the hierarchical levels consist of

aggregated representations of domains on the level below. In this way, the communication overhead is reduced because detailed state information is delivered only inside each domain, and only aggregated state information is transmitted across domain boundaries. The path computation process is reduced to finding domain crossing paths within each domain which are then concatenated to form the final path. The dissemination of aggregated states between domains has the advantages of both offering a level of security to each domain, and of decreasing the storage overhead.

3.2.1 Topology Aggregation

Topology aggregation [105] is the process of summarising the topological information (i.e. QoS metrics, inter-connectivity) of a given subset of given network elements in order to achieve scalability within large networks. It is not only motivated by the need for complexity reduction but also to hide the topology internals of different autonomous systems in the interest of security and to allow them autonomy in managing their domains.

3.2.1.1 Link Aggregation

Link aggregation refers to the process of representing a set of parallel links between two domains by a single logical link connecting the logical nodes which are the aggregate representation of those domains. The logical nodes are responsible for managing the link aggregation.

3.2.1.2 Nodal Aggregation

Nodal aggregation refers to the process of summarizing a peer group into a more compact representation that comprises a *logical node* at the next higher level of the hierarchy.

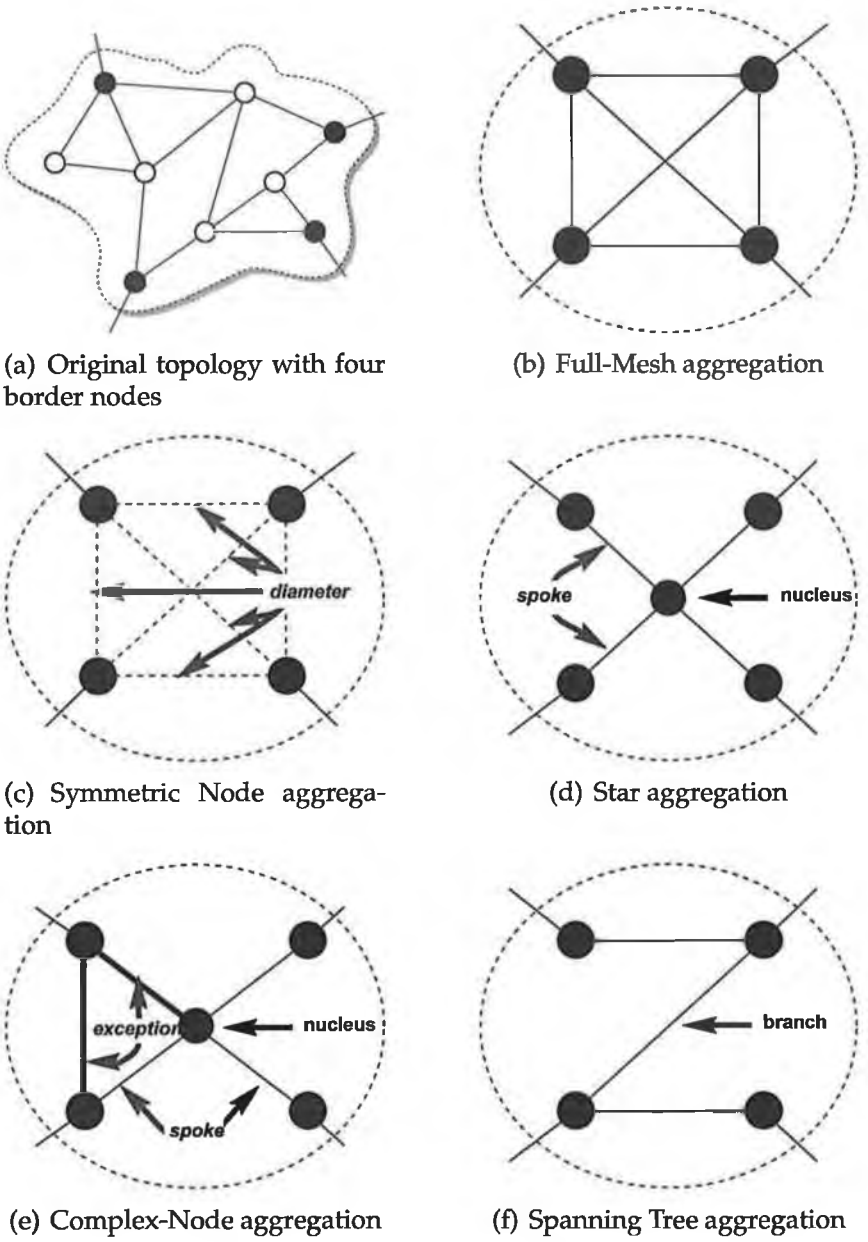
There are multiple nodal topology aggregation techniques [9, 105]. Figure 3.2

presents a range of aggregation techniques applied to the network topology with four border nodes shown in Figure 3.2(a). In all nodal aggregate representations, border nodes within the domain are represented as ports, while the costs[4] for traversing the network will become nodal costs for traversing the logical node. The main difference between the aggregated nodal representations is the way in which they store information regarding the connections between the ports of the logical node. Thus, nodal aggregation techniques represent different levels of trade-off between accuracy and compactness.

At one extreme there is the *Full-Mesh* representation [105] (Figure 3.2(b)) which offers the most accurate representation by providing information regarding all possible connections between all the ports. However, the amount of information to be advertised increases as the square of the number of border nodes. At the other extreme, there is the *Symmetric Node* aggregation method [105] (Figure 3.2(c)) which resembles the Full-Mesh representation, except that all logical links are identical and thus only a single link need be advertised. This approach offers the greatest reduction of advertised information, but does not adequately reflect any asymmetric topology information or capture any multiple connectivity in the original topology [105]. This inaccuracy will increase with the number of aggregation levels. The reduction of the information is gained by having a single nodal cost, regardless of the pair of ports through which the logical node is traversed, referred to as the *diameter*. The value of the diameter represents usually the "worst case" parameter value for traversing the logical node.

A compromise between the two extreme approaches is the *Star* approach [105] (Figure 3.2(d)). The center of the star is the interior reference point of the logical node, and is referred to as the *nucleus*. The logical connectivity between the nucleus and a port of the logical node is referred to as a *spoke*. The concatenation of two spokes represents a traversal of a symmetric network represented by the

[4]Such *costs* may represent any QoS constraint such as bandwidth, delay, delay variation and so on.



(a) Original topology with four border nodes

(b) Full-Mesh aggregation

(c) Symmetric Node aggregation

(d) Star aggregation

(e) Complex-Node aggregation

(f) Spanning Tree aggregation

Figure 3.2: *Examples of topology aggregation methods*

logical node. The Star topology is used as the default node representation within ATM PNNI. The main problem with this representation is that usually networks are not symmetric. They may contain "outliers" (i.e. nodes whose removal would significantly improve the peer group symmetry). This asymmetry can be modelled by a set of *exceptions*. Exceptions can be used to represent particular ports whose connectivity to the nucleus is significantly different from the default. Additionally, an exception can be used to represent connectivity between two ports,

i.e. a bypass, that is significantly better than that implied by traversing the nucleus. This represents the third conventional nodal aggregation technique: the *Complex-Node* (Figure 3.2(e)).

The *Spanning Tree* approach [104] (Figure 3.2(f)) represents another nodal aggregation technique which further encodes the Full-Mesh aggregation as a spanning tree connecting all border nodes. Thus, the amount of information to be advertised is proportional to the number of links.

The goal of aggregation, however, is not only to enhance the scalability but also to present an accurate representation of the physical topology. That is because by greatly reducing the routing protocol overhead, inaccuracy is introduced which typically has a negative impact on QoS routing performance [78]. The authors of [90] have implemented aggregation algorithms for asymmetric QoS-routing information (i.e. link residual bandwidth), using three aggregation approaches: Full-Mesh, Star, and Spanning Tree. The results have shown, as expected, that the *Full-Mesh* representation has the best performance. The only problem is the amount of the information to be advertised.

The problem of advertising too much routing information by using the *Full-Mesh* aggregation can be eliminated by using mobile agents in the routing process. This is because state information does not need to be disseminated through the network as it is consulted *in situ* as described in Section 2.5.

3.2.2 Hierarchical routing protocols

Private Network-to-Network Protocol (PNNI) [9] is the only QoS aware hierarchical routing protocol that has been standardised and implemented. There are also some research projects which propose other hierarchical routing strategies. Amongst them are: the Hierarchical Distribution Protocol (HDP), the Viewserver architecture and the Clearing House.

3.2.2.1 Private Network-to-Network Protocol (PNNI)

Private Network-to-Network Protocol (PNNI) [9] was already introduced in Section 3.2.2.1. Details about the aggregated format in which PNNI keeps its state information are presented in Section 2.2.2.1, while Section 3.2.1 describes different types of aggregation methods including the default aggregation method used by PNNI. Here I present the main advantages and disadvantages of this hierarchical routing protocol.

Its most important advantages are its maturity as the only hierarchical QoS routing protocol to be standardised, and its scalability (by allowing up to 104 hierarchical levels).

One of the shortcoming of this protocol is that the route computation load imposed by the PNNI routing scheme is unevenly distributed among the network nodes. There are situations where some nodes may be overloaded with route computation, while other nodes are rarely involved in this process [9].

Another PNNI drawback comes from the aggregation process, which leads to inaccurate state information advertisements [42, 78, 105]. Such inaccuracy may lead to inefficient utilisation of network resources.

3.2.2.2 Hierarchical Distribution Protocol (HDP)

The Hierarchical Distribution Protocol (HDP) [64] is a proposal for a hierarchical routing protocol within MPLS networks.

The main difference between PNNI and HDP architectures is that the latter features the logical node as a server, called *Bandwidth Broker (BB)*, that is separated from the physical nodes of the AS. Therefore, BBs are not routers or switches but cluster-based server farms that can grow in capacity based on the intensity of path creation requests.

An example of a networking hierarchy in HDP is depicted in Figure 3.3. HDP is executed in response to a request received by a physical node to create an MPLS

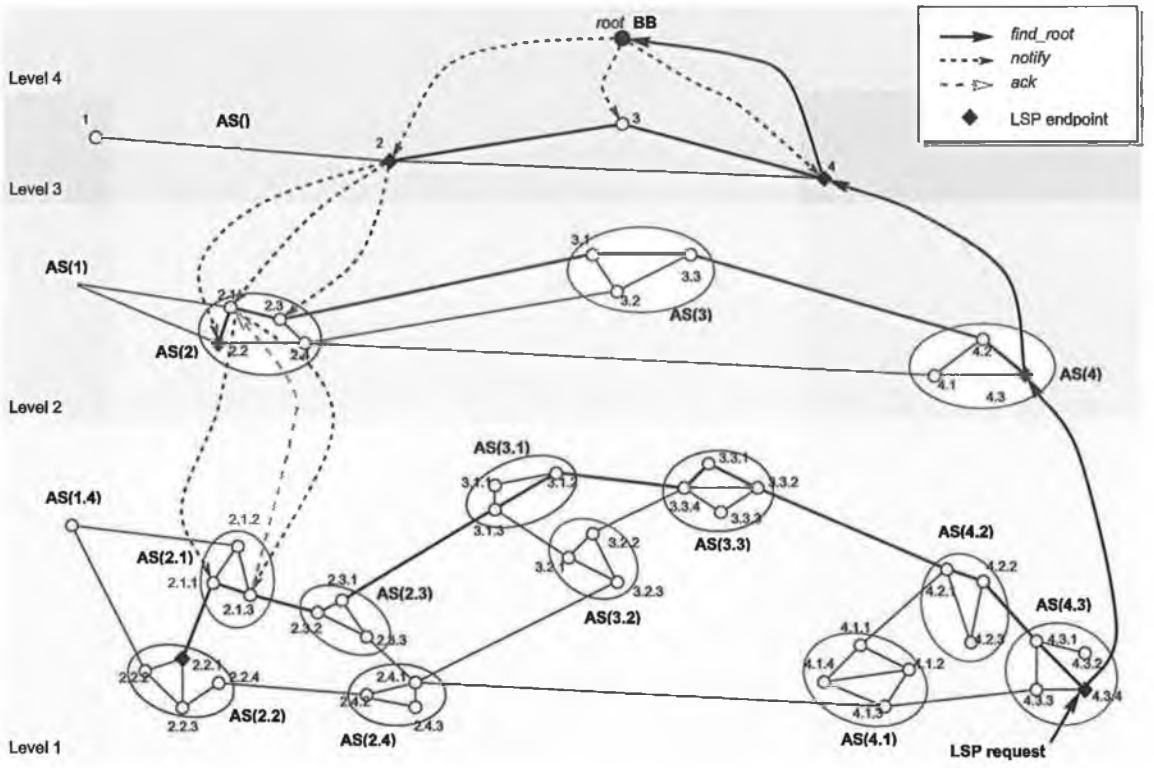


Figure 3.3: *Hierarchical Distributed Protocol (HDP)*

path. The request, which includes QoS requirements (e.g., bandwidth), is sent up the hierarchy as a *find_root* message, until it reaches the lowest level BB that has a view of both source and destination. Therefore, this BB is the root of that path management hierarchy and calculates an abstract route for the requested path at the next lowest level of the hierarchy. Then, it sends *notify* messages to all nodes (BB's from the next lower level) within that path which as well should search traversing paths for the domains they manage. This process repeats in parallel at each level, until it reaches the physical level, where the actual LSRs will try to reserve the resources. If there are enough resources, LSRs will send *ack* messages which will be propagated up to the root BB. Then, the state information maintained at different nodes of the hierarchy must be updated. Physical nodes start sending *update* messages up the hierarchy updating the state of each BB. If there are not enough resources for setting the path a *crankback* message is sent back to the root BB which will restart the algorithm searching for an alternative path.

The main advantage of this distributed system is that at every level, the path is computed in parallel. This, however comes at the expense of an increase in the number of messages [64]. Another issue is that although aggregation strategies are mentioned in the presentation of the algorithm, it is difficult to see how HDP can employ any while computing intermediate paths. Moreover, by starting the path computation at the top of the hierarchy and evolving downwards, even the existing aggregation strategies might not be used efficiently as some routing information might be already obsolete as the protocol reaches the lower levels of the hierarchy.

3.2.2.3 Viewserver

Viewserver [1] is an inter-domain routing protocol which seeks to overcome the problem of inaccurate routing information resulted from the aggregation process. Therefore, the path computation is done by the source node, which gathers centrally all the required routing information by traversing the hierarchy upwards (to find the parent *"view server"*) and downwards (to collect detailed routing information about transit and destination domains). A drawback of this protocol is that the setup time increases as the whole path is computed on a single node, while the state information gathered for an end-to-end path may raise scalability issues for very large networks.

3.2.2.4 Clearing House

The Clearing House [46] architecture uses a hierarchical network in order to make resource reservations over multiple network domains. This system is based on making reservations in advance in order to reduce the overall reservation setup time. These reservations are based on Gaussian predictors that estimate the future bandwidth usage. The performance of this architecture, however, relies on the algorithm used to predict the future traffic.

3.3 Multi-constrained routing

Multi-constrained routing problems can be divided into more distinct classes which are formally described below.

3.3.1 Problem Definition

As presented in Section 2.2.1, there are two main types of QoS metrics: *additive* and *non-additive*. In the case of additive metrics the value of a path is the sum of the link weights along that path. For non-additive metrics, the path value is the minimum (or maximum) link weight along that path. Thus, all the cases with non-additive QoS metrics are solved by pruning from the graph all links that do not satisfy the request. Therefore, most researchers consider only additive QoS metrics in the multi-constrained routing problem.

Definition 1 ***Multi-Constrained Path (MCP) problem*** *[44, 98]*

Let $G(N, E)$ be a directed graph which denotes a given topology, where N is the set of nodes and E is the set of links. Each link $(u, v) \in E$ is associated with m additive weights $w_i(u, v) \geq 0, i = \overline{1, m}$[5]. Given m constraints $c_i, i = \overline{1, m}$, the problem is to find a path P from source node s to the destination node d such that, for $i = \overline{1, m}$:

$$w_i(P) = \sum_{(u,v) \in P} w_i(u, v) \leq c_i \tag{3.1}$$

A path satisfying the condition (3.1) is called *feasible*. However, there may be multiple feasible paths between s and d. Thus, the next problem that arises is to find the optimal one.

Definition 2 ***Multi-Constrained Optimal Path (MCOP) problem*** *[98]*

Let $P_j, j = \overline{1, k}$ be all feasible *paths between the source node s and the destination node d that satisfy the condition (3.1). Given $l(P)$ to be the length of path P, the problem*

[5]The notation $\overline{a, b}$ represents the set $\{a, a + 1, \ldots, b\}$ where a and b are integers.

is to find the shortest path among the feasible ones, such that for $j = \overline{1,k}$*:*

$$l(P) \leq l(P_j) \tag{3.2}$$

A special case of MCOP is the *restricted shortest path* (RSP) which has the goal of finding the least-cost path among those that satisfy only one constraint: the delay.

All three types of multi-constrained routing problems mentioned above are known to be NP-complete [69]. Thus, researchers resort to heuristical solutions or approximation algorithms to solve such routing problems.

3.3.2 Overview of multi-constrained routing proposals

Several techniques have been devised to deal with multi-constrained routing scenarios. A coarse classification of these solutions is presented in [129].

- *the Lagrangian-based linear composition:* A linear combination of weighted link metrics is built to represent all individual metrics, so that:

$$W^*(P) = \sum_{i=1}^{m} d_i \cdot w_i(u,v) \tag{3.3}$$

  The individual weights d_i can be statically configured or determined dynamically. They also can be changed to get better paths. However, if the path is optimal in the single metric, does not follow that it is optimal in terms of the other metrics, and it may even not satisfy some of the constraints.

- *the Fallback Routing approach:* QoS metrics are considered one by one in a predetermined fallback sequence hoping that the optimal path with respect to a single parameter will also satisfy the other constraints.

- *Dependent QoS metrics:* Multiple metrics dependent on each other can be

reduced to one metric and the resulting single-metric problem can be solved in polynomial time[6].

Some multi-constrained proposals treat the delay-bandwidth [161] or the hop count-bandwidth [171] constrained problem. However, such solutions are not considered MCP problems by Definition 1. That is because bandwidth is not an additive constraint which makes the problem much easier and so it can be solved in polynomial time.

The authors of [117] show that when a class of *weighted fair queuing* (WFQ) scheduling algorithms is used, the problem of finding a path satisfying bandwidth, delay, jitter, and/or buffer space constraints is solvable by a modified version of the Bellman-Ford algorithm in polynomial time. That is because when WFQ-like scheduling algorithms are used, the delay, jitter and buffer space are no longer independent of each other and all of them become functions of bandwidth. This also is not an MCP problem because in MCP problems the weight functions are considered to be independent.

3.3.2.1 Jaffe's approximation

A heuristic algorithm with a polynomial time complexity, called Jaffe's approximation, is proposed in [84] to approximate the MCP problem. The algorithm finds a path that minimises $w_1(P) + d \cdot w_2(P)$, where d is a given constant. However, there are cases when minimising that function does not lead to a MCP solution. An example is depicted in Figure 3.4, for which the two constraints are $w_1(P) \leq 0.5$ and $w_2(P) \leq 0.5$. For the case $b = 1$, the path $a \rightarrow b \rightarrow d$ minimises $w_1(P) + w_2(P)$ but is not a solution, while the path $a \rightarrow c \rightarrow d$ is a solution but does not minimise $w_1(P) + w_2(P)$.

[6]See more about *polynomial time* in Appendix D.

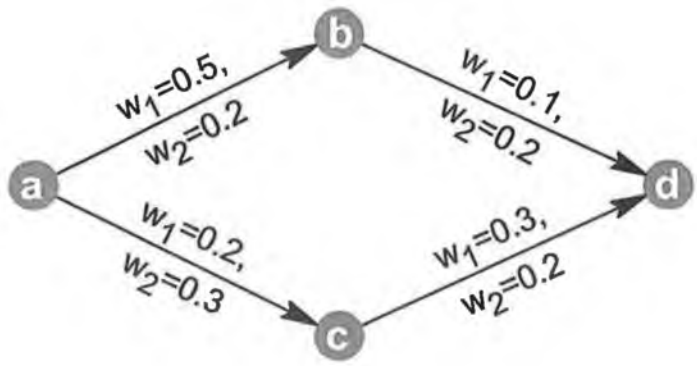


Figure 3.4: *Counter-example for Jaffe's approximation algorithm*

3.3.2.2 Chen's approximation

Chen and Nahrstedt [44] also proposed an approximation algorithm for the MCP problem. This algorithm tries to simplify the problem by scaling down $m-1$ (real valued) link weights to integer values so that, for $i = \overline{1,m}$:

$$w_i'(u,v) = \frac{w_i(u,v) \cdot x_i}{c_i} \tag{3.4}$$

where x_i is a given positive integer. Thus, the simplified problem is reduced to finding a path P that minimises the first (real valued) weight so that the other $m-1$ scaled down (integer) weights are within the stricter constraints x_i. To solve this simplified problem two algorithms were proposed: the Extended Dijkstra's Shortest Path algorithm (EDSP) and the Extended Bellman-Ford algorithm (EBF). These algorithms maintain a variable $d[v, k_2, \ldots, k_m]$, for each vertex v and every integer $k_i \in [0 \ldots x_i]$ where $i = \overline{1,m}$, which is an estimation of the smallest w_1 weight of those paths from source s to destination t whose w_i' weights are k_i. The value of $d[v, k_2, \ldots, k_m]$, initially set to $+\infty$, always follows Equation (3.5),

$$d[v, k_2, \ldots, k_m] \geq \min_{P \in \Gamma(v,k_2,\ldots,k_m)} \{w_1(P)\} \tag{3.5}$$

where $\Gamma(v, k_2, \ldots, k_m) = \{P | P$ is a path from s to t and $w_i'(P) = k_i$ for all $i = \overline{1,m}\}$. As the algorithm executes, EDSP (EBF) successively improves the estimation and $d[v, k_2, \ldots, k_m]$ converges to the minimum value presented in Equation (3.5). Thus

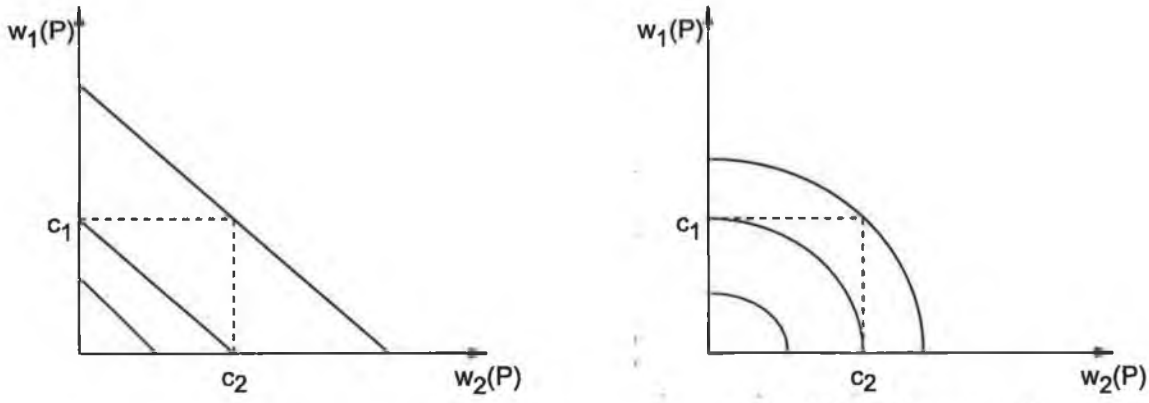
when EDSP (EBF) completes, the inequality in Equation (3.5) becomes equality.

3.3.2.3 TAMCRA and SAMCRA

The Tunable Accuracy Multiple Constraints Routing Algorithm (TAMCRA) [55] is based on three fundamental concepts:

1. a non-linear measure for the path length
2. the k-shortest path approach
3. the principle of non-dominated paths.

The first principle refers to the fact that instead of using a linear composite function for the path search as in Jaffe's approximation algorithm (Figure 3.5(a)), a non-linear function will be used (Figure 3.5(b)).



(a) Jaffe's approximation approach (b) TAMCRA's approach

Figure 3.5: *Searching a feasible path by minimising a linear composite function or a nonlinear composite function [98]*

Such an approach can be achieved by finding a path that minimises the distance function:

$$l(P) = \max_{1 \le i \le m} \frac{w_i(P)}{c_i} \tag{3.6}$$

A consequence of applying the first principle is that the sub-paths of the shortest paths are not necessarily shortest paths. This means that more paths have to be considered besides the shortest one, which leads to the second principle. In

TAMCRA the k-shortest path concept is applied to intermediate nodes t on the paths from s to d. Multiple paths for s to t are selected to be recorded based on the *non-dominated paths principle*. A path Q is *dominated* by a path P if $w_i(P) \leq w_i(Q)$ for all $i = \overline{1,m}$ where, for at least one value of i, $w_i(P)$ is strictly less than $w_i(Q)$. TAMCRA considers only non-dominated paths.

Any path P that satisfies $l(P) \leq 1$ is a feasible path for TAMCRA, and therefore an acceptable solution for the MCP problem. However, this path may not be the optimal one. Thus, a slightly modification of TAMCRA, called Self-Adaptive Multiple Constraints Routing Algorithm (SAMCRA) has been proposed in [158] to address this issue (and so solve the MCOP problem). This is done by allowing any number k of paths to be recorded. Unfortunately, in the worst case, this could lead to an exponentially growing k.

3.3.2.4 A Heuristic for MCOP (H_MCOP)

Korkmaz and Krunz [94] provided a heuristic algorithm, called H_MCOP, that tries to find a feasible path for any number of constraints. The search for a feasible path is done by approximating the nonlinear function (3.6), which is also used in TAMCRA.

To achieve its objective, H_MCOP executes two modified versions of Dijkstra's algorithm in the backward and forward directions. In the backward direction, it computes the shortest paths from every node to the destination node d with respect to a linear combination of weights. These paths are then used in the forward direction to predict if a path P could be feasible if traversing the intermediate nodes $t \in P$ for which exists the computed path from the source node s to t and the estimated remaining path from t to the destination node d.

3.3.2.5 Randomised algorithm

The same authors, Korkmaz and Krunz, propose in [93] a randomised algorithm for the MCP problem. It consists of two parts: the initialisation phase and the ran-

domised search. In the initialisation phase the algorithm computes the shortest paths from every intermediate node t to d considering each link weight and a linear combination of them. Then, the algorithm starts from s a randomised breadth first search (BFS), which only discovers nodes from which there is a good chance of reaching d. This is done by using the information collected in the initialisation phase.

3.3.2.6 Limited Path Heuristic

Yuan proposes in [173] two heuristics for the MCP problem: the *limited granularity* heuristic and the *limited path* heuristic. The limited granularity reduces the MCP problem to a problem solvable in polynomial time by transforming all QoS metrics except one to take bounded integer values. The limited path heuristics (LPH) is based on the Bellman-Ford algorithm and uses the last two fundamental concepts of TAMCRA. The difference between TAMCRA and LPH is that the latter stores the first k (not necessarily the shortest) paths and it does not check if a sub-path satisfies the constraints until it reaches the destination d.

3.3.2.7 A*Prune

Liu and Ramakrishnan consider in [112] the problem of finding K shortest paths that are within the constraints. They calculate all the paths from s to d in a similar manner to H_MCOP and return those that are within the constraints. If more that K paths are found, one is selected using Jaffe's linear length function.

3.3.2.8 Evaluation of MCP algorithms

The authors of [99] compared the MCP algorithms by simulation using Waxman graphs and square lattices. They obtained near-optimal success rates and small execution times for all MCP algorithms in the case of loose constraints. Thus, they focused their attention on the case of strict constraints. The simulations re-

vealed that the Bellman-Ford-based algorithms (i.e. Chen's approximation and the Limited Path Heuristic) require significantly more execution time than their Dijkstra-based counterparts. The *success rate*[7] simulations showed that the exact algorithms SAMCRA and A*Prune always give a success rate of one, while Jaffe's algorithm performs significantly worse than the others. The *normalised execution times*[8] the algorithms used to obtain the corresponding success rates show that all algorithms display a polynomial execution time for the class of Waxman graphs. For the class of lattices, the execution times of the exact algorithms (i.e. SAMCRA and A*Prune) grow exponentially, which is the price paid for exactness in hard topologies.

The complexity of multi-constrained path algorithms was also evaluated in [99]. Their results are presented in Table 3.1. The authors of [99] warn that these complexities should be interpreted with care because, for instance, the real execution time of H_MCOP will always be longer than that of Jaffe's algorithm under the same conditions, since H_MCOP executes Dijkstra's algorithm twice.

Table 3.1: *Worst case time and space complexities of multi-constrained (optimal) path algorithms evaluated in [99]*

Algorithm	time complexity	space complexity
Jaffe's approximation	$O(N \log N + mE)$	$O(N)$
SAMCRA, TAMCRA	$O(kN \log(kN) + k^2 mE)$	$O(kmN)$
EBF	$O(x_2 \dots x_m NE)$	$O(x_2 \dots x_m N)$
Randomised algorithm	$O(mN \log N + mE)$	$O(mN)$
H_MCOP	$O(N \log N + mE)$	$O(mN)$
A*Prune	$O(N!(m + N + N \log N))$	$O(mN!)$

The concluding remark in [99] is that SAMCRA-like algorithms that use a k-shortest path algorithm with a nonlinear length function while eliminating paths via the non-dominance[9] and lookahead[10] concepts, provide the best performance

[7]The number of times that an algorithm returned a feasible path divided by the total number of iterations.

[8]The execution time of the algorithm divided by the execution time of Dijkstra's algorithm.

[9]The non-dominance principle is a very strong search-space reducing technique, especially when the number of constraints m is small. When m grows, the lookahead concept together with the constraint values provide a better search-space reduction [99].

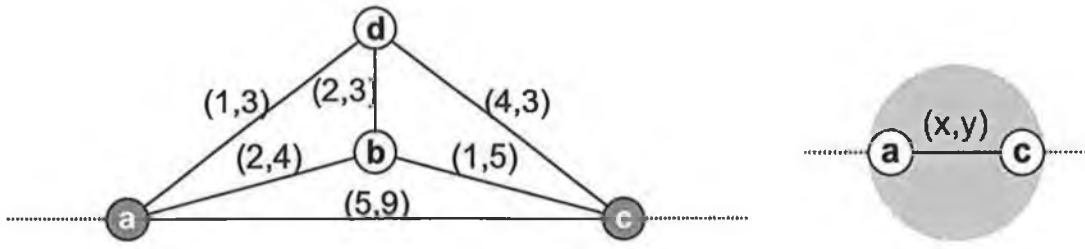
[10]This concept is based on information from destination-rooted path trees which are computed

amongst all algorithms evaluated.

3.3.3 Hierarchical multi-constrained routing

Within a hierarchical routing context, the multi-constrained problem becomes even more complex [92, 104, 114]. That is because it is very difficult to designate appropriate values for the multiple QoS parameters that are associated with a logical[11] link in order to build the aggregate nodal representations. That is because one path may be the best for one constraint but may not satisfy other constraints.

Figure 3.6 presents a network topology with each link having associated two additive weights. The issue here is what values should be associated with the logical link from the aggregated representation which connects nodes a and c.



(a) The physical topology (b) the *logical link*

Figure 3.6: *Example of topology aggregation considering two additive QoS metrics*

As depicted in Figure 3.7, there are five possible paths between the border nodes a and c. Their representation in the bi-dimensional space determined by the two weights is also depicted in Figure 3.7.

The conventional methods for solving the *"best" path* problem are described in [92, 104, 105].

- *Single path metric:* The path selection is made based on only one QoS metric w_i. The representative path P is then the one that satisfies: $w_i^* \leq w_i^{(j)}$ for all $j = \overline{1,m}$. The issue remains to select the actual metric. For the example in

in polynomial time. These paths are used to reduce the space-search and to facilitate the search for a feasible path [99].

[11] A *logical link* is an aggregation of all physical paths between two border nodes.

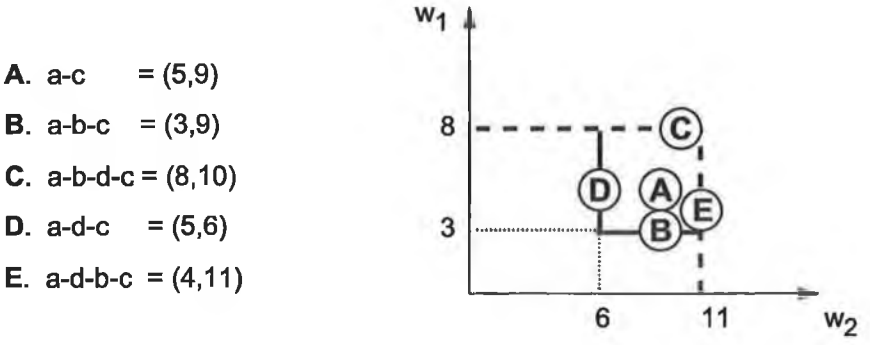


Figure 3.7: *The representation of the paths into the QoS-metric bi-dimensional space*

Figure 3.6 the chosen path will be (B) if w_1 is the representative QoS metric, or (D) if w_2 is the representative QoS metric.

- *The best case approach:* The aggregated link will have as QoS metrics $(w_i)_{i=\overline{1,m}}$ the *best* metric values amongst all paths, i.e. for all $i = \overline{1,m}$:

$$w_i = \min_{k=\overline{1,N}} w_i^{(k)} \tag{3.7}$$

where N is the total number of paths. For the example in Figures 3.6-3.7 the *best case approach* is $(3, 6)$.

- *The worst case approach:*The aggregated link will have as QoS metrics $(w_i)_{i=\overline{1,m}}$ the *worst* metric values amongst all paths, i.e. for all $i = \overline{1,m}$:

$$w_i = \max_{k=\overline{1,N}} w_i^{(k)} \tag{3.8}$$

where N is the total number of paths. For the example in Figures 3.6-3.7 the *worst case approach* is $(8, 11)$.

Other solutions which work only for bandwidth and delay constraints are also proposed in [92, 114].

However, the most important requirement in multi-constrained hierarchical routing is not to find a "best" representative for the logical link, but to find the appropriate sub-path for the *best* final path. A method to find such sub-paths

by considering more than one representative for a logical link is presented in Chapter 6.

3.4 New approaches to the QoS routing problem

QoS routing using mobile agents is performed by exploring all possible directions starting from a given point, the agents terminating as soon as they have found a path or end up in a cycle. Thus, the routing protocol works in a highly parallel and distributed manner, thereby reducing the computational burden on individual nodes. Moreover, no routing information has to be disseminated as the mobile agents are consulting it *in situ*. This means that storage overhead for non-local information does not exist. Moreover, the quantity of state information is no longer an impediment, so that more QoS constraints or more accurate aggregate representation for hierarchical routing can be used.

3.4.1 Mobile Agents in Network Routing

A number of research groups have introduced new mechanisms for routing based on the mobile agent paradigm [48, 59, 71, 88, 121].

Most of the mobile agent routing proposals were based on the *swarm intelligence* concept which was inspired by the behaviour of biological swarms (e.g., ants, bees). This was considered a good model for totally decentralised, yet robust and adaptive routing algorithms. The key element of this new concept is the *emergent behaviour* or *emergent intelligence* where a complex and often intelligent behaviour is gained through the interactions of thousands of autonomous swarm members, with simple primitives. The main principle behind these interactions is called *stigmergy*, or communication through environment. This term was introduced by Grasse in [75] to describe indirect communication among individuals through modifications induced in their environment. For instance, ants were able to find shortest paths using only the pheromone trail deposited by other ants.

In the routing approaches which use swarm intelligence, the swarm members are mobile agents. The advantages gained by using such models are many [26, 59, 88, 111], as described in Section 2.5.2.

AntNet is one of the pioneering approaches for routing with mobile agents. This proposal, presented in [59], is inspired by the observation of ant colony behaviour where the ants cooperate to achieve the common goal of finding food [75]. The agents are modelled to emulate the ant's pheromone technique for finding the shortest path. Here, individual agents leave traces of their presence in network nodes so that other agents can determine the probabilities of choosing specific paths to follow, until they all converge to a similar result (the shortest path). This work served as an inspiration for other research efforts [88, 97, 121]. The authors of [150] extended the ant-based routing by adopting the solution of multiple colonies of ants to search for optimal paths in order to mitigate the stagnation problem within ant colony optimisation algorithms. Stagnation occurs when the network state is unchanging and consequently the routing algorithm gets trapped in the local optima and is therefore unable to find new improved paths.

Mobile agent *cooperative schemes* were also introduced for routing purposes [97, 121, 169]. The authors of [121] proved that, within dynamic networks, a population of mobile, cooperating software agents are able to build and maintain routing tables which remain reasonably accurate even as the topology of the network changes over time.

Mathematical models for determining the behaviour of mobile agents in network routing were also built. Some analysed the growing and the jumping behaviour of an agent based routing algorithm [151], while others were concerned with the population distribution and probability of success of mobile agents [131]. Their results provide guidelines for future designs of agent-based routing systems.

All of the routing solutions presented above share the generic advantages of

mobile agents over the traditional technologies (e.g. link-state approaches), as discussed in Section 2.5.2. Moreover, they represent the best distributed (loop-free) solution for decentralisation. They share disadvantages in the areas of security and provability. The security problem constitutes an open area of research. Fundamental difficulties regarding security are inherent to all relatively open systems. Network behaviour which arises from the interactions of cooperating agents is difficult to model mathematically, and as a result is difficult to reason about deterministically. This means that only a significantly large deployment (in space as well as in time) of mobile agents will prove their reliability and predictability. The tests performed until now suggest [59] that mobile agent systems outperform both static and adaptive routing algorithms.

Mobile agents built using a general-purpose language such as Java or C++ tend to be difficult to program, because these languages are a better fit to the client-server architecture for distributed computed than to the mobile agent paradigm. Thus systems that use a dedicated language are, in general, more powerful, and their more elegant architecture outweighs the disadvantages of requiring a new language to be learned. Another issue of concern is the size of mobile agent generated by this dedicated code. A system that generates particulary compact code will now be described.

3.4.2 WAVE in Network Routing

The WAVE system has been used for building IP routing tables [143] as well as for QoS routing schemes within MPLS networks [72].

3.4.2.1 WAVE in IP networks

Within IP networks, the forwarding procedure is based on consulting the *routing tables* which tell the router which is the next hop node for a certain destination. The building of such tables is the responsibility of routing protocols. Sapaty [143]

proposes building optimum routing tables (OTR) using Wave technology.

This task is accomplished by using the shortest path-three (SPT) algorithm as in Figure 3.8. Starting from node *a*, the wave activity may spread in parallel to all neighbours via links (see Figure 3.8(b)), accumulating the growing distance from the starting node in a frontal variable F, which is incremented by the length L of the link passed.

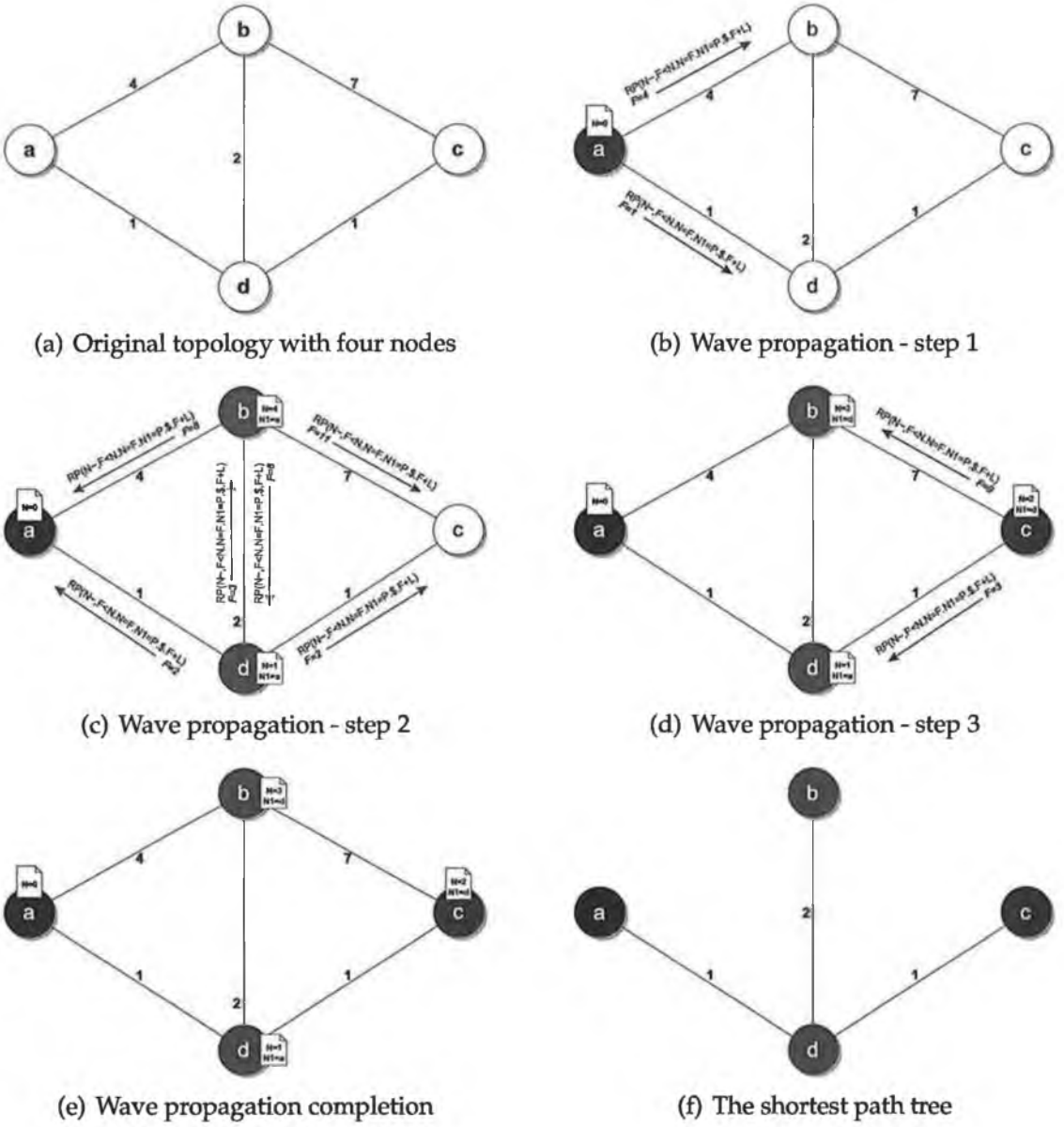


Figure 3.8: *Using* waves *to find the shortest path tree*

When *waves* arrive in nodes *b* and *d*, the value in F is compared with a nodal variable residing on these nodes, i.e. N. If this variable does not exist, it means

that the nodes were not yet visited, and therefore the variable N is created and initialised with the value of F. Then, the *waves* will spread further on to all neighbours of the current node (i.e. b and d), as in Figure 3.8(c). If the nodal variable N already exists in the targeted nodes (e.g., nodes a, b and d in Figure 3.8(c), and nodes b and d in Figure 3.8(d)), which means that those nodes were already visited, variables F and N are compared. If the value of F is smaller, the value of N is substituted with the value of F and the waves are propagated onwards; otherwise, the wave propagation stops.

Any modification of variable N (including its creation) is accompanied by the recording of the predecessor node which brought the latest distance F to the current node. This record is kept in nodal variable $N1$.

This parallel navigation process, which may continue only if *waves* can find shorter paths that the ones already found (i.e. $F < N$), converges rapidly (as in Figure 3.8(e)) to a shortest path-tree. Moreover, the shortest path-tree found (i.e. Figure 3.8(f)) is a loop-less one as any loop via links returning to a node can bring only a bigger distance to this node from the starting node.

Optimum routing tables (ORT) are calculated by initiating shortest path-tree (SPT) processes from all nodes of the network in parallel. The nodes from which the STP processes are initiated are the final destinations, while all the other nodes will receive the *best next hop* towards those destinations.

Solutions for more complex problems such as dynamic ORTs within networks with dynamic topologies are also treated in [143].

WAVE performance tests. The performance of Wave navigational techniques for their use in routing purposes was addressed in [74]. Here, all the basic navigation mechanisms (also called *migration strategies*) described in Section 2.5.4.2, are compared in terms of completion time. Tests were run on four different networks having 5, 10, 20 and 30 nodes, each with a random topology. The results show that the *depth-first spread* performed really poorly for networks larger

than 20 nodes. However, almost all basic spreading techniques performed almost 10000 times better than *depth-first spread* on networks with 30 nodes, except the *spiral spread* (which was only 1000 times better then the *depth-first spread*). The best performance was obtained by the *breadth-first parallel spread* technique. It was observed that the recursive implementations of some navigational strategies perform slightly better than their sequential (repetitive) implementation. The optimum routing tables (ORT) process described earlier in this section was also tested in [74].

The above algorithm's various implementation options were evaluated on a prototype software implementation of the Wave Interpreter installed on a 400-Mhz PC running RedHat Linux. The results for the *breadth-first parallel spread* executed in less than 1 ms for all networks considered, while the performance of the ORT implementation executed in 1.2 ms for networks of fewer than 30 nodes. The overall computation delay for both BFPS and ORT grew almost linearly within the considered networks.

3.4.2.2 Wave in MPLS networks

The use of Wave technology for routing purposes within MPLS networks was advocated in [72, 73]. Here *waves* were used to find multipoint-to-point path trees that satisfy QoS requirements in order to improve the current routing by addressing the Steiner Minimal Tree (SMT) problem [73]. Finding minimal cost trees leads to a minimisation of network resource usage and a better management of data streams while grouping them on a similarity basis. This is done by searching a QoS-compliant route for each individual ingress node whose final path to the root may coincide, at least partially, with that of one or more ingress nodes. The goal is to find the maximum number of edges where individual path intersections occur to minimise the overall cost of the tree. The authors of [72] show that this goal can be achieved by using *waves*. This is done however at the cost of generating bursty traffic that may cause congestion in the queues of the

agent processing units in the network when the colony of agents is launched to discover the mp2p tree.

The mobile agent technology presented above manages to generate very small mobile agent (packets) by using an entirely new programming paradigm that provides the fullest support for agent mobility [160]. Thus, this system is used for building the new hierarchical QoS routing algorithms, applicable within MPLS networks, and described in Chapters 5 and 6.

3.4.3 Active Networks in Network Routing

There is a debate as to whether active networks should be deployed in the field or not. For instance, the authors of [107] state: "... it is not *a priori* obvious that a programable network is a good idea. It clearly offers increased flexibility but at some cost." That means that even though active networking has the potential to solve many outstanding problems of current "passive" networks, the impact that such services will have no network performance should also be accounted for. Psounis identified two active network tradeoffs in [130]: a tradeoff between security levels and performance, and a tradeoff between usability/flexibility and complexity.

There are however some QoS-related problems which can be easily solved by an active network implementation. An example of such an application is described in [57, 58], where active networks are used to solve the problem of reservation gaps in networks with incomplete QoS support and their impact on QoS provisioning. To support a required QoS level across paths traversing non-QoS segments, the Q-nodes run Active Network services to monitor and provide information about the availability of resources across the reservation gap. The reservation gap is monitored and managed by Netlets [48, 48], a form of active code residing at Active Nodes.

Other researchers propose using *active networks* in the routing process [66, 119, 164]. For instance, the authors of [164] argue that active networking is a good candidate for QoS provisioning because it offers improved flexibility by allowing a user to individually choose QoS parameters and their method of calculation. The authors of [119] propose to use active networks in order to establish and control LSPs, to perform operations on label stacks, and to use flexible restoration techniques[12]. However, both proposals tend to involve customers/users in the routing process (i.e. to set routes, control LSPs, choose QoS parameters). The disadvantage of such systems is that not every customer/user knows the appropriate values to set for certain QoS parameters or how to identify the best route to select and most would not only want more involvement in sending their packets than selecting the destination. Moreover, even if a customer/user wants and knows how to use such systems he has to have a global view of the network to make a good decision.

There are also routing solutions inspired by the ants model [66] which are very similar to mobile agents implementations. A comparison between the two technologies (i.e. active networks and mobile agents) from the routing perspective is presented in Section4.1.

3.5 Summary

This chapter focused on QoS routing, presenting its main challenges and the existing solutions. The main source of these challenges are the communication, computational, and storage overhead introduced by QoS routing. A set of distinct solutions has been identified here for each of the three overheads. The most commonly used solution for reducing the communication overhead is topology aggregation. It also helps to reduce the storage and the computational overhead due to the smaller amount of aggregated routing information. The computational

[12] All of these are only proposals. No implementations of such services are available yet.

overhead can also be significantly reduced by distributed solutions to routing using modern techniques like active networks and mobile agents. Such distributed routing algorithms require state information to be available only locally, which significantly reduces the storage and communication overhead.

However, the most scalable topology aggregation techniques introduce inaccuracy which may have a negative impact on the routing performance, while the most accurate aggregation techniques require too much state information to be distributed. Moreover, even within hierarchical routing protocols which use topology aggregation, the actual route computation may generate communication (see HDP) or computational overhead (see PNNI and Viewserver). Such issues will be addressed in the remainder of this thesis.

The main techniques for topology aggregation along with the QoS capable hierarchical routing protocols were introduced.

Another important issue is the tractability of the multi-constrained problem. Heuristical and approximation solutions for flat as well as hierarchical QoS routing algorithms available in the literature were described.

This chapter concluded by describing of research proposals which use mobile agents and active networks for routing purposes. The next chapter compares these two new technologies in order to determine which is more suitable for implementing efficient QoS routing protocols.

CHAPTER 4

Active Networks or Mobile Agents?

> *"... the mobile agents paradigm and the active networks paradigm, although arising out of two different communities, have much in common."* Martin Collier [47]

The main similarity between the two paradigms is that they both use the mobile code concept. However, active networks and mobile agents are distinguished by their different architectures.

4.1 Comparison between the two worlds

The main difference between the two technologies, as the author sees it, consists in how the two technologies use the code and what impact the execution of such code has on current traffic. From this perspective, active networks have a direct impact on the existing/current traffic as the code either travels along with the (traditional) packets or waits for them on intermediate nodes in order to perform computations ranging from redirecting the packets to modifying their content. Mobile agents are more "passive" from this point of view as they travel indepen-

dently of data traffic with the primary aim to "discover" the network and collect information scattered into the network rather than to change data traffic. However, mobile agents if used for routing purposes, can indirectly modify the way that data traffic flows.

Viewing these features from the QoS routing perspective, mobile agents are a more powerful tool because they are able to search feasible paths before such paths are traversed by the actual traffic, as happens in most QoS routing schemes. Moreover, mobile agents are able to perform path search in a distributed and parallel manner.

In an active network context, on the other hand, the actual traffic can also be redirected by the *active code* travelling along it and/or by the *active nodes* traversed by it. The main difference compared with the mobile agent-based routing approach is that the routing decisions have to be made on the spot on a hop-by-hop basis. This has two main disadvantages:

1. routing decisions are made only based on the view of the network accessible from the current node;

2. the routing decision and the redirecting process have to be done at wire speed, which can be a serious impediment for most active network architectures as they propose the use of Java in the interest of portability and security.

There are however, as mentioned in Section 3.4.3, problems other than QoS routing, which can be easily solved only by using active networks. Section 4.2 present examples of such applications.

4.2 Adding functionality in future Access Networks

The separation between control and forwarding planes in MPLS brought great benefits in the trunk/core network. First, because it introduced a very simple and

efficient forwarding mechanism that can be implemented even at the hardware level[1]. And second because this separation allows the existence of one or more control planes. Thus, introducing tools like QoS and traffic engineering can be done transparently to the forwarding plane. Moreover, changing the forwarding mechanism will not require modifications in the control plane.

MPLS deployment is starting to reach beyond the core network, penetrating the access areas. The main benefit of wider MPLS deployment, beyond those which apply to MPLS in the core network, is the end-to-end continuity[2]. Therefore, many now advocate MPLS deployment in access networks. As a result, hardware vendors are already introducing such functionalities in access equipment.

There are also disadvantages to such MPLS expansion. One such disadvantage is that the MPLS forwarding mechanism is too "opaque" for access areas. There are situations where packets need to be processed[3] within the network. This is the case with firewalls, Web proxies, multicast routers, mobile routers or other similar services that need to access information from a packet header to decide whether the current packet should be dropped or how it should be forwarded.

Active Networks is a novel solution for implementing such services. The integration of MPLS with *active networks* as a solution for future access networks is thus advocated by this author. A novel architecture to achieve this is presented below. The resulting networks will provide all the features of MPLS, and in addition, will use active packets to support packet processing.

Active networks, as defined in Section 2.4, are packet switched networks in which packets may contain code fragments that are executed on intermediate

[1]This mechanism works well also in optical networks (see the Generalised Multiprotocol Label Switching (GMPLS) [20]).

[2]Reasons for control plane continuity are: a more efficient network management [86]; verifiable end-to-end Service Level Agreements (SLAs) [87]; and better provisioning of end-to-end QoS and traffic engineering [87].

[3]It is considered that packet processing takes place above layer 2 (link layer) and in particular to layer 3 (routing layer) packet processing.

nodes or else references to already implemented functions residing on intermediate nodes. Thus, the network is transformed from a *passive* carrier of bits into a programmable packet processing environment as well as a packet transmission one.

4.2.1 Integrating Active Networks and MPLS

In IP networks, routers can be augmented to have *active* capabilities. Moreover, *active routers* can coexist and inter-operate with legacy IP routers, which transparently forward packets in a traditional manner (see Figure 4.1).

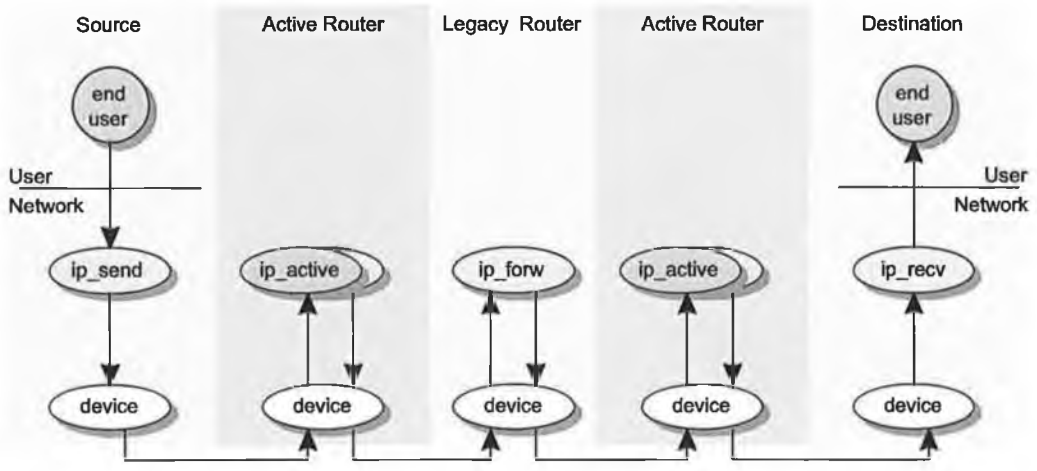


Figure 4.1: *Packet processing within the nodes of a legacy Active Network*

In the same way, as can be seen in Figure 4.2, I propose an MPLS network in which some LSRs are augmented with *active* capabilities. Legacy LSRs and active ones can coexist and inter-operate.

A more detailed architecture of an active LSR is shown in Figure 4.3. In a legacy LSR, a packet would follow the processing path indicated by continuous lines/arrows in Figure 4.3, while in the *active* LSR, the MPLS layer is modified to treat *active* packets separately. A packet is determined to be an *active* packet or not based on the label. If the packet is *active*, it is sent to the IP layer and from there to the relevant active code in order to be processed. After being processed, the packet is sent back to the MPLS forwarding plane. Here, the packet is assigned

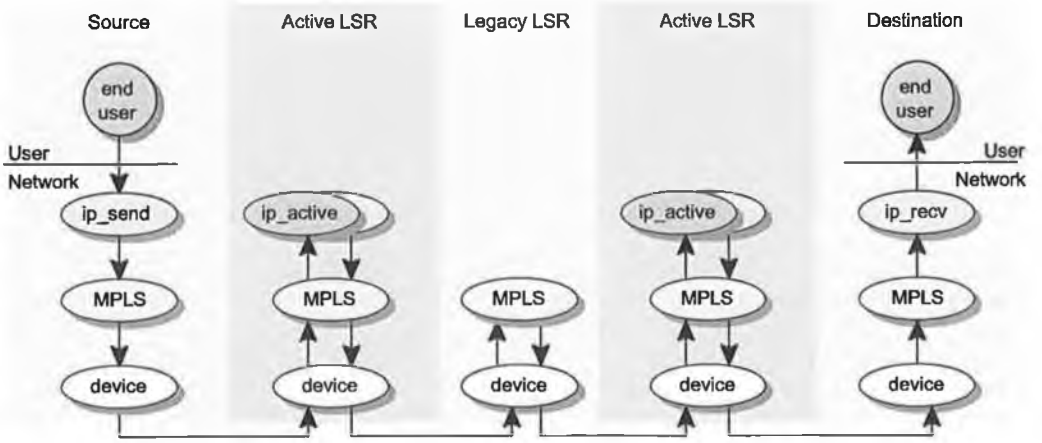


Figure 4.2: *Packet processing within the nodes of an MPLS Active Network*

to a Forwarding Equivalence Class and labelled accordingly to its class.

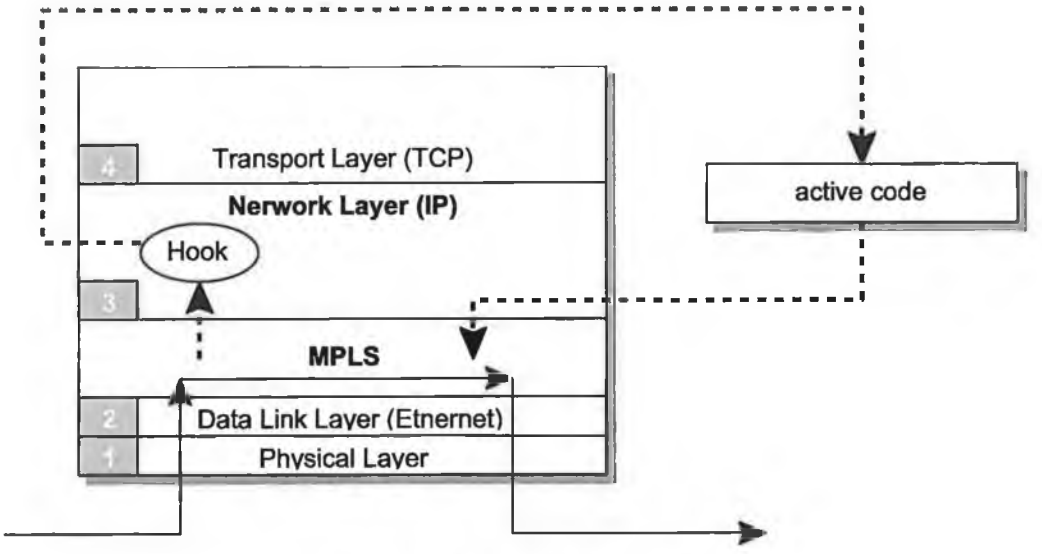


Figure 4.3: *An active MPLS architecture*

Using this architecture only the active packets (as identified by the label) that traverse an active LSR will be processed by the active code[4]. Conventional packets will be processed in accordance with the standard MPLS protocol.

4.2.1.1 The implementation

The framework has been implemented on the *Linux Operating System*. A simple network was emulated using *User-Mode Linux* [61]. The details of and the moti-

[4]The active code can be carried by the packet or reside on the node in which case the packet contains only a reference to the relevant active code.

vation for choosing Linux and User-mode Linux are presented in Appendix B. To configure the "active" LSR the *Netfilter* [136–138] framework, which is described in Appendix C, was used. The MPLS forwarding implementation chosen was the Sourceforge open-source project[5], called *mpls-linux*. The prototype featured static label assignment.

To prove the concept of integrating MPLS and active networks, a minimal MPLS network was configured containing four nodes: **LSR A, LSR B, LSR C,** and **LSR D,** connected with three links as shown in Figure 4.4.

Figure 4.4: *A minimal MPLS network*

The LSP **A - B - C** was set up, and contains an *active node*, i.e. **LSR B**. The active packets are labelled with a distinct label identifying them as such to the active node **LSR B**. Such packets contain code or references to code (residing on the node) that has to be interpreted and executed on their payload. Thus, the packets reaching **LSR B** and identified as being active are consequently sent up to the IP layer. Then, using the Netfilter framework as depicted in Figure 4.5, the content of the packets will be modified.

As a proof of concept, a trivial example was implemented which modified the packet's source address. Using the netfilter framework, the sample code registers to listen to the first hook defined by netfilter for IPv4 (NF_IP_PRE_ROUTING) which is located at the entry of the packet in the protocol stack, just after the sanity checks (i.e. not truncate, IP checksum OK, not promiscuous receive). It captures packets passing that hook and queues them for userspace. A userspace application then modifies the packet's source address.

The system was tested as shown in Figure 4.6, by sending ICMP (Internet

[5]See `http://sourceforge.net/projects/mpls-linux/`

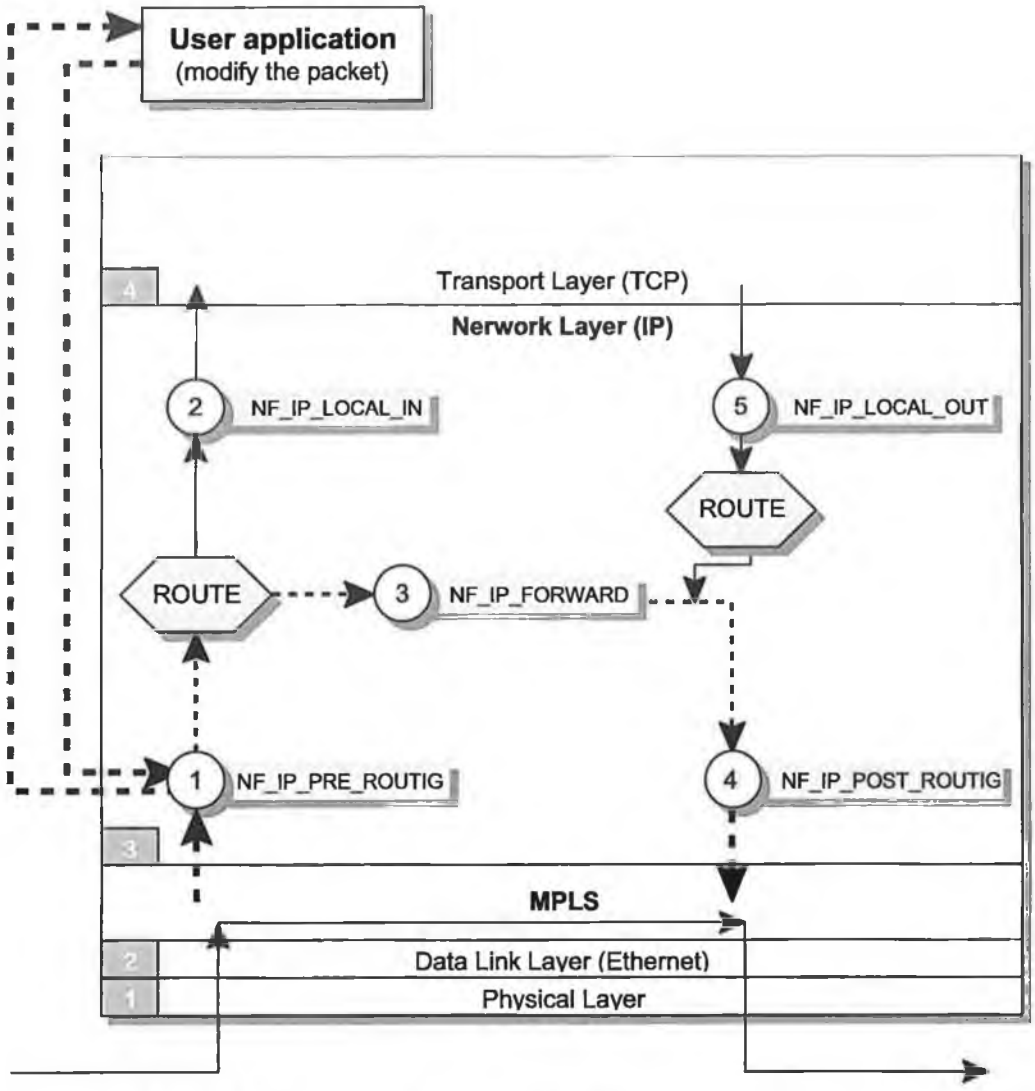


Figure 4.5: *An MPLS node modified to programme the packets*

Control Message Protocol) messages from **LSR A** to **LSR C**.

A response ICMP message should be received by **LSR A** as a reply for each ICMP message sent, but because packets from **LSR A** destined to **LSR C** go through **LSR B**, their source address is changed to that of **LSR D**. Therefore, when **LSR C** receives the ICMP messages but thinks that the packets come from a destination other than **LSR A** (the destination address having been altered) it sends the reply to that destination, which is **LSR D**.

This proof of concept demonstrates how a labelled packet can be identified as an active packet and passed to/retreived from userspace. A complete active network architecture for MPLS would require:

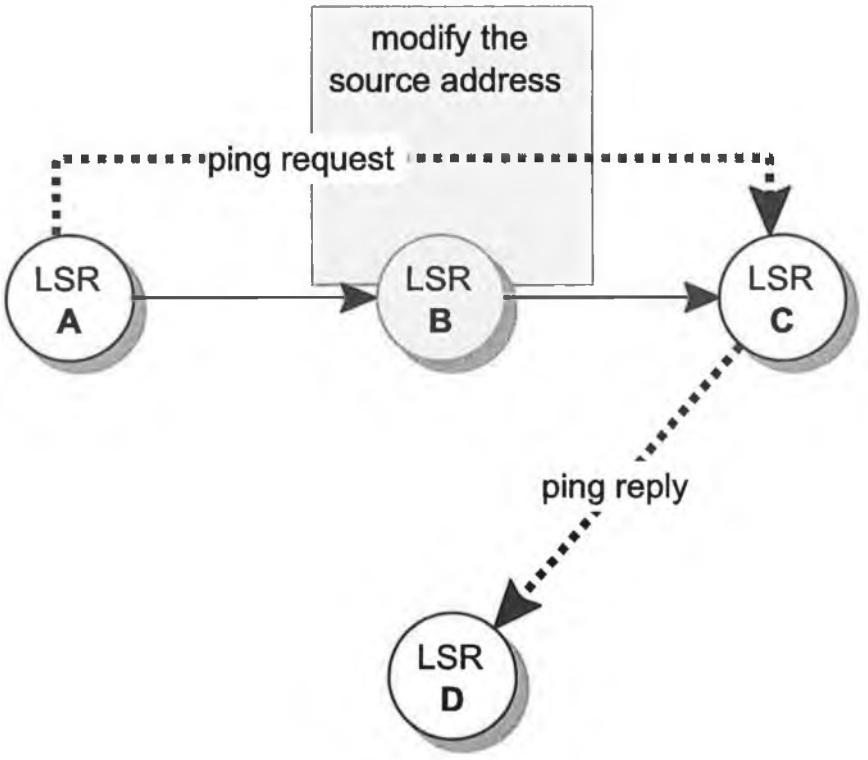


Figure 4.6: *Our experiment test*

- a mechanism to load, upload and maintain active code in userspace;
- a label management scheme which restricts the appropriate labels for use with active packets;
- a security mechanism.

The first requirement is generic to all active network architectures. An existing architecture, such as those discussed in Chapter 3, could readily be adapted to fit into the framework described here. Some minor changes to label management software in passive MPLS nodes would be required to ensure that they do not generate bogus active packets. Alternatively, the EXP bits in the MPLS header could be used to identify active packets. If no active packets are relayed by edge routers from the external network, and all routers in the MPLS cloud are trusted, the security mechanism can be much simpler than in IP networks.

4.2.2 Applicability examples

Some applications which could benefit from an integration of MPLS and active networks are presented below.

4.2.2.1 MPLS Web Switching

The solution jointly developed by this author and presented in [62] for the problem of overloaded web servers within an MPLS context, which required modifications to label assignments, can also be implemented using active networks. Here, a cluster of web servers is used for handling a large number of requests. The problem is how to assign a specific request to a server within the cluster. A dispatcher is positioned in front of the web server cluster in order to distribute the requests among the servers in a round-robin fashion.

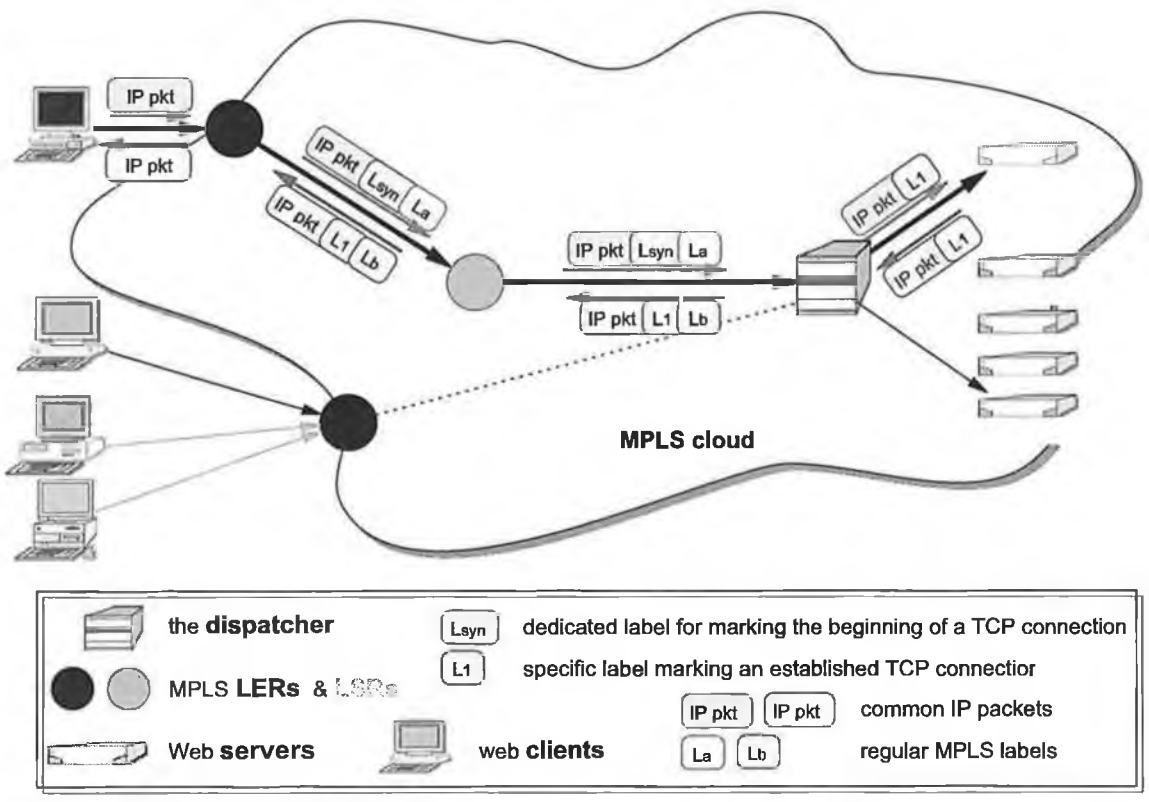


Figure 4.7: *The MPLS Web Switching scenario*

For treating the TCP continuity problem (i.e. all packets for one TCP connection should reach the same web server) a dedicated label was used for marking

the beginning of a TCP connection (e.g., L_{syn}). Thus, for every new connection the dispatcher will assign a new specific label which will be placed in a table within the dispatcher to specify the server to which that connection (e.g., L_1) has been assigned. Subsequent packets from that connection will contain the assigned label in their label stack so that the dispatcher will know to which server they should be sent. Such a scenario is illustrated in Figure 4.7.

By using active networks this mechanism can be implemented without the need of special labels (e.g. L_{syn} and L_1). First, the ingress nodes are augmented to have *active* capabilities. When the first packet (called *SYN*) from a TCP connection enters the MPLS cloud, the ingress router sends it directly to the dispatcher which recognises it as a *SYN* and assigns a server for the new TCP connection. After the dispatcher chooses the server, it sends an active packet to the ingress node which "instructs" the ingress node to redirect (using DNAT (destination network address translation)) all succeeding packets for that TCP connection directly to the corresponding server. The last packet of the TCP connection (called *FIN*) will delete the DNAT redirect established for this specific TCP connection within the ingress node.

Using active networks instead of dedicated/specific MPLS labels makes the communication between the servers and the clients more direct. Moreover, no more connection states need be kept within label edge routers (LERs) (e.g., L_1) or dispatcher. Since the LERs already have the responsibility of attaching labels to every packet entering the MPLS cloud, the added task of maintaining state for every TCP connection and its corresponding label is a significant extra burden. This is not required in the active solution. Also the passive solution uses as many L_1 labels as there are web connections, significantly reducing the number of available labels, compared with the active approach.

4.2.2.2 Online auctions

Another example concerns a web server running a live online auction by collecting and processing client bids for the available items. This server also responds to requests for the current price for an item as shown in Figure 4.8. For a large number of clients, the server can become overloaded and because of response time delay, the information about bids may be out of date by the time it reaches the client.

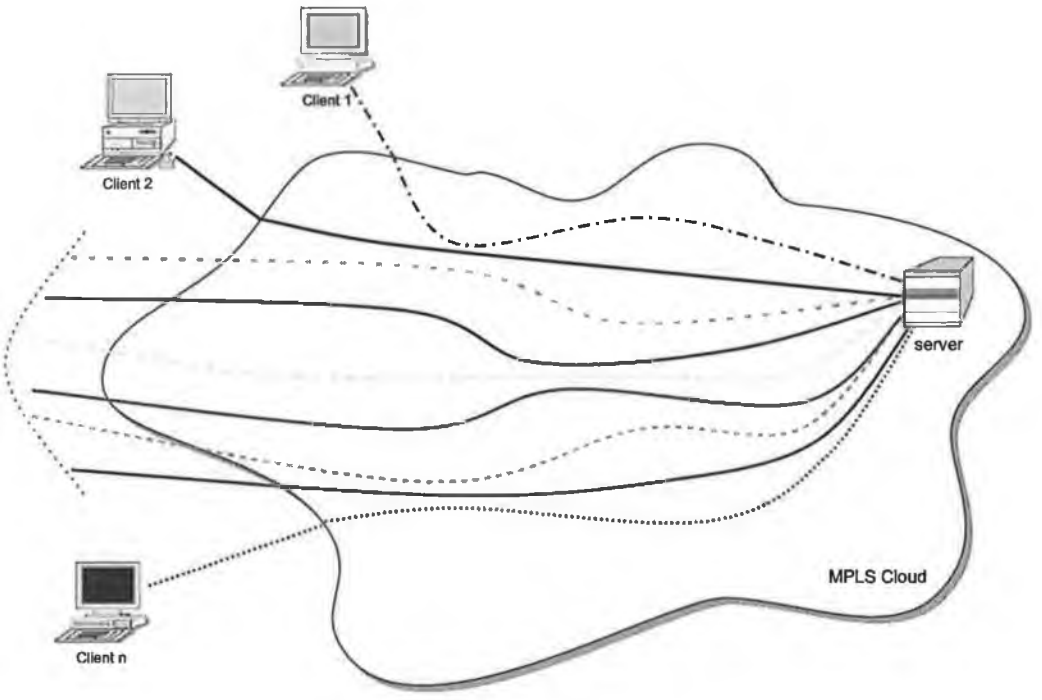


Figure 4.8: *The online auction scenario*

In an active network, low bids can be filtered out in the network (as in Figure 4.9), before they reach the server. When the server senses that it is heavily loaded, it can activate filters in the nearby network nodes and periodically update them with the current price of the popular item.

Filtering active nodes drops bids lower than this price and sends bid rejection notices to the appropriate clients. This frees up the server resources for processing competitive bids and reduces network utilisation near the server.

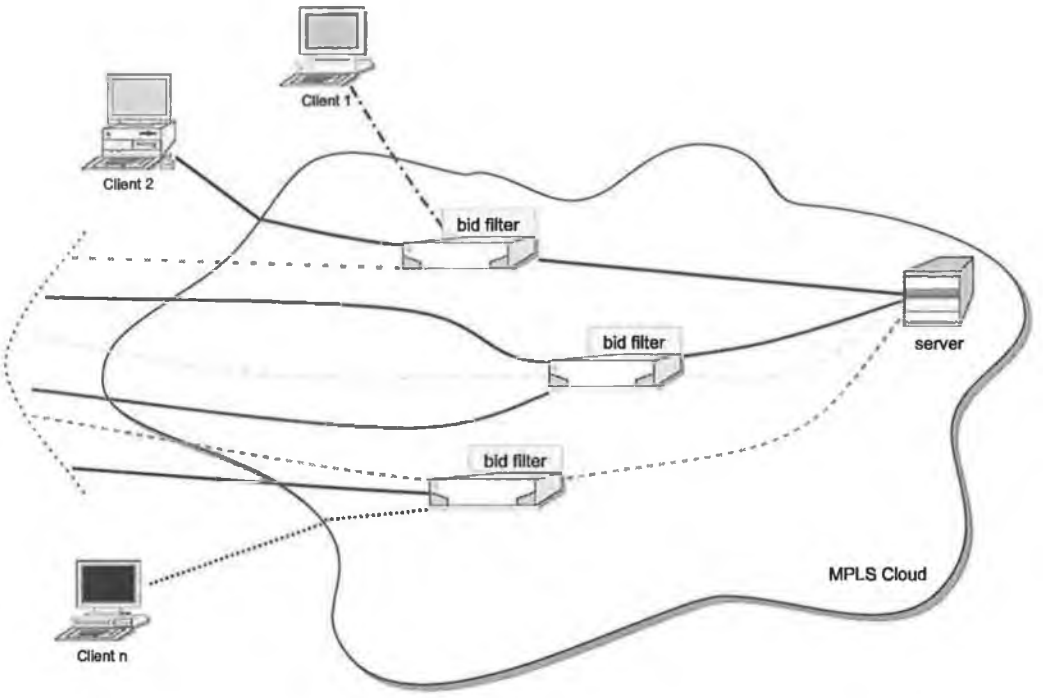


Figure 4.9: *Online auction using bid filters*

4.3 Applicability to QoS routing solutions

The suitability of both mobile agents and active networks to implement QoS routing algorithms is considered below. The features required of a ideal routing algorithm are:

- ***optimality**: the capability of the routing algorithm to select the best route;*
  - Mobile agents can in principle find the optimal route by exhaustively searching the network to find the best path, if such a path exists. Moreover, they can find all possible paths between a source and a destination through parallel searches performed by dispatching at each node a mobile agent through each outgoing link.
  - In an active network context, however, the *active code* travels along the traffic or awaits it on *active nodes*. Thus, a single path can be found, which may or may not be the optimal one.

- ***simplicity**: be as simple as possible;*
  - Both mobile agent and active networks offer potentially simple routing solutions by distributing the overall problem across mobile agents or active code entities.
- ***low overhead**: the algorithm should incur a minimum of software and utilisation overhead;*
  - Using mobile agents considerably reduces the computational and storage overhead because of their distributed nature. However, the evolution of the mobile agent population may result in a considerable communication overhead.
  - As with mobile agents, active networks routing solutions have low computational and storage overhead by distributing the computation. Moreover, the communication overhead is also very low because unlike mobile agents, active packets do not multiply.
- ***robustness and stability**: the algorithm should perform correctly in any unusual or unforeseen situation;*
  - Mobile agents have the ability to dynamically react to unfavourable situations. They can change, terminate or replicate in response to any network changes. For instance, mobile agents can dispatch to other nodes or terminate as a reaction to network node or link failures. This means a system/algorithm which uses a mobile agent population can typically survive many types of failures and unplanned behaviours at the individual agent level, without sacrificing task completion.
  - In an active network the loss of the current node or link can potentially be fatal, unless redundancy has been specifically designed into the active application.

- *flexibility: the algorithm should quickly adapt to changes.*
  - Both mobile agent and active networks are able to change (i.e. contained code can be easily changed) or extend their capabilities on-the-fly (e.g., by downloading code off the network).

The main advantage of mobile agent and active networks for QoS routing consists in treating the problem in a *"divide et impera"* manner. Thus, they may help reducing the overhead introduced by QoS routing:

- the state information is accessed *in situ*. Thus, the only communication overhead is the communication between active networks nodes or among the mobile agent population.
- the distributed path computation considerably reduces the computational overhead at any single node.
- there is no storage overhead because there is no need to gather the state information in one place.

Moreover, the modular structured code used by both paradigms allows their routing solutions to be more flexible (see above). It also allows the coexistence of multiple distinct routing techniques, used by different end-user applications.

The specific advantages of active network approaches consist in:

- They do not generate communication overhead (compared with mobile agents).
- End-users or applications are able to implement their own algorithms/procedures of packet routing.

Mobile agent approaches have as their advantages:

- A great power of adaptability. They can change, die, or reproduce in response to network changes. Therefore they are robust, fault tolerant and flexible.

- Distributed and parallel computations. By dispatching throughout the network, they can execute on different machines on parallel, reducing in this way the computational burden as well as the time required for the specific task at individual nodes.

- The "stigmergy"[6]. The indirect communication between entities of the same or different mobile agent colonies.

- *Emergent behaviour*[5]. Their ability to collectively exhibit complex behaviour, such as is required in solving the QoS routing problem, as a result of using simple primitives, i.e. mobile agents, which use simple interactions.

- They can easily evaluate a multiplicity of paths, maximising the likelihood of finding the optimal path.

It is apparent from the above discussion that, provided care is taken to manage the mobile agent population (thereby limiting the communication overhead incurred), mobile agents are a more powerful tool than active networks for use in network routing. A new approach to QoS routing is described in the next chapter, using mobile agents. It will be shown to offer advantages over conventional link-state approaches to this problem.

4.4 Deploying mobile agents in MPLS networks

Typically, mobile agent interpreters are part of the application layer. Thus, all mobile agents are encapsulated into IP packets and sent to the next node, where they have to traverse the protocol stack so they can get to the (local) interpreter to be processed. In general, a mobile agent interpreter uses a simple socket interface in order to communicate with corresponding peers, as in the first implementation of the Wave Interpreter [31]. Thus, the mobile agent system can run on either an MPLS or legacy IP network, without being recoded.

[6]See Section 2.5.2

Any large scale deployment of a mobile agent system in MPLS routers would require careful attention to be paid to performance issues. The ideal solution would be for each router to contain a hardware engine (either FPGA or ASIC) on which the agent could execute. A hardware interface would then need to be standardised between this engine and router resources such as routing tables and packet processing paths. In the absence of such a standard, it will need the co-operation of vendors to incorporate the mobile agent architecture in their processing engines. The security concerns of vendors regarding programmable networks would need to be addressed in advance of such deployments.

4.5 Summary

This chapter addresses the question of whether mobile code offers advantages in solving the QoS routing problem. The two paradigms considered are active networks and mobile agents. The problem domain being considered is that of MPLS networks, and thus the first concern is whether such networks can support these paradigms. A proof of concept by the author, implemented in Linux, shows that the active network paradigm, normally considered for use in IP networks, is also applicable in MPLS. A number of applications of active networks in MPLS networks are described to demonstrate that their deployment in an MPLS context would be worthwhile.

Mobile agent architectures are not tied to a specific networking protocol, unlike active networks. Thus their deployment in an MPLS cloud requires no special consideration.

It is concluded that the search capabilities of mobile agents make them the more powerful tool in implementing QoS routing, an application that will be explored in Chapter 5.

CHAPTER 5

Scalable Routing using Mobile Software Agents in MPLS networks

"The QoS routing is a key network function. It has two main objectives: finding routes that satisfy the QoS constraints and making the efficient use of the network resources." Shigang Chen [41]

Deploying QoS routing in large networks raises scalability issues in terms of higher storage, computational and update overhead. This is because the size of routing tables increase with the number of QoS constraints; the computational burden also increases when finding multiple-constraint paths or computing routes with a very fine granularity down to the level of a flow; update messages used for disseminating the link state information are larger and more frequent.

A solution to these issues is presented in this chapter. It uses mobile agents to determine routes, because of the benefits of this approach, as discussed in Chapter 4. The strengths of this solution are most apparent when applied to hierarchical QoS routing. This extension to the basic algorithm in described in Chapter 6.

5.1 Macro-routing

In this section a new protocol is proposed that addresses the problem of hierarchical routing within MPLS networks. It is called *Macro-routing* because, being an inter-domain routing protocol, its routing decisions at the higher levels are macro-decisions, as opposed to the detailed or micro-decisions made at the lowest level of the hierarchy.

Macro-routing is capable of both routing and signalling functionalities which are accomplished by the use of mobile agents. Thus, instead of advertising state information, small mobile agents are dispatched to process such information at each node. The advantage of this approach is that the information used to compute routes can be much more detailed that in traditional link-state protocols (e.g., it can feature multiple QoS constraints, or a Full-Mesh aggregated representation). Moreover, by using mobile agents which can replicate at each node and therefore analyse a large set of paths, route computations are done in a distributed and parallel manner which reduces the time required for path setup and distributes the processing burden amongst mobile agents.

5.1.1 Protocol description

The hierarchical organisation of *Macro-routing* consists at the lowest level of a number of domains which are typically independent administrative areas. The nodes within such domains are physical network nodes (i.e. router or switches). Each domain has a *managing node*, which must be able to interpret mobile agent code. It can either be selected from the nodes of the domain (as with PNNI) or it can be a distinct node (as in HDP). Its main function is to maintain an aggregated representation of the domain it is managing.

As the hierarchy is decided administratively, each domain at the lowest level of the hierarchy may choose its own routing strategy. For example it may use standard link-state methods, or may use mobile agents for route discovery. The

latter method implies the existence of a mobile agent interpreter on each router or switch. The only *Macro-routing* requirement is that the managing node must contain a *Full-Mesh* aggregate representation (see Section 3.2.1) of the managed domain. The maintenance of that aggregate representation is the responsibility of the domain administrator.

For the higher levels of the hierarchy the managing node creates an aggregated representation in four steps:

1. Each border node (including the source and destination nodes) activates a mobile agent (i.e. a *wave*)[1] that floods the domain by replicating itself at each node. Its goal is to find all possible paths to all the other border nodes within the same domain. Each mobile agent records the path it follows and processes the routing information at each node. If one mobile agent is revisiting a node, or the path it has traversed to date does not satisfy the given QoS constraints, it will be discarded. If it reaches another border node it will transmit the path used and its cost to the managing node.

2. The managing node chooses one optimal path between each pair of border nodes. In describing the protocol, only additive path cost constraints (e.g., total delay) are considered, for ease of explanation. However, the selection can be based on any QoS constraint.

3. A Full-Mesh aggregation topology is created using the selected paths. The costs of the selected paths will become *nodal costs* when computing paths at the next higher level of the hierarchy. Some or all of the other computed paths, which have not been selected for the Full-Mesh representation, can be cached for recovery purposes or as alternative paths.

4. There might be cases when the best path between two border nodes traverses other border nodes. This situation was not considered[2] during the

[1] see Section 5.1.2

[2] Every mobile agent terminates as soon as it reaches a border node, which means that no path

wave-based search. Thus, the final Full-Mesh aggregation has to be re-built by changing every direct border-to-border connection with another path which traverses other border nodes, if such path has better cost. The new paths can be obtained by concatenating paths already found during the wave-based search, while verifying that no node exists more than once in the concatenated paths except the border nodes that are the "concatenation points" for the given paths.

There are three major phases in the *Macro-routing* protocol whereby it finds and selects a QoS path from a given source to a given destination.

5.1.1.1 Determination of participant domains

The first phase involves determining the domains through which the path is likely to pass. It develops in two stages.

In the first stage, the source node initiates an "*upwards search*" in the hierarchy for the lowest level **parent node** which has a view of both source and destination, as in HDP and Viewserver.

In the second stage, the **parent node** initiates a "*downwards search*" in parallel to all its children. Recursively, the nodes reached will continue the search to all their children until they reach the lowest level of the hierarchy. All the physical domains reached by this search will be ***participant domains***.

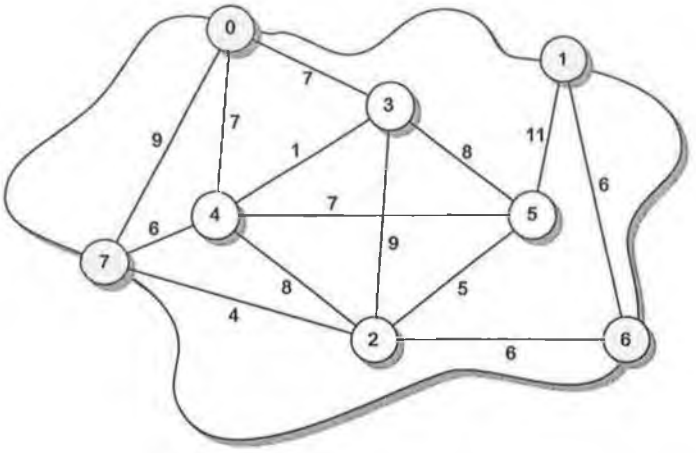
5.1.1.2 Path computation

The second phase involves the determination of the path. This can be done either on-line and/or off-line.

For on-line path computation, every managing node of the ***participant domains*** will create its own aggregate representation by calculating routes between all domain border nodes. Mobile agents search for paths that traverse the domain

between two border nodes will traverse another border node.

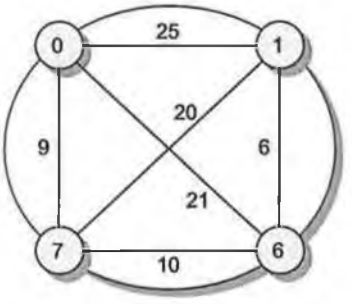
satisfying the QoS constraints. Amongst those paths a single *best* path is selected for each border-to-border pair in order to build the Full-Mesh nodal aggregation for the next level in the hierarchy. An example of Full-Mesh aggregation within *Macro-routing* is depicted in Figure 5.1.



(a) Physical domain topology

ingress node	egress node	list of traversing nodes	path cost
0	1	4, 5	25
0	6	4, 2	21
0	7	-	9
1	6	-	6
1	7	2, 5	20
6	7	2	10

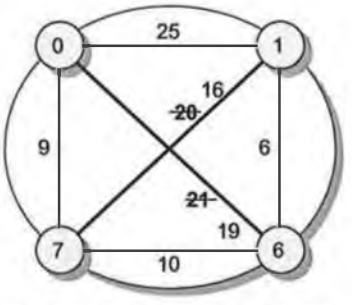
(b) the information content for the intermediate aggregated node



(c) the intermediate aggregated node

ingress node	egress node	list of traversing nodes	path cost
0	1	4, 5	25
0	**6**	**7, 2**	**19**
0	7	-	9
1	6	-	6
1	**7**	**6, 2**	**16**
6	7	2	10

(d) the information content for the aggregated node



(e) the final aggregated node

Figure 5.1: *The* Full-Mesh *nodal aggregation*

Mobile agents released from each border node traverse the domain multiplying themselves in search of all possible paths that satisfy the QoS constraints. At the end of its journey (i.e. upon reaching a border node different from the starting

one) each mobile agent sends the gathered information to the managing node of the current domain (see Figure 5.1(c)).

Sending the information to the managing node can be done either by piggy-backing on legacy messages or by using mobile agents. The WAVE system has routines incorporated in it that allow a "virtual" direct access between any two network nodes.

The information sent by a successful[3] *wave* to the managing node comprises the following:

- the starting border node: the *ingress node*
- the list of traversed nodes
- the path cost (representing the final QoS metric for the path containing the nodes from the list of traversed nodes)
- the final border node reached: the *egress node*

Starting from the second level of the hierarchy, *nodal costs* will be considered as well as link costs when computing the path cost[4]. The topmost domain will have as border nodes only the aggregated representation of the source and destination domains. The managing node of this domain will determine all the possible paths between its border nodes (the source and the destination) and so it can determine the optimal path based on their costs. The other paths found during this process can be used by fast recovery mechanisms or as alternative paths.

The paths for permanent and/or long-term connections can be determined off-line and only updated if/when necessary, while the paths used by short-term connections are computed online (i.e. on-demand). However, pre-computations or capacity planning can also be performed off-line.

[3] A *wave* that traverses the entire domain from one border node by another following a path that satisfies the constraints. Other *waves* terminate along their journey either because the path they follow does not satisfy the constraints or because they end up in a cycle.

[4] Costs include link costs and nodal costs. Nodal costs represent the cost of traversing a virtual node. This nodal cost is zero at the lowest level of the hierarchy.

5.1.1.3 Path reservation and set up

To accommodate the traffic for which the request has been made, the final path must be set up and the resources reserved. The overall path can be determined by traversing the hierarchy downwards and interrogating all the managing nodes along the chosen path about the detailed sub-paths across their domains.

The path set up and resource reservation can be done either by existing signalling protocols if they can set explicit paths, by existing resource reservation protocols (such as the Resource reSerVation Protocol (RSVP)), or by using suitably programmed mobile agents. The advantages of using mobile agents instead of RSVP for the reservation process are:

- **availability:** Mobile agent support is already available in the nodes, so there is no need to deploy or configure additional software;
- **parallelism:** RSVP has to traverse the path in a sequential manner twice: by using $PATH$ messages, from source to destination, to determine the path and then by using $RESV$ messages, from destination to source, to reserve the path. Mobile agents can do the reservation in a parallel and distributed manner so that, from each hierarchical level, within each domain, a mobile agent can be dispatched to reserve resources corresponding to each logical/aggregated link. Moreover, this process can be performed while determining the overall/"detailed" path as described previously.
- **hierarchical reservation:** By "hierarchical" reservation is meant the process of reserving the resources for the overall path in a manner which corresponds with the hierarchical representation used by the routing protocol (i.e. Macro-routing). Therefore, the first resources to be reserved are those corresponding to the links within the topmost (e.g., level k) domain within the hierarchy. If such resources are not available another k level path will be selected until there is a path with available resources. Then, the process

continues within the $k-1$ level domains which are represented by the virtual nodes traversed by the selected path at level k. The process stops when $k=0$. The advantage of such a reservation strategy is that any unavailable sub-path can be substituted on the spot while all other resources (previously addressed) remain reserved. RSVP, however, performs sequential reservation. This means that any failure is reserving a resource will result in an overall failure, which requires finding a new source-to-destination path and starting the reservation process all over again.

Setting up a hierarchical path in an MPLS network is straight-forward, using the MPLS label stack capability. Thus, every sub-path within every domain and every hierarchical level can be treated independently. Using mobile agents for setting up the LSP (Label Switched Path) is also preferred as mobile agents assigned to reserve the resources can also set up MPLS labels and then replicate and dispatch to advertise/distribute them. This can be done in a distributed and parallel manner using multiple mobile agents as described above. Any other label distribution protocol has to do the path set-up sequentially.

When all the resources have been successfully reserved and the overall path has been set up, the request is served and the traffic may flow. In the case of resource unavailability or link/node failure, alternative paths which are already computed can be used for a fast recovery.

5.1.2 Implementation details

The *Macro-routing* protocol can be implemented using any mobile agent technology. The implementation presented here uses Wave technology, which is described in Section 2.5.4 and Appendix A. The first[5] implementation of the Wave model, written using the C programming language under Solaris-Unix, was used.

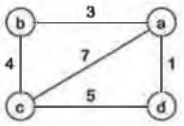
[5]There was one more implementation, in Common-Lisp, preceding this one, which was suffering on efficiency problems [29]. Thus, Peter Borst, the author of these implementations refers to the C implementation as the first.

This implementation of the Wave Interpreter was written by Peter Borst and described in [29].

Only the implementation of the *Path computation* phase of the protocol is described below. The other two phases (*Determination of participant domains* and *Path reservation and set-up*) have a straightforward implementation.

The *Path computation* phase starts from the lowest level of the hierarchy and sequentially progresses through the hierarchical levels until it reaches the root (the lowest level parent of both source and destination). Within a single level of the hierarchy, our protocol evolves in parallel within all domains. Each managing node initiates a search from each border node of its domain to all other border nodes. The implementation of this "wave-based path search" is described next.

The Wave approach to mobile intelligence requires us first to construct a virtual *Knowledge Network* (KN) which is an abstract view of the physical network assembled by the wave agents as they probe its links and nodes. A KN ([143]) can be created using very simple code[6] as in Figure 5.2.



(a) physical topology

```
CR(@#a.Fa=A.7#b.\
    4#c.Fc=A.5#d.2#Fa.1#Fc)
```

(b) WAVE code for creating the virtual Knowledge Network

Figure 5.2: *The Wave implementation for creating a four-nodes network*

The algorithm depicted in Listing 5.1 can be applied for the path search within first-level domains. The nodal costs at this level are zero as no aggregation has been performed yet. The corresponding Wave implementation is depicted in Figure 5.3. Since the Wave implementation is very compact, the *waves* used are very small.

Both algorithms in Listings 5.1 and Figure 5.3 describe distributed[7] and par-

[6]See a detailed explanation of this code in Appendix A (Section A.3).

[7]Every *jump* to another node results in the remaining operations being performed on this new node.

allel[8] computations performed by waves as they evolve while spanning the network. Due to its recursive spatial control, the use of control rules (e.g., RP and OS in Figure 5.3) in WAVE allows coordinated societies of mobile agents to evolve.

Pseudo-code Listing 5.1: *The path computation algorithm (for zero nodal costs).*

```
1   Input BorderNodes
2   Input requirements
3   Input managing_node
4   FOR every node ∈ BorderNodes DO
5       Other_Borders = BorderNodes \ {node}
6       netspan (node, ∅, NULL, Other_Borders, requirements, managing_node)
7   ENDFOR
8   // the definition of the netspan subroutine
9   netspan (node, Path, path_cost, Borders, req, managing_node)
10      jump to node // jump is a WAVE-specific command
11                   // which results in an effective jump to the specified node
12      IF Path ≠ ∅ THEN
13          link_cost = cost of the passed link
14      ENDIF
15      IF node ∉ Path AND path_cost + link_cost < req THEN
16          Path = Path ∪ {node}
17          path_cost = path_cost + L
18          IF node ∈ Borders THEN
19              RETURN Path and path_cost to the managing_node
20              TERMINATE // current wave terminates upon finding a valid path
21          ELSE
22              FOR every N ∈ Neighbours(node) DO
23                  netspan (N, Path, path_cost, Borders, req, managing_node)
24              ENDFOR
25          ENDIF
26      ELSE
27          TERMINATE // cycle or unavailable resources have been encountered
28      ENDIF
```

```
Falg=RP(OS(Fcost==0,A/~Fborders).\
        A/~Fpath.Fcost<Required.Fpath&A.Fcost+L.♯).\
Fborders=ARG1.Frequired=ARG2.Fmanaging=ARG3.\
@♯Fborders.Fcost=0.\
Falg.\
Fpath&A.Fcost+L.@♯Fmanaging.Npath&Fpath.Ncost&Fcost
```

Figure 5.3: *The Wave implementation of Figure 5.1.*

At the next level of the hierarchy the algorithm must consider non-zero nodal costs before it does a *local broadcast to all neighbouring nodes*. Specifically, since the nodal cost depends on the outgoing link, each agent must (to use Wave terminology) make an *explicit jump* to a specific next-hop node, rather than a *broadcast jump* (to all downstream nodes), adjusting the nodal cost as appropriate. The extension of the algorithm for this operation is depicted in Listing 5.2.

[8]Multiple societies of mobile agents are generated to perform similar tasks in parallel (e.g. a different society is generated from each border node).

Pseudo-code Listing 5.2: *Modification of the path computation algorithm of Listing 5.1 to account for non-zero nodal costs.*

```
    ...
22  FOR every N ∈ Neighbours(node) DO
→       nodal_cost = corresponding cost for traversing node from Predecessor(node) to N
→       path_cost = path_cost + nodal_cost
23      netspan (N, Path, path_cost, Borders, req, managing_node)
24  ENDFOR
    ...
```

5.1.3 Macro-routing performance

The *Macro-routing* protocol will be compared with HDP since this protocol was also designed to work in MPLS networks. For the example network depicted in Figure 5.4, the path chosen by HDP has a cost of 85, while the Macro-routing protocol found a path with a cost of 59. The HDP approach to topology aggregation resulted in a suboptimal path being chosen.

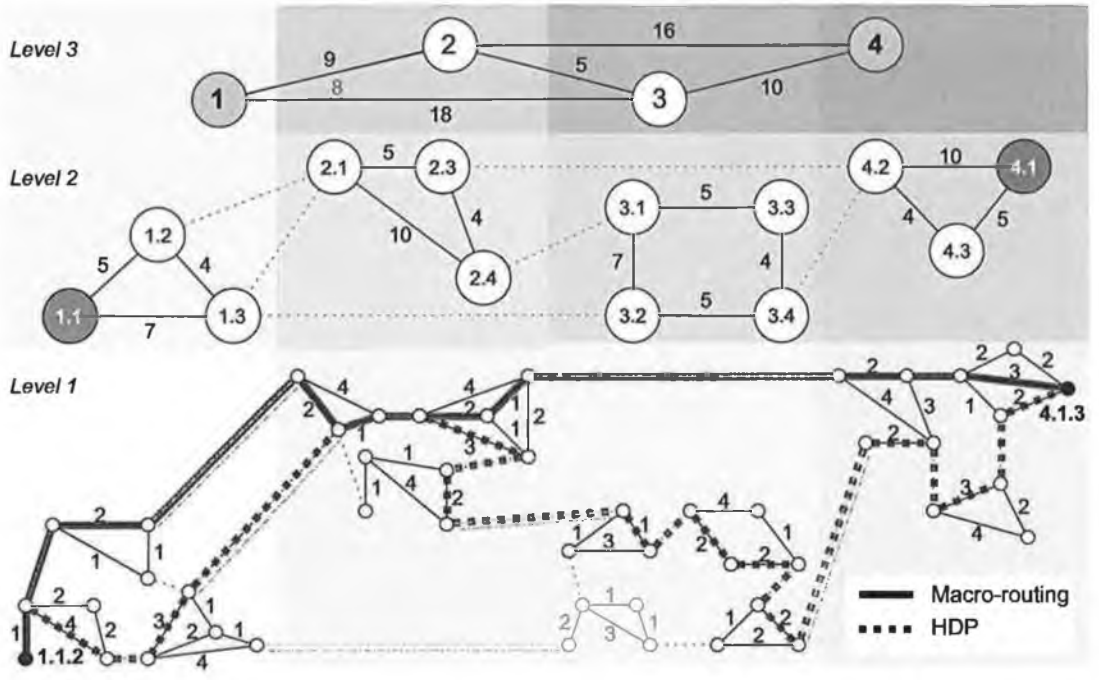


Figure 5.4: *Macro-routing versus Hierarchical Distribution Protocol*

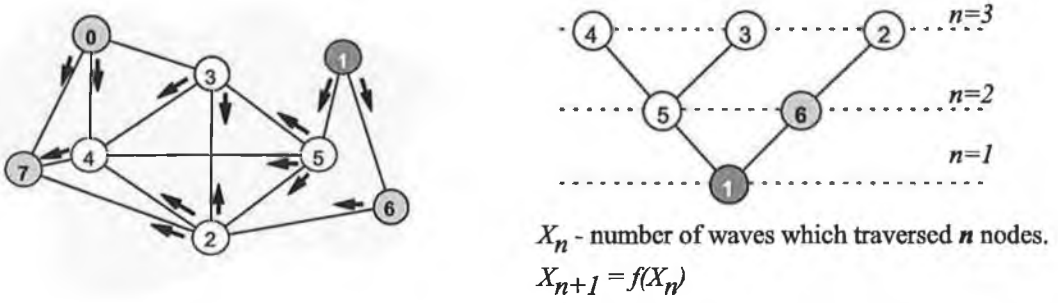
Macro-routing rapidly finds the optimal path using path computations executed in parallel not only in different domains at the same hierarchical level but also between any two border nodes of a domain. However, this parallel operation can result in a large number of *waves* traversing the network. Simulations and an-

alytical models are used below to investigate this overhead, which, if excessive, would render Macro-routing impractical.

To determine the amount of traffic generated by the protocol both the size and the number of mobile agents (*waves*) involved in the routing have to be determined. The size of a mobile agent (in bits) depends on the mobile agent technology used but, unless excessive, is not likely to limit the protocol's scalability. Counting the number of mobile agents requires either simulating the protocol operation over different topologies (such as Figure 5.5(a)) or developing a mathematical formula. Both approaches are considered below. For both cases the worst case scenario where all paths between source and destination satisfy the QoS constraints was considered, since this gives rise to the most *waves*.

5.1.3.1 The mathematical model for calculating the traffic overhead

The model considers only the *waves* generated by a single border node (see Figure 5.5(b)).



(a) simulations (b) analytical model

Figure 5.5: *Wave counting tests*

A recursive mathematical formula to yield the number of *waves* which traverse $n+1$ nodes based on the number of *waves* traversing n nodes is described by Markov Chain branching processes (also known as Galton-Watson processes) [35]: $X_{n+1} = \sum_{X_n}^{k=1} Z_n^{(k)}$, where $Z_n^{(k)}$ is the number of *waves* generated by node k.

It is assumed that $Z_n^{(k)} = 0$ if the current *wave* revisits the node k (i.e. if there is a cycle) or if node k is a border node (i.e. if a path has been found) and that

$Z_n^{(k)} = \alpha$ otherwise, where α is the average node degree. Hence,

$$X_{n+1} = (X_n - C_n - B_n) \cdot \alpha,$$

where C_n is the number of *waves* which end up in cycles and B_n is the number of *waves* which found a border node. Moreover, if $C_n = X_n \cdot p_{cn}$ and $B_n = X_n \cdot p_{bn}$, where p_{cn} and p_{bn} are the respective probabilities of a *wave* ending up in a cycle, and of finding a border node, results $X_{n+1} = X_n \cdot (1 - p_{cn} - p_{bn}) \cdot \alpha$. This formula gives us the number of *waves* generated by one border node. The values of the parameters p_{cn} and p_{bn} may be estimated by simulation.

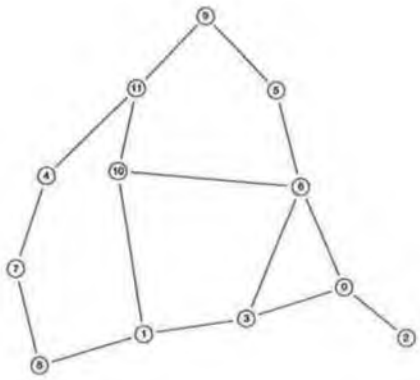
5.1.3.2 The simulation model

Simulation tests were performed to continue the analysis from the mathematical model.

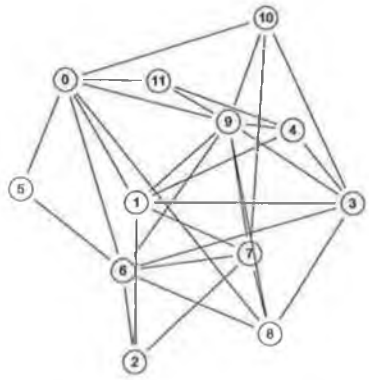
For all tests the *Georgia Tech Internetwork Topology Models* (GT-ITM) [175] was used to generate random network topologies. A few examples of such topologies are presented in Figure 5.6. The corresponding values of number of nodes, number of links and connectivity degree for each topology used in our simulations are shown in Table 5.1.

One or multiple constraints were associated to each link. The main metrics used were *administrative cost* $\in [1, 15]$ and *delay* $\in [2ms, 45ms]$. For each link, the corresponding metric was randomly chosen.

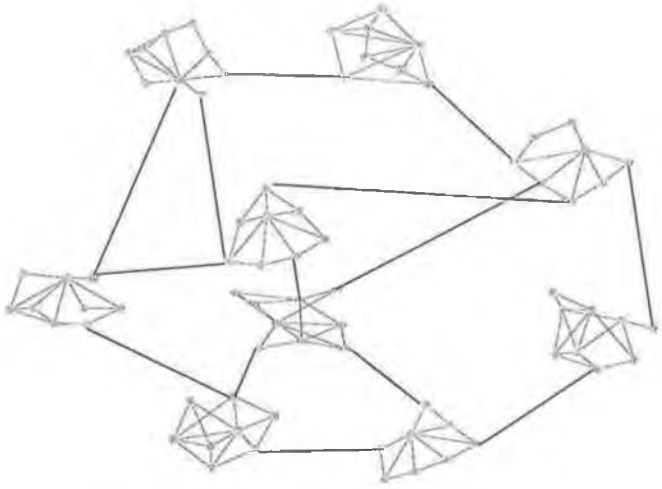
Two sets of tests were performed. For the first set of tests, the random topologies with associated metrics were used to create *virtual Knowledge Networks*. Then, *waves* implemented as in Figure 5.3 were dispatched within these networks as described in Section 5.1.2. Thus, one border node initiated a *wave* by using the *winject* utility as described in Appendix A.4. The aim of the first set of tests was to count the number of waves generated and determine how many of these waves are useful for the routing process and how many are not (i.e. end up in a cycle).



(a) Low connectivity topology



(b) High connectivity topology



(c) Two-level hierarchical topology

Figure 5.6: *Topologies used in simulation*

This allows the cost of searching for "long paths" to be estimated.

The second set of tests were performed on two-level hierarchical networks (see Figure 5.6(c)). Rather than deploying waves on these networks, an application was developed to generate a list of the paths the waves would discover. This application is presented in Appendix E.

The first step was to generate a random two-level network.

The GT-ITM utility was used to generate flat (single-level) topologies. The specification files generated by the GT-ITM contain information about nodes and links. So, the nodes are labelled and the links are presented in the format: *from_node to_node*. The test application reads these topology specification files and associates a randomly chosen cost with each link, so that a specific link is now characterised

Table 5.1: *Details about topologies used in simulation*

Flat topologies used for counting the number of waves		
No. of nodes N	**No. of links** L	**Connectivity degree** $2L/N$
< 9	[5, 10]	[1.66, 2.85]
9	[8,36]	[1.77, 8]
12	[11, 66]	[1.83, 11]
13	[12, 78]	[1.84, 12]
50	[57, 74]	[2.28, 2.96]
Two-level hierarchical topologies		
No. of nodes N	**No. of links** L	**Connectivity degree** $2L/N$
$4 \times 4 = 16$	24	3.00
$9 \times 9 = 81$	124	3.06
$10 \times 10 = 100$	149	3.98
$12 \times 12 = 144$	[216, 372]	[3, 5.166]
$13 \times 13 = 169$	282	3.33
$15 \times 15 = 225$	368	3.27
$20 \times 20 = 400$	600	3.00

by its associated cost as well. Such cost can contain one or multiple metrics. Using these flat topologies, the hierarchy is automatically created by taking one flat topology for level two and using the other flat topologies to expand its nodes to represent the first level of the hierarchy. The hierarchy is created in such a way that no node would have a *degree*[9] smaller than two, otherwise the hierarchy was considered *invalid*. To increase the average node degree, the topologies from the first hierarchical level with the fewest number of links (i.e., the smallest average node degree) were associated with the nodes of highest degree from the topology at the second level of the hierarchy. Thus, domains corresponding with such topologies would have more border nodes (i.e. would be aggregated as nodes with highest degree).

For each test from the second set, one source and one destination node were randomly chosen so that no direct connection exists between them (neither at the first level of the hierarchy nor the second). Therefore, the context of each test is

[9]The *degree* for a specific node is considered to be the number of (inter- and intra-domain) links that are incident to it.

defined by the hierarchy, the randomly assigned costs for each link, the source and the destination nodes.

The application processes in turn each domain/topology from the first level of hierarchy, so that, for each of them the costs of all paths between all border node pairs are computed, and the path with the best cost is selected for the aggregated representation. The search for all paths between two border nodes is done using an iterative backtracking algorithm described in Listing 5.3.

Pseudo-code Listing 5.3: *Iterative backtracking for finding all paths between two nodes*

```
Input source_node
Input destination_node
//Path - vector containing the nodes within current path
//Neighbour - vector containing the neighbours of the current node in the path
i = 1                                          // current index for Path
j = 1                                          // current index for Neighbour
Path[i] = source_node
Neighbour[j] = source_node
WHILE j>0 DO                                   // Neighbour ≠ ∅
    add_neighbor = FALSE
    p = j
    FOR EACH node for which exists link between itself and Neighbour[p] DO
        IF node ∉ Path THEN
            j++
            Neighbour[j] = node
            add_neighbor = TRUE
        ENDIF
    ENDFOR
    IF NOT add_neighbor THEN
        WHILE Path[i] == Neighbour[j] and j > 0 DO
            i--
            j--
        ENDWHILE
    ENDIF
    i++
    Path[i] = Neighbour[j]
    WHILE Path[i] == destination_node and j > 0 DO
        RETURN Path
        j--
        WHILE Path[i-1] == Neighbour[j] and j > 0 DO
            i--
            j--
        ENDWHILE
        Path[i] = Neighbour[j]
    ENDWHILE
ENDWHILE
```

After finding each path, its cost is computed correspondingly with its each metric's type (e.g. by adding the link costs if the metric is additive, or multiplying the link costs if the metric is multiplicative). From all paths generated by the iterative backtracking, the best is selected. When only one metric is used, the selection is straightforward. The selection mechanisms used in the case of multiple

constraints are described in Sections 6.1.3 and 6.1.4.

These selected paths are used to build the *aggregated nodes* for the domain in the second level of hierarchy. Then, using the same iterative backtracking algorithm, all paths between the aggregated "source" node and the aggregated "destination" node are found. Computing the path costs at the second level of hierarchy requires nodal costs to be accounted as well. These nodal costs are stored in an *extended matrix*. So that, if we consider that the link costs are stored in an $N \times N$ matrix called X, where

$$X[u, v] = \text{ cost of the link between } u \text{ and } v,$$

its extended version would be an $(N+1) \times (N+1) \times (N+1)$ matrix, where

$$\begin{cases} X[u, 0, v] - \text{ the link cost between } u \text{ and } v, \\ X[u, v, z] - \text{ the nodal cost of node } v \text{ if traversed from node } u \text{ to } z, \\ X[0, u_0, u_1] - \text{ the "nodal" cost of the source node,} \\ X[u_{k-1}, u_k, 0] - \text{ the "nodal" cost of the destination node .} \end{cases}$$

In the actual Macro-routing implementation, the source and destination were considered border nodes, connected by virtual (zero-cost) links to virtual empty domains. Therefore, the costs for $source - to - border\_node$ paths in the source domain and $border\_node - to - destination$ paths in the destination domain were also considered nodal costs. All considered costs are depicted in Figure 5.7.

The costs of one such path (e.g. $p = \{source = u_0, u_1, u_2, \ldots u_k = destination\}$), if for instance additive metrics are involved, would be

$$C(p) = X[0, u_0, u_1] + \left(\sum_{i=1}^{k-1} X[u_{i-1}, 0, u_i] + X[u_{i-1}, u_i, u_{i+1}]\right) + X[u_{k-1}, 0, u_k] + X[u_{k-1}, u_k, 0].$$

The paths generated by this application[10] are, in fact the paths the *wave* pop-

[10]This application in presented in detail in Appendix E.

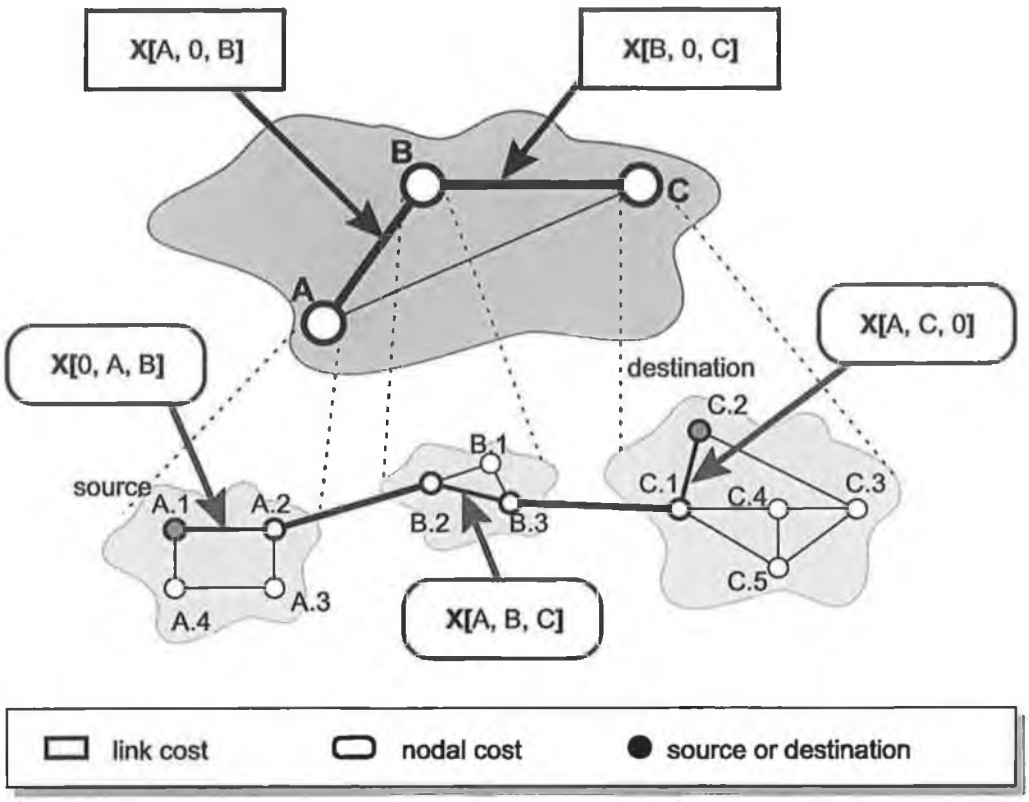


Figure 5.7: *Entry in the hierarchy specification file*

ulation will find (i.e. all possible paths). This application generated cycle[11] paths as well as successful paths. The total number of these paths are in fact the number of generated *waves*, because each path is the record of nodes visited by a *wave*, if *waves* were used to find them. The final goal of these tests was to determine the effect on performance of limiting the lifespan of a wave. Thus, the number of waves as well as the Macro-routing algorithm's performance were observed for different lifespan values so as to determine the optimal value of this parameter.

5.1.4 Simulation results

5.1.4.1 The *wave* population evolution within flat domains

To observe how the wave population develops in different contexts, multiple topologies with different connectivity degrees were investigated. The results depicted in Figure 5.8 show that a higher connectivity (i.e. $2L/N$), and thus larger

[11]The cost of cycle paths was considered -1.

number of links (i.e. L), leads to a more rapid *wave* population increase than does a larger number of nodes (i.e. N).

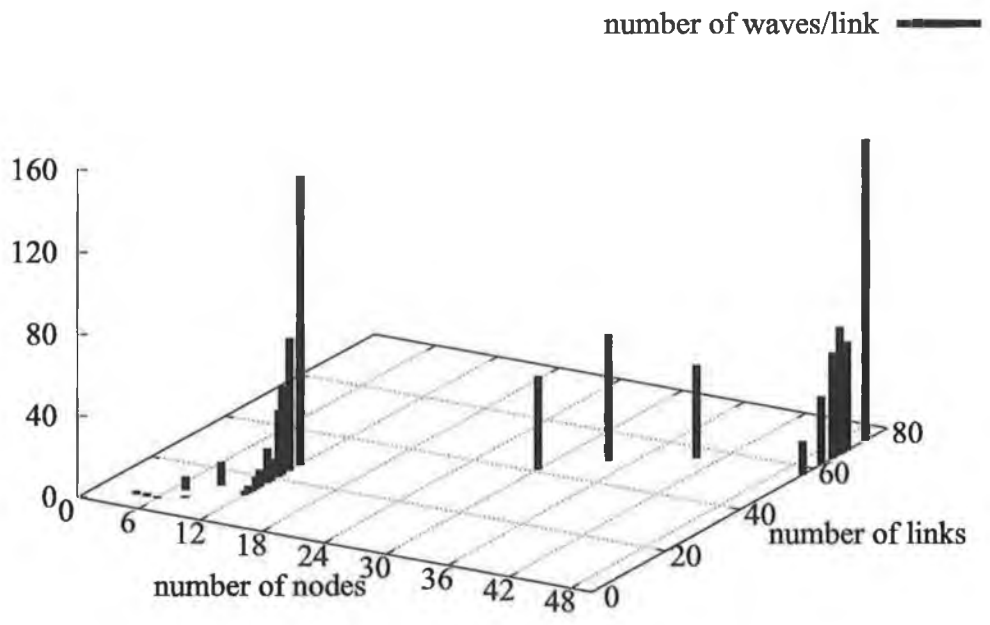


Figure 5.8: *Number of waves/link in various topologies*

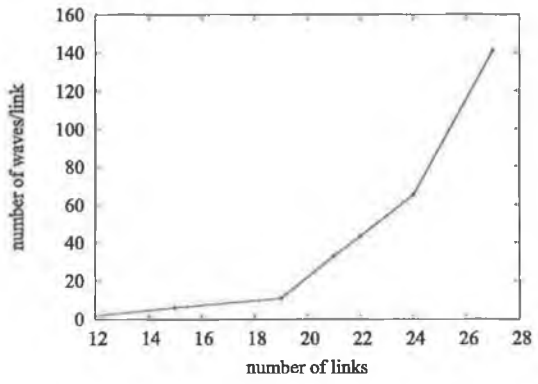
Similar results are presented in Figure 5.9(a) for topologies having the same number of nodes and varying number of links, while Figure 5.9(b) shows that the number of waves which end in cycles (and which thus do not contribute to route discovery) is very high with high connectivity.

For the next tests, the wave population was divided into various classes in accordance with Definition 1.

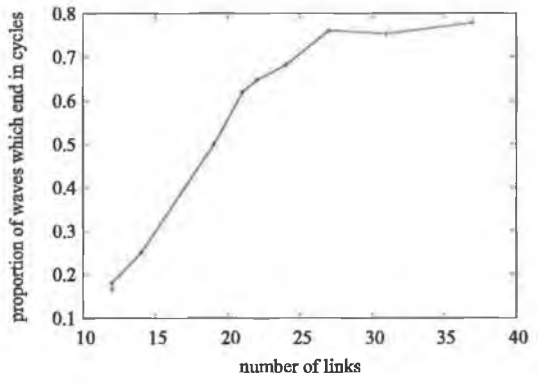
Definition 1 ***Border, cycle, and alive waves***

Let n be the number of nodes visited by the wave population. I define:

- ***border waves*** *- waves which reach a border node and thus have found an n node path - $B_w^{(n)}$ is the number of such waves;*
- ***cycle waves*** *- waves which reach (as the n-th node) an already visited node, and thus enter a cycle - $C_w^{(n)}$ is the number of such waves;*



(a) The average number of waves/link in topologies with 12 nodes as a function of connectivity degree (number of links)



(b) The proportion of waves which end in cycles relative to the total number of waves in topologies with 12 nodes as a function of connectivity degree (number of links)

Figure 5.9: *The* wave *population evolution on different topologies as a function of connectivity (i.e. number of links)*

- ***alive waves*** *- waves that are still alive and continue to evolve in the network in search of a path as they have not encountered any border node nor entered a cycle before or when visiting the* n^{th} *node -* $A_w^{(n)}$ *is the number of such waves;*

The total number of waves which already visited n *nodes is given by:*

$$T_w^{(n)} = B_w^{(n)} + C_w^{(n)} + A_w^{(n)} \tag{5.1}$$

Figure 5.10 presents the number of *waves* within a network of 13 nodes and 20 links as a function of the path length (i.e. number of nodes already visited). The total number of *waves* increases with the number of nodes visited until the path length is nine. Thereafter the high probability of *waves* ending in a cycle or finding a border node causes the total number of *waves* to decrease. A path length of thirteen results in a *wave* population of zero, since the probability of a *wave* ending up in cycle is unity (all nodes have been visited).

These results allow the *cycle probability* and the *path effort*, as defined below, to be calculated.

Definition 2 ***Cycle probability***

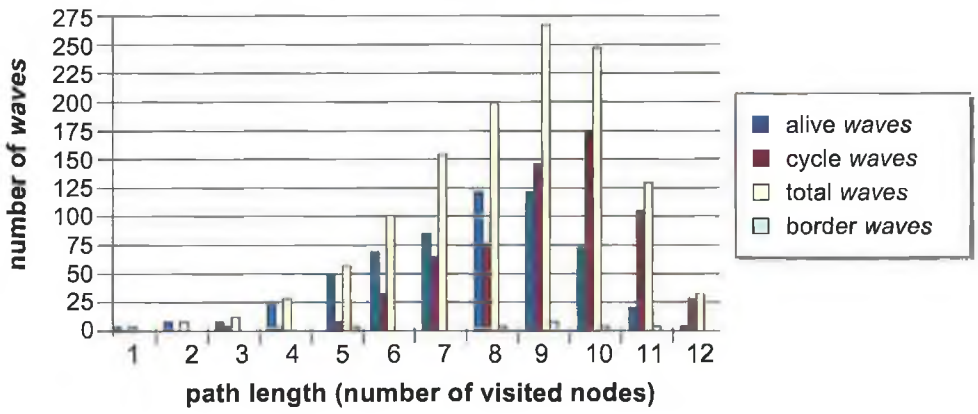


Figure 5.10: *The number of waves propagating from a border node.*

The cycle probability, *or the probability of one wave entering a cycle upon visiting the n^{th} node is estimated as:*

$$p_{cn} = \frac{C_w^{(n)}}{T_w^{(n)}} \tag{5.2}$$

Definition 3 ***Path effort***

The path effort *is the ratio of the number of ineffective waves which end up in cycles after n nodes have been visited to the number of waves which might find a path:*

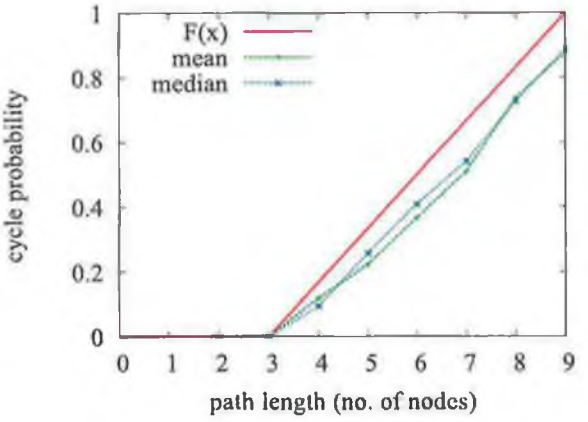
$$E_n = \frac{C_w^{(n)}}{A_w^{(n)}} \tag{5.3}$$

The tests for finding the *cycle probability* and the *path effort* were performed on three main classes of topologies, i.e. with 9, 12, or 13 nodes. Within each class, the connectivity degree is varied from the minimal level of connectivity (i.e. $L = N-1$ links), close to the full mesh connectivity (i.e. $L = \frac{N(N-1)}{2}$ links).

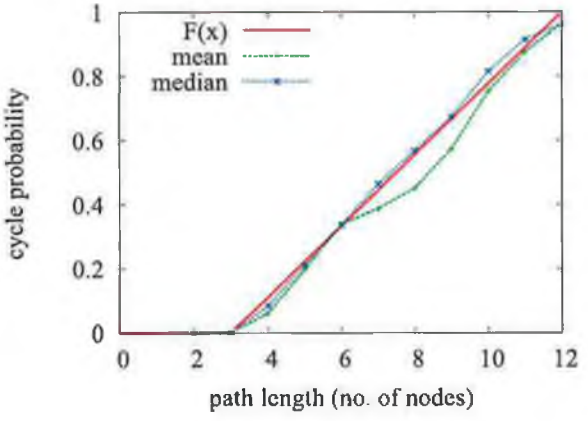
The results obtained for *cycle probability*, depicted in Figures 5.11, closely match the following function,

$$F(x) = \begin{cases} 0, & \text{if } x \leq 3 \\ \frac{x-3}{N-3}, & \text{if } x > 3 \end{cases} \tag{5.4}$$

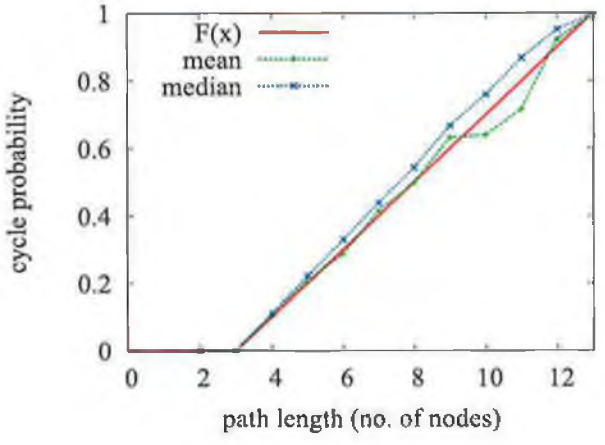
where x is the number of nodes already visited (i.e. path length) and N is the number of nodes. The value 3 from (5.4) means that there have to be at least 3 nodes in order to exist cycles.



(a) $N = 9$ and $L = \overline{8, 36}$



(b) $N = 12$ and $L = \overline{11, 66}$



(c) $N = 13$ and $L = \overline{12, 78}$

Figure 5.11: *The cycle probability (see Definition 2)*

The metric called *path effort* was considered in order to determine the threshold above which the protocol becomes impractical. Figure 5.12 shows that the effort involved in finding long paths is excessive when more that γ nodes are visited, where $\gamma = 7$ in the 9 node topology and $\gamma = 9$ in the 12 and 13 node topologies. From this results it can be concluded that the protocol must be modified to ensure its scalability.

5.1.4.2 Limiting the population of mobile agents

A parameter called *lifespan* is introduced to the algorithm which resembles the TTL field used in the IP protocol. Its purpose is to limit the number of *waves* generated during route search by reducing the number of generations which the parent *wave* can produce. The rationale for this is that the law of diminishing returns is assumed to apply - it is unlikely that an exhaustive search of every possible path is necessary to find the optimal path. The modified algorithm is no longer guaranteed to find the optimal path (and indeed will find no path if the destination is more than *lifespan* hops away).

The next set of tests verifies the influence of the *lifespan* parameter on the protocol performance. The metric used to evaluate it, and the terms used to describe the range of possible outcomes of the revised algorithm, are given below.

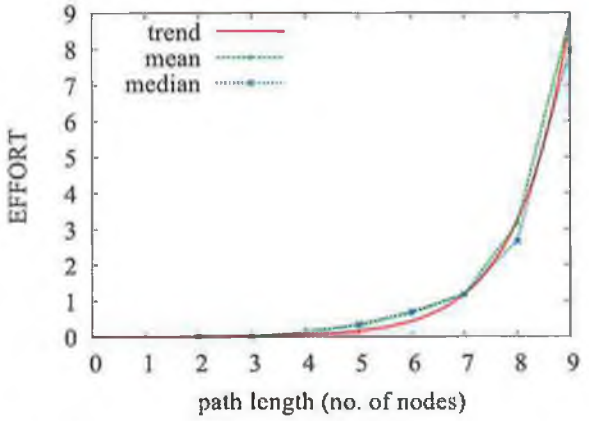
Definition 4 ***Efficiency***

Let C_{opt} be the optimal path cost between a specific source and destination, and C_{act} the actual path cost obtained by the (sub-optimal) lifespan-limited Macro-routing algorithm. Macro-routing's efficiency *is then:*

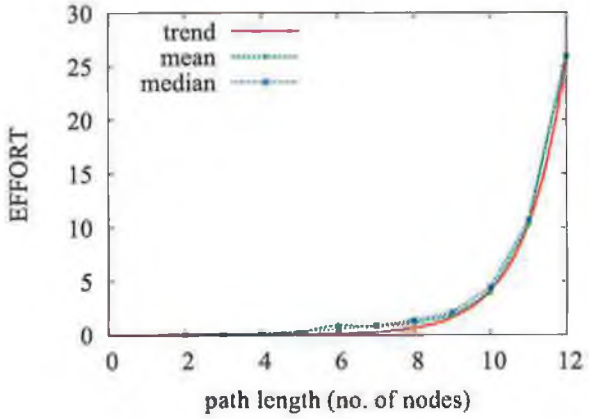
$$E = \frac{C_{opt}}{C_{act}}, \tag{5.5}$$

Definition 5 ***Failure, success, best***

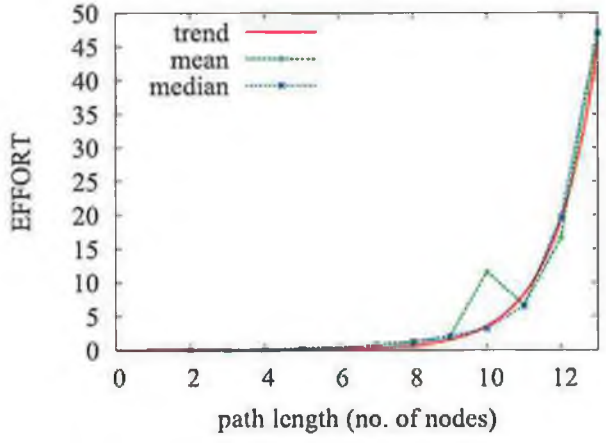
Let E be Macro-routing's efficiency as defined in Definition 4. I define Macro-routing's results as:



(a) $N = 9$ and $L = \overline{8, 36}$



(b) $N = 12$ and $L = \overline{11, 66}$



(c) $N = 13$ and $L = \overline{12, 78}$

Figure 5.12: *The path effort (see Definition 3)*

- failure: *no path is found (because of too low a lifespan value):* $E = 0$
- success: *paths satisfying the requirements are found, and:* $E > 0$
- best: *the paths found include the optimal path, i.e. :* $E = 1$.

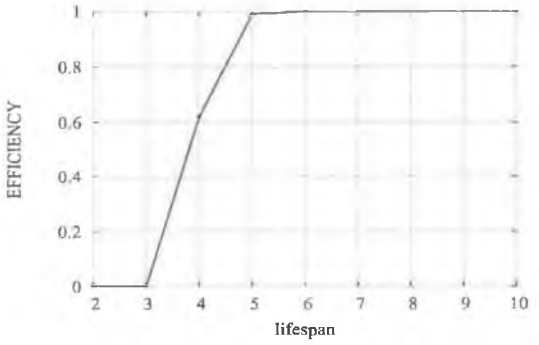
The next set of tests were performed on two-level hierarchical networks with connectivity varying from 3 to 5.166 and $12 \times 12 = 144$ nodes. These topologies were divided into three different classes based on their connectivity (c_d) varying in the following intervals:

1. $c_d \in [3, 3.66]$
2. $c_d \in [4, 4.5]$
3. $c_d \in [5, 5.166]$

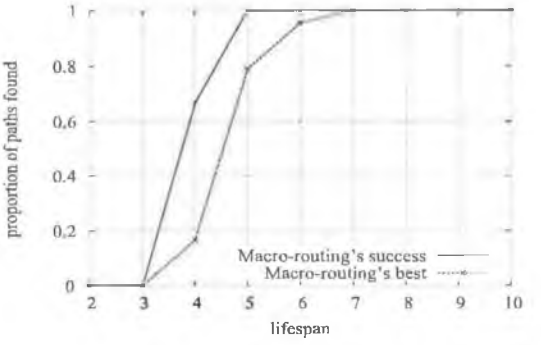
The mean values of the results obtained across the three classes are depicted in Figure 5.13. They show that on the given topology (with $12 \times 12 = 144$ nodes) a lifespan value above 5 does not affect Macro-routing's efficiency in a significant manner (see Figures 5.13(a) and 5.13(b)), while the number of waves is greatly reduced (see Figure 5.13(c) for the ratio between the cycle and the alive waves and Figure 5.13(d) for the total number of waves/link). Moreover, when the value of lifespan is set to 7 there is no loss in Macro-routing's efficiency, while its overhead (i.e. the number of waves) is considerably reduced.

The average communication overhead generated by waves, as depicted in Figure 5.13(d) is significant for lifespan values above 5, considering the relatively small size of the network. This is due to the large networks connectivity as can be seen in Figure 5.14, which shows the considerable variation in the number of waves generated on three different classes of topologies.

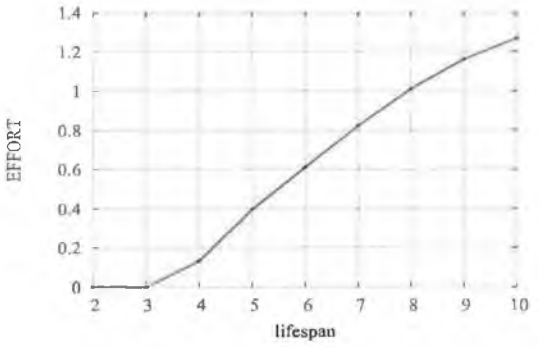
The number of waves generated by Macro-routing is overestimated in this results due to the assumption that all paths satisfy the constraints. Demanding constraints would significantly limit the number of compliant paths, and thus the wave population.



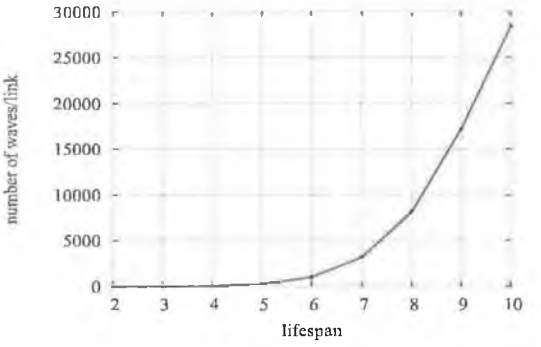
(a) Macro-routing's Mean *Efficiency*



(b) Mean proportion of *success* and *best* Macro-routing paths found



(c) Macro-routing's Mean *Effort*

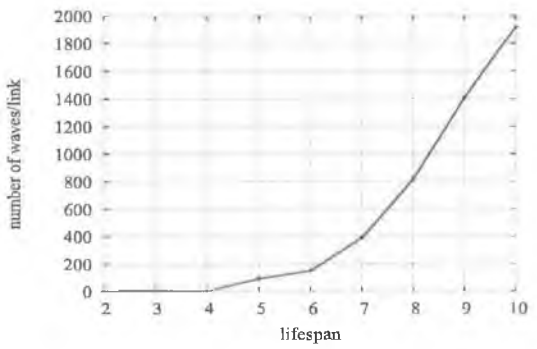


(d) Mean number of waves/link generated

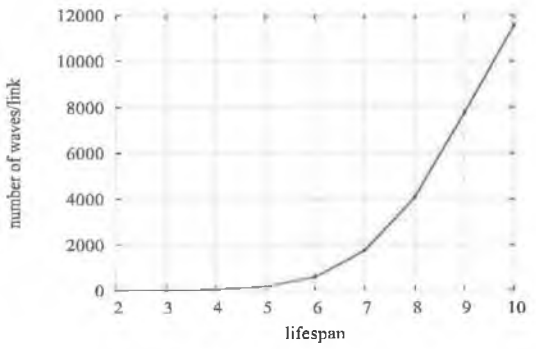
Figure 5.13: *Macro-routing's performance when* lifespan *is limited*

5.2 Summary

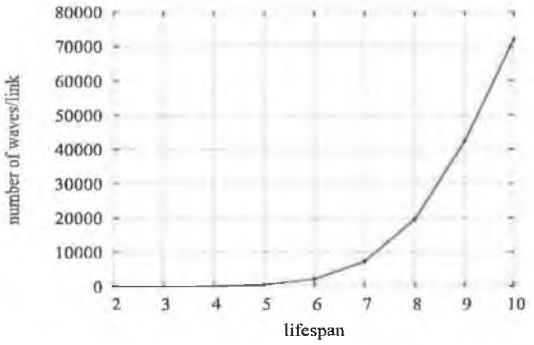
Deploying QoS routing strategies in large networks generates storage, computational and communication overhead. In this chapter a new routing solution was presented that addresses these scalability issues: a hierarchical routing protocol for MPLS networks, called *Macro-routing*, which also has signalling and resource reservation capabilities. By using mobile agents (called *waves*) for the routing process, Macro-routing distributes the computational and diminishes the storage overheads. The communication overhead generated by disseminating routing information does not exist in Macro-routing. That is because the routing information is consulted "in-situ". This also allows the most accurate aggregation technique, i.e. *Full-Mesh*, to be used. However, due to its exhaustive search for sub-paths, Macro-routing dispatches waves which might generate a considerable communication overhead. This overhead is reduced by two strategies: terminat-



(a) Mean number of waves/link generated on topologies with $c_d \in [3, 3.66]$



(b) Mean number of waves/link generated on topologies with $c_d \in [4, 4.5]$



(c) Mean number of waves/link generated on topologies with $c_d \in [5, 5.166]$

Figure 5.14: *Macro-routing's communications overhead on different topologies*

ing any wave that reaches a border node and imposing a *lifespan* limit on the wave population. Moreover, the size of one *wave* packet is significantly smaller (i.e., less than one KB) compared with the link-state packets used to distribute QoS routing information (see details about the PNNI packet sizes in [9]). Also, due to the dynamic nature of the routing information, such link-state packets need to be broadcast frequently. In contrast, waves are only dispatched as needed. Thus, it can be tentatively assumed that the communication overhead generated by the wave population is, at worse, no greater than the overhead generated by the link-state packets when centralised QoS routing mechanisms are used. This qualitative argument will need to be confirmed by quantitative studies, which is a topic for further research.

The tests performed considered the worst-case scenario where the maximum number of *waves* is produced, which happens when all available paths satisfy the

QoS requirements. In such cases simulations show that the number of *waves* is sensitive to the connectivity degree (i.e. the number of links) rather than to the number of nodes. Moreover, for highly connected networks, most long paths terminate in a cycle, and are thus useless for routing purposes. Therefore, the *lifespan* parameter is a desirable feature. Tests run on the two level hierarchical networks showed that there exists a threshold over which this parameter significantly reduces the communication overhead without significantly impairing protocol performance.

The benefits of using an agent-based routing protocol are most pronounced when there are multiple path constraints. This is the subject of the next chapter.

CHAPTER 6

Hierarchical QoS Routing with Multiple Path Constraints

"Routing subject to multiple path constraints (e.g., cost and delay constraints) is a desirable feature in today's integrated networks in spite of its intractability" Whay C. Lee [106]

QoS routing for finding feasible paths that satisfy simultaneously multiple constraints is called *multiple-constrained routing*.

6.1 Multi-constraint Macro-routing

The task of finding a path in the network satisfying multiple constraints is particularly challenging. This problem is even more complex in the case of hierarchical networks (as discussed in Section 3.3.3). The problem here is that since a logical link at one level of the hierarchy corresponds to P paths in the physical network, a complete characterisation of the logical link, where there are M constraints to be met, requires $M \times P$ parameter values to be recorded. To do so would negate the

benefits of topology aggregation that a hierarchical routing scheme brings, and so some means must be found to reduce the amount of data used to characterise the logical links.

In the case where $M = 1$ (the single constraint problem) this is straightforward. The lowest cost path among the set of P paths is used to represent the full set, since this is the "best" path. This solution is *not* available when $M > 1$, since identifying the "best" path is problematic. The optimal path may be a compromise between the various constraints, and it will not be apparent *a priori* which of the P paths this is.

The obvious solution is to record the metrics of a candidate set of best paths, in the hope that one of these will be the appropriate path. This causes two difficulties:

1. the amount of data used to describe a logical link , while reduced, is still considerable;

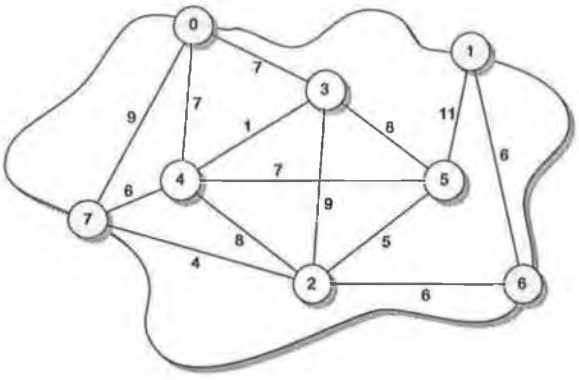
2. some mechanism is needed to record and select the candidate paths.

It is proposed here to address the first problem by extending *Macro-routing* to handle multiple constraints. The use of mobile agents to perform routing reduces the amount of data that needs to be propagated around the network, compared to link-state protocols, and the parallel action of multiple agents accelerates the process of route discovery. The second problem will be addressed by using a new method to represent the aggregated topology of a domain, called the *extended Full-Mesh*.

6.1.1 The *extended Full-Mesh* (EFM) aggregate representation

The *extended Full-Mesh* (EFM) aggregate representation extends the Full-Mesh (described in Section 3.2.1), by allowing not only one "best" path between any two border nodes but multiple satisfactory paths. The method shall be illustrated

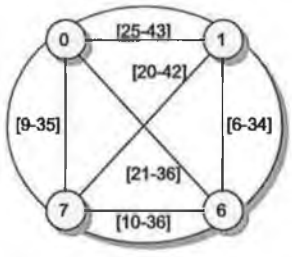
using an example network already considered in Figure 5.1. This is redrawn for convenience in Figure 6.1(a). This example features only a single constraint, for purposes of illustration. The *extended Full-Mesh* representation is found in two steps, illustrated in Figure 6.1(c) (and the associated table in Figure 6.1(b)) and Figure 6.1(e) (and the associated table in Figure 6.1(d)) respectively.



(a) Physical domain topology

from **node** to **node**	ordered path costs
from **0** to **1**	25, 26, 27, 31, 32, 33, 42, 43
from **1** to **6**	6, 22, 32, 34
from **6** to **7**	10, 20, 22, 24, 26, 36
from **7** to **0**	9, 13, **14**, **19**, 20, 21, 23, 24, 25, 28, 30, 34, 35
from **0** to **6**	21, 22, 23, 26, 27, 36
from **1** to **7**	20, 24, 26, 30, 32, 42

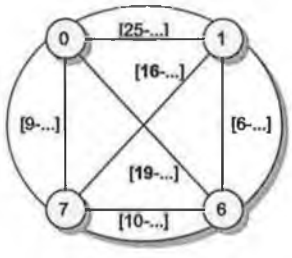
(b) ordered path costs for every path found between any pair of border nodes



(c) the intermediate *extended Full-Mesh*

from **node** to **node**	ordered path costs
from **0** to **1**	25, 26, 27, **28**, **29**, 31, 32, 33, **40**, 42, 43, ...
from **0** to **6**	**19**, 21, 22, 23, 26, 27, **29**, **31**, **32**, 33, 36, ...
from **0** to **7**	9, 13, 14, 19, 20, 21, 23, 24, 25, 28, 30, **31**, ...
from **1** to **6**	6, 22, 32, 34, **47**, ...
from **1** to **7**	**16**, 20, 24, 26, **28**, 30, 32, **34**, **35**, **36**, 42, ...
from **6** to **7**	10, 20, 22, 24, 26, **30**, **32**, 36, **38**, ...

(d) final ordered path costs for every path found between any pair of border nodes



(e) the *extended Full-Mesh*

Figure 6.1: *Building the* extended Full-Mesh *aggregate representation of a network in two steps*

The first step involves searching all the possible paths between any two border nodes and calculating their costs. Then, for each pair of border nodes the path costs are sorted from *best* to *worst* (that is, in increasing order for this example as it uses an additive metric). The results are presented in the table shown in Fig-

ure 6.1(b). For instance, there are eight possible paths from node 0 to node 1 with costs ranging from 25 to 43 and four possible paths from node 1 to node 6 with costs in the range from 6 to 34. These path costs are used in building the *extended Full-Mesh* aggregate representation by using intervals of metrics instead of single metric values (see Figure 6.1(c)). This is an intermediate result. The next step involves searching for border-to-border paths that traverse other border nodes, as explained in Section 5.1.1. Their resulting path costs are also considered for building the EFM, and therefore they are inserted (the bold entries Figure 6.1(d)) in the vector of path costs obtained in the first step. The final EFM representation, shown in Figure 6.1(e), does not specify the upper bounds of the path costs vectors/intervals. These bounds can be determined by a variety of methods, as will be discussed in Section 6.1.3.

Next, some abstract concepts, used to describe the EFM aggregate representation are introduced. These are the *EFM interval*, the *cost matrix*, the *metric vector*, the *path vector* and the *EFM feasible path*.

Definition 1 ***EFM interval***

Let

- *g be a graph describing a flat topology, with $g = \{V, L, B\}$, where V is the set of nodes, L is the set of links and B is the set of border nodes, so that $B \subset V$.*
- *$P_i(u, v)$ be the i^{th} path between border nodes u and v.*
- *M be the number of metrics associated with each link and/or node from the graph g. These metrics are considered while computing the paths between any pair (u, v) of border nodes, $u, v \in B$.*
- *$n_{u,v}$ be the number of paths connecting any pair of border nodes $u, v \in B$.*
- *$\mathbf{C}$ be the* cost matrix *for the paths from u to v, where $C_{i,j}^{u,v}$ is the value of metric j on path $P_i(u, v)$. This is a matrix of dimension $n_{u,v} \times M$. The column vector $\xi_j^{u,v}$ is*

the j-th column of $\mathbf{C}^{u,v}$ and is termed a metric vector *as it lists the value of metric j for all possible paths from u to v. The row vector $\zeta_i^{u,v}$ is the i-th row of $\mathbf{C}^{u,v}$ and is termed a* path vector *as it lists the value of all metrics for path $P_i(u,v)$.*

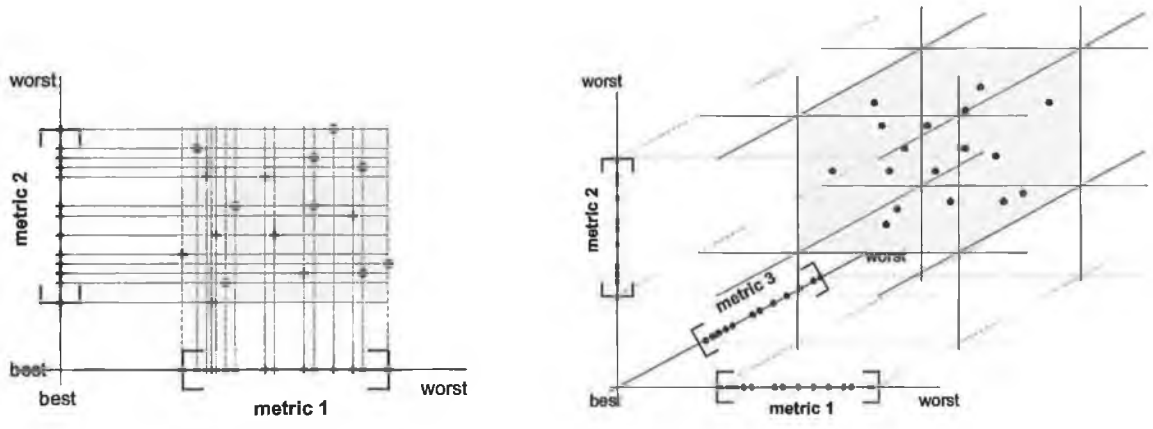
Definition 2 ***Feasible paths***

Consider a cost matrix $\mathbf{C}^{u,v}$ and a constraint vector $\Delta^{u,v}$ of dimension M which lists the constraint values for each metric. If for some value of i, and $0 < j \leq M$

$$\begin{aligned} & C_{i,j}^{u,v} \leq \Delta_j^{u,v}, \quad \textit{where the } j\textit{-th metric is additive, multiplicative or convex} \\ \textit{or} \quad & C_{i,j}^{u,v} \geq \Delta_j^{u,v}, \quad \textit{where the } j\textit{-th metric is concave} \end{aligned} \tag{6.1}$$

then $P_i(u,v)$ is a feasible path.

The path vector defines a point in an M-dimensional space. The cost matrix defines $n_{u,v}$ such points as illustrated in Figure 6.2. The shaded regions denote an M-dimensional "volume" bounded by the minimum and maximum values of each metric. Such a region will be referred to as an *EFM interval*. The EFM interval and the cost matrix together constitute the EFM representation of the network domain.



(a) for two metrics (b) for three metrics

Figure 6.2: *Visualization of the* cost matrix

If all paths satisfying the QoS constraint are retained in the EFM representation, there would be no aggregation at all. However, this would make the size

of a *wave* excessive, especially when there are multiple constraints (since a wave has to carry $M \times k$ values for M constraints and k path vectors). Therefore, a threshold, T, is introduced to limit the number of path vectors in the recorded cost matrix. Methods for reducing the number of path vectors and the effect of imposing such a threshold will be discussed in Section 6.1.3.

6.1.2 Macro-routing with the *extended Full-Mesh* aggregation

The hierarchical routing protocol, *Macro-routing*, can be extended to find multi-constraint paths using the EFM aggregate representation. To explain how EFM works in a hierarchical network, consider a very simple hierarchical example of three domains and two hierarchical levels. This example is depicted on the left hand side of Figure 6.3. The path search is performed using two metrics; one is additive and the other is concave. The same algorithm can be applied should there be only additive or multiplicative metrics, without modification. The elements of the metric vector corresponding to the additive metric are enclosed in square brackets [ ], and this will be referred to as the *additive metric vector*, whilst the elements of the metric vector corresponding to the concave metric are enclosed in chain brackets {} - this is called the *concave metric vector*.

I define "&" to be the operator for the concave metric and "+" to be the operator for the additive metric. These operators are applied to metric vectors in accordance with the following rules, where x, y, z and t are scalar metric values.

$$x\&y = min(x, y),$$

$$x\&y\&z = min(x, y, z) \text{ and}$$

$$\{x, y\}\&z = \{min(x, z), min(y, z)\}$$

$$\{x, y\}\&\{z, t\} = \{min(x, z), min(y, z), min(x, t), min(y, t)\}$$

$$[x, y] + z = [(x + z), (y + z)] \text{ and}$$

$$[x, y] + [z, t] = [(x + z), (y + z), (x + t), (y + t)]$$

The algorithm starts from the *source* node a by initiating, as in *Macro-routing*, the search for all the participant domains. Then, within each participant domain at *Level 0* (all three domains in this example), *Macro-routing* initiates a *wave*-based search for all possible paths between all border[1] nodes. They will be used to build the aggregate representations in *Level 1* as depicted in Figure 6.3.

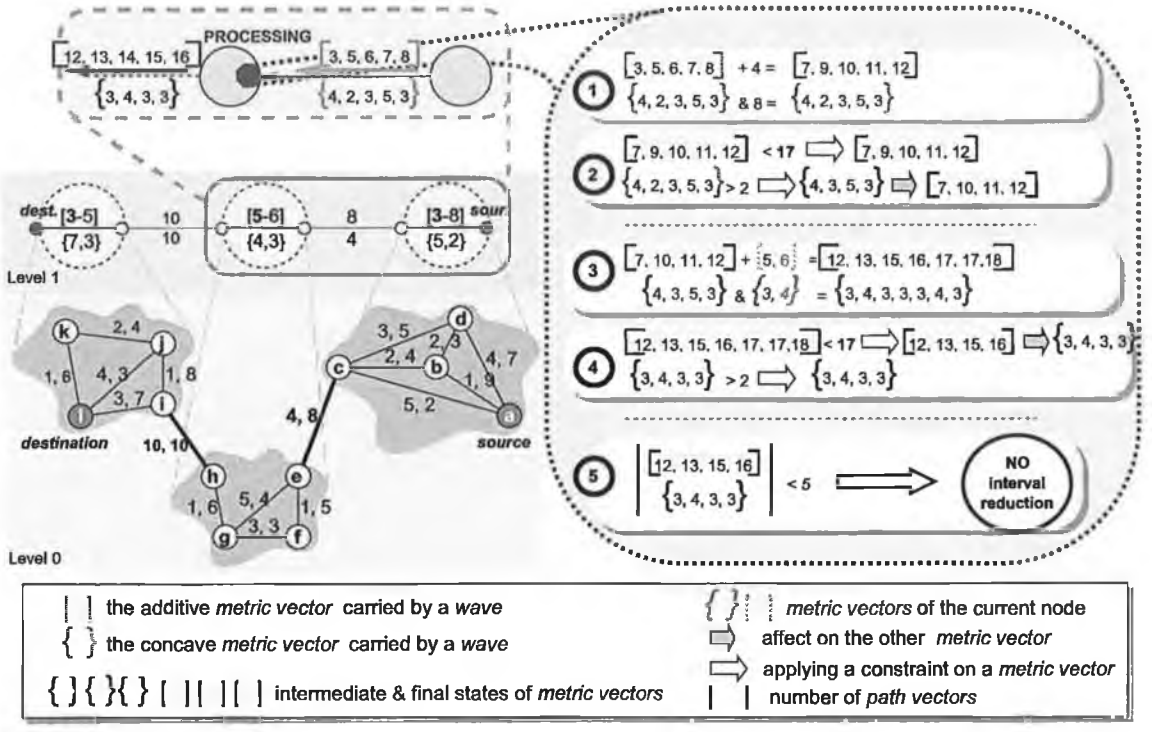


Figure 6.3: *Example for two metrics (additive and concave)*

There are within *Level 0* three participant domains: the source domain, one transit domain and the destination domain.

In the source domain there are two border nodes: a (source) and c. Thus, the EFM aggregate representation for the source domain will have only a single link in this case. The computations for determining its cost matrix start by searching all possible paths between the two border nodes a and c.

$$a \rightarrow c = 5, 2$$

$$a \rightarrow b \rightarrow c = (1 + 2), (9\&4) = 3, 4$$

$$a \rightarrow b \rightarrow d \rightarrow c = (1 + 2 + 3), (9\&3\&5) = 6, 3$$

[1]Source and destination nodes are also considered to be border nodes (see the description of Macro-routing in Section 5.1.1).

$$a \rightarrow d \rightarrow c = (4+3), (7\&5) = 7, 5$$

$$a \rightarrow d \rightarrow b \rightarrow c = (4+2+2), (7\&3\&4) = 8, 3$$

Therefore, the cost matrix for the source domain is

$$\begin{pmatrix} 3 & 5 & 6 & 7 & 8 \\ 4 & 2 & 3 & 5 & 3 \end{pmatrix}^T$$

In the transit domain are two border nodes: e and h. The possible paths between them are:

$$e \rightarrow g \rightarrow h = (5+1), (4\&6) = 6, 4$$

$$e \rightarrow f \rightarrow g \rightarrow h = (1+3+1), (5\&3\&6) = 5, 3$$

Thus, the cost matrix for the transit domain is

$$\begin{pmatrix} 5 & 3 \\ 6 & 4 \end{pmatrix}$$

At the next level of the hierarchy, i.e. *Level 1*, there is only one domain containing three nodes, all representing the aggregate format of the domains from *Level 0*. After all these aggregated representations have been composed (i.e. once *Level 1* is complete), the managing node of the *Level 1* domain initiates the search for all paths traversing this domain. A *wave* is released from the source node to find all possible paths to the destination node. When leaving the source node, the *wave* will carry with it the cost matrix for traversing the source node, i.e. the two metric vectors $[3, 5, 6, 7, 8]$ and $\{4, 2, 3, 5, 3\}$. At the next node, this information (carried by the *wave*) will be used for the "partial" path processing phase. This process is presented in the right hand side of Figure 6.3 and proceeds in 5 steps:

1. **"update" the cost matrix to take account of the metrics of the link just passed (from the source node to the transit node).** This involves modifying the two metric vectors as follows:

   $$[3, 5, 6, 7, 8] + 4 = [7, 9, 10, 11, 12] \text{ and}$$

$$\{4,2,3,5,3\}\&8 = \{4,2,3,5,3\}$$

The resulting cost matrix is:

$$\begin{pmatrix} 7 & 9 & 10 & 11 & 12 \\ 4 & 2 & 3 & 5 & 3 \end{pmatrix}^T$$

2. **apply the constraint tests for each metric vector**. On the right hand side of Figure 6.3, the second element from the concave metric vector was eliminated as it did not pass the constraint test. Consequently, the element with the same index had to be eliminated from the additive metric vector as well.

3. **"update" the cost matrix resulting from phases 1 and 2 to take account of the cost matrix for traversing the current node**. This involves modifying the two metric vectors as follows:

$$[7,10,11,12] + [5,6] = [12,15,16,17,13,16,17,18] \text{ and}$$
$$\{4,3,5,3\}\&\{3,4\} = \{3,3,3,3,4,3,4,3\}.$$

After eliminating any identical path vectors (i.e. $(16,3)$ in this case) and sorting them in order of the additive metric, the cost matrix becomes:

$$\begin{pmatrix} 12 & 13 & 15 & 16 & 17 & 17 & 18 \\ 3 & 4 & 3 & 3 & 3 & 4 & 3 \end{pmatrix}^T$$

If there are more nodal traversing possibilities, there will be a separate computation for each traversal. The incoming wave will duplicate with the number of outgoing links and each wave will carry its corresponding cost matrix.

4. **apply the constraint tests once more**. Here, the last three elements within the additive metric vector are eliminated. This will have the same effect on

the concave metric vector as well, resulting in

$$\begin{pmatrix} 12 & 13 & 15 & 16 \\ 3 & 4 & 3 & 3 \end{pmatrix}^T$$

as the final cost matrix.

5. **apply the *threshold* test**. If the number of path vectors within the EFM interval exceeds the threshold, some path vectors have to be eliminated using one of the techniques presented in Section 6.1.3.

6.1.3 Decreasing the number of path vectors

The number of *path vectors* may increase to a value which can negatively influence the path computation process. That is because more candidate paths require more computations, and the size of a *wave* may increase so that the algorithm generates too much control traffic. The information a *wave* has to carry, in addition to that of a standard *Macro-routing wave*, is $M \times k \times c$ bytes where M is the number of metrics, k is the number of *path vectors*, and c is the number of bytes occupied by a metric value.

To control the *wave* size the number of *path vectors* can be limited by a threshold T. Each path vector relates to a candidate path, hereafter referred to as an EFM path. The main factors that govern the number of EFM paths are:

a) The topology of the network. The number of EFM paths may increase with the network connectivity. This, is true especially when there are enough resources to satisfy the constraints.

b) The number of border nodes per domain. This does not directly influence the number of EFM paths when building the EFM aggregate representation at a single level. However, at the next level of hierarchy the number of border nodes becomes the node degree of the aggregated domain. The arguments of condition **a)** then apply.

c) The number and severity of the constraints. One expects, in general, to find few EFM paths where the set of constraints is large and/or demanding.

d) The availability of resources.

It may be concluded that sparse networks, a large or demanding set of constraints, and a poverty of resources are all contexts in which a threshold may be unnecessary since the number of EFM paths expected to be found is low. In other contexts, for the algorithm to be efficient a threshold value is required to limit the number of EFM paths, and thus the number of path vectors, evaluated by *waves*. This however is not a trivial problem as it is not obvious *a priori* which path vector to discard when there are too many. In the remainder of this chapter possible selection criteria are investigated in order to determine their effectiveness.

Two classes of techniques are presented below for reducing the number of considered path vectors. One (*truncation*) is a greedy method through which the best resources are occupied first; the other method (*random*) tries to keep a larger spectrum of alternative path vectors.

In describing these techniques below, the following special[2] cases are considered:

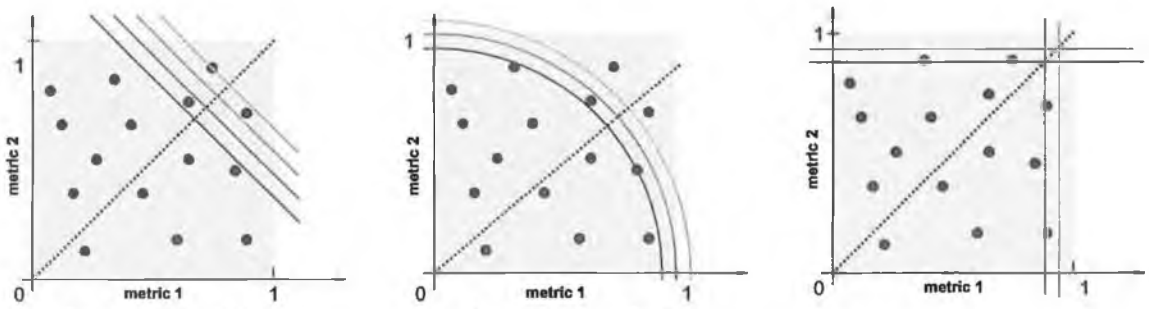
- The methods are applied to a two-dimensional EFM interval. (although they are applicable on any n-dimensional EFM interval with $n \geq 2$).
- All metrics are normalised to lie in the range from 0 (*best*) to 1 (*worst*).

6.1.3.1 Truncation of the EFM interval

Here paths starting from the *"worst"* down until only T path vectors remain are eliminated.

A difficulty is that it is not possible to clearly distinguish the *worst* multi-constraint paths. Three possible methods to select these paths are considered below. They are schematically represented in Figure 6.4.

[2]The restriction to these cases is purely for clarity of explanation.



(a) *normal* method (b) *radius* method (c) *simple* method

Figure 6.4: *Methods for truncating a two-dimensional EFM interval*

The first method, depicted in Figure 6.4(a) progressively eliminates the path vectors that are traversed by the line $f(x,t) = 2t - x$, which lies perpendicular to the diagonal from the *worst-worst* (1,1) corner down to the *best-best* (0,0), as the parameter t decreases from 1 towards 0, until (at some value $t = t_0$) (at most) T path vectors remain. The path vectors that satisfy the inequality $f(x,t_0) > y$ are the selected ones[3].

Another method, presented in Figure 6.4(b), is similar but the delimitation between the selected and the unselected paths is not a line but an arc. This arc is part of the circle centred at $x = 0, y = 0$. The circle's radius, r, varies from $\sqrt{2}$ down to 0 until no more than T path vectors remain. When the appropriate radius $r = r_0$ is found, all path vectors inside the arc are selected. Thus, an path vector is retained if it satisfies the inequality $\sqrt{x^2 + y^2} < r_0$.

Figure 6.4(c) depicts the third truncation method, where the *worst* path with respect to a single metric is eliminated. The designation of the considered metric is done either starting with a randomly chosen metric and then choosing other metrics for the next path elimination in a round-robin fashion or based on a *priority* metric attribute. The chosen attribute may change with every path elimination allowing another metric with a higher priority to be the next decision factor. If only one metric is used in the selection process (without alternation), this method is referred to as *truncate single*.

[3] x and y are the two metrics in this two-dimensional example.

The first two models proposed for truncation techniques resemble solutions proposed for solving the multi-constraint path problems[4] (see Section 3.3.2.3). The difference is that the latter were used to select final paths, while I use them to build an aggregate representation for hierarchical routing.

6.1.3.2 Random selection

Another method for reducing the number of EFM paths is to reduce the density of path vectors within the EFM interval though random selection. Thus, T EFM paths are picked at random from the available paths.

Another "random" path selection that can be implemented in a real system is to consider the first T paths reported back to the managing node. This mechanism is inherently biased and can also be considered a truncation selection because it tends to favor paths with low delay and few hops.

6.1.4 Simulation results

The simulation environment used for Macro-routing's tests, as presented in Section 5.1.3.2, was extended for multiple metrics/constraints. Algorithms for the path selection mechanisms proposed in Section 6.1.3 were also implemented. In the remainder of this chapter the implemented path selection techniques will be referred to by the acronyms presented in Table 6.1.

Table 6.1: *The acronyms used for the path selection techniques used for simulation tests*

EFM path selection technique	acronym
TRUNCATE SINGLE using metric j	$TS[j]$
TRUNCATE NORMAL	TN
TRUNCATE RADIUS	TR
QUASI RANDOM	QR

The quasi-random method implemented selects the T paths for the EFM aggregate representation from the list of paths generated by the backtracking ap-

[4]The *normal* method (Figure 6.4(a)) uses a similar linear function to the Lagrangian relaxation techniques (e.g. Jaffe's approximation [84]), while the *radius* method (Figure 6.4(b)) uses a nonlinear function that resembles the one used in the TAMCRA algorithm [55].

plication described in the second part of Section 5.1.3.2, preserving the order in which the paths are generated by this application. The selected paths are distributed across the list, to minimise any possible correlations between them. Hence the i-th path selected is that in position $l(i)$ in the list, where

$$l(i) = i \cdot \left\lfloor \frac{P}{T} \right\rfloor,$$

with $1 \leq i \leq T$ and P is the total number of paths selected.

30 sets of tests were run on different hierarchical topologies with two hierarchical levels (see Table 6.2).

Table 6.2: *Topologies used in the simulation tests*

Topology /nodes	*intra+inter* domain **links**	**connectivity** $2L/N$
4 domains × 4 = **16**	20+4=**24**	3.00
9 domains × 9 = **81**	110+14=**124**	3.06
10 domains × 10 = **100**	137+12=**149**	2.98
12 domains × 12 = **144**	208+15=**223**	3.09
13 domains × 13 = **169**	265+17=**282**	3.33
15 domains × 15 = **225**	343+25=**368**	3.27
20 domains × 20 = **400**	565+35=**600**	3.00

Two additive metrics were used, *administrative cost* and *delay*. For each individual test, the values for the two metrics were randomly chosen, for every physical link within the hierarchical topology, from a given interval (i.e. *administrative cost* $\in [1, 15]$ and *delay* $\in [2ms, 45ms]$).

For all tests, the number of EFM paths to be selected was limited to T (e.g., $T = 5$ for the first set of tests). This means that the EFM aggregate representation will have $\Omega_{u,v}$ path vectors for every pair of border nodes $u, v \in B$, where:

$$\Omega_{u,v} = \min\{T, n_{u,v}\} \tag{6.2}$$

Each test performed on one topology resulted in different values for the best or the worst costs of the designated metrics. In order to compare them, these

results were normalised so that 0 represents the best value and 1 represents the worst value of each metric.

The results on all topologies were similar. Figure 6.5 shows the results for tests run on the 20×20 node topology.

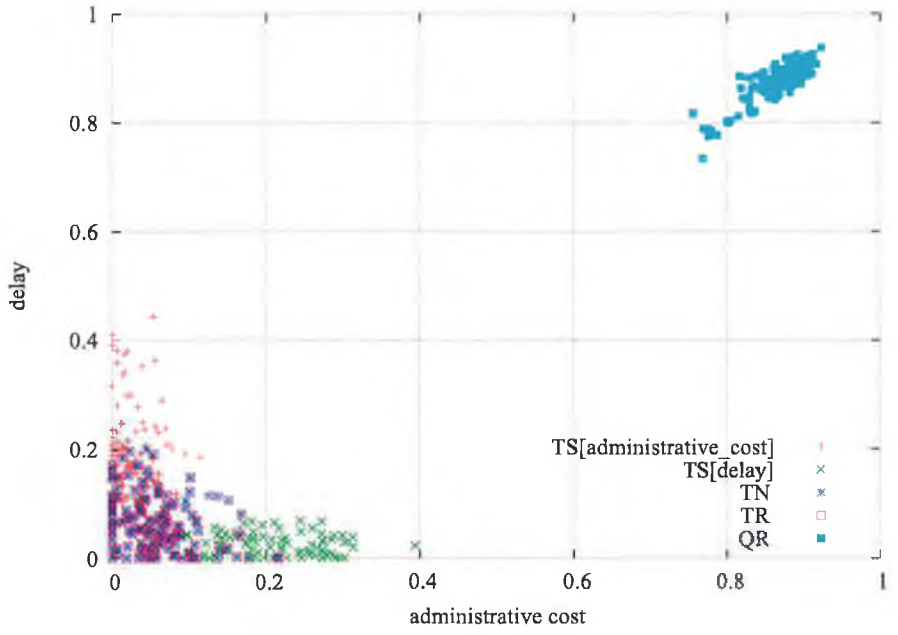


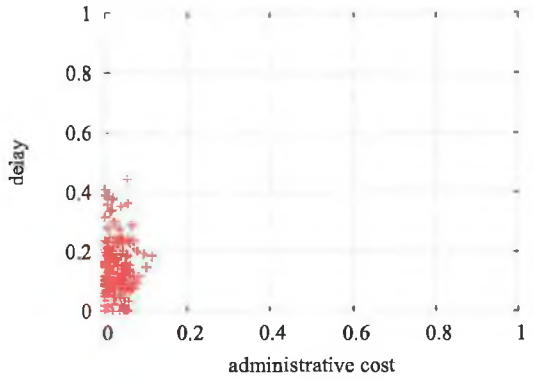
Figure 6.5: *Comparing four of the EFM methods on a 20×20 node topology*

A separate view on the three methods (i.e. $TS[administrative\_cost]$, $TS[delay]$, and QR) is presented in Figure 6.6. Similar results were obtained on other topologies (e.g., the topologies described in Table 6.2).

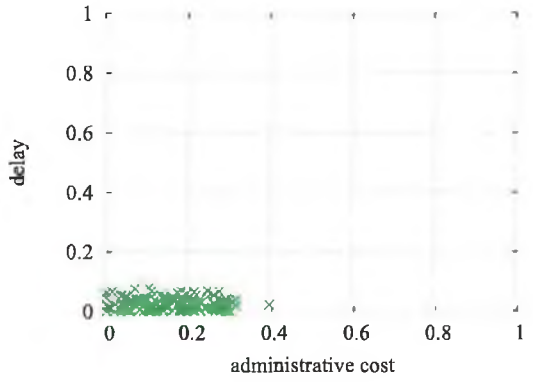
These tests showed that:

- $TS[j]$ leads to better results when considering only metric j, but may produce worse results overall compared with other approaches (e.g., TN and TR);
- There are a few cases when the QR approach finds the best path costs. However, such cases are rare and form no discernible pattern.

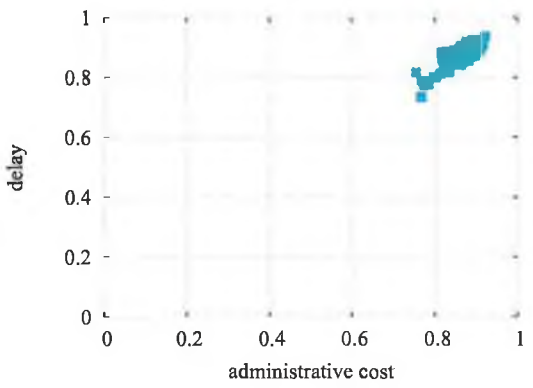
These patterns are not observed on the 4×4 node topology, as is shown in Figure 6.7. It is conjectured that too few paths are available in such a small network for the method of path selection to be significant.



(a) TRUNCATE SINGLE using administrative cost



(b) TRUNCATE SINGLE using delay



(c) QUASI RANDOM

Figure 6.6: *Detailed comparison of $TS[administrativecost]$, $TS[delay]$ and QR methods on a 20×20 node topology*

The difference between the TRUNCATE NORMAL and TRUNCATE RADIUS methods on the tests performed on the 20×20 node topology are insignificant, as verified by the detailed/enlaged results in Figure 6.8. This suggests that any truncation rule which is fair (i.e., which favours no metric in its selections) will offer similar performance.

To establish whether this was true in general, the next set of tests were performed on different topologies with the same number of nodes (i.e. $12 \times 12 = 144$) but with different connectivity degrees (i.e. $c_d \in [4, 4.5]$, and $c_d \in [5, 5.3]$). Then, the upper and lower bounds for each metric were determined from the EFM interval.

This will be denoted by the (self-explanatory) names min_{adm_cost}, MAX_{adm_cost}, min_{delay}, and MAX_{delay}.

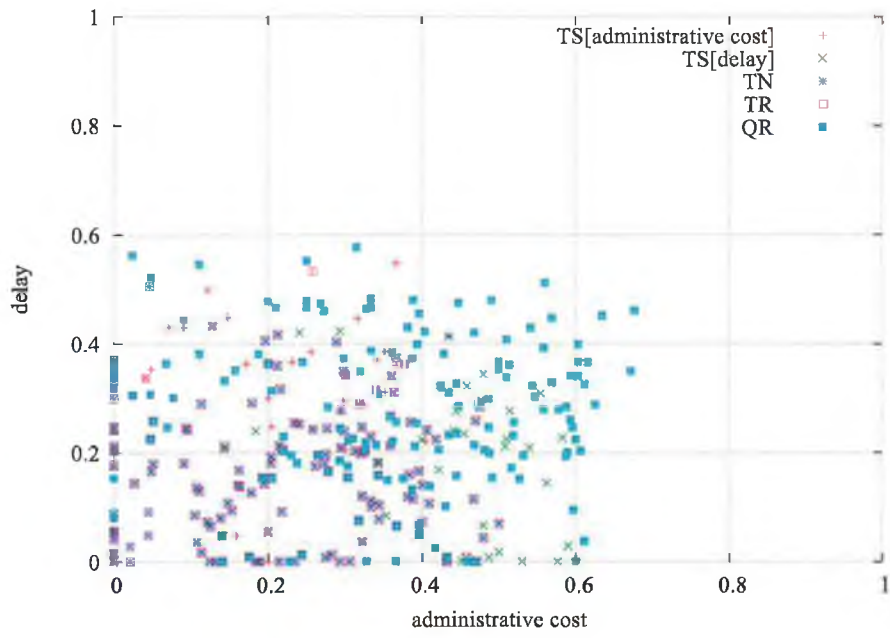
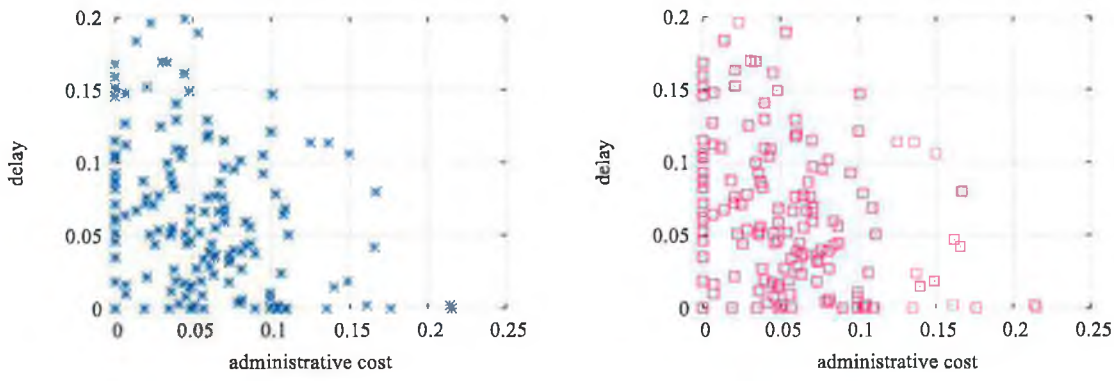


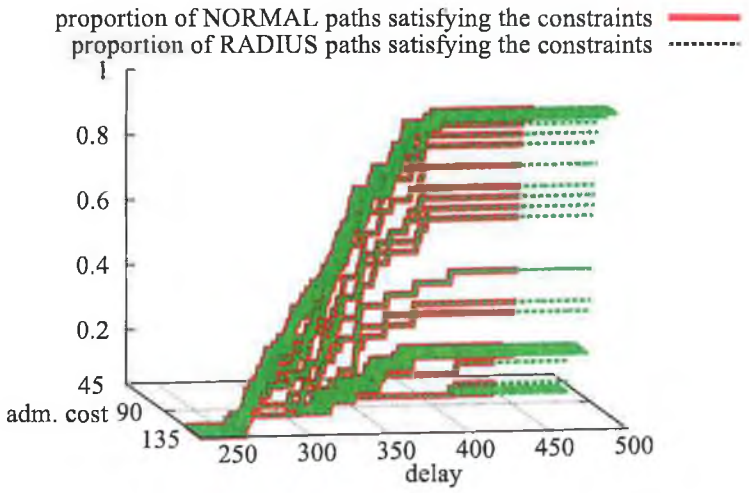
Figure 6.7: *A comparison of four of the EFM methods on a 4×4 node topology*



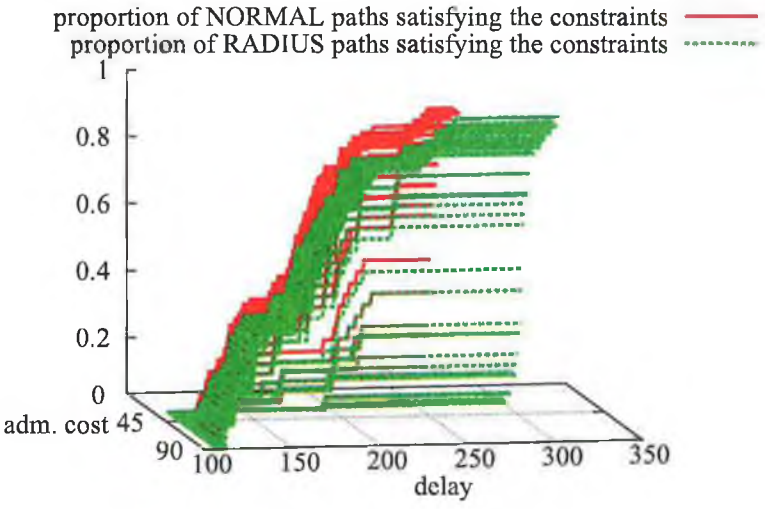
(a) TRUNCATE NORMAL (b) TRUNCATE RADIUS

Figure 6.8: *Detailed comparison of TN and TR methods on a 20×20 node topology*

For each metric interval obtained (i.e., $[min_{adm\_cost}, MAX_{adm\_cost}]$ and $[min_{delay}, MAX_{delay}]$), the proportion of paths satisfying constraints lying within these intervals were plotted. Figure 6.9 shows that differences between the two path vector reduction methods (i.e. TRUNCATE NORMAL and TRUNCATE RADIUS) occur only in topologies with higher connectivity degree (i.e. $c_d \in [5, 5.3]$), where the path vectors found by the NORMAL method are satisfying stricter constraints in a larger proportion than RADIUS paths. Moreover, the TRUNCATE NORMAL approach presents in both Figures 6.9(a) and 6.9(b) smaller min-MAX intervals.



(a) $c_d \in [4, 4.5]$



(b) $c_d \in [5, 5.3]$

Figure 6.9: *Comparing TN and TR methods on 12×12 node topologies*

Next, the macro-routing approach was compared to an existing hierarchical routing algorithm. The macro-routing algorithm was *Multi-constraint Macro-routing* with EFM aggregation (with $T = 4$) and the TRUNCATE NORMAL path selection mechanism. This was compared to a scheme with Full-Mesh aggregation and the *Lagrangian-based linear composition*[5] for selecting the representative path for each port-to-port connection used for building the Full-Mesh aggregate representation. All metrics were considered equally weighted with $d_i = \frac{1}{n}$, where n is the number of metrics. Thus, the *Lagrangian-based linear composition* consid-

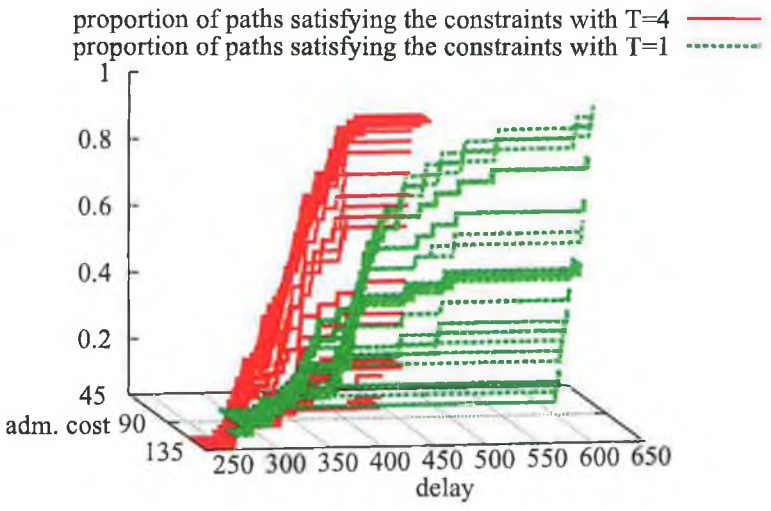
[5]See Section 3.3.2.

ered was:

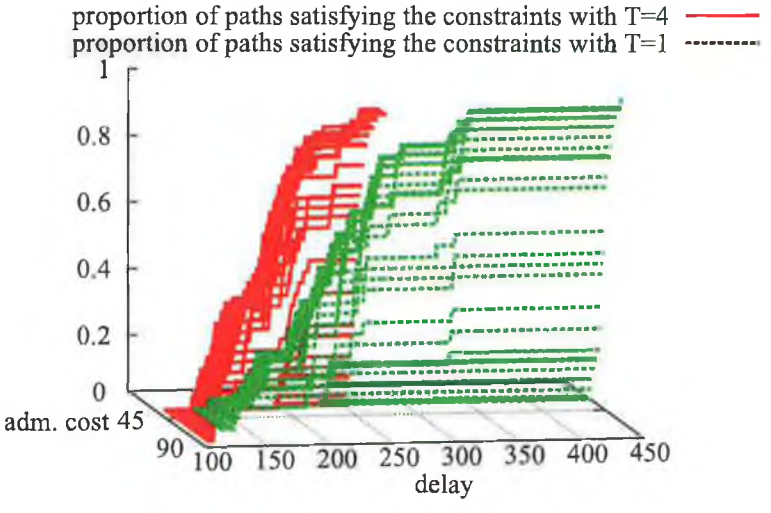
$$W_i = \frac{\sum_{k=1}^{n} c_i^k}{n} \tag{6.3}$$

The selection mechanism based on (6.3), however, leads to the same results as would *Multi-constraint Macro-routing* when the TRUNCATE NORMAL path selection mechanism is applied with $T = 1$. Thus, the two different protocol implementations will be referred to as $T = 4$ and $T = 1$.

The results depicted in Figure 6.10 show that $T = 4$ generates much better results that $T = 1$.



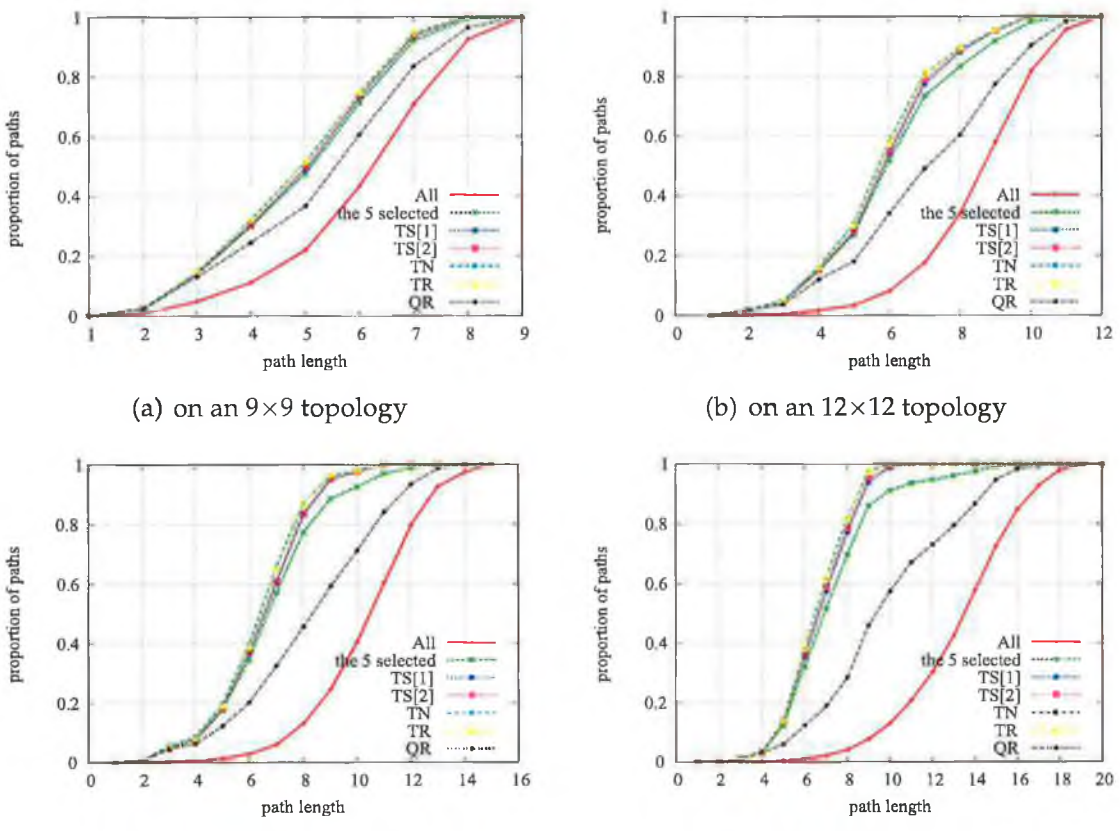
(a) $c_d \in [4, 4.5]$



(b) $c_d \in [5, 5.3]$

Figure 6.10: *Compare TN with $T = 4$ and $T = 1$ on 12×12 node topologies*

The next set of results show the distribution of the length (i.e. number of nodes visited) of all paths found within each domain of the hierarchy. The results depicted in Figure 6.11, show the proportion of paths whose length does not exceed l, as a function of l. These distributions are shown for the general population of paths in the network, as well as for the set of paths found using the various selection schemes. It may be observed that the paths selected using one of the implemented methods tend to be shorter compared with all paths found. For instance, Figure 6.11(d) shows that in a 20 node topology, non of the paths selected by any of the *truncate* methods is longer than 12, while the *QUASI RANDOM* method finds paths with the maximum length of 18 nodes.



(a) on an 9×9 topology

(b) on an 12×12 topology

(c) on an 15×15 topology

(d) on an 20×20 topology

Figure 6.11: *The convergence of all and the 5th selected paths in terms of lifespan*

This results provide further evidence that the *lifespan* parameter presented in Section 5.1 restricts the *wave* population expansion without significantly impairing the performance of the *Multi-constrained Macro-routing* protocol.

6.2 Summary

This chapter addresses the hierarchical multi-constrained problem. Thus, the *Extended Full-Mesh (EFM)* aggregate representation was proposed, as a compromise between the Full-Mesh aggregation and no aggregation. The Full-Mesh representation works well for finding hierarchical paths based on a single metric, as presented in Chapter 5. However, it does not guarantee to find a hierarchical path based on multiple metrics, even if such a path exists. That is because it is very difficult to designate a single "best" multi-constrained path for every pair of border nodes in order to build the Full-Mesh aggregate representation. EFM allows more paths to be considered, and thus increasing the chances of finding a viable path. A disadvantage of the EFM aggregation when used with the *Macro-routing* protocol is that it can potentially generate too much routing traffic. Thus, the size of an EFM representation has to be limited. A number of techniques for limiting the EFM representation were presented and discussed in this chapter. The tests conducted showed that two of these techniques, i.e. *truncate normal* and *truncate radius* obtained the best results. Moreover, for highly connected networks *truncate normal* obtained better results than *truncate radius*. It was also shown that the EFM representation obtained better results than Full-Mesh.

The results, concerning the length of feasible paths, obtained from this new set of tests provided further evidence that the use of the *lifespan* parameter will not significantly affect the performance of the protocol, while reducing the communication overhead.

A final observation is that the amount of overhead generated by the Macro-routing protocol with EFM is, in principle, self-limiting. A large number of *wave* packets will be generated only when a range of routes meet the multiple constraints. However, in most cases, these constraints will include bandwidth, so that a large population of *waves* will be generated only in network conditions where capacity is available to transport them.

CHAPTER 7

Conclusions

Traditional routing techniques are no longer adequate for newly emerging network applications. Although QoS routing can satisfy the requirements of such applications, it is not a mature solution yet and therefore, ongoing research is required on efficient QoS routing strategies.

Most QoS routing research proposals consider computing the paths using source routing and global state information, and thus they introduce considerable computational, storage and communication overhead. The solutions advocated here for overcoming the overhead induced by traditional QoS routing mechanism are hierarchical aggregation and distributed routing using one of two modern approaches to network software, i.e. mobile agents and/or active networks. The main advantage of using these two technologies is that by dispatching mobile code the routing information can be consulted *in-situ*. This means that state information need only be maintained locally and that the most accurate aggregation technique can be used, i.e. the Full-Mesh.

7.1 Contributions

This thesis addressed the question whether mobile code offers advantages in solving the QoS routing problem. The two paradigms considered were *active networks* and *mobile agents*. The problem domain under consideration is that of MPLS networks, and thus the first concern was whether such networks can support these paradigms. Therefore, a Linux proof of concept was implemented. It shows that the active network paradigm, normally considered in the context of IP networks, is also applicable in MPLS. A number of applications of active networks in MPLS networks were also described, to demonstrate that their deployment (and thus that of an active network architecture) in an MPLS context would be worthwhile. Mobile agents are not tied to a specific networking protocol. This gives them an advantage over active networks, since their deployment in an MPLS cloud requires no special code to be written. Mobile agents were identified as the preferred paradigm since their search capabilities make them a more powerful tool in implementing QoS routing.

Mobile agents called *waves* were used to implement a new scalable hierarchical QoS routing protocol named *Macro-routing* which is also capable of efficient resource reservation and (label switched) path setup. The Wave architecture itself was developed elsewhere [139]. Compared with traditional QoS routing schemes which use source routing and global state, Macro-routing generates less computational and storage overhead as the routing information is consulted *in-situ* in a distributed manner by the *waves*. The performance offered by Macro-routing which finds all feasible paths simultaneously amongst which is the optimal one (where it exists), might come at the cost of generating too much communication overhead. The tests performed here proved that the expansion of the wave population, which is the source of the communication overhead, can be restricted without significantly impairing Macro-routing's performance. Another strength of Macro-routing is that the distributed route computation using local state al-

lows the use of the most accurate aggregation method, i.e. *Full-Mesh*. However, when processing multiple QoS constraints, even the Full-Mesh representation is deficient as it is very difficult or impossible to designate a single "best" path between any two border nodes in order to build the aggregate representation. Thus, a new aggregation technique has been proposed named *Extended Full-Mesh (EFM)* which is a compromise between Full-Mesh aggregation and no aggregation. This new technique allows more than one border node - to - border node links to be considered (if they exist). The tests conducted here show that this representation offers better results than Full-Mesh. A disadvantage of the EFM aggregation technique when used with the *Macro-routing* protocol is that it can potentially generate excessive routing traffic. Thus, the size of an EFM representation has to be limited. A number of techniques for limiting the EFM representation were presented and discussed in this thesis. The tests conducted showed that two of these techniques, i.e. *truncate normal* and *truncate radius*, obtained the best results. Moreover, for highly connected networks *truncate normal* obtained better results than *truncate radius*. The final observation resulted from tests was that the amount of overhead generated by the Macro-routing protocol with EFM is, in principle, self-limiting. A large number of *wave* packets will be generated only when a range of routes meet the multiple constraints. However, in most cases, these constraints will include bandwidth, so that a large population of *waves* will be generated only in network conditions where capacity is available to transport them.

7.2 Future work

My experience and results of the research conducted by others support the idea that mobile agents are a technology that will become very useful, and perhaps even critical, in many areas of distributed system such as network routing. Recent developments in the use of the Internet underline the importance of agents in

information-intensive applications. Thus, my future focus will be on exploiting the work presented here in two main directions described as follows.

The Macro-routing protocol, as developed during the programme of research documented here, is very much a prototype. An obvious next step is to augment the current implementation to make it deployable in the field. This would allow its scaling properties to be quantitatively evaluate and compared to reval link-state protocol such as PNNI. This would require the existing interface between the routers for routing table maintenance and the routing protocol in soft MPLS routers to be reverse-engineered. Deployment in commercial routers would require a measure of vendor support. Another issue to investigate is the number of paths to be stored in the EFM representation, and this will be undertaken in conjunction with a study of a wider range of path selection mechanisms. Finally the tradeoff between domain size and the number of hierarchical levels in a large-scale deployment merits investigation.

A second area of investigation is to study the relevance of Macro-routing in contexts other than MPLS in the Internet core. An area of particular interest is its suitability for deployment in wireless network. The difference is at the link layer and below, plus the possibility of node mobility, would provide new challenges for the protocol. Issues to consider would be whether the broadcast features of wireless networks could be used to accelerate wave deployment, whether the mobile agents could support novel forms of handover, and, in general, whether mobile code is a natural fit to the routing problem in mobile networks.

7.3 Concluding remarks

The deployment and development of efficient QoS routing strategies, although very difficult in the current Internet, is essential to provide reliable transport for critical future traffic (e.g., multimedia, real-time). In this thesis I have presented specific QoS issues and current solutions to address such issues. I also proposed

new and more efficient QoS routing, resource reservation and path setup solutions which use modern techniques (i.e. active networks and mobile agents).

These solutions, if deployed in a live network, should assist the migration from a best-effort service model to a future Internet that robustly supports real-time applications and others with stringent QoS requirements.

APPENDIX A

The Wave Concept

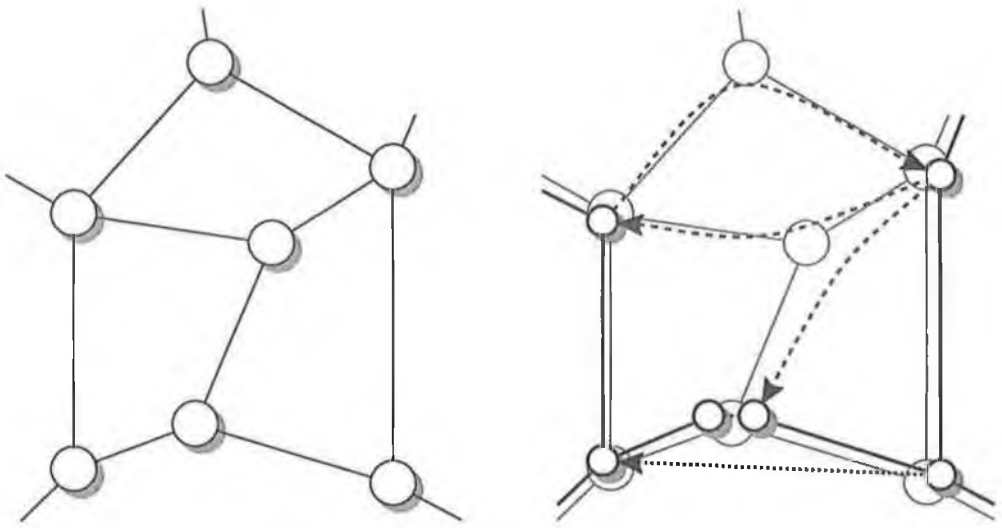
The Wave concept is best described in Sapaty's[1] publications [139–146], and Borst's[2] publications [29–31]. This appendix is a mere schematically presentation of basic elements.

A.1 Knowledge Networks

The Wave system creates and processes knowledge networks (KNs), which consist of nodes connected by directed or non-directed links. As depicted in Figure A.1, a KN maps into the real network such that one or more KN nodes may reside on one physical node, while KN links connect KN nodes, mapping on one or more physical links.

All KN nodes have unique addresses, which consist of two consecutive parts: a unique address of a node in a computer's memory and a unique address of the computer in a computer network. Both KN nodes and links have individual contents (or names) represented as arbitrary strings of characters.

[1]Peter S. Sapaty is the creator of the WAVE concept.
[2]Peter M. Borst wrote the first C implementation of a Wave interpreter.



(a) The physical network (b) The Knowledge Network

Figure A.1: *The mapping of the* Knowledge Network *on the physical network*

Any KN node may be accessed from any other KN node either directly, by their contents or addresses (*direct hops*), or by propagating through KN links (*surface hops*). Both direct or surface hops may be either *selective* (i.e. to a particular node), or *broadcasting* (i.e. to more than one node, including the cases where all other KN nodes are accessed by a direct hop, or only all neighbouring nodes by a surface hop).

A.2 The organisation of the WAVE language

The general syntax of the Wave language, as described in Figure A.2, reflects its operation mode, that is to propagate (parallel and asynchronous) through distributed data represented as a Knowledge Network (KN).

As already stated in Section 2.5.4.1, a Wave program, or a *wave*, consists of recursive sequences of parallel (separated by comma) and/or sequential (separated by period) spatial actions over KNs, called *moves*.

Moves can be either *simple moves* (i.e. sequences of *data units* separated by elementary operations, or *acts*), or *compound moves* (i.e. arbitrary *waves* in parenthesis which may be preceded by control *rules*).

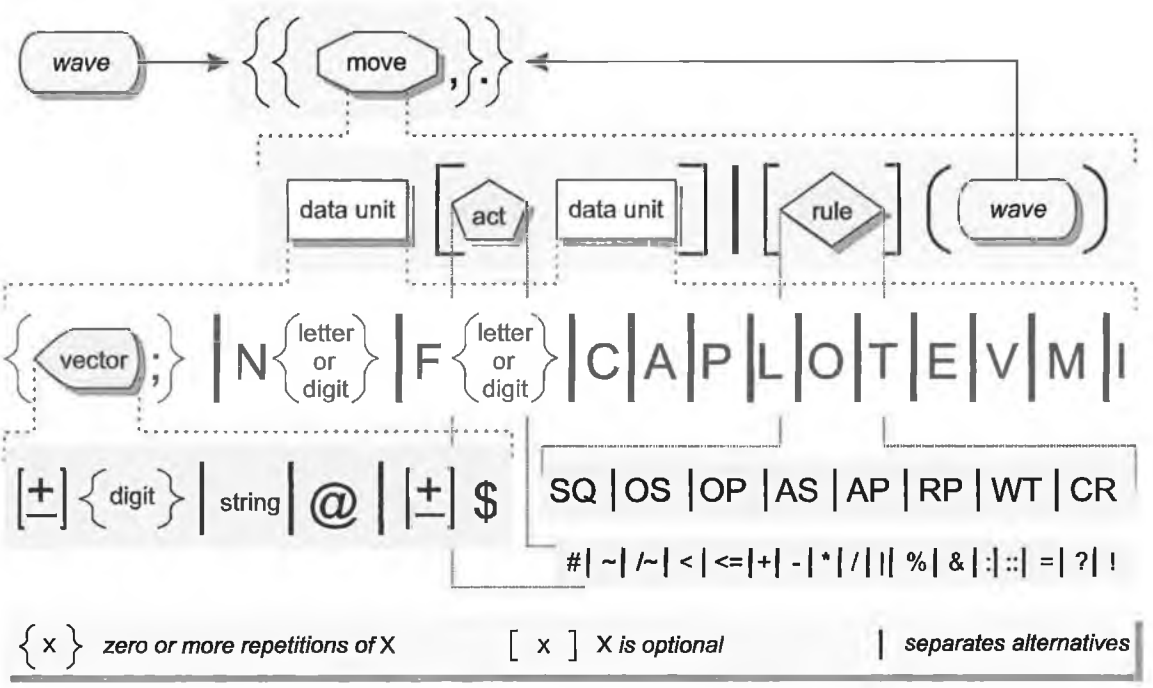


Figure A.2: *The syntax of the Wave Language*

Data units are either sequences of *vector* elements separated by semicolon, or *spatial variables*. Elements of a vector are string elements or scalars and they may be accessed, using *vector acts* (see Table A.3), either by their indices or by contents. Indices may address vector elements from the beginning, starting with 1 and being positive, or from the end, starting with -1 and being negative. The number of vectors elements is not declared in advance and may vary (i.e. increase or decrease) during computations.

Strings are delimited by single quotation marks, which can be omitted if the string contains only letters, digits and the underscore sign. It is recommended that strings starting with capital letters be placed in single quotes, as they may be interpreted as *spatial variables*, *rules*, or special scalars. A common scalar is a string optionally prefixed by a sign (+ or −). A special scalar is one of the reserved words in Table A.1.

Spatial variables are physically distributed in a KN space. They are divided into two classes, as depicted in Table A.2, as *task variables* serving different user algorithms, and *environmental variables* allowing access to different elements of

Table A.1: Special scalars *in Wave*

LONG NOTATION	SHORT NOTATION	EXPLANATION
DIRECT	@	used as left operand of a jump, provides the direct access to the node(s) determined by the right operant (e.g., @♯a)
INFINITE	$	(used to be used as local broadcast)
ANY		used as a left or right operand of a jump provides broadcasting wave propagation
STAY		represents an empty *move* which performs no operation
NONE		represents empty vector elements, or used as right operand in an assignment removes the left operand object.
ABORT	3	state that triggers an emergency abortion of the entire wave program.
TRUE	2	state that indicates "full success" and allows the development of further waves from the current node.
DONE	1	state that reflects the completion of that wave program while forbidding further development of it.
FALSE	0	state that indicates "failure" of the wave program with blocking of its development.

both virtual (KN) and physical (computer network) environment. Environmental variables C, A, P, L, O, T, E, V are stationary (i.e. reside on a node), while M, I are mobile (i.e. travel within a *wave*).

Acts, or elementary operations, as described in Table A.3, can be either *control acts* performing local data management and control in KN nodes, or *fusion acts* processing local data within nodes.

Rules are special Wave language constructs which set up constraints in the development of waves they precede. They may be nested recursively in a wave string. Usually they suspend the remainder of the wave program directly following the wave they control and release it after the completion of the ruled wave. The presentation of Wave rules is contained in Table A.4.

Table A.2: *Wave* spatial variables

Task variables	
Nodal Variables	▷ start with capital letter N ▷ reside on KN nodes
Frontal Variables	▷ start with capital letter F ▷ belong and travel within a *wave*
Environmental variables	
CONTENT (C)	▷ content of the current node ▷ assignment of an empty (NONE) to C removes this node from KN
ADDRESS (A)	▷ address of the current node
PREDECESSOR (P)	▷ predecessor node address
LINK (L)	▷ content of a passed link ▷ assignment of an empty (NONE) to L removes this link from KN
ORIENTATION (O)	▷ orientation of a passed link: + if traversed along its orientation - if traversed opposite its orientation
TERMINAL (T)	▷ special read and write variable which represents a terminal accessible from the current node
ENTRY (E)	▷ holds the name of the entry node which is a full network name of the current computer
VICINITY (V)	▷ vector of names of computers that have Wave interpreters activated and linked directly to the current interpreter
MAP (M)	▷ the map of KN ▷ vector of networks name and addresses of computers into which the created KN is distributed
IDENTITY (I)	▷ alfanumeric string representing the identity of a *wave* (individual color)

Table A.3: *Wave* acts

Control acts		
Hop	♯	hop between KN nodes
Filters	~	belongs
	/ ~	does not belong
	==	equal
	/ ==	not equal
	<	less
	<=	less or equal
	>	greater
	>=	greater or equal
Assignment	=	assignment
State generator	!	generates one of the states: `FALSE (0)`, `DONE (1)`, `TRUE (2)`, `ABORT (3)`
Code injection	∧	allows injection of arbitrary strings into a wave and executed immediately as program code
Fusion acts		
Arithmetic acts	+	add
	−	subtract
	*	multiply
	/	divide
Vector acts	&	append vectors
	:	find/record by index
	::	find/record by content
	=	assignment
Vector-String conversions	|	split string into vector
	%	merge vector into string
String operations	&&	string concatenation
	>>	concatenation of string vectors
External calls	?	enables access to any other systems on the same host via the basic operating system

Table A.4: *Wave* rules

	Splitting and branching rules
SQ	SEQUENCE ▷ activates all branches sequentially regardless of their resulting state
OS	OR SEQUENTIAL ▷ activates branches sequentially until a branch returns TRUE or DONE
OP	OR PARALLEL ▷ activates branches in parallel and selects the branch that first replies with the state TRUE or DONE while discarding all other branches
AS	AND SEQUENTIAL ▷ activates the branches sequentially if all return TRUE or DONE and stops if any returns FALSE
AP	AND PARALLEL ▷ activates all branches in parallel and discards all branches still running if any returns FALSE
RN	RANDOM ▷ randomly chooses a single branch among those formed by the splitting procedure.
	Other rules
RP	REPEATE ▷ first lets the wave to develop freely in KN, and if terminates in TRUE, the entire program, consisting of the RP-embraced wave followed by the wave remainder, will be reapplied on the discovered set of KN nodes
WT	WAIT ▷ suspends the wave remainder until the entire controlled wave terminates, resulting in logical wave synchronisation
ID	INDIVISIBLE ▷ treats the wave as an indivisible operation in the relation with the current node, while blocking all shared node resources from being used simultaneously by other waves
CR	CREATE ▷ supplies the wave embraced with the power to create or to extend the KN topology, while navigating in space.
RL	RELEASE ▷ makes the wave it embraces independent from the main program control.

A.3 Wave example: Creating a Knowledge Network

Arbitrary networks can be created in Wave using a wave embraced by a CR ("create") rule. This rule allows a distributed topology to be created without repeating CR for each element because the CR-rule is inherited when waves replicate [142]. There are a variety of possible strategies for creating the same topology. Two strategies for creating a reference four node network topology are described below.

The first strategy, depicted in Figure A.3, which is a more detailed version of Figure 5.2, operates sequentially. Here the network is created in six stages. In

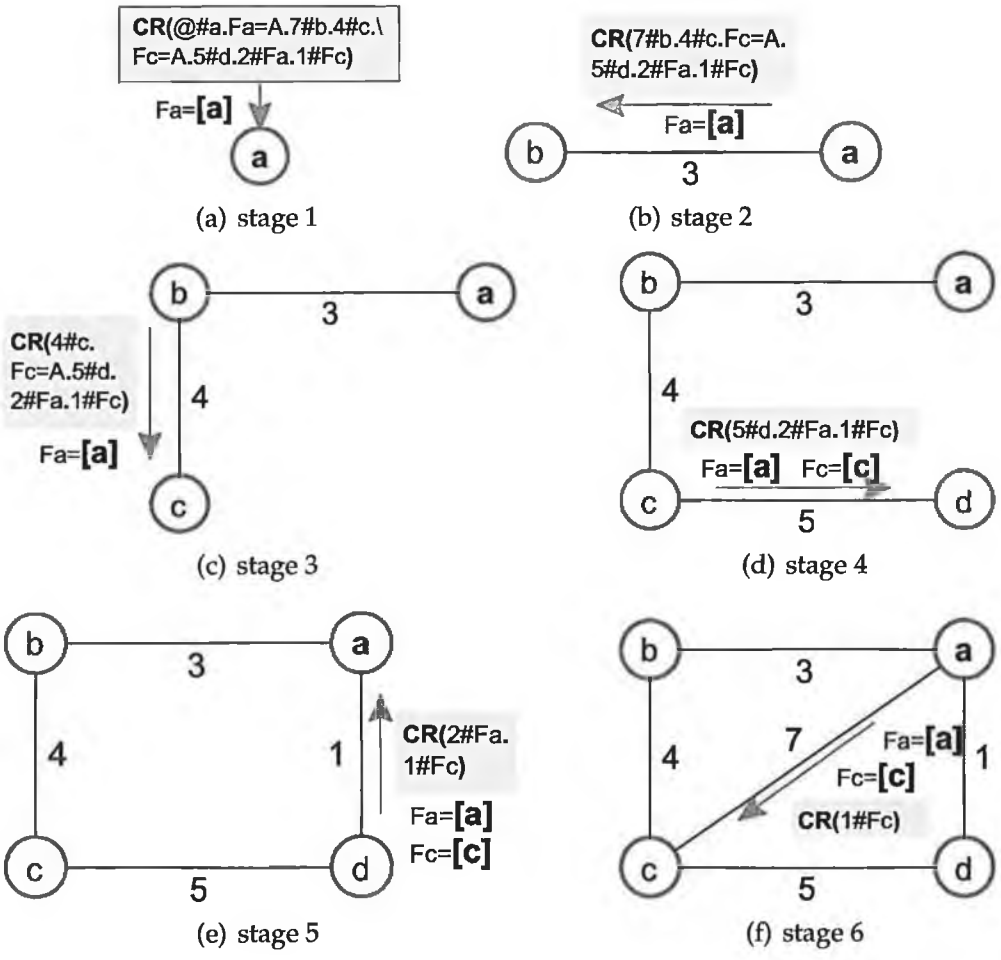


Figure A.3: *Sequential creation of a four-nodes network topology*

the first stage a direct hop is performed to node *a*, which is then created as a single-node KN. Moreover, the address of node *a* is saved in the frontal variable

Fa. In the second stage, this newly created KN is enlarged, by the addition of one more node *b*, which is connected to node *a* through a non-directed link with the associated attribute 7. The network is further expended in the following stages by adding two more nodes (i.e., *c* and *d*) and four more links with the attributes 4, 5, 2, and 1 respectively. Any time an explicit hop is performed on an already existing node (e.g., *a* or *c*), its absolute address is used. These addresses are obtained by accessing a special variable *A* (ADDRESS), which returns the absolute address of the current node. These addresses are recorded in frontal variables (e.g., *Fa* and *Fc*) and carried with the waves for further use. If the name of the node is used instead of its address the result would be the creation of duplicate nodes with the same name but different addresses.

The second strategy for creating the reference network topology (also described in detail in [143]) is depicted in Figure A.4 and contains only four stages, since it uses parallelism.

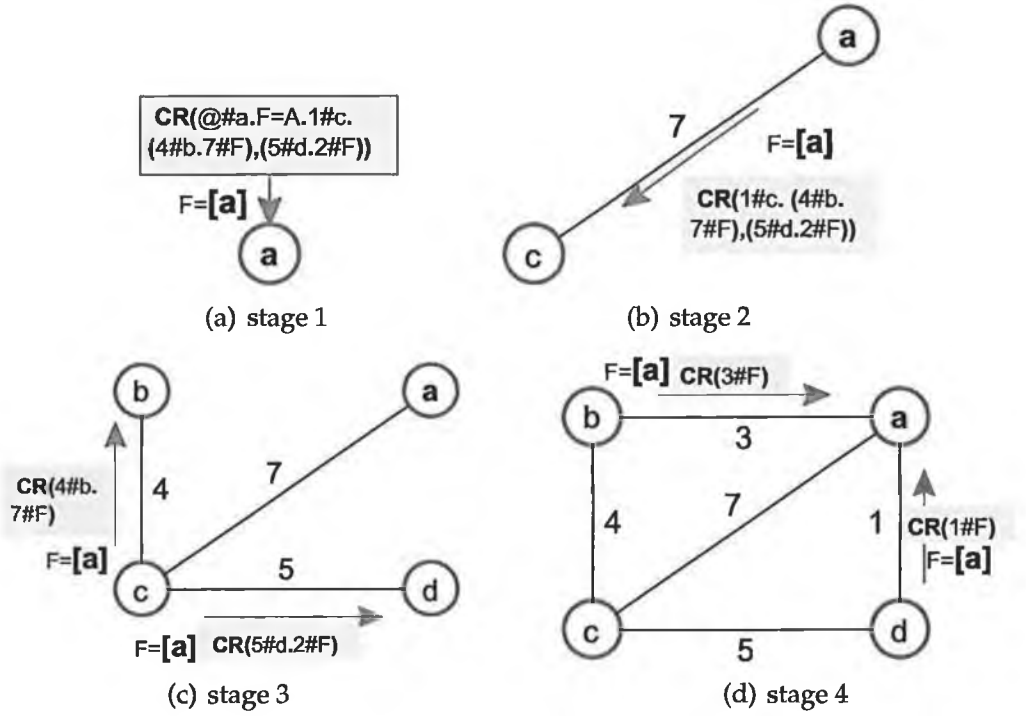


Figure A.4: *Parallel creation of a four-nodes network topology*

The parallel sequences are separated by a comma (,). Clearly this second strategy is more efficient, but it requires a prior analysis of the network topology.

A.4 The Wave Interpreter

A.4.1 The first C implementation of the Wave Interpreter

The C implementation of the Wave Interpreter was written by Borst, while he was a student at the University of Karlsruhe, and is described in detail in [29, 31]. The basic information required to *inject* a Wave program, so that it will start navigating the network and performing arbitrary computations in a parallel and distributed manner, is presented below.

The structure of the Wave interpreter, is depicted in Figure A.5. The *wkernel* process is the central element of the architecture, and implements the processing and control of the Wave language. The process which performs the interface between the Wave interpreter and UNIX is called *wexec* and executes the external calls of the Wave language which specify a sequence of Unix commands.

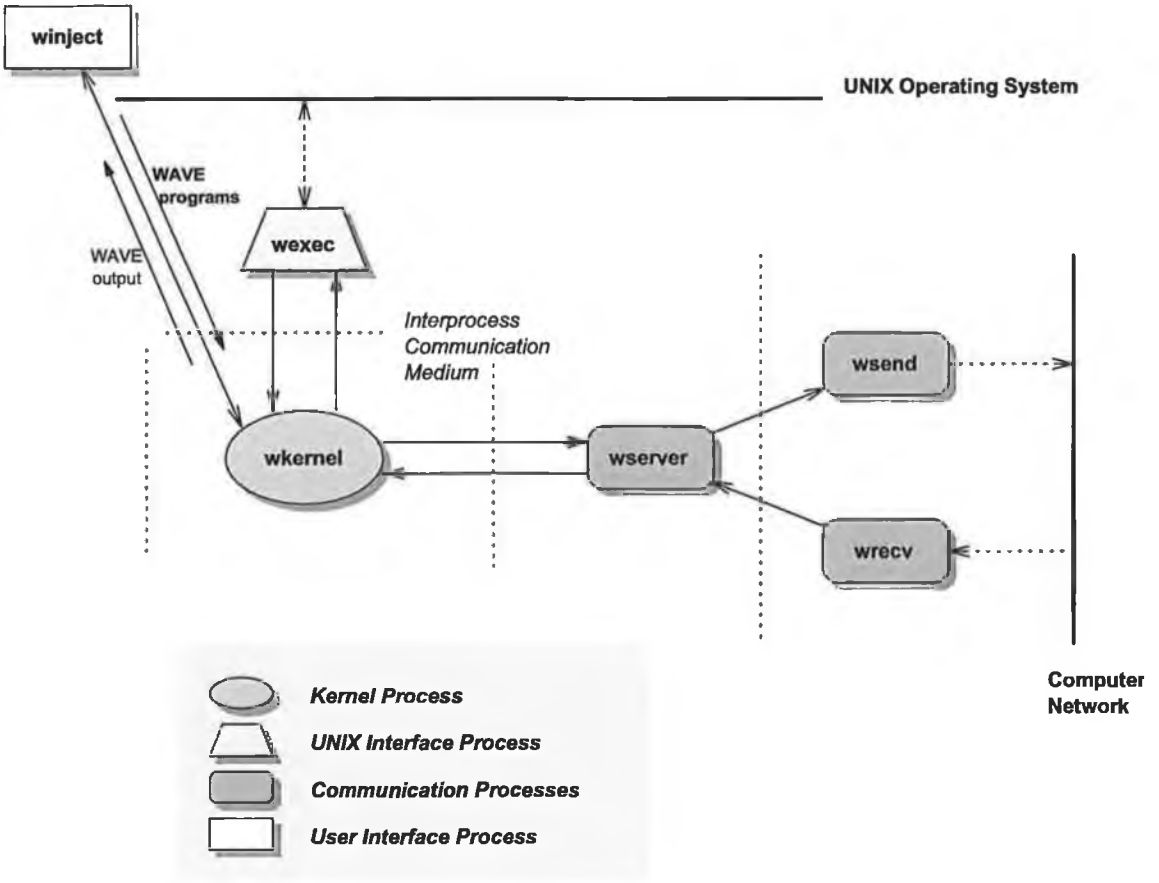


Figure A.5: *The Wave Interpreter*

The communication processes are connected to the Wave kernel through the *wserver* process, which is responsible for the management of message buffers and message exchange with the Wave kernel. Message transmission is performed by the *wsend* communication process, while message reception is the responsibility of the *wrecv* communication process.

A further process, which does not appear in Figure A.5, is called the *wstart* process. It is responsible for the initialisation and termination of the five processes mentioned above (i.e., *wkernel*, *wexec*, *wserver*, *wsend*, and *wrecv*). During the startup of the Wave system, *wstart* creates the IPC structures for the interprocess communication and then activates all other processes in turn, synchronising them so that they start simultaneously [29].

The user interface to the Wave system is performed by the *winject* process, which is fully embedded in the UNIX operation system and may be called like any other UNIX command. Multiple *winject* processes may be active at the same time, each having a unique query identifier, automatically attached to the submitted query[3]. Thus, the Wave kernel can distinguish between different sources of injection and knows where to return the output of the Wave programs. Hence, multiple input and output streams between different *winject* processes and the *wkernel* process can be served simultaneously.

The injection of Wave code can be done, by using the *winject* utility in either interactive (using the -i option) or noninteractive modes. In the interactive mode, the user is provided with a prompt at which may directly type Wave programs. The termination of the Wave program coincides in this case with the termination of the *winject* process. In the noninteractive mode, *winject* interprets its arguments to be the names of files containing the Wave programs to inject. If no files are supplied, it is assumed that they will appear on the standard input. If more than one file is supplied, *winject* submits them to the Wave kernel sequentially and terminates only after all Wave programs (that have been injected) terminate. Parallel

[3]Queries overlapping in one interpreter are processed concurrently in a time sliced fashion.

injection of Wave programs is also possible, simply by invoking separate *winject* processes.

A.4.2 Wave interpretation. The Wave Automaton

Basic description of the components of the Wave implementation presented in Section A.4.1, is given below. More details about the architecture of an Wave Interpreter are given in [142, 143].

The main components of a Wave Interpreter, as described in [142, 143], are depicted in Figure A.6.

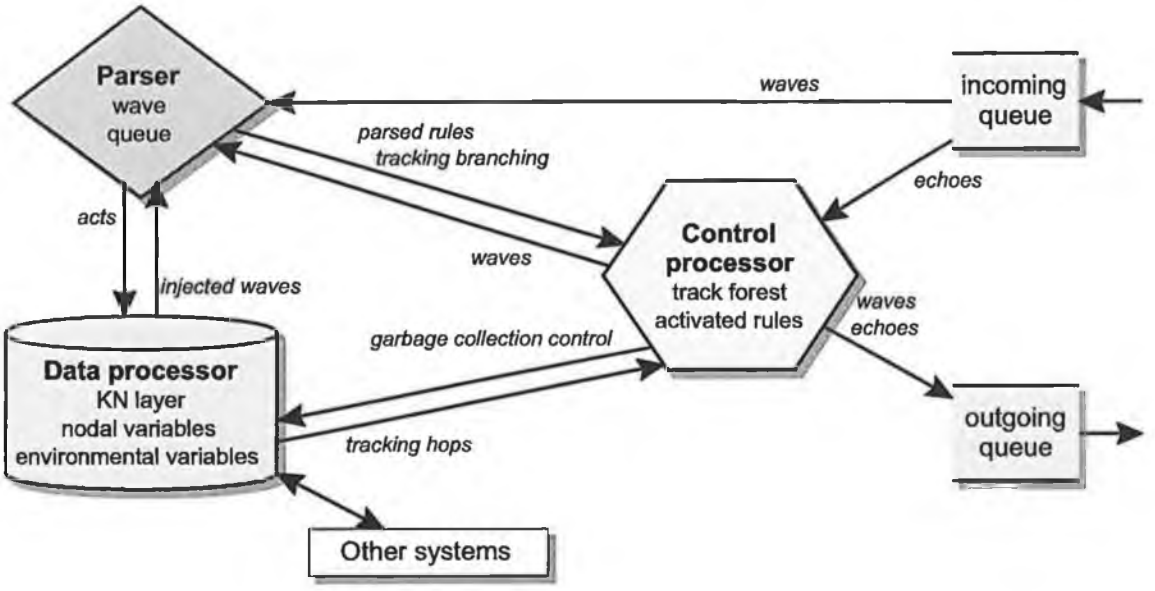


Figure A.6: *The Wave Interpreter Architecture*

The *incoming queue* collects waves and/or echoes received from other Wave interpreters, while the *outgoing queue* collects waves and/or echoes, already processed by the current interpreter, and which must be sent to other interpreters.

The main elements of the Wave Interpreter architecture are the three functional units: *parser*, *data processor*, and *control processor*.

The first unit an incoming wave will encounter is the *parser*, which performs all the necessary manipulations for the recognition, decomposition, modification and execution of the code [143]. These operations include:

- Removing unnecessary elements: redundant parenthesis, spaces, comments;
- Replacing the name of the rules by their abbreviations, the constants by their values, and expanding waves with new waves specified as procedures;
- Decomposition of waves into their heads and tails, and splitting the head into sectors and replicating the tail for each sector if necessary;
- Tracing hops by new tracks, and determining dynamic track branches corresponding with the rules encountered in waves;
- Maintaining a pool of suspended waves, which are waiting for their destination set of nodes (on which they are supposed to execute) to be defined;
- Determining the operations to be performed by the *data processor*.

When the head of the parsed wave contains an act, nodal or environmental variables that must be evaluated, it is sent to the *data processor*. This functional unit executes all (control as well as fusion) acts (see Table A.3), and manages all nodal and environmental variables as well as the KN topology residing on the current node. Upon executing an act, the data processor sends either an acknowledgment back to the parser, or new waves processed or retrieved[4] as data. It invokes the control processor for linking newly created nodal and/or frontal variables to tracks. The data processor also makes any connections with other systems that are resident on the same host (e.g., the established interface between the UNIX Operating System and the implementation of the Wave interpreter described in Section A.4.1).

The *control processor* maintains the track forest, which represents the main spatial control structure of the distributed language interpretation [143], and the active rules. Thus, the control processor executes the rules discovered in waves by the parser, and based on their result creates/updates the corresponding tracks.

[4]Nodal or frontal variables may include new procedures.

It also processes echoes, merging them in track nodes. Upon terminating waves, the control processor triggers the process of garbage collection in the data processor (for deleting redundant variables), which is associated with the deletion of tracks.

A.5 Wave Programs

The implementation introduced in [29], and summarised in Section A.4.1, uses three main classes of Wave programs.

Plain Wave programs are "normal" Wave programs, written using the syntax of the Wave language as described in A.2 and injected by using the *winject* process. They usually have the *.wave* extension.

Parametrised Wave programs are Wave programs with special placeholders for parameters. The parameter substitution is performed by a utility program called the Wave preprocessor. Thus, such programs have the *.wpp* extension.

Wave applications are complex UNIX shell scripts which implement user queuing, preprocessing, composition and injection of Wave programs.

In the Wave implementation described in Section 5.1.2 (Figure 5.1) a parameterised Wave program was used, which was invoked by using the following command line:

```
wpp -f my_alg.wpp 'a0;a3;a4;a7;a8' '100' 'a2' | winject > output_geo20
```

Here $a0; a3; a4; a7; a8$ are the border nodes between which all paths are to be found, 100 is the cost constraint, and $a2$ is the managing node responsible for collecting and and analysing the results in order to prepare the aggregated topology. After the algorithm is preprocessed by the *wpp* utility, it is injected in the network (through the Wave interpreter).

A.6 Messages in Wave

The communication in Wave is based on a *message passing* model [31]. In order to assess Macro-routing's communication overhead, a closer look at the packets/messages generated by the Wave system is needed. The main Wave messages are [31, 143]:

Waves, which contain Wave programs;

Echoes, which are messages used by the rules to be sent backwards via tracks to assess success or failure of the whole wave branch at the root;

Tails, which are remainder of Wave packets suspended upon rule execution and reactivated upon notification by echoes.

The structure of these messages is depicted in Figure A.7 and their possible content in Table A.5 [31].

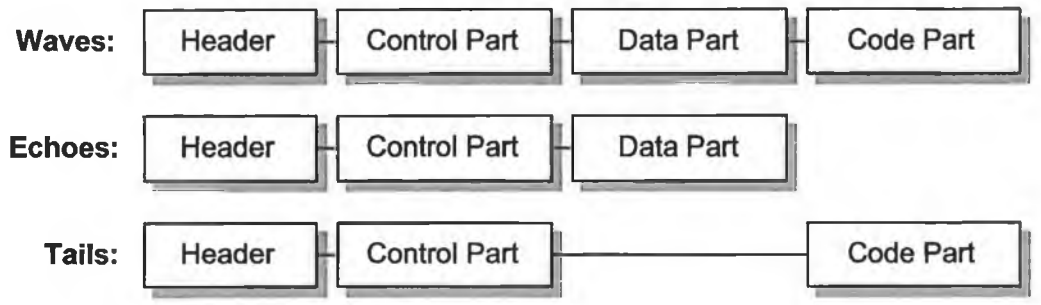


Figure A.7: *The Wave Interpreter*

Table A.5: *Contents of Wave messages*

HEADER		CONTROL PART		DATA PART		CODE
Type	Destination	Track	Flags	KN Ref.	Local Data	
WAVE	KN node	Pred.	✓	✓	✓	✓
ECHO	Track node	Succ.	-	-	✓	-
TAIL	Track node	Pred.	✓	-	-	✓

The destination of an echo and/or tail message is a track node. That is because echo and/or tail messages are generated when rules are used. Moreover,

as indicated by the "Track" field in the Control part of the echo message, such messages travel backwards in the KN, following the trails left by rule-embraced waves (i.e. tracks). The mode flags in the control part are present only in wave and tail messages. They usually define special properties relating to the control rules (e.g. CR) as well as to garbage collection [31, 143]. KN references, available only in a wave message, correspond to the mobile environment (e.g., the source of the wave, the link passed, the orientation of the link passed). Local data is available in wave massages, containing the colour or identity of the wave and its frontal variable, and in echo messages, containing the state transmitted (e.g., TRUE, DONE, FALSE, ABORT). The code part, present in wave messages and tails, contains the actual wave code.

APPENDIX B

Linux, MPLS-LINUX, and User-Mode Linux

B.1 The Linux Operating System

Linux is a Unix-type operating system. It had its origins in 1991 when a young student from the University of Helsinki in Finland, called Linus Torvalds wrote a kernel for a new Unix-like operating system which he called LINUX. Since then, the Linux Kernel was rewritten many times by its creator together with many other programmers worldwide and continues to develop under the GNU General Public License (GPL)[1] reaching in January of 2005 the stable version 2.6.12.

Its advantages, of free distribution, functionality, adaptability and robustness, have made it the main alternative to proprietary Unix and Microsoft operating systems for servers. IBM, Hewlett-Packard and other giants of the computing world have embraced Linux and support its ongoing development. Its use as a home and office desktop operating system is also on the rise. The operating

[1]See: `http://www.gnu.org/copyleft/gpl.html`

system can also be incorporated directly into microchips in a process called "embedding" and is increasingly being used in this way in appliances and devices.

Networking support in Linux is advanced and superior to most other Operating Systems. Since the people developing Linux collaborated and used the Internet for their development efforts, networking support came early in Linux development. As an Internet server, Linux is a very good choice, often outperforming Windows NT, Novell and most UNIX systems on the same hardware (even multiprocessor boxes). Linux is frequently chosen by leading businesses for superior server and network performance.

B.1.1 The *mpls-linux* project

The *mpls-linux* project is an open source effort to create a set of MPLS signalling protocols and an MPLS forwarding plane for the Linux operating system.

MPLS for Linux became a Sourceforge project in 30 November 2000. It consists of two packages:

- **mpls-linux** - the Linux Kernel based forwarding plane released under the GNU General Public License (GPL);
- **ldp-portable** - a portable version of the LDP protocol released under the Lesser General Public License (LGPL).

More details, the description and code sources of the two packages are available at: `http://sourceforge.net/projects/mpls-linux/`.

It is also possible to manipulate MPLS tables from userspace, add/manipulate incoming and outgoing labels and establish (manually) Label Switched Paths (LSPs). The main tool to do this is *mplsadm*[2].

[2]This utility was used to set up a LSP path for the sample application described in Section 4.2.1.1

B.1.1.1 The *mplsadm* utility

This utility is a userspace application that uses the MPLS subsystem parts to manually create LSPs from the command line. Figure B.1 presents the command line and the parameters for the *mplsadm* utility.

usage: ***mplsadm [ADBUdhvT:FL:I:O:i:o:m:]***

-A		add modifier
-B		bind modifier
-D		delete modifier
-U		unbind modifier
-d		toggle debug
-h		this message
-v		verbose info
-T	<tunnel name>:<dest addr>	mpls tunnel
-L	<interface name>:<label space>	set the label space for an interface
-l	<interface name>:<label space>	disable the label space for an interface
-I	<gen|atm|fr>:<label>:<label space>	create|delete an incoming label
-O	<key>	set a key for an outgoing label
-i	<opcode:opcode_data>+	specify instructions for an incoming label
-o	<opcode:opcode_data>+	specify instructions for an outgoing label
-m	<mtu>	specifies the MTU

Figure B.1: *Command line usage for* mplsadm

An example of using *mplsadm* to set an outgoing label is presented in Figure B.2. Here a new outgoing generic label (with label value 16) is allocated to be used between the host LSR and its downstream LSR identified by address 128.104.17.130 and which can be reached via the interface eth0.

More details and many more examples can be found at the Sourceforge site: `http://mpls-linux.sourceforge.net/`. Specific steps to be followed for setting up a Label Switched Path (LSP) are also presented there. However, the *mplsadm* utility would not suffice for setting up an LSP. The *netfilter*[3] framework

[3]Netfilter is described in Appendix C

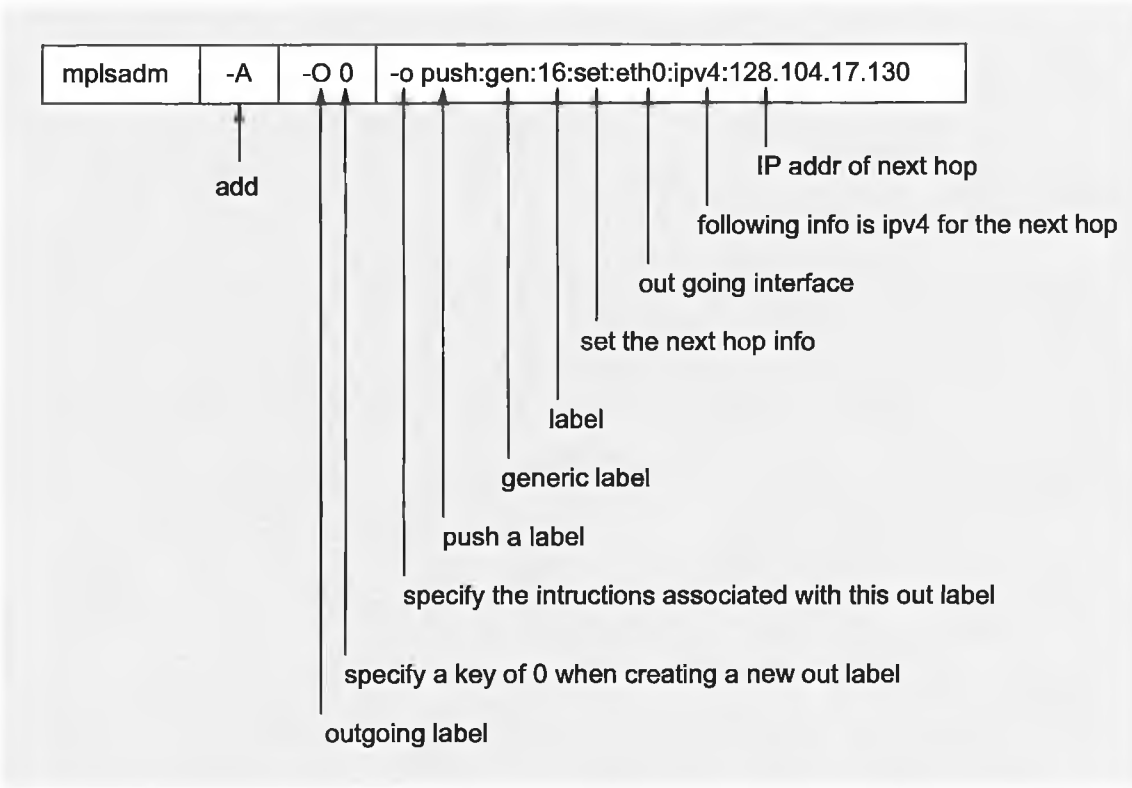


Figure B.2: *Allocate / Establish an out-going label using* mplsadm

is also used for mapping the actual traffic to the LSP by specifying *Forwarding Equivalence Classes* for the ingress Label Switched Routers.

B.1.2 User-Mode Linux

User-Mode Linux (UML)[61] is a way of running multiple *virtual* machines on a single physical one. Each virtual machine is able to run a different Linux instance[4] and all instances have separate resources.

The partitioning of a large machine into a number of virtual machines also has security advantages. If a virtual machine gets compromised, it would not affect the other virtual machines. Thus, UML is most frequently used for:

- kernel development and debugging
- process debugging

[4]They can be different Linux distributions or feature differen kernel versions.

- safely playing with the latest kernels
- trying out new distributions

It can also be used in education when students need a dedicated machine. With a proper configuration which gives limited access, nothing that is done on a virtual machine can change or damage the real computer, or its software. Moreover, UML virtual machines can be interconnected to each other, to the host, and to other physical machines in order to emulate a network. Thus, many universities which are running courses on OS internals and networking use UMLs.

The actual architecture of User-Mode Linux is the one depicted in Figure B.3(b), where it can be seen that the *virtual* machines run as common Linux processes.

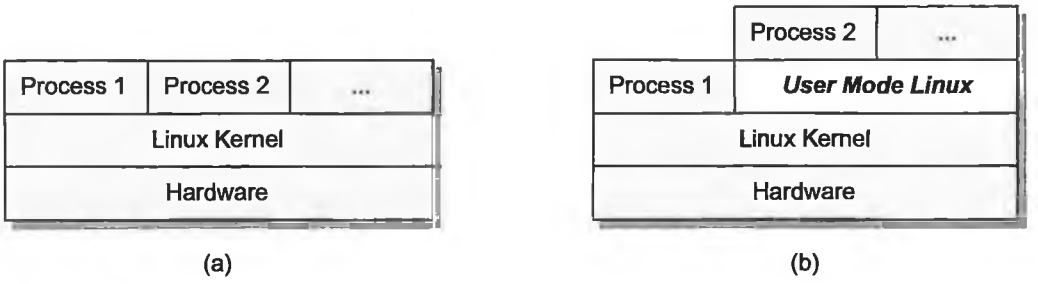


Figure B.3: *(a)The Linux Kernel (b)The User-Mode Linux Kernel*

That means that if normally, the Linux Kernel communicates directly with the hardware (video card, keyboard, hard drives, etc), and any programs which run ask the kernel to operate the hardware as shown in Figure B.3(a), the User Mode Linux Kernel is different. Instead of talking to the hardware, it talks to the real Linux kernel (called the *"host kernel"*), like any other program. Programs can then run inside User-Mode Linux as if they were running under a normal kernel, as shown in Figure B.3(b).

The Sourceforge site (i.e. `http://user-mode-linux.sourceforge.net/`) presents detailed guidelines for User-Mode Linux installation.

APPENDIX C

Netfilter

Netfilter [136–138] is an advanced framework within the Linux kernel that offers the possibility of packet *mangling* (i.e. modification of the header or payload contents). For each networking protocol netfilter implements "hooks" that are well defined points in a packet's traversal of that protocol stack. In IPv4 there are 5 such *hooks* placed as illustrated in Figure C.1:

NF_IP_PRE_ROUTING - just after entering IPv4 and before any route module;

NF_IP_LOCAL_IN - after the point where the routing module decides that the packet is local and before exiting IPv4 for an upper (transport) layer;

NF_IP_FORWARD - after the point where the routing module decides that the packet should be forwarded;

NF_IP_POST_ROUTING - after the point where the routing module decides where to forward the packet;

NF_IP_LOCAL_OUT - just after the IPv4 entry point from an upper (transport) layer and before any route module.

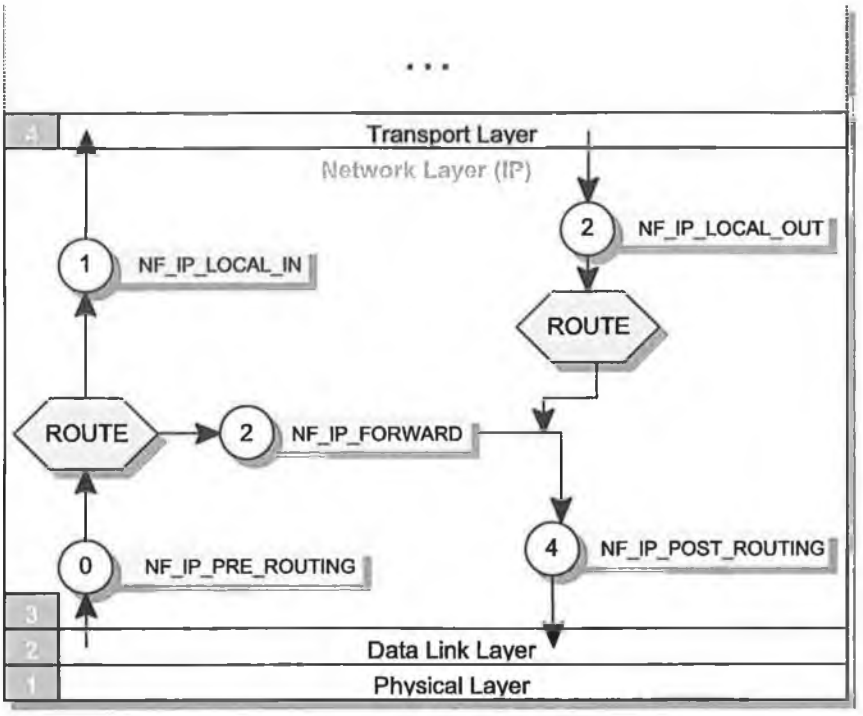


Figure C.1: *The Netfilter* hooks *defined in IPv4*

Parts of the kernel (i.e. kernel modules) can register to listen to the different hooks for each protocol. That means that when a packet is passed to the netfilter framework, it checks to see if a module is registered for that protocol and hook. Then, whoever registered for that protocol and hook can perform one of the following operations on the packet's content:

- NF_DROP - discard the packet;
- NF_ACCEPT - continue traversal as normal;
- NF_STOLEN - tell netfilter to forget about the packet;
- NF_QUEUE - queued the packet for userspace.

Queued packets are sent to userspace, where a userspace process can examine the packet, can alter it, and re-inject it at the same or different hook from which it left the kernel.

Complex packet manipulation tools such as *iptables* can be built on top of the netfilter framework.

C.1 Packet selection: IP Tables

IP Tables[1][2] is a packet selection system built upon the netfilter framework and is used for packet filtering (the *filter* table), Network Address Translation (the *nat* table) and general pre-route packet mangling (the *mangle* table).

C.1.1 Packet filtering

The iptable filter mechanism can only filter packets and never alter them. It hooks into netfilter at the NF_IP_LOCAL_IN, NF_IP_FORWARD and NF_IP_LOCAL_OUT points. That means that for any given packet there is one possible place to filter it. Thus, iptables filtering is smaller and faster than ipchains.

C.1.2 Network Address Translation (NAT)

This table is slightly different from the *filter* table, in that only the first packet of a new connection will traverse the table. The result of this traversal is applied to all future packets in the same connection.

As its name says, the *nat* table performs network address translations. Based on that, it is divided into two parts: *source* NAT (where the first packet is source altered) and *destination* NAT (the first packet is destination altered). For non-local packets the netfilter hooks used are are NF_IP_PRE_ROUTING and NF_IP_POST_ROUTING. They can have either the source or the destination address altered. For local packets the netfilter hooks used for the destination address alteration are NF_IP_LOCAL_IN and NF_IP_LOCAL_OUT.

[1] It is directly descendant of *ipchains* (that came from *ipfwadm*, that came from BSD's *ipfw* IIRC), with extensibility.

[2] More details at `http://www.netfilter.org/`

C.1.3 Packet mangling

The packet mangling table is used for actual changing of packet information. It hooks into netfliter at the NF_IP_PRE_ROUTING and NF_IP_LOCAL_OUT points.

APPENDIX D

Complexity Classes

There are different complexity classes. According to such classes the problems can be divided into *easy* or *tractable* problems which can be solved by polynomial-time algorithms and *hard* or *intractable* problems which require superpolynomial time.

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(10^n)$

represents the scale of some of the most common complexities from the easiest to the hard complexities. On such scale, the tractable problems are considered to be the ones which on inputs of size n, their worst-case running time is $O(n^k)$ for some constant k. They are representatives of the **P** class and can be solved in polynomial time. The other problems (e.g., with *exponential complexity* $f(n) = O(x^n)$) are considered intractable problems and are members of the **NP** class.

D.1 P (*Polynomial-time*)

This class is the set of problems for which a solution can be found in polynomial time (i.e. in time $O(n^k)$ for some constant k, where n is the size of the input to the problem). Different subsets of **P** class are depicted in Figure D.1.

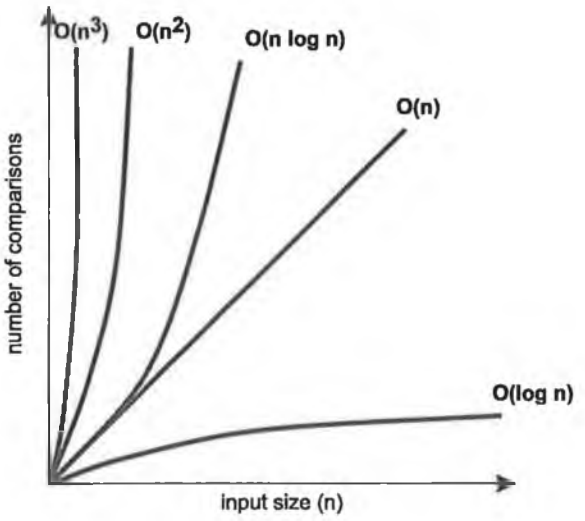


Figure D.1: *Polynomial-time complexity classes*

It can be seen that the best performance is obtained by the algorithms in the *logarithmic complexity class*, followed by the algorithms from the *polynomial complexity class* (e.g., $O(n^k)$, $k > 0$). The most efficient algorithms from the *polynomial complexity class* are the ones with *linear complexity* (i.e. $O(n) = O(n^1)$).

D.2 NP (*N*ondeterministic *P*olynomial time)

The set of problems for which a candidate solution can be verified in polynomial time, but nothing is known about how long finding the solution takes (i.e. could be exponential (e.g. $O(2^n)$)), are called **NP** problems.

It is generally believed that P is a proper subset of NP (i.e. $P \neq NP$). That means that any problem in P is also in NP, or $N \subset NP$.

D.3 NPC (*N*ondeterministic *P*olynomial time *C*omplete)

The set of problems with the following properties are called **NP-Complete** [51].

- the problem is also an **NP** problem,
- it is also **NP-hard**, i.e. every other problem in **NP** is reducible to it in a

polynomial time.

Informally, this means that an NP-complete problem is considered any problem that is as "hard" as any problem in NP. This implies that if any NP-complete problem can be solved in a polynomial time, then every NP problem has a polynomial time solution [52]. This can prove that $P = NP$. However, no polynomial-time algorithm has yet been discovered for any NP-complete problem. Thus, most theoretical computer scientists view the relationships among P, NP and NPC as depicted in Figure D.2.

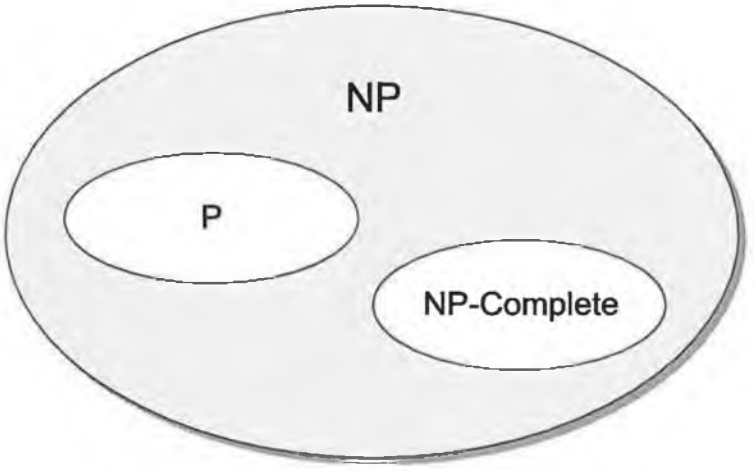


Figure D.2: *The relationship between the three complexity classes, P, NP, and NPC, where $P \subset NP$, $P \subset NP$, and $P \cap NPC = \emptyset$*

The **NP-Complete** problems are considered to be the "toughest" problems in **NP** in the sense that they are the ones most likely not to be in **P**. Thus, all known algorithms for NP-complete problems require a time that is superpolynomial in the input size. It is as yet unknown whether there are any faster algorithms to solve such problems. Therefore, one of the following approaches is used in these situations:

- **Approximation:** Finding a suboptimal solution that is within a certain (known) range of the optimal one.
- **Probabilistic approach:** Only a given distribution of the problem instances (one that assigns low probability to "hard" inputs) are considered.

- **Special cases:** Only problem instances belonging to a certain special case are considered.
- **Heuristic:** Using an algorithm that works "reasonably well" on many cases, but for which there is no proof that it is always fast and always produces a good result.

APPENDIX E

The Multi-constrained Macro-routing application model

A complex Linux application using C programming language modules, AWK scenarios and bash scripting was developed, as depicted in Figure E.8 for replicating the list of paths that the Multi-constrained Macro-routing protocol would discover in a two hierarchical level network, based on multiple constraints.

Each set of tests can have either one or thirty iterations[1]. Each iteration develops in three steps as follows:

1. A new hierarchy is built based on specification from the current configuration file.

2. A simulation that implements the Multi-constraint Macro-routing model computes the final paths.

3. Results are plotted.

A series of flat random topologies were generated using GT-ITM, by using

[1]As determined by executing the *One.sh* or *All_in_one.sh* shell scripts respectively.

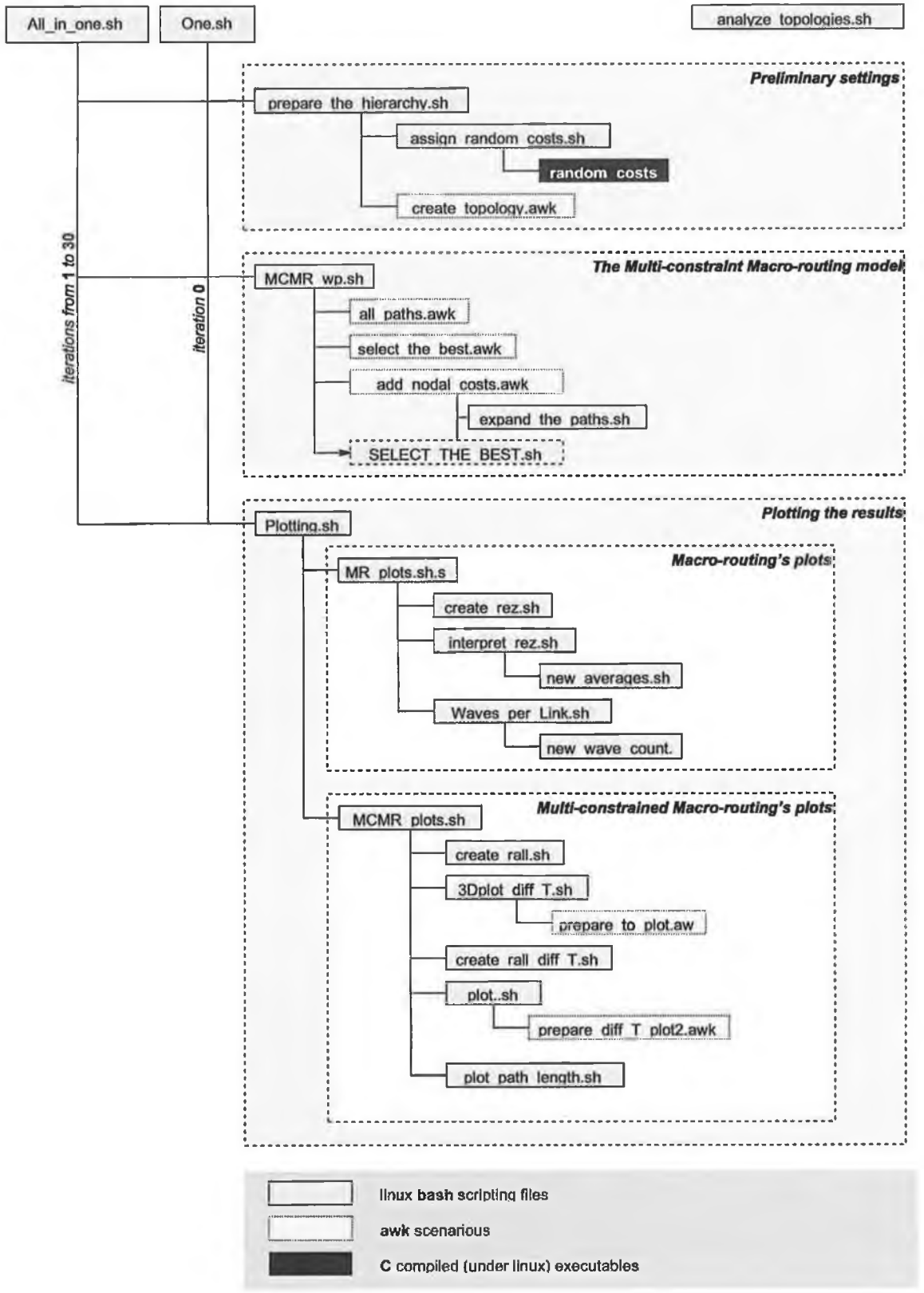


Figure E.1: *The Linux application generating the Multi-constrained Macro-routing paths*

similar configuration files as the ones depicted in Figure E.2. For further details please refer to [174, 175].

```
<method keyword> <number of graphs to generate>
<number of nodes> <space dimension> <generating method> <alpha>
```

(a) generic

```
geo 5
4 44 3 0.8
```

(b) for 4 nodes

```
geo 13
12 50 3 0.9
```

(c) for 12 nodes

```
geo 21
20 100 3 .12
```

(d) for 20 nodes

Figure E.2: *Samples of GT-ITM configuration files to generate flat random topologies*

A separate shell application (i.e. *analyze_topologies.sh*) was created to analyse the topologies used to build the hierarchy in terms of number of nodes, number of links and the connectivity degree. These computations are performed on all topology files used for a single hierarchy. The average connectivity degree for all these topologies is also computed.

A configuration file is used to set the parameters specific to each iteration. A sample of such a configuration file is depicted in Figure E.3.

The configuration file specifies such parameters as the files containing the topology and the hierarchical specifications, the source and the destination, the lifespan, the T parameter (i.e. the EFM interval maximum size), the EFM path selection method used, and the type, number and values of the metrics/constraints used for that iteration.

E.1 Preliminary settings

Step 1 involves building a hierarchy and assigning random costs corresponding with every metric. Figure E.4 depicts the combination of programs used to accomplish these tasks, as well as their input and output files.

The GT-ITM utility was used to generate flat (single-level) topologies. The topology specifications generated by GT-ITM, are *.alt* files, a typical specimen of

```
## file containing the names of the files with the domains/topologies to be aggregated
HDomains=tests/geo12_3/hgeo12_50_3_45-2.rel

## the hierarchical relation file
HRel=tests/geo12_3/hgeo12_50_3_45-2.rel.rc

## the upper topology
UpTop=tests/geo12_3/geo12_50_3_45-2.rc.alt

## the source as:  DOMAIN.NODE
Source=2.3

## the destination as:  DOMAIN.NODE
Destination=5.3

## the path limit (max no.  of nodes allowed)
PathLimith=

## the Extended-Full-Mesh limit = the max.  no.  of paths considered for one nodal link
## for Full-Mesh EFMLimit=1
EFMLimit=5

## if EFMLimit>1, a path selection method has to be specified.  It can be:
## 0 - TRUNCATE SINGLE ++ using only one metric (the first or specify with 0.i)
## 1 - TRUNCATE NORMAL ++ using the metric M=sum(Xi)/n i=1,n
## 2 - TRUNCATE RADIUS ++ using the metric M=sqrt(sum(Xi*Xi)) i=1,n
## 3 - DISPERSING RANDOM ++ using random selection
EFMSelectionMethod=1


## MetricType can be:
## A = additive
## C = concave :  MAX
## D = convex :  MIN
## M = multiplicative
## NOTE: the number of metric types must be the same with the number of
## metrics presented within the topology files.  If too few are specified in MetricType,
## the last one is considered for the rest.
MetricType=A;A;

## MetricOrder can be:
## < increasing order
## > decreasing order
## Same NOTE as above!
MetricOrder=<;<;

## MetricCosts contains the name of the files which have the possible cost values.
MetricCosts=costs/administrative_cost;costs/delay_ms;
```

Figure E.3: *Sample of configuration file*

whose content is depicted in Figure E.5(a). Here *vertices* are what I refer to as nodes and *edges* are what I call links. The *assign_random_costs.sh* utility assigns random costs[2] to every link in every[3] topology specification file belonging to the hierarchy. A representative specimen of output it generates is depicted in Figure E.5(b).

[2]The random costs are specified as a list of values in a separate *cost* file for each metric.

[3]Both the *.alt* file representing the topology at the second level of hierarchy (i.e., specified in the configuration file as *UpTop*), and all *.alt* files relevant to the lower hierarchical level named in the *.hrel* file depicted in Figure E.6(a).

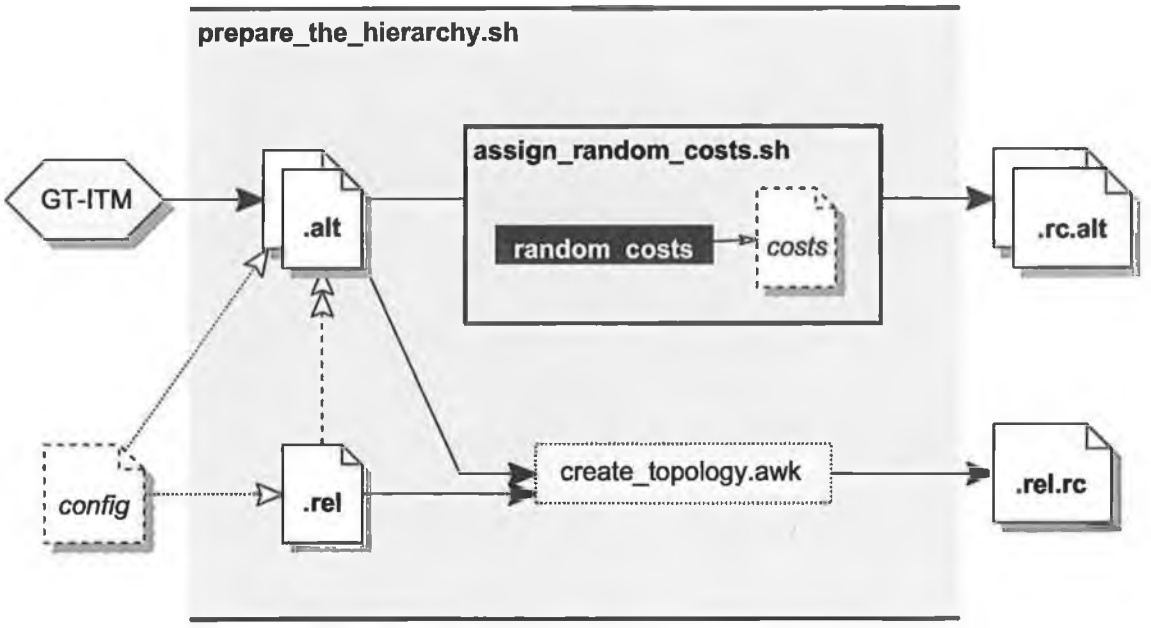


Figure E.4: *Preparing the hierarchy diagram*

```
...

VERTICES (index name):
0 0
1 1
2 2
3 3

EDGES (from-node to-node):
0 3
0 1
1 2
2 3
```

(a) Relevant *.alt* content

```
...

VERTICES (index name):
0 0
1 1
2 2
3 3

EDGES (from-node to-node):
0 3 14;44;
0 1 7;45;
1 2 4;27;
2 3 12;21;
```

(b) Relevant *.rc.alt* content for two metrics

Figure E.5: *Sample of topology specifications for a 4 node network*

The hierarchy is built as described in Section 5.1.3.2 by the *create_topology.awk* script, which uses the lower level topology specifications listed in the *.hrel* file and the topology aggregation file specified by the *UpTop* parameter in the configuration file to create an optimal hierarchy. The result is a hierarchy specification file, as depicted in Figure E.6(b), where each line contains information about one domain at the first hierarchical level in the format specified in Figure E.7.

This comprises:

- a *filename* representing a topology specification file with the random costs

```
geo4_50_3_3.alt
geo4_50_3_4.alt
geo4_50_3_5.alt
geo4_50_3_6.alt
```

(a) *.hrel* file

```
geo4_50_3_3.rc.alt 0    1 2    3 3
geo4_50_3_4.rc.alt 1    0 3    2 0
geo4_50_3_5.rc.alt 2    1 1    3 0
geo4_50_3_6.rc.alt 3    0 0    2 2
```

(b) *.hrel.rc* file

Figure E.6: *Sample of hierarchical specifications for a* 4×4 *node network*

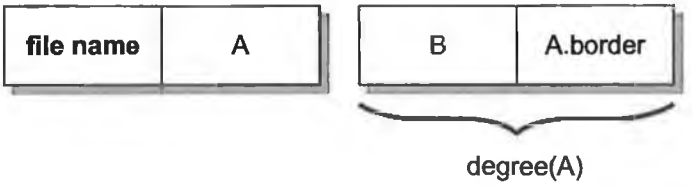


Figure E.7: *Entry in the hierarchy specification file*

(i.e., *.rc.alt*);

- A, the corresponding name of the node from the domain at the second level of the hierarchy;
- a list of pairs $(B, A.border)$, where the size of the list corresponds to the degree of node A (i.e. the number of incident links to A), and where B and $A.border$ represent the following:
  - B is a node in the second level of hierarchy connected to A (i.e. corresponding with the information from the topology specification file of the domain in the second level of hierarchy);
  - $A.border$ is the border node from the domain corresponding to A connected to another border node of the domain corresponding to B.

During this phase a source and a destination node are also designated[4] and placed in the configuration file so that they can be used during the next application phase, which is described in the following section.

[4]The process of electing the source and the destination nodes is described in Section 5.1.3.2.

E.2 The Multi-constraint Macro-routing model

The process of searching all possible paths in the hierarchy and their corresponding path-costs along with the aggregations required is depicted in Figure E.8.

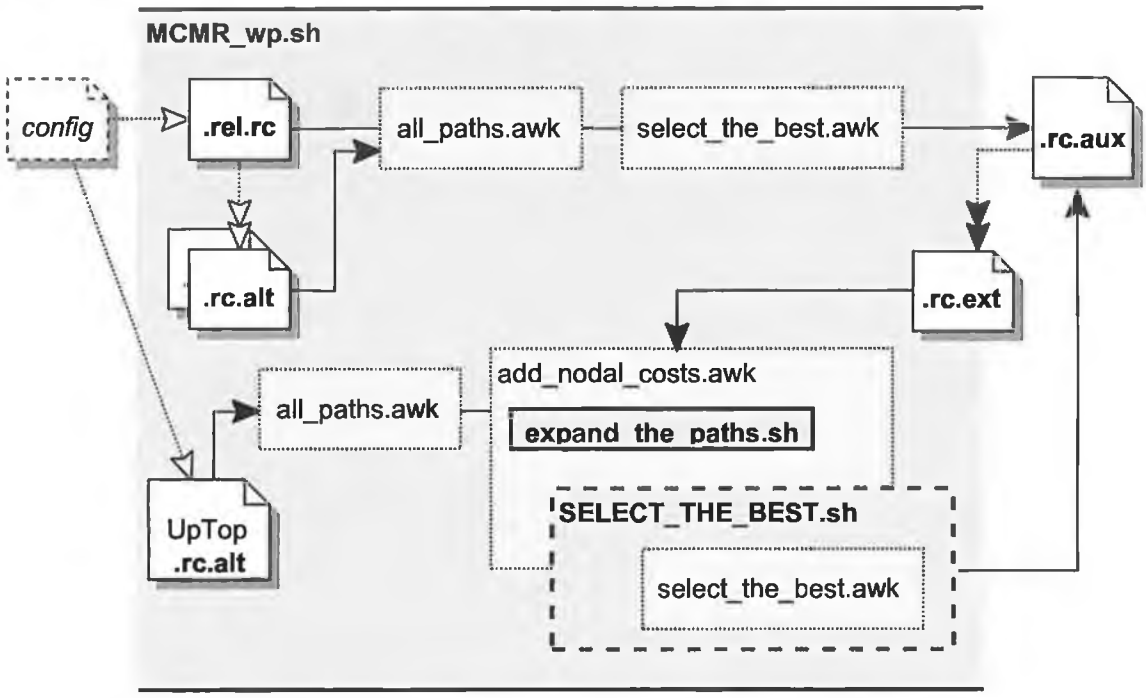


Figure E.8: *The Multi-constrained Macro-routing model*

It starts by finding all paths between all border nodes within domains at the first hierarchical level, i.e. all paths across the transit domains. The border nodes are explicitly specified in the *.hrel.rc* file and the paths between them can be computed based on the information from the corresponding topological specification (i.e., *.rc.alt*) files. This task is performed by *all_paths.awk*. The *select_the_best.awk* script chooses only a number of these, the number of paths selected and the selection technique being specified in the configuration file. All the intermediate results are placed in the *.rc.aux* file. Any final paths found between two border nodes are also recorded in the extended[5] matrix in the *.rc.ext* file.

All paths between the source node and all border nodes in the source domain as well as the paths between all border nodes in the destination domain and the

[5]See the description of the extended matrix in Section 5.1.3.2.

destination node are also computed in a similar way to the above. These results are also documented in the *.rc.aux* and the *.rc.ext* files.

At the second level of the hierarchy, all paths between the source and the destination are computed using *all_paths.awk*. These paths are then expanded by including the *nodal* costs (as computed by *add_nodal_costs.awk* from the extended matrix stored in the *.rc.ext* file). A number of these *expanded* paths are the selected using the selection mechanism specified in the configuration file. These are the final paths the Multi-constrained Macro-routing protocol would discover.

E.3 Plotting the results

I used this application to obtain paths for both *Macro-routing* and *Multi-constrained Macro-routing*.

All results obtained with this application are plotted using the *gnuplot*[6] utility.

E.3.1 Macro-routing results

Macro-routing's tests were performed on two level hierarchies, by using a single metric in conjunction with the Full-Mesh aggregation technique.

That part of the application which is responsible with the interpretation of these results is schematically presented in Figure E.9.

This can evaluate:

- Macro-routing's *efficiency*[7];
- The proportion of paths found, which are Macro-routing's *success* or *best*[8];
- Macro-routing's *effort*[9] in searching for long paths;

[6] *Gnuplot* is a freely distributable and portable command-line driven interactive data and function plotting utility. See more details on `http://www.gnuplot.info`.
[7] See Definition 4 from Section 5.1.4.2.
[8] See Definition 5 from Section 5.1.4.2.
[9] See Definition 3 from Section 5.1.4.

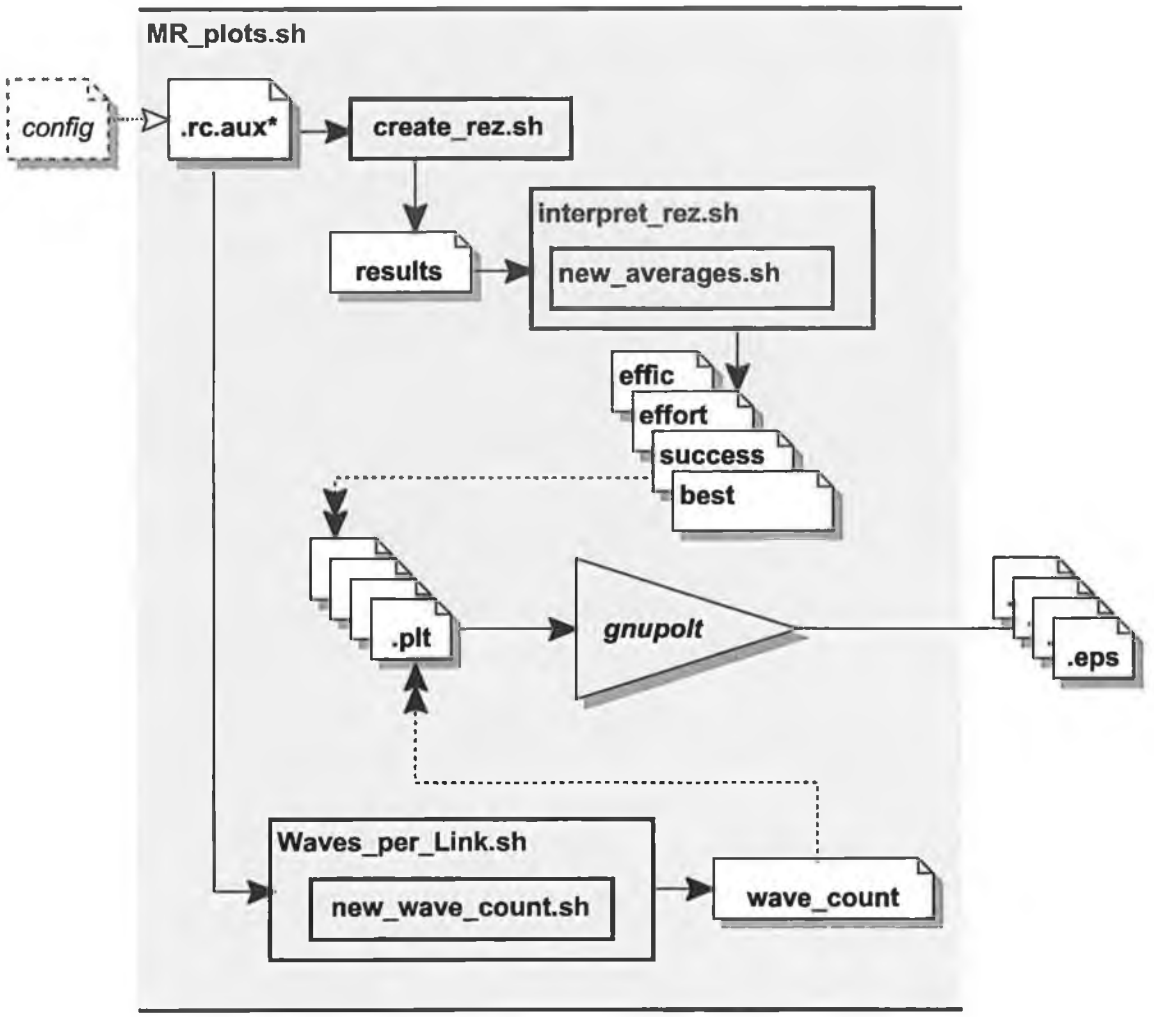


Figure E.9: *The interpretation process for Macro-routing's test results*

- The communication overhead generated by Macro-routing in terms of the number of waves per link.

In order to perform the first three tasks, only the final paths (as recorded on the last line of the *.aux* files) from each test[10] are selected by *create_rez.sh* and placed in the *results* file. These results are gathered for different *lifespan* values, subject to the constraint that the final path must be no longer than the lifespan. Finally, the application computes:

- the ratio between the path cost obtained by Macro-routing when no lifespan

[10]With each iteration there was one test result for each EFM path selection mechanism implemented. Thus, for 30 iterations and 5 EFM path selection mechanisms, $5 \times 30 = 150$ distinct tests were performed.

is imposed and the path cost obtained by Macro-routing when a lifespan is imposed, for various lifespan values;

- the number of tests which find paths that have the best cost (i.e. Macro-routing's *best*);
- the number of tests which at least find a viable path even if that path does not have the best cost (i.e. Macro-routing's *success*).

The subset of paths generated by Macro-routing with different lifespan values and documented in *.aux_limit* files are processed in order to determine the average number of waves/link.

All gathered data were plotted using *gnuplot* scripts (i.e., *.plt*) similar to that depicted in Figure E.10. The plotted results were placed in *.eps* files.

```
reset
clear
set terminal postscript eps color "Times-Roman" 25
set output "MReffic.eps" # otherwise writes the postscript into the screen
set size 1,1 # for plots which have a long x-axis
set key right bottom

unset border
set grid

set xlabel "lifespan"
set ylabel "EFFICIENCY"

set xrange [2:10]
set yrange [0:*]

plot "effic" using 1:2 notitle with linespoints lw 2
```

Figure E.10: *Sample of gnuplot scripts used to plot Macro-routing's test results*

E.3.2 Multi-constrained Macro-routing results

That part of the application which is responsible for the interpretation of the Multi-constrained Macro-routing test results is schematically presented in Figure E.11. It is mainly concerned with making comparisons between:

I. results generated by all EFM path selection mechanisms;

II. *TRUNCATE NORMAL* and *TRUNCATE RADIUS* while considering the proportion of paths satisfying various constraints;

III. *FULL-MESH* and *EXTENDED FULL-MESH* while considering the proportion of paths satisfying various constraints;

IV. the length of paths generated all EFM path selection mechanisms;

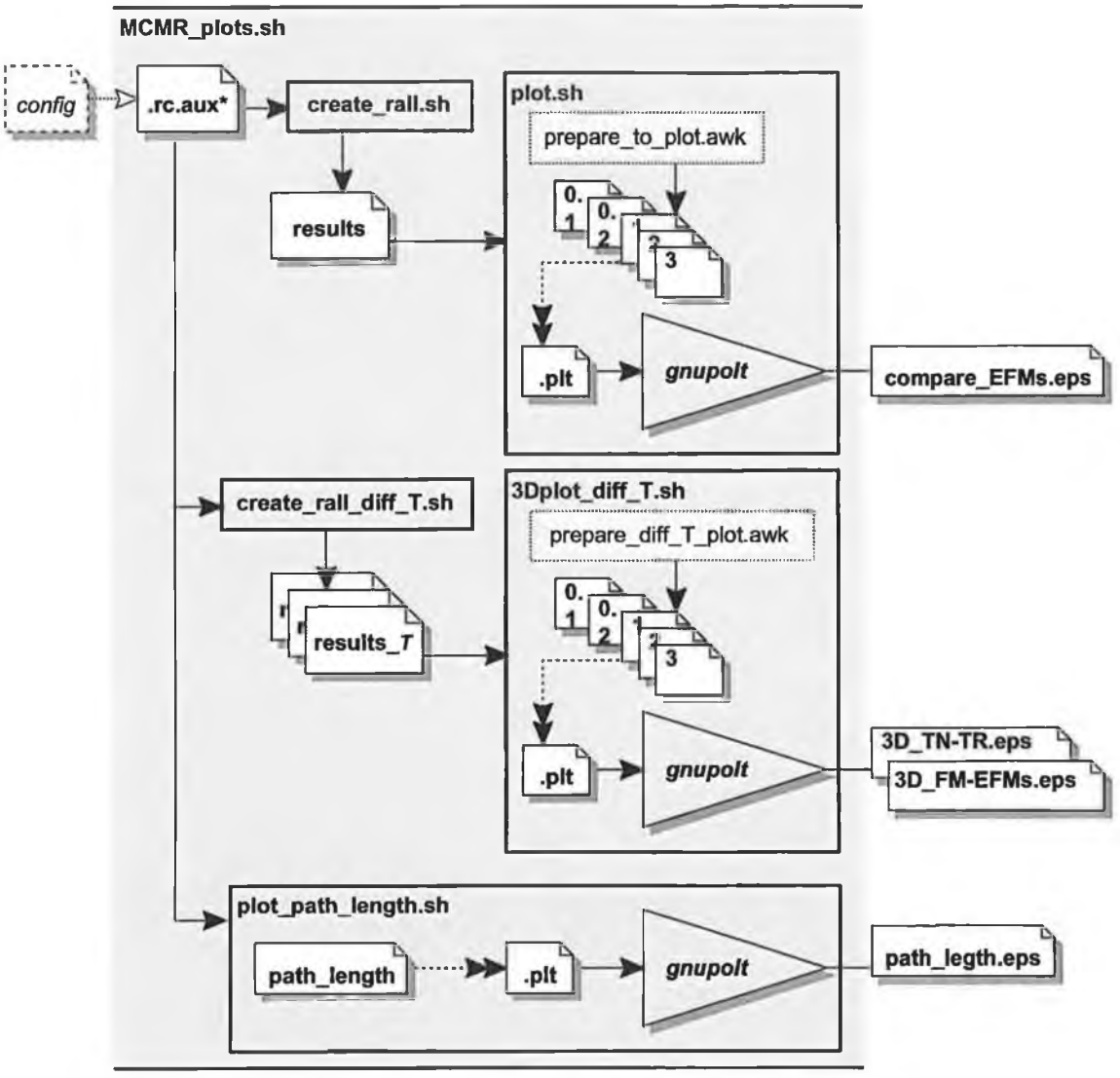


Figure E.11: *The interpretation process for Multi-constraint Macro-routing's test results*

The results in set I. are generated using *create_rall.sh* and *plot.sh*. The *.aux* files are used in a similar way to that described in Section E.3.1 to obtain the *results* file. The data in this file is segmented in distinct files corresponding with each

EFM path selection mechanism used. As presented in Figure E.3, $0.i$ corresponds to *TRUNCATE SINGLE using metric* i, 1 corresponds to *TRUNCATE NORMAL*, 2 corresponds to *TRUNCATE RADIUS* and 3 corresponds to *DISPERSING RANDOM*. The values for each path cost are normalised with respect to the best path cost corresponding to each metric and then the results are plotted using a gnuplot script *compare_EFMs.plt*, which documents the plot in an *.eps* file.

The results for sets II. and III. are generated using *create_rall_diff_T.sh* and *3Dplot_diff_T.sh*. They process the data in *.aux* files in a similar way to those for set I., but also consider various values for the EFM interval dimension (i.e., T). The *awk* script also counts the proportion of paths meeting the constraints as the values of constraints vary (see Figures 6.9 and 6.10). A sample gnuplot script for plotting the resulting three-dimensional data (for the case of two constraints) is given in Figure E.12.

```
reset
clear
# set time
set terminal postscript eps color "Times-Roman" 20
set output "3D_FM-EFMs.eps" # otherwise writes the postscript into the screen
# set size 1.2,1 # for plots which have a long x-axis
set key right top

set xrange [*:*]
set yrange [*:*]
set zrange [*:*]

# set nosurface
# set contour
set view 80,80
show view

set xlabel "adm.  cost"
set ylabel "delay"
# set zlabel "proportion of paths satisfying the constraints"

set ticslevel 0

set xtics 45
set ytics 50
# set ztics 40


splot "T4_1" with dots title "proportion of paths satisfying the constraints with T=4",
      "T1_1" with dots title "proportion of paths satisfying the constraints with T=1"
```

Figure E.12: *Sample of plotting file*

The results for set IV. are found as follows. The number of paths whose length is below a given parameter is performed separately on each *.aux.p* file, where *p* is the value for the corresponding EFM path selection method. The results obtained are gathered in the *path_length* file, and then plotted using gnuplot in a similar manner to that used for set I.

BIBLIOGRAPHY

[1] Cengiz Alaettinoglu and A. Udaya Shankar. The Viewserver Hierarchy for Interdomain Routing: Protocols and Evaluation. *IEEE Journal of Selected Areas in Communications*, 13(8):1396–1410, 1995.

[2] D. Scott Alexander, William A. Arbaugh, Michael W. Hicks, Pankaj Kakkar, Angelos D. Keromytis, Jonathan T. Moore, Carl A. Gunter, Scott M. Nettles, and Jonathan M. Smith. The SwitchWare Active Network Architecture. *IEEE Network Magazine Special issue on Active and Controllable Networks*, 12(3):29–36, May/June 1998.

[3] Loa Andersson, Paul Doolan, Nancy Feldman, Andre Fredette, and Bob Thomas. LDP Specification. RFC 3036, IETF, January 2001. Status: STANDARDS TRACK.

[4] Loa Andersson and George Swallow. The Multiprotocol Label Switching (MPLS) Working Group decision on MPLS signaling protocols. RFC 3468, IETF, February 2003. Status: INFORMATIONAL.

[5] George Apostolopoulos, Roch Guerin, Sanjay Kamat, and Satish K. Tripathi. Quality of Service Based Routing: A Performance Perspective. In *ACM SIGCOMM 98*, pages 17–28, Vancouver, Canada, August 1998.

[6] George Apostolopoulos and Satish K. Tripathi. On Reducing the Processing Cost of On-Demand QoS Path Computation. In *the 6th International Conference on Network Protocols (ICNP)*, pages 80–89, Austin, Texas, USA, October 1998. ISBN: 0-8186-8988-9.

[7] George Apostolopoulos, Doug Williams, Sanjay Kamat, Roch Guerin, Ariel

Orda, and Tony Przygienda. QoS Routing Mechanisms and OSPF Extensions. RFC 2676, IETF, August 1999. Status: EXPERIMENTAL.

[8] Grenville Armitage. MPLS: The Magic Behind the Myths. *IEEE Communication Magazine*, 38(1):124–131, January 2000.

[9] ATM Forum. Private network-network interface specification, version 1.0. Technical Report af-pnni-0055.000", ATM Forum, March 1996.

[10] ATM Forum. Multi-Protocol Over ATM Version 1.0. Specification af-mpoa-0087.000, ATM Forum, July 1997.

[11] Daniel O. Awduche. MPLS and Traffic Engineering in IP networks. *IEEE Communication Magazine*, 37(12):42–47, December 1999.

[12] Daniel O. Awduche, Lou Berger, Der-Hwa Gan, Tony Li, Vijay Srinivasan, and George Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209, IETF, December 2001. Status: STANDARDS TRACK.

[13] Daniel O. Awduche, Angela Chiu, Anwar Elwalid, Indra Widjaja, and Xipeng Xiao. Overview and Principles of Internet Traffic Engineering. RFC 3272, IETF, May 2002. Status: INFORMATIONAL.

[14] Daniel O. Awduche, Joe Malcolm, Johnson Agogbua, Mike O'Dell, and Jim McManus. Requirements for Traffic Engineering Over MPLS. RFC 2702, IETF, September 1999. Status: INFORMATIONAL.

[15] Anand Balachandran, Andrew T. Campbell, and Michael E. Kounavis. Active Filters: Delivering Scaled Media to Mobile Devices. In *7th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 125–134, St. Louis, Missouri, USA, May 1997.

[16] Albert Banchs, Wolfgang Effelsberg, Christian Tschudin, and Volker Turau. Multicasting Multimedia Streams with Active Networks. In *IEEE Local Computer Network Conference (LCN)*, pages 150–159, Boston, Massachusetts, USA, October 1998.

[17] Joachim Baumann. Mobility in the Mobile-Agent-System Mole. In *3rd CaberNet Plenary Workshop*, Stuttgart, Germany, January 1997.

[18] Richard Ernest Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.

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

[20] Lou Berger. Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description. RFC 3471, IETF, January 2003. Status: STANDARDS TRACK.

[21] Lou Berger, Der-Hwa Gan, George Swallow, Ping Pan, Franco Tommasi, and Simone Molendini. RSVP Refresh Overhead Reduction Extensions. RFC 2961, IETF, April 2001. Status: STANDARDS TRACK.

[22] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, 1992. ISBN: 0-13-200916-1.

[23] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura. Active Networking and the End-to-End Argument. In *5th IEEE International Conference on Network Protocols (ICNP)*, Atlanta, Georgia, USA, October 1997. IEEE Computer Society.

[24] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura. An Architecture for Active Networking. In *IFIP TC6 7th International Conference on High Performance Networking*, pages 265–279, White Plains, New York, USA, April 1997. ISBN: 0-412-82070-6.

[25] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura. Self-Organizing Wide-Area Network Caches. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 600–608, San Francisco, California, USA, April 1998.

[26] Andrzej Bieszczad, Bernard Pagurek, and Tony White. Mobile Agents for Network Management. *IEEE Communications Surveys*, 1(1), 1998.

[27] Uyless Black. *MPLS and Label Switching Networks*. Prentice Hall, 2001. ISBN: 0-13-015823-2.

[28] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An Architecture for Differentiated Services. RFC 2475, IETF, December 1998. Status: INFORMATIONAL.

[29] Peter M. Borst. The first implementation of the WAVE system for UNIX and TCP/IP computer networks. Technical Report 18/92, Dept. of Informatics, Univ. of Karlsruhe, Karlsruhe, Germany, December 1992.

[30] Peter M. Borst. Towards an Architecture for WAVE Interpretation in Open Distributed Systems. Technical Report 14/95, Dept. of Informatics, Univ. of Karlsruhe, Karlsruhe, Germany, May 1995.

[31] Peter M. Borst. *An Architecture for Distributed Interpretation of Mobile Programs*. PhD thesis, Faculty of Informatics, University of Karlsruhe, Karlsruhe, Germany, 2001.

[32] R. Braden, Dave Clark, and Scott Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633, IETF, June 1994. Status: INFORMATIONAL.

[33] R. Braden, Lixia Zhang, Steve Berson, Shai Herzog, and Sugih Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205, IETF, September 1997. Status: STANDARDS TRACK.

[34] Terry Bradley, Caralyn Brown, and Andrew Malis. Inverse Address Resolution Protocol. RFC 2390, IETF, September 1998. Status: STANDARDS TRACK.

[35] Pierre Bremaud. *Markov chains : Gibbs fields, Monte Carlo simulation and queues*. Springer, New York, USA, 1999. ISBN: 0-387-98509-3.

[36] Ross W. Callon. Use of OSI IS-IS for routing in TCP/IP and dual environments. RFC 1195, IETF, December 1990. Status: PROPOSED STANDARD.

[37] Kenneth L. Calvert, Samrat Bhattacharjee, Ellen W. Zegura, and James P.G. Sterbenz. Directions in Active Networks. *IEEE Communications Magazine*, 36(10):72–78, October 1998.

[38] Luca Cardelli. A Language with Distributed Scope. In *22nd ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 286–297, San Francisco, California, USA, 1995.

[39] Ravikumar V. Chakaravarthy. IP Routing Lookup: Hardware and Software Approach. Master's thesis, Texas A&M University, 2004.

[40] Daniel T. Chang and Danny B. Lange. Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, San Jose, USA, October 1996.

[41] Shigang Chen. *Routing Support for Providing Guaranteed end-to-end Quality-of-Service*. PhD thesis, Engineering College of the University of Illinois at Urbana-Champaign, Urbana, Illinois, 1999.

[42] Shigang Chen and Klara Nahrstedt. An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions. *IEEE Network Magazine*, 12(6):64–79, December 1998.

[43] Shigang Chen and Klara Nahrstedt. Distributed Quality-of-Service Routing in High-Speed Networks Based on Selective Probing. Technical report, University of Illinois at Urbana-Champaign, Department of Computer Science, 1998. The extended abstract was accepted by LCN98.

[44] Shigang Chen and Klara Nahrstedt. On Finding Multi-constrained Paths. In *IEEE International Conference on Communications (ICC)*, pages 874–879, Atlanta, Georgia, USA, June 1998.

[45] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile Agents: Are They a Good Idea? RC 19887, IBM, Yorktown Heights, New York, USA, 1994. (December 21, 1994 - Declassified March 16, 1995).

[46] Chen-Nee Chuah, Lakshminarayanan Subramanian, Randy H. Katz, and Anthony D. Joseph. QoS provisioning using a clearing house architecture. In *8th International Workshop on Quality of Service (IWQoS)*, pages 115–126, Pittsburgh, Pennsylvania, USA, June 2000.

[47] Martin Collier. Mobile Agents and Active Networks: Complementary or Competing Technologies? Technical report, Dublin City University, Dublin, Ireland, 1998.

[48] Martin Collier. Netlets: the future of networking? In *1st IEEE Conference on Open Architectures and Network Programming*, San Francisco, California, USA, April 1998.

[49] Douglas E. Comer. *Computer Networks and Internets*. Prentice Hall, 3rd edition, 2001. ISBN: 0-13-091449-5.

[50] Alex Conta, Paul Doolan, and Andrew G. Malis. Use of Label Switching on Frame Relay Networks Specification. RFC 3034, IETF, January 2001. Status: STANDARDS TRACK.

[51] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *3rd annual ACM symposium on Theory of computing*, pages 151–158, Shaker Heights, Ohio, USA, 1971. ACM Press.

[52] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge , Massachusetts London, England, 2nd edition, 2001. ISBN: 0-262-03293-7.

[53] Eric S. Crawley, Raj Nair, Bala Rajagopalan, and Hal Sandick. A Framework for QoS-based Routing in the Internet. RFC 2386, IETF, August 1998. Status: INFORMATIONAL.

[54] Bruce Davie, Jeremy Lawrence, Keith McCloghrie, Eric Rosen, George Swallow, Yakov Rekhter, and Paul Doolan. MPLS using LDP and ATM VC Switching. RFC 3035, IETF, January 2001. Status: STANDARDS TRACK.

[55] Hans De Nerve and Piet Van Mieghem. TAMCRA: A Tunable Accuracy Multiple Constraints Routing Algorithm. *Computer Communications*, 23:667–679, 2000.

[56] Dan Decasper and Bernhard Plattner. DAN: Distributed Code Caching for Active Networks. In *IEEE Conference on Computer Communications (INFOCOM)*, volume 2, pages 609–616, San Francisco, California, USA, April 1998.

[57] Kalaiarul Dharmalingam and Martin Collier. An Active Network Solution to RSVP Reservation Gaps. In *IEE London Communication Symposium*, London, England, September 2002.

[58] Kalaiarul Dharmalingam, Karol Kowalik, and Martin Collier. RSVP Reservation Gaps: Problems and Solutions. In *IEEE International Conference on Communications (ICC)*, volume 3, pages 1590–1595, Anchorage, Alaska, USA, May 2003.

[59] Gianni Di Caro and Marco Dorigo. Mobile Agents for Adaptive Routing. In *31st Hawaii International Conference on System Science (HICSS)*, Kohala Coast, Hawaii, USA, January 1998.

[60] Edsger Wybe Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[61] Jeff Dike. User Mode Linux. Running Linux on Linux. *Linux Magazine*, April 2001. `http://www.linux-mag.com/2001-04/user_mode_01.html`.

[62] Radu Dragoş, Sanda Dragoş, and Martin Collier. Design and implementation of an MPLS based load balancing architecture for Web switching. In *15th ITC Specialist Seminar*, pages 24–32, Würzburg, Germany, July 2002.

[63] William N. Eatherton. Hardware-based Internet Protocol Prefix Lookups. Master's thesis, Washington University Electrical Engineering Department, 1999.

[64] Mohamed El-Darieby, Dorina C. Petriu, and Jerry Rolia. A Hierarchical Distributed Protocol for MPLS path creation. In *7th IEEE International Symposium on Computers and Communications (ISCC)*, pages 920–926, Taormina, Italy, July 2002.

[65] Adrian Farrel. *The Internet and Its Protocols: A Comparative Approach*. Morgan Kaufmann, April 2004. ISBN: 155860913X.

[66] Giuseppe Di Fatta, Salvatore Gaglio, Giuseppe Lo Re, and Marco Ortolani. Adaptive Routing in Active Networks. In *3rd IEEE Conference on Open Architectures and Network Programming (OpenArch)*, Tel-Aviv, Israel, March 2000.

[67] Werner Feibel. *Encyclopedia of Networking & Telecommunications*. The Network Press, 3rd edition, November 1999. ISBN: 0-7821-2255-8.

[68] Lester Randolph Ford and Delbert Ray Fulkerson. Flows in Networks. *Princeton University Press*, 1962.

[69] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, 1979. ISBN: 0716710455.

[70] Richard J. Gibbens, Frank P. Kelly, and Peter B. Key. Dynamic Alternative Routing - Modelling and Behavior. In *12th International Teletraffic Congress*, Torino, Italy, June 1988.

[71] Sergio González-Valenzuela. QoS-Routing for MPLS Networks through Mobile Processing. Master's thesis, Electrical and Computer Engineering, Faculty of Graduate Studies, University of British Columbia, January 2002.

[72] Sergio González-Valenzuela and Victor C. M. Leung. Qos routing for mpls networks employing mobile agents. *IEEE Network Magazine*, 16(2):16–21, May/June 2002.

[73] Sergio González-Valenzuela, Victor C. M. Leung, and Son T. Vuong. Multipoint-to-Point Routing with QoS Guarantees Using Mobile Agents. In *3th International Workshop on Mobile Software Agents (MATA)*, pages 63–72, Montreal, Canada, August 2001.

[74] Sergio González-Valenzuela and Son Vuong. Evaluation of Migration Strategies for Mobile Agents in Network Routing. In *4th International Workshop on Mobile Software Agents (MATA)*, pages 141–150, Barcelona, Spain, October 2002.

[75] Pierre-Paul Grassé. La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La theorie de la stigmergie: Essai d'interpretation des termites constructeurs. *Insectes Sociaux*, 6(1):41–81, 1959.

[76] Eric W. Gray. *MPLS. Implementing the technology*. Addison-Wesley, 2001. ISBN: 0-201-65762-7.

[77] Shaw Green, Leon Hurst, Brenda Nangle, Padraig Cunningham, Fergal Somers, and Richard Evans. Software Agents: A Review. Technical Report TCS-CS-1997-06, Trinity College, Dublin, Ireland, 1997.

[78] Roch Guerin and Ariel Orda. QoS-based Routing in Networks with Inaccurate Information: Theory and Algorithms. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 75–83, Kobe, Japan, April 1997.

[79] Fang Hao and Ellen W. Zegura. Scalability Techniques in QoS Routing. Technical Report GIT-CC-99-04, College of Computing, Georgia Institute of Technology, 1999.

[80] Fang Hao and Ellen W. Zegura. On Scalable QoS Routing: Performance Evaluation of Topology Aggregation. In *21st IEEE Conference on Computer Communications (INFOCOM)*, volume 1, pages 147–156, Tel-Aviv, Israel, March 2000.

[81] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A Programming Language for Active Networks. *3rd ACM SIGPLAN International Conference on Functional Programming Languages*, 34(1):86–93, 1999.

[82] Geoff Huston. Next Steps for the IP QoS Architecture. RFC 2990, IETF, November 2000. Status: INFORMATIONAL.

[83] IEEE Communication Society. *A Brief History of Communications*, chapter Communications Technology: Before 1952–21st Century, pages 4–44. IEEE, 2002. ISBN: 0-7803-98825-4.

[84] J. M. Jaffe. Algorithms for Finding Paths with Multiple constraints. *Networks Magazine*, 14:95–116, 1984.

[85] Bilel Jamoussi, Loa Andersson, Ross Callon, Ram Dantu, Liwen Wu, Paul Doolan, Tom Worster, Nancy Feldman, Andre Fredette, Muckai K. Girish, Eric Gray, Juha Heinanen, Timothy E. Kilty, and Andrew G. Malis. Constraint-Based LSP Setup using LDP. RFC 3212, IETF, January 2002. Status: STANDARDS TRACK.

[86] Antti Kankkunen. MPLS and Next Generation Access Networks. In *1st European Conference on Universal Multiservice Networks (ECUMN)*, pages 5–16, Colmar, France, October 2000.

[87] Antti Kankkunen. Extending MPLS Service Edge Closer to the Customer. In *MPLS World Congress*, Paris, France, February 2004.

[88] Ioannis N. Kassabalidis, Mohamed A. El-Sharkawi, Robert J. Marks II, Payman Arabshahi, and Andrew A. Gray. Swarm Intelligence for Routing in Communication Networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, San Antonio, Texas, November 2001.

[89] Yasuhiro Katsube, Ken ichi Nagami, and Hiroshi Esaki. Toshiba's Router Architecture Extensions for ATM: Overview. RFC 2098, IETF, February 1997. Status: INFORMATIONAL.

[90] Yu-Kung Ke and John A. Copeland. Aggregation Algorithms for Asymmetric QoS-Routing Information. In *IEEE Global Telecommunications Conference (GLOBECOM)*, volume 4, pages 2209–2214, San Antonio, Texas, USA, November 2001.

[91] Srinivasan Keshav. *An Engineering Approach to Computer Networking: ATM Networks, the Internet and the Telephone Network.* Addison-Wesley, 1997. ISBN: 0-201-63442-2.

[92] Turgay Korkmaz and Marwan Krunz. Source-Oriented Topology Aggregation with Multiple QoS Parameters in Hierarchical ATM Networks. In *7th International Workshop on Quality of Service (IWQoS)*, pages 137–146, London, UK, January 1999.

[93] Turgay Korkmaz and Marwan Krunz. A randomised algorithm for finding a path subject to multiple QoS. *Computer Networks*, 36(2-3):251–268, 2001.

[94] Turgay Korkmaz and Marwan Krunz. Multi-Constrained Optimal Path Selection. In *22st IEEE Conference on Computer Communications (INFOCOM)*, volume 2, pages 834–843, Anchorage, Alaska, April 2001. ISBN:0-8186-8061-X.

[95] David Kotz, Robert Gray, and Daniela Rus. Transportable Agents Support Worldwide Applications. In *the 7th ACM SIGOPS European Workshop*, pages 41–48, Connemara, Ireland, September 1996.

[96] Karol Kowalik. *Design Issues in Quality of Service Routing*. PhD thesis, School of Electronic Engineering, Dublin City University, Dublin, Ireland, January 2004.

[97] Kwindla Hultman Kramer, Nelson Minar, and Pattie Maes. Tutorial: Mobile Software Agents for Dynamic Routing. *Mobile Computing and Communications Review*, 3(2):12–16, 1999.

[98] Fernando A. Kuipers, Turgay Korkmaz, Marwan Krunz, and Piet Van Mieghem. An Overview of Constraint-Based Path Selection Algorithms for QoS Routing. *IEEE Communications Magazine*, 40(12):50–55, December 2002. special issue on IP-Oriented Quality of Service.

[99] Fernando A. Kuipers, Turgay Korkmaz, Marwan Krunz, and Piet Van Mieghem. Performance Evaluation of Constraint-Based Path Selection Algorithms. *IEEE Network Magazine*, 18(5):16–23, September/October 2004.

[100] Danny B. Lange and Mitsuru Oshima. Seven Good Reasons for Mobile Agents. *Communications of the ACM*, 42(3):88–89, March 1999.

[101] Mark Laubach. Classical IP and ARP over ATM. RFC 1577, IETF, January 1994. Status: STANDARDS TRACK.

[102] Mark Laubach and Joel Halpern. Classical IP and ARP over ATM. RFC 2225, IETF, April 1998. Status: STANDARDS TRACK.

[103] Scott Seongwook Lee, Shirshanka Das, Giovanni Pau, and Mario Gerla. A Hierarchical Multipath Approach to QoS Routing: Performance and Cost Evaluation. In *IEEE International Conference on Communications (ICC)*, Anchorage, Alaska, USA, May 2003.

[104] Whay C. Lee. Spanning Tree Method for Link State Aggregation in Large Communication Networks. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 297–302, Boston, Massachusetts, USA, April 1995. ISBN: 0-8186-6990-X.

[105] Whay C. Lee. Topology Aggregation for Hierarchical Routing in ATM Networks. *ACM SIGCOMM Computer Communication Review*, 25(2):82–92, April 1995.

[106] Whay C. Lee, Michael G. Hluchyi, and Pierre A. Humbert. Routing subject to Quality of Service Constraints. *IEEE Integrated Communication Networks*, 9(4):46–55, July/August 1995.

[107] Ulana Legedza, David Wetherall, and John Guttag. Improving the Performance of Distributed Applications Using Active Networks. In *IEEE Conference on Computer Communications (INFOCOM)*, volume 2, pages 590–599, San Francisco, California, USA, April 1998.

[108] Tony Li. MPLS and the Evolving Internet Architecture. *IEEE Communication Magazine*, 37(12):38–41, December 1999.

[109] Tony Li and Yakov Rekhter. A Provider Architecture for Differentiated Services and Traffic Engineering (PASTE). RFC 2430, IETF, October 1998. Status: INFORMATIONAL.

[110] Ying-Dar Lin, Nai-Bin Hsu, and Ren-Hung Hwang. QoS Routing Granularity in MPLS Networks. *IEEE Communications Magazine*, pages 58–65, June 2002.

[111] Steffen Lipperts and Birgit Kreller. Mobile Agents in Telecommunications Networks - A Simulative Approach to Load Balancing. In *5th International Conference on Information Systems, Analysis and Synthesis (ISAS)*, pages 231–238, Orlando, Florida, USA, 1999.

[112] Gang Liu and K. G. Ramakrishnan. A*Prune: an algorithm for finding k shortest paths subject to multiple constraints. In *20th IEEE Conference on Computer Communications (INFOCOM)*, pages 743–749, Anchorage, Alaska, April 2001.

[113] James V. Luciani, Dave Katz, David Piscitello, Bruce Cole, and Naganand Doraswamy. NBMA Next Hop Resolution Protocol (NHRP). RFC 2332, IETF, April 1998. Status: STANDARDS TRACK.

[114] King-Shan Lui, Klara Nahrstedt, and Shigang Chen. Hierarchical QoS Routing in Delay-Bandwidth Sensitive Networks. In *25th Annual IEEE Conference on Local Computer Networks (LCN)*, pages 579–588, Tampa, Florida, USA, November 2000.

[115] Qingming Ma. *QoS Routing in the Integrated Services networks*. PhD thesis, CMU-CS-98-138, January 1998.

[116] Qingming Ma and Peter Steenkiste. On Path Selection for Traffic with Bandwidth Guarantees. In *5th International Conference on Network Protocols (ICNP)*, pages 191–202, Atlanta, Georgia, USA, October 1997. IEEE Computer Society. ISBN:0-8186-8061-X.

[117] Qingming Ma and Peter Steenkiste. Quality-of-Service Routing with Performance Guarantees. In *4th International IFIP Workshop on Quality of Service (IWQoS)*, pages 115–126, New York, USA, May 1997.

[118] Gary Scott Malkin. RIP Version 2. RFC 2453, IETF, November 1998. Status: STANDARDS TRACK.

[119] Nicholas F. Maxemchuk and Steven H. Low. Active Routing. *IEEE Journal on Selected Areas in Communications (JSAC)*, 19(3):552–565, March 2001.

[120] Daniel A. Menascé and E. Casalicchio. QoS in Grid Computing. *IEEE Internet Computing*, 8:85–87, July/August 2004.

[121] Nelson Minar, Kwindla Hultman Kramer, and Pattie Maes. *Cooperating Mobile Agents for Dynamic Network Routing*, chapter 12. Springer-Verlag, 1999. ISBN: 3-540-65578-6.

[122] John Moy. OSPF Version 2. RFC 2328, IETF, April 1998. Status: STANDARDS TRACK.

[123] Ken-ichi Nagami, Yasuhiro Katsube, Yasuro Shobatake, Akiyoshi Mogi, Shigeo Matsuzawa, Tatsuya Jinmei, and Hiroshi Esaki. Toshiba's Flow Attribute Notification Protocol (FANP) Specification. RFC 2129, IETF, April 1997. Status: INFORMATIONAL.

[124] Peter Newman, W. L. Edwards, Robert M. Hinden, Eric Hoffman, Fong Ching Liaw, Tom Lyon, and Greg Minshall. Ipsilon Flow Management Protocol Specification for IPv4 Version 1.0. RFC 1953, IETF, May 1996. Status: INFORMATIONAL.

[125] Peter Newman, W. L. Edwards, Robert M. Hinden, Eric Hoffman, Fong Ching Liaw, Tom Lyon, and Greg Minshall. Ipsilon's General Switch Management Protocol Specification Version 1.1. RFC 1987, IETF, August 1996. Status: INFORMATIONAL.

[126] Peter Newman, Greg Minshall, and Tom Lyon. IP Switching: ATM Under IP. *IEEE/ACM Transactions on Networking*, 6(2):117–129, April 1998.

[127] Kathleen Nichols, Steven Blake, Fred Baker, and David L. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, IETF, December 1998. Status: STANDARDS TRACK.

[128] Ariel Orda. Routing with end-to-end QoS guarantees in broadband networks. *IEEE/ACM Transactions on Networking (TON)*, 7(3):365–374, 1999. ISSN: 1063-6692.

[129] Pragyansmita Paul and S. V. Raghavan. Survey of QoS routing. In *15th IEEE International Conference on Computer Communication (ICC'02)*, volume 1, pages 50–75, Mumbai, Maharashtra, India, 2002.

[130] Konstantinos Psounis. Active Networks: Applications, Security, Safety, and Architectures. *IEEE Communications Surveys*, 2(1):1–16, 1999.

[131] Wenyu Qu and Hong Shen. Some Analysis on Mobile-Agent Based Network Routing. In *International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, pages 12–17, Hong Kong, China, May 2004.

[132] Yakov Rekhter, Bruce Davie, Dave Katz, Eric Rosen, and George Swallow. Cisco Systems' Tag Switching Architecture Overview. RFC 2105, IETF, February 1997. Status: INFORMATIONAL.

[133] Yakov Rekhter and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, IETF, March 1995. Status: STANDARDS TRACK.

[134] Eric C. Rosen, Dan Tappan, Yakov Rekhter, Guy Fedorkow, Dino Farinacci, Tony Li, and Alex Conta. MPLS Label Stack Encoding. RFC 3032, IETF, January 2001. Status: STANDARDS TRACK.

[135] Eric C. Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol Label Switching Architecture. RFC 3031, IETF, January 2001. Status: STANDARDS TRACK.

[136] Paul "Rusty" Russell. Netfilter: Packet Mangling in 2.4. In *6th International Linux Kongress*, Augsburg, Germany, September 1999.

[137] Paul "Rusty" Russell. Writing a Module for netfilter. *Linux Magazine*, June 2000. `http://www.linux-mag.com/2000-06/gear_01.html`.

[138] Paul "Rusty" Russell and Harald Welte. Linux netfilter Hacking HOWTO. `http://www.netfilter.org/documentation/HOWTO/ /netfilter-hacking-HOWTO.html`.

[139] Peter S. Sapaty. The WAVE paradigm. Technical Report 17/92, Dept. of Informatics, Univ. of Karlsruhe, Karlsruhe, Germany, July 1992. Also published in Proc. Post-Conference Joint Workshop on Distributed and Parallel Implementations of Logic Programming Systems, JICSLP'92, pages 106-148, Washington, D. C., Nov. 13-14, 1992.

[140] Peter S. Sapaty. A brief introduction to the WAVE language. Technical Report 8/93, Dept. of Informatics, Univ. of Karlsruhe, Karlsruhe, Germany, February 1993.

[141] Peter S. Sapaty. Mobile Wave Technology for Distributed Knowledge Processing in Open Networks. In *4th International Conference on Information and Knowledge Management (CIKM): Workshop on New Paradigms in Information Visualization and Manipulation*, Baltimore, Maryland, USA, November/December 1995.

[142] Peter S. Sapaty. Mobile processing in open systems. In *5th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, Syracuse, New York, USA, August 1996.

[143] Peter S. Sapaty. *Mobile Processing in distributed and Open Environments*. Wiley, 2000. ISBN: 0-471-19572-3.

[144] Peter S. Sapaty. Organisation and Management of Distributed Dynamic Systems in WAVE. In *Symposium on Unmanned Systems*, Baltimore, MD, USA, July 2001.

[145] Peter S. Sapaty and P. M. Borst. An overview of the WAVE language and system for distributed processing in open networks. Technical report, Dept. of Electronic and Electrical Eng., Univ. of Surrey, Surrey, UK, June 1994.

[146] Peter S. Sapaty and Peter Borst. WAVE: Mobile Intelligence in Open Networks. In *1st Annual Conference of Emerging Technologies and Applications in Communications (ETACOM)*, Portland, Oregon, USA, May 1996. IEEE Computer Society Press.

[147] Beverly Schwartz, Wenyi Zhou, Alden W. Jackson, W. Timothy Strayer, Dennis Rockwell, and Craig Partridge. Smart packets for active networks.

In *IEEE Open Architectures and Network Programming (OPENARCH)*, pages 90–97, New York, USA, March 1999.

[148] Anees Shaikh, Jennifer Rexford, and Kang G. Shin. Efficient Precomputation of Quality-of-Service Routes. In *8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 15–27, New Hall, Cambridge, UK, July 1998.

[149] Scott Shenker, Craig Partridge, and Roch Guerin. Specification of Guaranteed Quality of Service. RFC 2212, IETF, September 1997. Status: STANDARDS TRACK.

[150] Kwang Mong Sim and Weng Hong Sun. Multiple Ant-Colony Optimisation for Network Routing. In *1st IEEE International Symposium on Cyber Worlds (CW)*, pages 277–281, Tokyo, Japan, November 2002. ISBN: 0-7695-1862-1.

[151] John Sum, Hong Shen, Chi sing Leung, and Gilbert H. Young. Analysis on a Mobile Agent-Based Algorithm for Network Routing and Management. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):193–202, March 2003.

[152] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003. ISBN: 0-13-066102-3.

[153] David L. Tennenhouse, Stephen J. Garland, L. Shrira, and M. Frans Kaashoek. From Internet to ActiveNet. Request for comments, MIT, January 1996.

[154] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[155] David L. Tennenhouse and David J. Wetherall. Towards an Active Network Architecture. *Computer Communication Review*, 26(2):5–18, April 1996.

[156] Bob Thomas and Eric Gray. LDP Applicability. RFC 3037, IETF, January 2001. Status: INFORMATIONAL.

[157] Luca Valcarenghi. Quality of Service in Global Grid Computing. In *13th Symposium on High Performance Interconnects (HOTI'05)*, pages 4–5, Stanford University, Stanford, California, USA, 2005.

[158] Piet Van Mieghem and Hans De Nerve. Hop-by-hop quality of service routing. *Computer Networks*, 37(3-4):407–423, October 2001.

[159] Arun Viswanathan, Nancy Feldman, Rick Boivie, and Rich Woundy. ARIS: Aggregate Route-Based IP Switching. DRAFT viswanathan-aris-overview-00, IETF, March 1997. Status: INTERNET DRAFT. Expiration Date: September 1997.

[160] Son T. Vuong and Ivailo Ivanov. Mobile Intelligent Agent Systems: WAVE vs. JAVA. In *1st Annual Conference of Emerging Technologies and Applications in Communications (ETACOM)*, Portland, Oregon, USA, May 1996.

[161] Zheng Wang and Jon Crowcroft. QoS Routing for Supporting Resource Reservation. *IEEE Journal on Selected Areas in Communications (JSAC)*, 14(7):1288–1294, September 1996.

[162] Zheng Wang and Jon Crowcroft. Quality of Service Routing for Supporting Multimedia Applications. *IEEE Journals on Selected Areas in Communications (JSAC)*, 14(7):1228–1234, September 1996.

[163] Jeff Weldon, Hui Zhang, and Li Lin. IP over ATM with Scalability and QOS. Tag Switching vs. the IP Switch. Project Report EE228A, University of California Berkeley, Electrical Engineering and Computer Science, 1996.

[164] Michael Welzl, Alfred Cihal, and Max Mühlhäuser. An Approach to Flexible QoS Routing with Active Networks. In *4th IEEE Annual International Workshop on Active Middleware Services (AMS)*, pages 75–82, Edinburgh, UK, July 2002.

[165] Michael Welzl, Leopold Franzens, and Max Mühlhäuser. Scalability and Quality of Service: A trade-off? *IEEE Communications Magazine*, 41, 2003.

[166] David J. Wetherall, John V. Guttag, and David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE Open Architectures and Network Programming (OPENARCH)*, San Francisco, California, USA, April 1998.

[167] David J. Wetherall and David L. Tennenhouse. The ACTIVE IP Option. In *7th ACM SIGOPS European Workshop*, Connemara, Ireland, September 1996.

[168] James E. White. Telescript technology: The foundation of the electronic market place. General Magic white paper, 1995.

[169] Johnny S.K. Wong and Armin R. Mikler. Intelligent mobile agents in large distributed autonomous cooperative systems. *Journal of Systems and Software*, 47:75–87, 1999.

[170] John Wroclawski. Specification of the Controlled-Load Network Element Service. RFC 2211, IETF, September 1997. Status: STANDARDS TRACK.

[171] Xipeng Xiao and Lionel M. Ni. Internet QoS: A Big Picture. *IEEE Network Magazine*, 13(2):8–18, April 1999.

[172] Yechiam Yemini and Sushil Da Silva. Towards programmable networks. In *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italy, October 1996.

[173] Xin Yuan. Heuristics algorithms for multi-constrained quality-of-service routing. *IEEE/ACM Transactions on Networking*, 10(2):244–256, 2002.

[174] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to Model an Internetwork. In *IEEE Conference on Computer Communications (INFOCOM)*, volume 2, pages 594–602, San Francisco, California, USA, March 1996.

[175] Ellen W. Zegura, Kenneth L. Calvert, and Michael J. Donahoo. A Quantitative Comparison of Graph-based Models for Internet Topology. *IEEE/ACM Transactions on Networking*, 5(6):770–783, December 1997.

[176] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network Magazine*, 7(5):8–18, September 1993.