QoS-ROUTING FOR MPLS NETWORKS THROUGH MOBILE PROCESSING

By

SERGIO GONZALEZ VALENZUELA

B.E. (E.E.) Instituto Teconológico de Sonora, Ciudad Obregón, México, 1995

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

January 2002

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical & Comp Eng.

The University of British Columbia
Vancouver, Canada

Date 25/JAN/02

DE-6 (2/88)

Abstract

The Multi-Protocol Label Switching technology promises to introduce enhanced features to the Internet in the near future, while contributing to the overall end-to-end provision of Quality of Services. However, to achieve such an improvement, this new technology requires enhanced capabilities from routing algorithms currently available.

A routing system based on the use of mobile software agents is presented, which offers comparable or better results than those obtained using classical routing paradigms. The proposed scheme overcomes the problems faced by existing routing algorithms, and adapts well to the needs of future networking technologies that support the overall QoS provision.

Simulations suggest that the use of mobile software agents facilitate the discovery of routes that meet the Quality of Service constraints, while obtaining efficient routing results based on real-time information. It can be shown that a routing scheme based in mobile processing can minimize the computational complexity of the algorithms significantly. Results also show that the time spent by the routing algorithm during the discovery of QoS-compliant paths follows a logarithmic pattern, as the number of nodes requiring service from the routing algorithm also increases.

Table of Contents

List of Figures

Acknowledgements

I dedicate this work to both my wife for her unconditional support through good and rough times, and my son for bringing so much happiness to our family. My endless appreciation goes also to my parents, my sister, her family, and my wife's family for their constant support and sound advise. I love you all.

I sincerely thank Dr. Victor Leung for his fine supervision, invaluable guidance and constant patience, during my thesis work at UBC. I would also like to thank Dr. Son Vuong for his positive influence and contagious enthusiasm, as well as professors Cyril Leung and Ed Casas and for their interesting lectures. I express my gratitude to my friends in México and my classmates at UBC for those interesting talks, either technical or non-technical. My gratitude goes also to Kristy Barclay for helping to edit my thesis.

I am endlessly grateful to CONACYT México and the NSERC of Canada for partially supporting my graduate studies at UBC, and allowing me to achieve one of my lifetime goals.

Finally, I want to take a moment to remember those beloved ones who departed this world before us. There will always be a special place for you all in our hearts and in our memory.

Thank you all.

Sergio González-Valenzuela

Chapter 1 Introduction

The objective of this chapter is to provide to the reader preliminary information on the main ideas informing this thesis. The reader will then have the context of the general concepts of this investigation, which will be described in more detail through chapters 2 and 3. This work deals with concepts that apply to the area of telecommunications networks, and specifically on backbone networks such as the Internet core. This work encompasses four major areas of study in network communications: Multi-Protocol Label Switching (MPLS), Quality of Services (QoS), network routing, and mobile software agents. To understand how these technologies can interact together to reach the goal, it is first necessary to understand, to a sufficient level, how the networking technology operates and how it has evolved over the last few years. This will help to explain how these new technologies became major areas of current research, as well as their strengths and relative weaknesses.

Background on routing will be provided in section 1.1, which briefly explains the importance of routing in communication networks, as well as the routing schemes currently used, although a much more detailed presentation will be given in chapter 2. Section 1.2 provides the motivation of this research work and illustrates the importance of creating new schemes to address routing issues for future data networks. This topic will be explained in detail throughout chapter 3. In section 1.3, the objectives and goals of the research will be introduced, and section 1.4 will present an abstract of what is to come in subsequent chapters.

1.1 Preliminary Background on Routing

Routing is one of the major tasks in a communications network. It has been described as a procedure used to determine the best next hop for a data packet traversing a network, and bounded to a specific destination [52]. This is accomplished by forwarding data across a given network depending on previous computations, while seeking to optimize network resources *and* attempting to meet the data requirements [12], [31], [52]. It can be said that there is a trade off involved in the routing procedure; this is because users would usually want data carried across a network as fast and as reliably as possible. However, limitations imposed by network resources imply some cost to honouring the user's request. Therefore, on current data networks such as the Internet, user data is transported from source to destination according to a best effort scheme: the network will attempt to carry data across the network as fast and as reliably as possible, but no guarantees are made.

This method might have been sufficient to cope with the necessities of early data networks, such as the ARPANET, the precursor of the Internet. In this early network, routing was primarily concerned with connectivity issues, and the intention of transporting data to its desired destination [52]. No attempts were made to provide guarantees to user data, other than to getting data across the network to its final destination.

As technology has evolved, so the needs of modern society needs have imposed new challenges, requiring current networks to provide better services that meet actual communications needs. Such needs range now from audio and video (multimedia) applications, to high-speed data transfers, high-profile telemetry, and control of remote devices, among others. Connectivity is nowadays basically taken for granted, and is no longer in question, while better quality of network services is. This has lead to the introduction of the Quality-of-Service (QoS) concept in communication networks over the past decade. A number of frameworks have been proposed

since then [8], [96], and some of those are being introduced and experimentally developed for the near-future implementation of new communication technologies. Advances in communications hardware have opened the gate for a new era of information technology, which has also contributed to support for new proposals that will finally attempt to meet QoS constraints.

As the need for data transport across the network has evolved, one might expect that the old routing strategies would have evolved as well to cope with the new necessities. However, it will be later seen that this has not necessarily been the case.

1.2 Motivation

After carrying out a comprehensive survey, it was observed that routing has not grown in parallel to the growth of other networking technologies. A number of extensions and patches have been proposed to current routing schemes in order to keep up with the needs previously mentioned [4], [57]. Such routing schemes are still being used and/or explored to carry on for use in future network technologies. Getting data across complex networks, such as the Internet, using the same old routing schemes will become a major problem for the future networking technologies. Quality of Service frameworks, such as the Differentiated Services and Integrated Services architectures, will help provide a state-of-the-art QoS control mechanism. Also, new forwarding schemes, such as MPLS, will provide the means to perform high-speed data forwarding. Therefore, it is imperative to realize the rising need for new proposals to overcome issues facing existing routing protocols. It is also imperative to be aware that the global QoS provision cycle won't be closed until new routing mechanisms are studied and concurrently implemented to achieve a parallel evolution of network technologies.

Another motivation for this work is the need to address the routing problem in a different manner. As will be seen in later chapters, current routing schemes rely on the estimation of best routes by locally computing data that was gathered throughout the network in a distributed manner. One can easily see how this paradigm could become self-constricting, as the requirements for network support reveal themselves to be highly demanding, making the complexity of the routing computations intractable [43], [96]. It is reasonable to predict that future network implementations will impose greater demands on routing protocol performance, as well as a higher computing burden on network nodes, which will likely cause current routing schemes to struggle. On the other hand, there is the mobile software agents' paradigm, which has gained a good deal of attention in recent years. Important research efforts have been performed to study the potential of their implementation in different fields, including telecommunications. Therefore, it is believed that a novel and efficient approach can be put forward to contribute to the ongoing effort to find more efficient solutions to the routing problem.

Another important aspect observed during the survey conducted was that many of the approaches using mobile agent technology deal with solutions that could be considered predominantly theoretical. Thus, it is necessary to build and study communication systems that actually reflect the circumstances that concern the real problems found in current or forthcoming network architectures and QoS frameworks. Given this motivation, the objectives and goals of this thesis work are described next.

1.3 Objective and Goals

The primary objective of this work is to present and investigate an alternative solution to routing problems that supports the needs of near-future networking technology. To achieve this

objective, the potential of the mobile agents paradigm will be considered with regard to routing issues. Theoretical routing problems will be addressed as part of the research work; however, special emphasis will be given to practical considerations. In essence, the majority of observations are based on results obtained through simulations. Additional goals now follow:

- To design an efficient algorithm based on the mobile agents paradigm, with the purpose of finding alternative and effective ways to perform routing.
- The proposed solution should address the issue of scalability in communication networks. Specific emphasis will be placed on working with the Internet backbone infrastructure.
- Analytical approaches will be presented when possible. Since no mathematical models have been established to formally describe and model the behaviour of mobile software agents, the analysis presented will mostly involve traditional approaches addressed in graph theory, but adapted to the mobile agents paradigm.
- Results obtained by means of simulations should suffice to give a good understanding of the type of traffic generated by a routing scheme based on mobile agent technology.

1.4 Research Outline

In light of the objectives and goals just presented, an outline of the research work is given next.

1. A survey that deals with the interaction of a number of networking technologies will be presented in Chapter 2. Such technologies include MPLS, Diffserv, QoS, Routing and Mobile Agents.

2. Chapter 3 proposes how the previously mentioned technologies could mutually interact to achieve the final goal, while taking into account previous work reported on the subject throughout the literature.
3. A preliminary approach, based on mobile agents addressing the QoS-routing problem, will be presented in Chapter 4, which helps in understanding the basic characteristics of a routing solution based on mobile agents.
4. A second and more robust solution will be presented in Chapter 5, which includes enhanced features that enable it to run under dynamic circumstances.
5. A simple analysis will also be addressed in Chapter 5, after a presentation of the practical results of the simulations. This analysis will help to better explicate the characteristics of a routing solution based on mobile agent technology.
6. Finally, Chapter 6 concludes with some important considerations and final remarks.

Chapter 2 Routing Primitives in Data Networks

The objective of this chapter is to provide the reader with preliminary information on the routing problem in data communication networks, specifically, the Internet. As previously stated, this work will focus on networking technologies that have good potential for implementation in the near future in the Internet backbone. Routing in data networks has been widely addressed in the literature, and has been catalogued as a highly complex problem [80]. An essential background of how routing works will be presented in the first section, while the subsequent sections will address the interaction between routing, MPLS networks and Quality of Service. Finally, the last section will discuss a novel experimental routing approach using mobile software agents, including an introduction to the Wave technology. Because of the broad scope these topics cover, *the theory presented here will be mainly centred on the concepts relevant to this work.* Some familiarity with the concepts of routing and forwarding is required.

2.1 Routing on the Internet

As described in Chapter 1, the main issue facing earlier data networks was the provision of connectivity among their terminals [20]. A logical consequence of this circumstance was that the first data networks were also concerned with finding the shortest route between end-peers, so that a minimal number of resources would be used in any given data-transfer session. The first routing protocols made use of algorithms that work according to a concept known as *shortest path routing*, in which numerical values or "weights" are assigned to edges of the network to represent specific network metrics. The final objective is then to find the route with shortest

distance between two network nodes. A visual example of this concept is shown in the next figure:

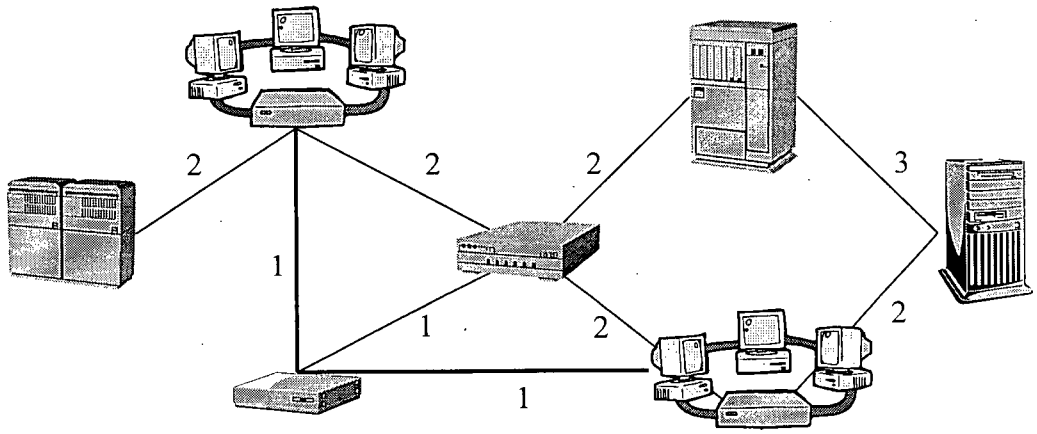


Figure 2.1 Shortest distances between sub-networks LAN A, and LAN B

Two basic algorithms designed under this premise were incorporated to perform routing in data networks: the Bellman-Ford algorithm [31], and Dijkstra's algorithm [26]. Since then, a number of routing algorithms have been studied and proposed; yet, the former algorithms prevail as the choice for implementation in routing protocols even today. To better comprehend the practical implications of using routing protocols that implement shortest path algorithms, it is necessary to examine the network structure of the Internet. This is addressed in the following section.

2.1.1 The Internet Backbone Network

After the creation of the first data network, known as the ARPANET, the National Science Foundation (NFS) in the United States provided support to create a backbone network, known as NFSNET [81]. By then, the Transmission Control Protocol (TCP) and the Internet Protocol (IP) had also been introduced to provide a method of interconnecting individual networks in either local or wide areas. This gave birth to the contemporary term of *internetworking*, which along with the NFSNET backbone, finally gave rise to what is now globally known as the *Internet*. Thus, the Internet is a set of *inter*connected *net*works, which gives the appearance of a single

larger network. The set of communication devices that provide the main physical connectivity among the networks comprising the Internet is commonly known as a *backbone*. As the primary bonds between individual networks, these devices must provide higher capacity in regards to the amount of information that can be carried across such links. A backbone link can be seen as a single communication line, but in most cases, it is actually a complete network. The current Internet backbone consists of a very large communications' infrastructure interconnecting Local Area Networks (LAN), Metropolitan Area Networks (MAN), and Wide Area Networks (WAN). A number of privately managed backbone-networks are deployed and interconnected around the globe.

2.1.2 Routing in Packet Switching Networks

This work focuses on issues regarding interconnectivity at the network layer level, represented by two important functions: routing and delivery. Explicit emphasis will be placed on routing, although the delivery, or *forwarding* function, will also be discussed later on this chapter during the introduction to the MPLS technology. It is first necessary to define the actual concepts of routing and forwarding.

- ***Routing:*** Defined as a decision made by a router to receive and forward data, based on current knowledge of network topology and state, incurring the lowest possible cost.
- ***Forwarding:*** Action taken by a network data-switch or router to dispatch incoming data to an outgoing link, based on a routing decision.

From the preceding definitions, it can be readily seen that the concepts of routing and forwarding are intimately related. There are three basic ways of delivering data packets across networks: datagram packet switching, virtual-circuit packet switching and circuit switching [80]; however, for this work, only the first two types will be addressed.

In packet switching networks (e.g. the Internet), user data is encapsulated (preceded) by additional information according to lower-layer protocol standards, in order to be transported to a final destination, as depicted in figure 2.2.

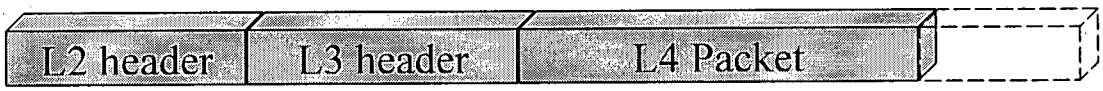


Figure 2.2 Higher-layer packet proceeded by lower-layer headers

In regards to network layer encapsulation, the Internet Protocol (IP) is used in conjunction with the Transmission Control Protocol (TCP) as the protocol suite to transfer data through the Internet, commonly known as the TCP/IP suite. The IP encapsulation header includes data fields that contain network addresses of both the source and destination. Each router along the transmission path uses the destination address contained in the IP header to make a routing decision, and then forwards the packet to the next hop to get it closer to the intended destination. Looking up the destination address of the IP packet in a routing table contributes to making routing decisions. A router obtains the entry associated with the destination address query being served, and determines an outgoing link for the data forwarding. This procedure is repeated as the packet travels through the network until it reaches the intended destination, as shown in figure 2.3.

For datagram packet switching, no path set-up procedure is previously arranged before any packet is actually transmitted, allowing packets with a similar source to travel through different paths toward a common destination. In the case of virtual-circuit packet switching, a predefined route is set-up before any data is forwarded across the network.

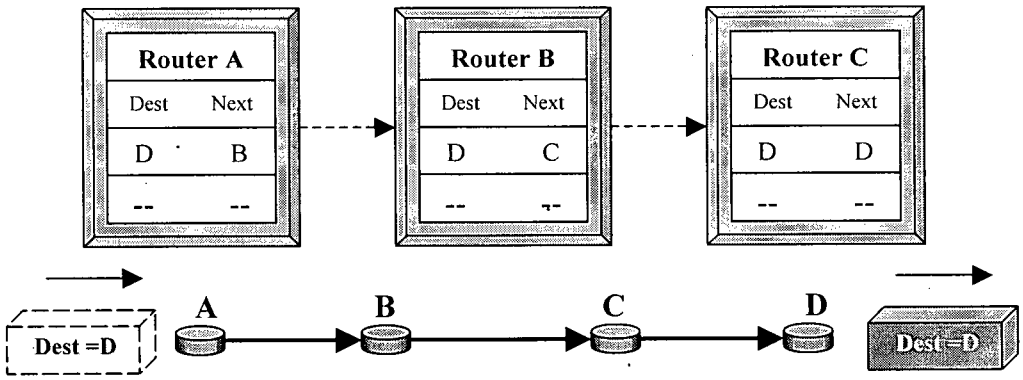


Figure 2.3 Routing-table entry lookup for packet delivery across a network

The computation of routing tables in each router relies in the exchange of remote network state information among the routers participating in the network. Each router determines its current data-traffic state and sends this information using a flooding strategy to all the other routers in a network [12]. This information is then used to compute the routing tables at each router. This procedure is known as distributed routing [80], and is performed in a hierarchical fashion, following either an intra-network, or an inter-network scheme.

2.2 The Multi-Protocol Label Switching Architecture

Presented in [71], Multi-Protocol Label Switching technology (MPLS) enables new and enhanced possibilities in the field of networking. MPLS emerges as an evolution of the label-swapping paradigm, originally introduced by ATM technology [66], [69], and then followed by Ipsilon's IP-switching [59], [60], and Cisco's Tag-switching [51], [70]. MPLS is not defined as a new layer in the communication model, but rather as a sub-layer. Figure 2.4 shows the place of MPLS in the OSI communication model.

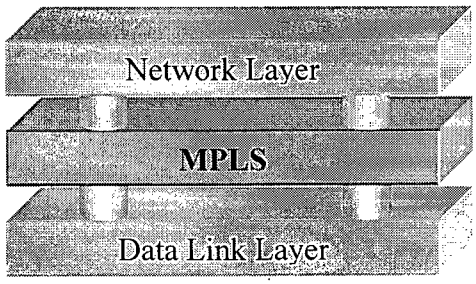


Figure 2.4 Place of MPLS in the OSI communications' model

In MPLS, data packets are labelled according to a pre-established agreement between two network switches, also known as Label Switching Routers (LSRs), and forwarded across the network, depending on the actual labels' identities. There are two ways to carry out the labelling of a data packet in MPLS:

a) by embedding the label into a L2 field designed for a similar purpose (e.g. ATM's VPI/VCI fields)

b) by adding a "shim layer" in between Layer-2 and Layer-3 encapsulations

Each method provides different features. When defined as a separate "shim" layer, the MPLS header consists of 32 bits, sub-divided into four fields: the label field (20 bits), the experimental field (3 bits), the stack bit (1 bit), and the time-to-live field (8 bits). When ATM is used as the L2 underlying protocol, the first three fields are encoded into the VPI/VCI fields of the ATM cell header (24 bits). The label field is where the actual label value is placed during the forwarding procedure; the experimental field is open for use with different purposes depending on the application, and the stack bit indicates whether there are additional labels placed in a stack to serve other purposes. The next figure shows the encapsulation format for MPLS labels.

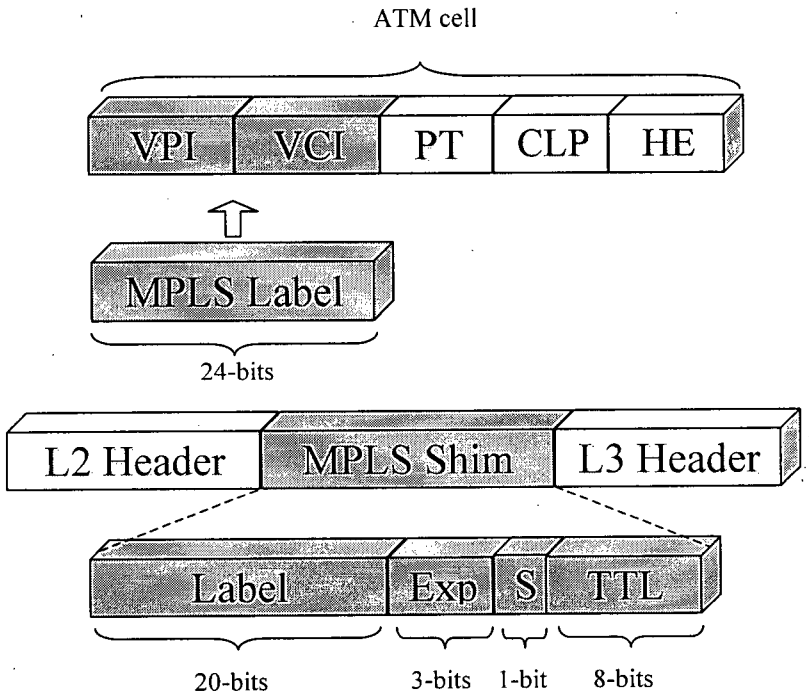


Figure 2.5 Higher-layer packet preceded by lower-layer headers

2.2.1 The MPLS Forwarding Paradigm

For the creation of the labelled packets in an MPLS network, in accordance with [71], each switch along a Label Switched Path (LSP) assigns a numeric value (i.e. a label) to each outgoing packet, depending on the destination. The actual value of the label is predetermined by a protocol carried out between each LSR and its peering nodes for which there is a direct connection to. Therefore, a given switch (say SW1) having a direct outgoing connection to another node (say SW2), requests a label that will be assigned to each packet traversing the physical link between SW1 and SW2 (i.e. the LSP). In this case, SW2 locally determines the value of the label depending on the outgoing link that the packet coming from SW1 will follow. This label is sent back to SW1 and stored in a table for further use. When a given packet arrives at SW1, it will detach the label from the packet, and look it up in a forwarding table. This indexed search will help to determine the label's value for the next outgoing link (e.g. towards SW2), which will in

turn be interpreted by the next node along the path being traversed. Each node in the network performs the same procedure. Fig. 2.6 helps to better describe this process.

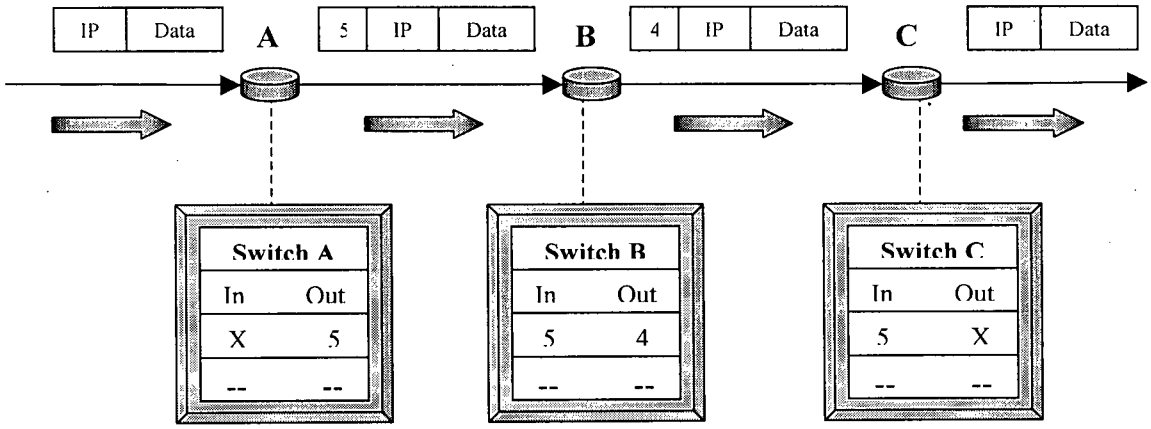


Figure 2.6 Data forwarding procedure in MPLS

Packets can be assigned different labels according to diverse premises: source/destination, type of message being transmitted, network treatment required and so forth, which opens a broad field of new possibilities. Data packet grouping in MPLS gives rise to the concept of a Forwarding Equivalence Class (FEC), which means that a number of data packets sharing similar characteristics and requiring the same level of commitment from the network can be grouped into classes, so that a single label can be used for the differentiation of packets while they traverse the LSP [71]. Once the label or FEC of the packet has been assigned, no further operations along the LSP are necessary within the MPLS network, besides the actual forwarding function. Thus, MPLS permits a higher level of data management in networks, providing ease of implementation a number of more complex tasks: explicit routing, quality of service (QoS) support, Virtual Private Networks (VPNs) design, multicasting, and multipoint-to-point connections [5], [6], [9], [21], [37], [39], [71], [98].

2.2.2 Distribution of Labels

So far, only the general concept of label swapping in MPLS networks has been presented; however, it is necessary to elaborate on the label distribution techniques between participating nodes in such a network [2]. First, it is important to state that the label distribution in MPLS networks is always done in a *downstream* fashion, which means that an LSR is located downstream with respect to a label-FEC binding, as previously explained.

In essence, two main types of label distribution techniques can be used in MPLS networks: downstream on-demand and downstream unsolicited [6], [39], [71]. If unsolicited, an LSR assigns a label for each of the entries stored in its own routing table according to a predefined premise (packet's precedence), and then it distributes this label/FEC binding to LSRs located downstream with respect to these bindings. This means that for every possible hop-to-hop route available with a given precedence, there may be an LSP already assigned to it. If on-demand, an LSR makes an explicit label request for each of the entries stored in its routing table, which causes the LSR to assign a label/FEC binding at the time of request. It then it responds with the requested label to the LSR that requested it. The labels stored in each LSR will be available for use on any packet that needs to be transmitted across the MPLS network.

In the simplest case, a given packet will have its label stripped when reaching an LSR, and will have a new one assigned according to its precedence and destination, following classic label-swapping handling. In specific cases, a packet might carry a stack of labels, which are determined by a Next Hop Label Forwarding Entry (NHLFE), giving indications on how to treat the label stack for proper packet forwarding [71]. This feature can be used to forward specific control messages within the MPLS network to establish pre-defined LSPs. This procedure facilitates the technique known as *explicit routing*, enabling a number of features mentioned before, which will be further explained next.

2.2.3 MPLS Approach to High-Speed Networking

So far, an elemental introduction of MPLS has been made with regards to its functionality, as well as some basic features. The most relevant MPLS applications are these:

- Virtual Private Networks (VPNs): When campuses of large companies or universities spread out in different geographical areas, interconnectivity may be performed by means of the Internet. This might be undesirable, as information belonging to a specific group or organization, including sensitive information, may travel across general access network facilities. MPLS can alleviate this issue by assigning labels to packets that originate at specific locations (and with specific destinations), so that sensitive information may be forced to traverse more reliable channels [83], [98]. This type of network service is known as a Virtual Private Network, as illustrated in figure 2.7.

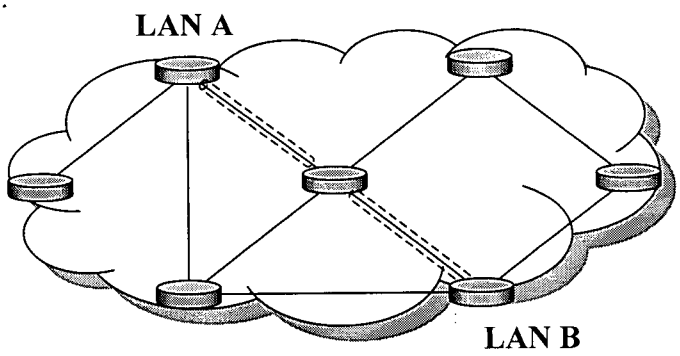


Figure 2.7 Creation of a VPN through explicit routing tunnelling

- Traffic Engineering (TE): The amount of information traversing the Internet may become excessively high at times, depending on several circumstances, which in turn affects the throughput offered by a network. The MPLS technology offers support to attack this problem. Efficient manipulation of labels can be used to direct or re-direct data flows across the network to alleviate data links occupancy in times of congestion [7], [9], [97].

- QoS support: At the network layer, routing can be used to find paths that comply with QoS requirements, later passing this information on to a reservation protocol to explicitly set-up the QoS-compliant path. MPLS can support QoS requirements by enabling the setting-up of such paths, but also, by strategically labelling packets so that network nodes can identify packets with higher precedence. Thus, pre-defined agreements among MPLS network nodes help to differentiate data packets by means of labels, and to determine when certain data flows require superior treatment when reaching such nodes.

2.3 Quality of Service in Data Networks

In this section, the concept of quality of services in data networks is introduced to an appropriate degree so as to illuminate the most important concepts of this subject, giving background for future topics, while also introducing the most important QoS frameworks to date. Emphasis will be given to how QoS can be provided at L2 and L3 by means of MPLS networks.

2.3.1 Introduction to QoS

As mentioned before, the quality of services issue will play an essential role in the future of network communications. A general definition of "quality of service" in communications networks might simply be the level of resources and commitment that a given network provides to user data. Notice that both 'resources' *and* 'commitment' are specified, since having only one or the other available for the user may not be enough. QoS has become an important issue, mainly due to the following reasons:

- The growth of demand for enhanced services, such as multimedia, requires the network to cope with the needs of high-level applications

- The need to offer a wider variety of Internet services obeys a number of economical drivers according to the QoS provided by a network.
- Current network services only provide best-effort services to user data, which are mainly focused on connectivity issues.

QoS is an issue that must be addressed at the various layers in a communication model, and it is not only an issue found in routing [45], [61], [82], [93]. Implementing and deploying specific technologies solely at some layers in a communication model does not ensure the accomplishment of QoS in an end-to-end manner. In order to achieve this complex task, it is necessary to deploy different technologies throughout a layered communication model [8]. For this work, emphasis will be placed on QoS technologies intended for use at L2 and L3.

2.3.2 QoS frameworks at the Network Layer

A number of QoS frameworks have been presented in the literature in past years [8], [14]. In the case of the network layer, one of the most important propositions for a QoS provision is the Differentiated Services Architecture (Diffserv). Formally introduced in [13], the Diffserv architecture offers ease of implementation and reduced complexity. Classes of services are previously determined by establishing a Service Level Agreement between users and ISPs. Some of the most important features of Diffserv are:

- Network nodes implement proper QoS treatment by examining a labelled field in the IP header, which determines the precedence to be applied to each packet.
- All data-flows containing the same label value recognized by network nodes for QoS treatment undergo the same queuing precedence, which makes the Diffserv architecture highly scalable in backbone networks.

The inherent level of aggregation provided by Diffserv makes it a suitable solution for a QoS provision at the network nodes. The actual QoS provision is accomplished by implementing efficient scheduling mechanisms oriented to forwarding packets in a fair manner, in accordance with the SLAs defined. Therefore, data packets belonging to preferential classes of services will experience better treatment at the network nodes, which consequently define improved QoS by reducing QoS-related parameters. Data packets are metered, marked and shaped by edge routers prior to entering a Diffserv network domain, so that intermediate nodes can apply the appropriate precedence according to their class of service. This feature makes Diffserv a suitable candidate for working on a QoS provision scheme in conjunction with MPLS, due to their inherent labelled functionality, as detailed through Chapter 3.

2.3.3 QoS-Routing Support

Much work has been devoted to developing and extending currently used routing protocols to support QoS provisions [47], [65], [80] [81], [96]. QoS-routing (or more generally, constraint-based routing) has been described as a mechanism used to determine the best paths for data-flows, based on both the flow's requirements and the knowledge of resource availability across the network [20], [68]. As mentioned in an earlier section of this chapter, current routing protocols have their foundations in best effort schemes, using algorithms that find the shortest paths between two given points in the network. Two important objectives that need to be addressed by QoS-routing are: the dynamic formulation of possible paths and preserving network optimization [63]. For this work, only QoS-routing within autonomous systems is considered. One of the most used routing protocols for routing in autonomous systems is the Open Shortest Path First (OSPF) protocol [58]. This protocol is based on the exchange of routing tables among nodes within an AS, in an attempt to advertise the state of the links attached to each node.

Periodic updates are carried out by each node to update local routing tables, so that remote nodes can also receive a copy of a needed table, and bring up to date their own tables by means of proper routing re-computations. The overall operational concept of this protocol implies that the periodic transmitting of entire routing tables tends to generate more traffic as the network grows, which also implies the consumption of more bandwidth [20], [80], while incurring on increased computation overhead. Computing routes for a QoS flow while taking in account two types of metrics is an extremely difficult problem [96]. A practical solution is to first find routes that comply with the QoS requirements of the data-flow, and then, to compute a shortest path algorithm in the pruned network [20], [94].

Several approaches have been proposed to address this problem; however, most of them still rely on mechanisms that follow the same conceptual paradigms. A novel approach to performing routing has been recently proposed; yet, it is one that follows a radically new conceptual foundation, as explained next.

2.4 Mobile Software Agents in Network Routing

As described in previous sections, contemporary routing schemes rely on the estimation of best routes by locally computing data representing remote network state. The acquisition of knowledge in large distributed systems requires a significant amount of time; thus, a global view of the system as seen by individual computing network entities, may often be incomplete or become incoherent [94]. Routing is seen as a multi-objective optimization problem in a dynamic environment, which makes it essentially a distributed task [25]. These issues were the main drivers towards the formulation of the mobile agents paradigm as stated in [43] and [48]. The basic theory behind mobile agents is discussed next.

2.4.1 The Mobile Agent Paradigm

Mobile agents are pieces of software code, whose objective is to perform custom computation tasks on behalf of the user. They are designed to migrate among a number of hosts in a given network, while carrying both their execution state and code [43]. Mobile agents can be programmed to work in cooperative schemes, where each agent performs a specific task to obtain partial results [29], [42]. Later, the agents may use such partial information in order to achieve a collective goal. This is the type of mobile agent modelling that will be addressed throughout this work.

2.4.2 Mobile Agent Technologies

A number of technologies have been created to develop applications using the mobile agent paradigm. Some of them rely on their own programming foundations, and others make use of programming utilities developed with the Java programming language. Some of the most relevant mobile agent systems are introduced next [40]:

- *Telescript*: Developed by *General Magic*, is an object-oriented language with enhanced support for security and access control. It basically works by using cooperative schemes, where both stationary and mobile agents are deployed. The system provides migration capabilities by defining its own programming primitives. The need to learn a completely new language proved unworkable for telescript, which evolved to a java-based foundation now known as Odyssey.
- *Tacoma*: Originally, this was designed as a joint project between Cornell University and the University of Tromso in Norway. It is based on the Tcl language, with the facility to interact with other languages as well. Migration is also supported by the nature of its programming design, which works in a script-like fashion. Although security is an issue

on this platform, the system supports both synchronous and asynchronous communications between agents.

- *Agent Tcl*: Developed at Dartmouth College, it is also a scripting language where the agents possess the ability to migrate with the entire code to a node loaded with the system's interpreter. Mobile agents can establish communication by means of direct streaming connections, or by using a messaging scheme. This system provides security enhancements to limit unauthorized access of some resources.
- *Java Aglets*: Introduced by IBM, this mobile agent system is based upon a programming extension made to the Java language, which also provides an enhanced level of migration. On execution, aglets invoke a series of methods directly linked with events on the system the agents traverse; however, they only support intercommunication by means of message passing. Although security support is not a strength of the system, other mobile agent systems are based on the Java language due to its worldwide popularity and support, such as the Voyager system, developed by Object Space, and the Concordia system, created by Mitsubishi Electric.

2.4.3 The Wave Paradigm

Although initially conceived several years ago, the Wave platform was only recently developed as a true emerging technology, capable of addressing a number of issues inherent to open distributed systems [75], [76]. Wave is described as a set of defined strings representing operations, functions and data that propagate across a communications network. Tasks such as optimization, modelling, topology analysis data control and management can be efficiently and asynchronously addressed by the Wave platform in a highly parallel and distributed manner. Classical centralized data-computation is based on the sequential execution of fetched instructions, which operate over blocks of data loaded in a memory device. Such data may stand

as an abstract representation of a real-world system. As part of the mobile agent paradigm, Wave integrates a number of features to overcome the limitations of the centralized schemes. The Wave code strings, or just *waves*, may start their algorithmic execution at any node in the network and propagate in a controlled, virus-like fashion, conquering space as the code execution evolves in time [75]. During this navigation process, the "conquered" network nodes become part of a logical (virtual) knowledge network (KN) that behaves as a true intelligent entity distributed in space. It is important to emphasize that a KN created by a Wave system running on top of a physical network does not embrace or control physical networks the way closed systems traditionally do. Rather, it spreads and 'conquers' a part of such a network and treats it as an open system, or distributed supercomputer, thus facilitating the creation and expansion of the KN. This in turn helps in the solution of problems in a parallel manner, without any centralized supervision. These features make Wave a viable tool for use in telecommunications applications.

The fact that Wave technology was chosen over other platforms relates to the fact that Wave was indeed designed for utilization in environments with specific requirements, such as those of communication networks. Other languages have been widely used as platforms for mobile agent design. Java appears to be the most widely used platform for mobile agent modelling, which is oftentimes the language of choice for Internet applications. In [87], differences and similarities of both Wave and Java platforms, and even a combination of both are anticipated to be plausible for achieving a combined and more robust platform for mobile agents. Figure 2.8 shows the layering structure of the Wave automata as depicted in [75]. Wave is an efficient and flexible system for distributed simulation as well as global cooperative distributed processing.

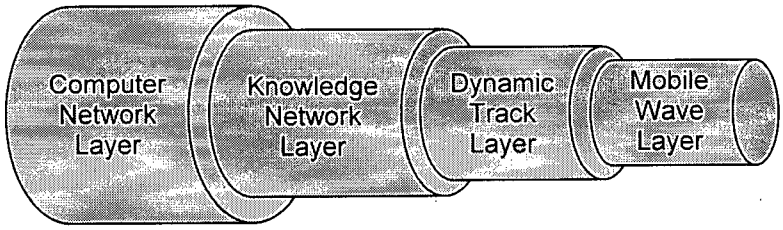


Figure 2.8 Layered Wave Model

2.4.4 The Wave Language

As in other types of programming languages, Wave defines its own method of accomplishing tasks. Wave can be viewed as a scripting tool, possessing an individual syntax that is understood by a Wave interpreter loaded in a computer. The conceptual theory behind the Wave language is now briefly explained.

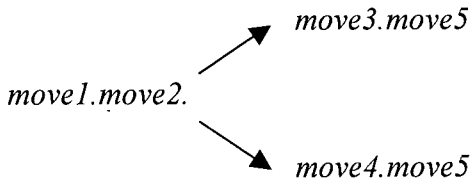
The Wave syntax

As defined in [75] and [76], a Wave program embraces a sequence of spatial actions called *moves*, which propagate and process data across a KN. Groups of independent *moves* can be separated by either a period or a comma, depending on the rationale behind the specific Wave program. When such moves are separated by periods, the moves are executed individually in sequence. Moves can be converted into independent pieces of executing code when commas separate them. In such cases, the interpreter independently executes each move in a parallel fashion, and the remaining moves are treated as separate and independent waves that inherit all the functional characteristics of the original. A visual depiction appears in the following example:

A wave structured in the following way:

move1.move2.move3,move4.move5

Is seen by the Wave interpreter in resulting way:



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. While the FALSE state halts the execution of the current wave, it might allow other waves to execute; however, an ABORT state causes an emergency halt of the whole Wave program before being cloned by using a comma, or rule, as seen 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, the left and/or the right accompanying it. Moves may also be comprised of other operations, such as assignments to variables, filter operations applied to variables, and rules. A description of the operations performed by different types of acts used in Wave is defined in Appendix A.

2.4.5 Navigation Methods in Wave

In accordance with the searching methods followed by a number of algorithms commonly found in the literature, Wave pursues equivalent models, defined as *navigations methods*, in which the actual search is performed by the mobile software agents traveling across a simulated or real network [75]. These methods embrace the fundamental philosophy behind mobile agents, since they actually roam a network in a predefined fashion so as to accomplish a general objective,

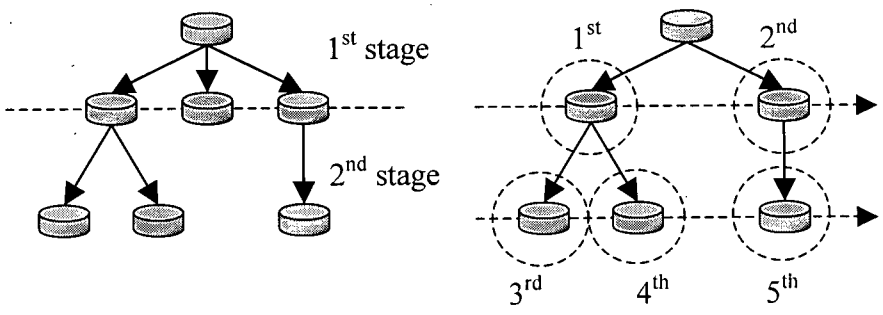
meaning that the actual problem solving is realized in-situm, whereas traditional methods work over cached data in local memory. The most important navigation methods, as presented in [75], are discussed next.

Breadth-first parallel spread

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's 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 synchronize their pace of conquering while they traverse the network. This feature is fully supported by the very functional design of the Wave technology.

Evolving spread

This navigation method also 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 along a list with nodes previously visited, which mean that traces of other waves are not left all over the network. Inherently, this navigation method prevents the formation of loops; however, in large networks, waves following this navigation technique may become larger 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.



(a) Breadth-first search (b) Evolving-spread search

Figure 2.9 Breadth-first search and evolving spread navigation methods

Spiral spread

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.

Depth-first sequential spread

This technique is extensively used in solving graph theory problems. Here, waves are generated so as 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 alternate 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.

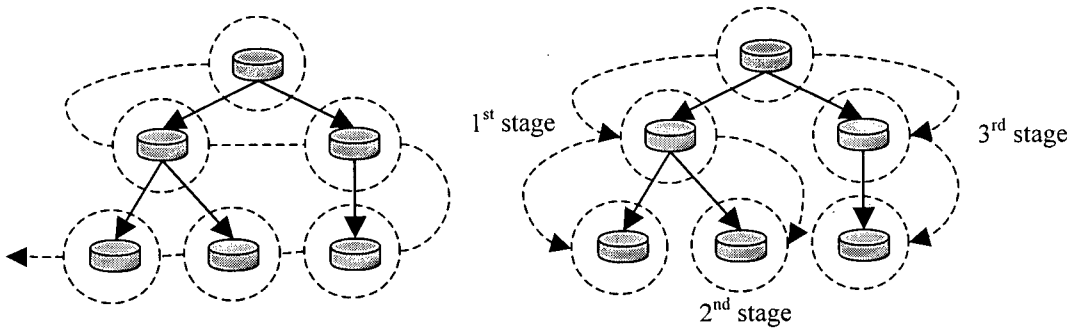


Figure 2.10 Spiral spread and depth-first navigation methods

2.5 QoS-Routing With Mobile Agents

With the intention of finding alternative solutions to solving networking issues, including routing, a number of research groups have pioneered efforts to introduce new mechanisms based on the mobile agents paradigm [23], [29], [30], [34], [53] and [84]. Although there is no hardware infrastructure available in current network routers/switches to efficiently support mobile agent technology, it is important to continue research to understand the benefits when compared to current routing schemes. Some approaches to performing routing with mobile agents have been reported [56], [73], [89]. The results are useful in determining the strengths of the proposed mechanism, but also in detecting areas requiring further analysis [43], [48]. A brief synopsis of relevant work found in this area now follows.

2.5.1 Cooperative Agents Schemes

A variety of papers have been published in regards to cooperative mobile agents [28], [29], [30], [32], [41]. In this regard, mobile agents have been designed in such a way that each agent performs a specific task and obtains a result from it; then, it shares the acquired knowledge with other agents to achieve a general goal. Some of the most interesting cooperative mobile agent applications observed are now presented:

- Adaptive routing: Originally introduced in [25], Ant Net is one of the pioneering approaches for routing with mobile agents. This approach is inspired by the observation of ant colony behaviour that cooperate to achieve the common goal of finding food [11], [33]. The agents are modeled to emulate the ants' 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 at a similar result (the shortest path). Several papers make reference to this work, which served as an inspiration for other research efforts.
- Information distribution: Remarkable work was conducted and presented in [54] and [55] to show how a mobile agents system was used to distribute information across a simulated wireless communication network for dynamic routing and network mapping. A similar work was presented in [94], in which the author develops a mobile agents system whose objective is to study the efficiency of the agents' movement along a network.

2.5.2 QoS-Management With Agents

A number of mobile agent schemes have been proposed to approach the QoS management problem. Some of them address specific issues, such as routing, and others have been proposed as complete architectures or frameworks to provide QoS guarantees in an end-to-end fashion. A more detailed description is next described:

- QoS frameworks and architectures: Several publications exist with proposals to enhance QoS provisions by means of mobile agent technology. They mainly deal with structural and procedural issues, rather than with technical solutions. They also present similar ways in which mobile agent systems can be deployed to support end-

to-end QoS guarantees. Most of the presented architecture models follow layered schemes. For details of these frameworks refer to [1], [22], [35], [49] and [67].

- Research dealing with specific technical issues to preserve QoS guarantees has also been reported. Reference [30] introduces a mathematical model to determine when a mobile agent should initiate a QoS renegotiation procedure after a threshold violation has been detected. In [50], a mobile agent bandwidth negotiation model is introduced. Furthermore, the ARS system extends the work of AntNet, and proposes a framework for mobile agent cooperation through monitoring of QoS levels in separated layers [62].

2.5.3 Reservation and Admission Control Agents

Work has also been reported where mobile agents are used to allocate resources along the network as required by specific connections [90]. This is a clear example of mobile agent technology being used specifically to perform a series of remote operations on behalf of the user. References [17], [87] and [78] are specific efforts carried out by one research team addressing this issue. Similar objectives are pursued with the intention of implementing both resource reservation and admission control schemes on IP networks [23], [24]. This effort is very meaningful, since it represents an interesting way of introducing the mobile agent scheme for problem solving over networking technology already implemented.

Chapter 3 Problem Definition

3.1 Computing Multipoint-to-Point Trees in MPLS Networks

It has been mentioned before that MPLS facilitates the process of setting up paths to carry data along the network. However, as previously mentioned, MPLS *does not* perform routing operations itself; it relies on current routing protocols to obtain a path that connects two end systems. Once the paths have been set-up by the proper distribution of labels among peering MPLS capable switches (Label Switching Routers or LSR), the nodes are then ready to perform the forwarding operation that speeds up the data transfer. Such fact leads to the first discussion of this work as explained next.

3.1.1 The MPLS Forwarding-Routing Paradox

The following figure helps to have a clearer picture of how forwarding makes use of results obtained by routing to ultimately perform the proper data forwarding procedure. It can be clearly seen that the forwarding tables are computed according to the information stored in the routing tables. The routing/forwarding procedure works as follows:

1. A routing protocol (e.g. OSPF) finds the next best hop for the path intended
2. Peering LSRs exchange labels,
3. The above steps are repeated as necessary to build the Label Switched Path (LSP)
4. Data packets are forwarded along the LSP

It should be noticed that, even though the routing/forwarding processes have been decoupled by means of MPLS, *it is the forwarding element that was re-defined; yet, the routing scheme still remains the same.*

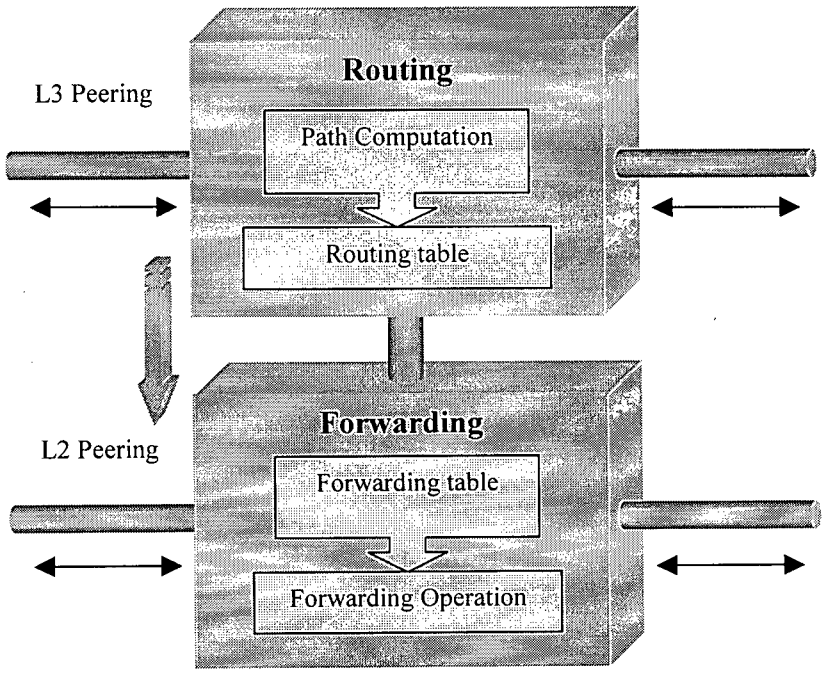


Figure 3.1 Routing support for data forwarding in MPLS

MPLS defines the proper label assignment for packet forwarding, but it still needs an external procedure to indicate what the next hop will be, so that the proper label can be assigned [71]. Figure 3.2 shows the relationship of the current protocol architecture.

The fact that such an effective forwarding scheme still relies on routing protocols that might become inappropriate, or at least troublesome becomes paradoxical, as the complexity, necessities and requirements of the communications networks increase. Both routing and forwarding are complementary to each other; therefore, they should evolve in a parallel fashion. However, it is the forwarding technology that seems to have advanced the most in past years.

3.1.2 Importance of Multipoint-to-Point Trees

MPLS supports the establishing of any type of connection: point-to-point (p2p), point-to-multipoint (p2mp) or multipoint-to-point (mp2p) [71]. A multipoint-to-multipoint connection can be seen as a generalization of the latter two.

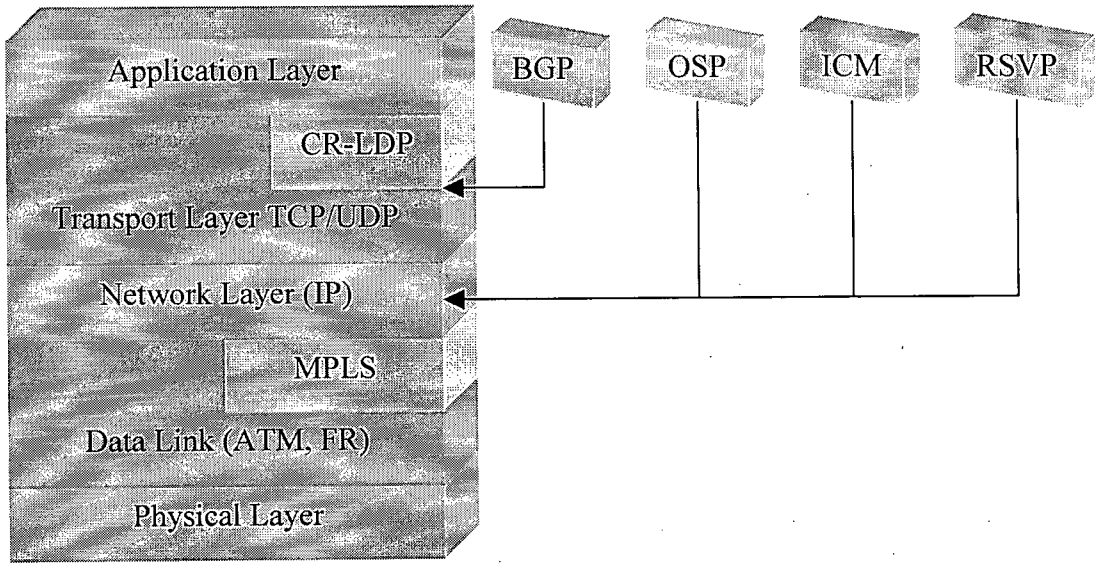


Figure 3.2 MPLS in the OSI communications model and supporting protocols

Point-to-point connections have been addressed most in any of the existing routing schemes; p2mp connections come second, as they support the conceptual foundation of multicasting. Multipoint-to-point connections, however, lack sufficient research attention, which leads to the second concern of this work.

The lack of attention to the mp2p trees is explicable if seen from the datagram-paradigm point-of-view, simply because there was no need to develop sophisticated mp2p strategies. Besides, no mp2p connections were supported on the Internet, until recent modifications to existing hardware were proposed (e.g. VC/VP merge modifications for ATM switches [21], [71], [91]). Nonetheless, the eased stream merging support (i.e. mp2p connections) became readily available with the introduction of the MPLS concept [6], [71].

It would appear at first that the need for mp2p connections might not be as critical as for p2p or p2mp. However, with the introduction of new frameworks to support QoS guarantees, this assumption is no longer plausible [6], [13]. It is not hard to see why this situation has changed. Consider the case of MPLS support for Diffserv services on the Internet backbone [3], [44], [96]. MPLS facilitates the management of data streams on Diffserv-capable switches by assigning

labels to identify specific flows before they are processed in any given node [39]. Even if different QoS frameworks are considered, there will always be situations where a number of data steams traveling across a network might coincide in a given node to follow the same path, either to the same egress node, or to diverge on to a different path further along the network [9], [36], [37].

The benefit of using mp2p trees in the above situations becomes evident. The number of labels that a given node has to manage can be significantly reduced by assigning a single label to all incoming data flows that may traverse a similar outgoing path, either all the way to the destination node, or at least to some extension [6], [44]. This same procedure can be followed by other merge nodes downstream, which can also function as merge points of the already aggregated data-flows (i.e. in a hierarchical fashion). The following picture provides a good visual example of this scenario:

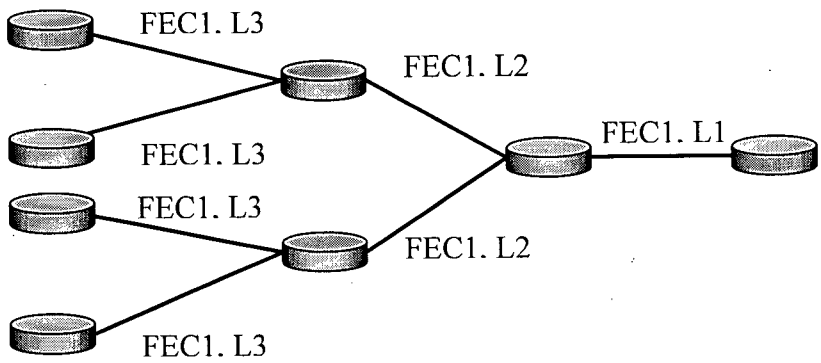


Figure 3.3 Creation of mp2p trees by aggregation of FECs

It should be noted, however, that the fact that mp2p trees are well supported by MPLS does not mean that such trees should become a fixed framework to follow, since there might be specific reasons why stream merging might not be desirable in individual cases.

To enable efficient set-up of mp2p trees, MPLS technology needs a routing protocol capable of finding explicit (strict or loose) routes before the LSP is either established or modified [6]. The

computation of explicit routes becomes complex and computationally costly using the current protocols [39]. After an explicit route has been computed, a label distribution protocol can establish the LSP for the main tree, or secondary branches if necessary. Again, because of the lack of a networking infrastructure to efficiently support mp2p trees, the current routing protocols were not designed for such purposes, not even for multicasting situations (p2mp trees). Specific protocols had to be designed to support multicasting throughout a network (e.g. Mbone, MOSPF [57]), whereas no mp2p protocol has yet been widely deployed for the mp2p case. Current routing algorithms may help to support mp2p routes set-up by individually finding partial p2p routes to build the 'reverse' tree, but no readily available protocol implementation has yet been found that carries out such an intractable task all at once, neither for statically nor for dynamically created routes, as explained next.

3.1.3 Multipoint-to-Point Tree Set-Up Mechanism

There are two ways that can be used to define an mp2p tree: statically or dynamically. If static, a given p2p connection is first established, and can then be augmented to a number of streams to join the evolving mp2p tree. This can be done assuming that joining such a tree brings benefits to the network in regards to connection management, without causing deterioration to existing or the new connections. The other way is to set-up the mp2p dynamically [27], [46], [92], [95]. This solution involves the reconfiguring of the mp2p tree, should one or more connections decide to join or leave the tree [50]. This dynamic tree reconfiguration should ensure that the original constraints of the connection scheme are preserved. Dynamic reconfiguring of the mp2p tree is also an intractable task [36]. An intuitive approach would be to borrow current solutions found in the p2mp connection counterpart (multicasting) to address routing issues in the mp2p ('reverse') tree. The multicasting problem has proven difficult to handle, and the same can be expected for its mp2p counterpart if similar strategies are used [74], [88].

So far, the need for supporting the discovery of mp2p routes as part of the MPLS context for defining FECs has been stressed. However, as MPLS does not solve the QoS provision issue at the network layer by itself, it requires interaction with a QoS-control framework. Thus, it is also important to define how such interaction affects the need to support mp2p routing in MPLS networks. As presented in the next section, it will be seen that such interaction not only supports, but also enhances the need to create a scheme for the discovery of mp2p trees.

3.2 QoS Routing Support for the Multipoint-to-Point Tree

Under the current routing paradigms, finding routes to a given point in the network becomes a more difficult task when QoS guarantees are introduced [4], [20]. A great deal of work has been reported on path computation with QoS guarantees for both p2p connections and for p2mp (multicasting) connections; however, only a few reports have addressed this problem for the mp2p case [9], [36], [37]. Most of the literature found on QoS support for mp2p trees deals largely with the traffic-engineering (TE) problem, which is related to the QoS routing problem. This can be achieved by means of explicit routing, which is well supported by MPLS [71]. The following sections will present a perspective in which QoS can be achieved by a combination of individual features provided by the networking technologies discussed so far.

3.2.1 MPLS and Differentiated Services

The Differentiated Services Architecture has been most favoured as the QoS-control framework of choice for future implementation, due to its superior scalability features [3], [80], [96]. This pertains to the fact that the Diffserv aggregated-flow scheme not only reduces flow state overhead, but also enhances the performance of MPLS by preserving the number of labels to be

managed, as long as mp2p trees are used [3], [44]. Favouring Diffserv as the preferred QoS-provision framework brings up another important issue to address when implementing the Diffserv over MPLS architecture, explained next.

MPLS allows two ways of carrying data-flows across a Diffserv capable network through LSPs. These are addressed in detail in [44], and they are E-LSP, L-LSP. The E-LSP scheme states that a single LSP can be used to carry up to eight QoS-differentiated flows of any given FEC, also known as behaviour aggregates (BAs). On the other hand, the L-LSP procedure indicates that a separate LSP can be established for a single FEC. By explicitly assigning a determined scheduling precedence to this LSP at the time of set-up, an LSR can infer the attribute later on just by looking at the packet's label.

To determine a suitable LSP scheme for QoS support in a Diffserv network, it is necessary to take into account the type of hardware needed to support MPLS in the near future. On one hand, MPLS is a L2-L3 hardware-independent technology, as stressed previously. However, independent of the hardware framework used, the use of a shim header MPLS scheme implies the need to perform segmentation and reassembly at the LSPs intermediate nodes to examine the MPLS label and Diffserv dropping precedence value, which is impractical and more costly. The L-LSP mechanism seems to be the most viable for two good reasons [13]:

- Intermediate nodes would only have to look at the packet's label to determine scheduling precedence.
- Individual per-traffic stream labels are easier to handle and have only local significance, which provides finer granularity in terms of the LSP.

3.2.2 Multipoint-to-Point Trees in Diffserv Networks

The way in which QoS services are provided in a given Diffserv domain have a direct effect as to how the mp2p trees are established. Therefore, it is first necessary to understand the possible

scenarios in which an mp2p tree might be built. As suggested in [3] and [96], a Diffserv network could be used to support three major types of QoS ranges:

- Guaranteed (or premium): This service encompasses those connections that require superior network performance, including low delay and jitter, bandwidth availability, and so forth. SLAs for this type of service might be best suited to dynamic connections for monetary reasons. In any case, users would have to request these services before sending any data.
- Stochastic (assured): Here, the provider makes an agreement in which a certain level of service quality is assured. This level may differ depending on the agreement, where higher assurances usually mean higher costs as well. Depending on network traffic, preference will be given to customers whose SLAs are more strict, so occasional QoS violations might occur for some connections, depending on the SLA.
- Default: This type of service is better known as best-effort service. Here, a network commits to transport data as fast and reliably as the circumstances allow. No guarantees or assurances of any kind are made.

Subsequently the final decision on the creation of an mp2p path greatly depends on the type of SLA to be honoured. However, is possible with a dynamic creation scenario of mp2p trees for the premium SLAs, and static ones for the last two.

3.3 Overall Approach

It is now important to determine how mobile agent technology concept can be used for supporting the work proposed here. It can be seen that although a great deal of research has been

reported, it mostly elaborates on hypothetical/theoretical networking models. While many of the ideas presented are significant, it is necessary to come up with proposals that have a better chance of being implemented in actual data networks. It is important to recall that there is currently no network infrastructure deployed to fully support mobile agent technology. Therefore, coming up with schemes that better reflect real networking issues, and providing efficient solutions to recurrent problems, are key aspects that need to be taken in account [42]. It is the purpose of this work to provide an efficient QoS provision mechanism by means of mobile agent technology, to support routing for flow aggregation schemes in MPLS networks.

The QoS functional approach proposed comprises a Diffserv framework running over an MPLS network supported by a mobile agents system for both enabling flow-aggregation and accomplishing efficient QoS-routing. Therefore, mp2p tree connections with a QoS provision can be obtained by means of a novel routing scheme (based on mobile agent technology) and the explicit routing facility provided by the MPLS network. A cooperative agent scheme will be deployed and studied to verify its effectiveness. Any such scheme should be kept as simple and effective as possible to facilitate the agents' job, and keep computations to minimum complexity.

Chapter 4 Multipoint-to-Point Routing Approach

This chapter introduces foundations of a mobile agent technology based algorithm as a plausible solution to the multipoint-to-point routing problem. First, a short introduction to the Steiner Tree problem and its theoretical applicability to this thesis will be given. Second, a number of assumptions and some conceptual reasoning will be given as a preamble to the design of a mobile agent based algorithm approaching the mp2p tree creation problem. The next section will present and explain in detail the methodology followed in the design of the proposed algorithm, as well as its operational insights. Finally, a number of results obtained during the testing of such algorithms will be presented and discussed.

4.1 Computing the Multipoint-to-Point Constraint Tree

Diffserv manages data packets by differentiating them according to the grade of service they require, which means that a network node that supports Diffserv treats data packets with similar requirements equally. It is also clear that a number of data-streams originating in diverse sources may eventually converge at some node in the network while traversing their individual paths towards a common egress node. This gives rise to the mp2p routing problem: a way to efficiently establish paths so as to support flow-aggregation of data in MPLS networks and accomplish complete QoS support at L3. Addressing the mp2p routing issue may be approached from a more formal point of view, which can be accomplished by means of graph theory, as explained next.

4.1.1 The Steiner Tree Problem

Problems involving trees in graph theory have been extensively studied during the past decades [44]. They have been addressed as one of the most important types of graphs, for they have a broad range of applications [76]. In their simplest form, a tree can be defined as a graph, G, containing no cycles. This merely implies that a tree lacks any path whose starting and ending nodes are the same, where a path is defined as a sequence of nodes V connected by edges E. Figure 4.1 shows some examples of trees.

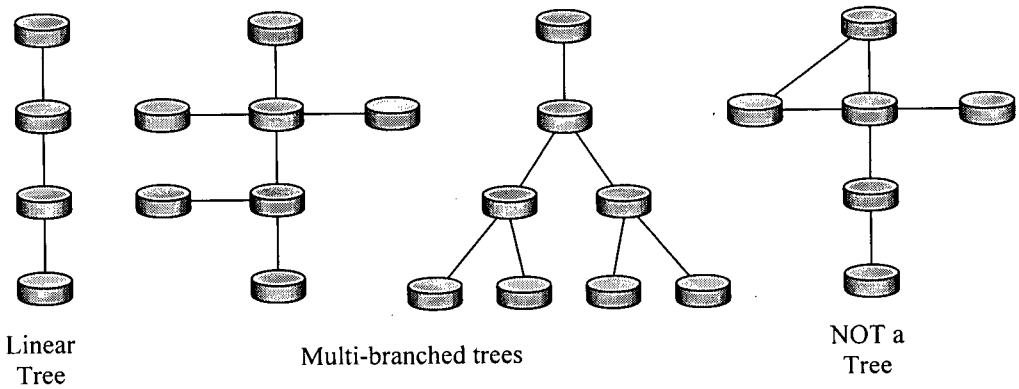


Figure 4.1 Graphs representing trees

Consider a graph G with N vertices, which belong to a tree. Then the following are characteristics of a tree [44], [72], [76]:

- G is connected and contains no cycles.
- G contains N-1 edges.
- If G has a new edge added to it, exactly one circle is created (and the tree is lost).
- A unique path joins every two nodes of G.
- If any edge is removed, the remaining graph is not connected.

The letter T usually denotes a tree. In such trees, a vertex of one degree (edges adjacent to it) is called a *leaf*. A vertex that is not a leaf is then called an *internal vertex*. The *Steiner Tree problem* is a special case, as follows: *given a finite set of points in metric space, find a tree that*

connects these points with the shortest possible length [44]. Such tree is also known as the *Steiner Minimal Tree (SMT)*.

The SMT has a wide range of applications. In the case of telecommunications, SMTs are of special interest for representing problems related to networking, such as routing. In this regard, a special case of the problem is known as SMT in a graph. An even more specific problem definition addressed in this work is known as the *Constraint SMT* in graphs. The term *constraint* relates to the QoS attribute. The objective of the constraint SMT, or *grade of service* Steiner tree, is therefore to define a tree with a minimal total cost when a given grade of service has been assigned to each edge in a graph, when each link is capable of providing the minimal QoS required at each end node [21].

4.1.2 Steiner Minimal Trees in Graphs with Quality of Service

To understand the difficulty of computations with the introduction of QoS in Steiner trees, consider the following factors:

- A data network is often regarded as a stochastic environment, for its state depends on a number of variables that have direct inference in its overall behaviour.
- Any QoS-routing scheme should consider the varying nature of the network parameters at the time it performs pertaining computations. Therefore, changes in the availability of network resources should be taken into account for constructing the routing tables.
- Any routing protocol implementing an SMT problem-solving scheme would trigger constant re-computations of the tree, depending on the recurrence network state updates, which is directly related to the availability of remote resources in the network.

In reality, current routing protocols do not implement SMT problem-solving algorithms because of their high complexity. However, they employ other methods for obtaining Minimum Spanning

Trees (MST) as explained in previous chapters. This helps to maintain routing computations complexities tractable.

4.1.3 A Brief Background on Complexity Theory

Often, the complexity of an algorithm is measured by to the time it takes to perform a computation and provide the desired result. Accordingly, algorithms may then be categorized as either *polynomial* or *exponential*. Polynomial algorithms are capable of solving a problem in accordance with a polynomial function $p(n)$, such that for any input of size n, the calculation takes at most $p(n)$ steps [72]. In algorithms defined by an exponential time function, the input complexity measurement grows exponentially. Therefore, these kinds of algorithms should be avoided, since even moderate level problems can become intractable, even if powerful computers are used [76].

The complexity measurements of an algorithm also introduce the notion of complexity classes. In the NP class, decision problems are capable of being solved in polynomial time in a non-deterministic way, where a given state in the algorithm may determine many "next" states [76]. Thus, *NP* stands for Non-deterministic Polynomial. A problem is said to be *NP-hard* if it is as "hard" as other problems in an NP class, and *NP-complete* if it is both *NP-hard* and in the *NP* class. The Steiner problem is known to be *NP-complete* [44], while some proposed solutions claim that the SMT in graphs with QoS is *NP-hard* [21].

If a routing protocol were to implement algorithms to solve SMTs with QoS, some important aspects would have to be considered:

a) The complexity assessed for the SMTs would imply the need to implement such protocols in fairly powerful computers.

b) The network state's updating mechanisms would have to be carefully designed to find an optimal balance between the need to reduce excessive network traffic due to such updates, and to keep the uncertainty of remote network states to a minimum.

Finding optimal trees for routing purposes is regarded throughout the literature as impractical due to their high complexity. In this work, it is suggested that this difficult dilemma may not have been addressed appropriately. In the remaining sections of this chapter, an alternative solution following a different approach will be presented.

4.2 Design Foundations Towards the MP2P Routing Solution

Now that the conceptual foundations of the constrained mp2p tree have been defined, a set of principles will be introduced which serve as a foundation for an algorithm's design within the mobile agent paradigm. Before detailing this foundation, a number of assumptions will be considered.

4.2.1 Design Assumptions

First, it is important to define some necessary assumptions that serve as background for upcoming explanations:

a) It is assumed that the routing scheme presented here is intended for deployment in a Diffserv over MPLS network architecture.

b) All the nodes in the network are assumed to be MPLS-capable. No hybrid schemes are considered.

c) Nodes in the network follow a Diffserv QoS-architecture. As such, there are a number of *edge nodes* in charge of classifying, metering, marking, shaping and dropping as

per [36]. Similarly, *core nodes* are defined as those interconnecting either other core nodes or edge nodes in the network whose primary tasks are forwarding and dropping.

d) As described before, it is assumed that an MPLS L-LSP data managing technique is used, where independent labels are used to differentiate data-flows, as explained in Chapter 2.

e) It is assumed that the network nodes have the capability of running an interpreter-like program based on mobile agent technology, while also providing an interface that allows message/results passing between the MPLS switching architecture and the interpreter as shown in figure 4.2.

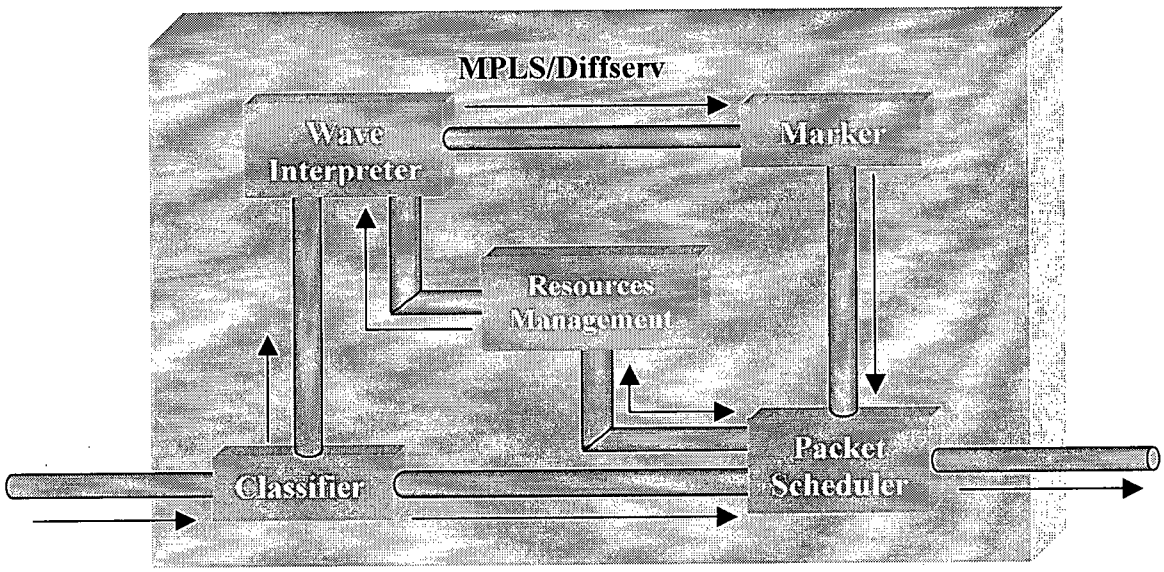


Figure 4.2 Parallel processing for raw data and mobile agents in a given node

f) The mobile agent interpreter of choice is the Wave interpreter [61].

g) Data-packets containing mobile *wave* code are specially marked for further identification throughout a given AS. Unconditional preferential treatment at each network node should be provided, for they are considered to be special network-control data.

h) Upon reception and identification of a wave data packet, a network node immediately passes code to a different queue for service (i.e. the own interpreter's queue). After service completion, the agent is re-inserted in the general data-stream with top priority.

i) In accordance with the previous note, it is assumed that network technology supporting this kind of service possesses an architecture that offers a parallel structure in the node's internal hardware that can process raw data and mobile agent code separately, each with its own processor.

j) All nodes are aware of the characteristics of every other node in the AS they belong to. That is, each node possesses a table with information pertaining to other network's nodes such as IP address, type of node (i.e. edge or core node), connection type (i.e. direct connection to node or indirect access), and so on.

k) Each node is responsible of individually managing its resources. Mapping and assigning hardware and bandwidth are tasks to be performed according to a predefined policy, possibly assigned by a network administrator.

l) Different "colonies" of mobile agents are defined, with explicit tasks and goals for each colony.

4.2.2 Design Foundations

Having defined the proper assumptions in regards to the functionality of the network, a number of design foundations will be described next. These will serve as the principles the algorithmic structure of the wave programs will be based upon. Some important characteristics will also be considered in the agents' design:

- *Simplicity*: The proposed solutions should dwell under the light of simplicity, for it is a key characteristic inherent to mobile agents. Mobile code should be kept as simple

as possible to achieve faster processing times, to occupy minimal bandwidth, and to enhance ease of interaction with other agents or network entities.

- *Robustness*: The agents should have a sufficient level of reliability to cope with different kinds of situations they might encounter.
- *Autonomy*: Each agent should be able to independently achieve its own objective. Although a cooperative agent scheme may be used to reach a final goal, agents should be able to operate and migrate among different nodes in the network without having to depend on the operational aspects of other agents.

A number of design considerations are now ready to be realized according to the theoretical concepts learned through extensive survey.

4.2.3 Heuristic Approach

The general objective is discovering routes to build the QoS-compliant mp2p tree, while attempting to minimize the number of network resources, that is, communication links (edges) and switches/routers (nodes) in a network. It is clear that working in the minimization of one parameter will have a direct effect on the other. Let X denote a given edge in the network represented by a graph G with nodes V and edges E. The cost of a single shortest path from a given source s to a destination d may be represented thus:

$$C(SP) = \sum_{e=1}^{d} X_e \quad \forall\, e \in E; d \in V$$

This expression states that the total cost of the shortest path is the sum of the individual edges from the source to the destination. For an mp2p case, the total cost of the tree $T \in G$ containing nodes $V' \in V$ and edges $E' \in E$, is taken from the sum of all individual paths participating in this type of connection $T(V', E')$. For a Diffserv over MPLS scheme, each individual path begins at a given edge node in the AS:

$$C(SP) = \sum_{v}^{n} \sum_{e=1}^{d} X_e^v \quad \forall e \in E', v \in V'$$

However, in an attempt to optimize network resources and fulfill the original motivation of the mp2p trees (i.e. reducing the number of labels used in MLPS, and grouping similar data-streams for better management in Diffserv), the goal is to find QoS-routes from the participating ingress nodes, whose paths may coincide, at least partially, on their way to the common root. The higher the number of partial routes that match, the better. The optimization problem should then be focused on attempting to *maximize the number of coinciding partial routes*. Then, the cost of an individual path that shares similar routes with other connections could be expressed:

$$C(SP) = \Sigma \textit{ (non-coinciding edges in the path)} + \Sigma \textit{(coinciding edges in the path)}$$

Let X_α denote the non-coinciding edges in a single path, and let X_β represent the coinciding edges, the expression now becomes this:

$$C(SP) = \sum_{\alpha \subset e \in E'} X_\alpha + \sum_{\beta \subset e \in E'} X_\beta$$

Therefore, the optimization problem can be clearly defined as follows:

$$\sum_{e \in E'} X_e = \min\{ \sum_{\alpha \subset e \in E'} X_\alpha + \max\{ \sum_{\beta \subset e \in E'} X_\beta \} \}$$

Where $\max\{ \sum_{\beta \subset e \in E'} X_\beta \}$ denotes the sum of edges utilized by individual connections that intersect those of the other connections. A single edge between nodes *i* and *j*, that serves as a partial shared instance of individual paths in *k* connections with a common destination, is seen thus:

$$X_\beta = \bigcap_{k>1} X_{ij}^k \quad \forall k \subset e \in E'$$

Forming the summation over all the edges *m* participating in the intersections of *k* connections, the total cost of such instances becomes this:

$$\sum_{\beta \subset e \in E'} X_\beta = \sum_{m} \bigcap_{k} X_m^k \quad \forall m \subset v \in V', k \subset e \in E', \alpha \subset e \in E'$$

The optimization problem could then be expressed this way:

$$\sum_{e \in E'} X_e = \min\{ \sum_{\alpha \subset e \in E'} X_\alpha + \max\{ \sum_{m} \bigcap_{k} X_m^k \} \} \quad \forall m \subset v \in V', k \subset e \in E', \alpha \subset e \in E'$$

The problem clearly lies in finding the maximum number of edges where individual path intersections occur, so as to minimize the overall cost of the tree. Finding all possible SPTs for each participating connection, and using information to choose the final paths that will participate in the final tree topology, leads to obtaining the desired result. An additional procedure can be used to identify the intersecting edges of individual paths that may contribute to realizing optimized solutions by pruning the paths not contributing to such optimization. Finally, the most suitable solution is found by evaluating the remaining paths against each other to determine the best path for each connection. A more detailed explanation of how to implement these procedures by means of the Wave paradigm now follows.

4.2.4 Definition of the Supporting Architecture for QoS-Routing

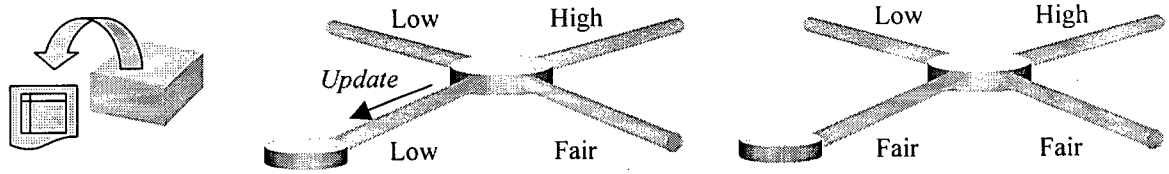
In order to simplify the procedure for achieving the QoS-routing objective, the proposed architecture based on mobile agents is divided into two parts: QoS-KN, and routing discovery. Each of them is now explained.

Construction of the QoS-KN

In accordance with remarks 'k' and 'e ' of section 4.2.1, a Wave interpreter running on top of an MPLS/Diffserv switch would have the benefit of obtaining local QoS-related information directly from the switch. In this case, the QoS information can be used by a colony of static agents designed to build and maintain one or more knowledge networks that reflect QoS availability at individual nodes. Therefore, in the proposed architecture, a colony of agents can be implemented with the above-mentioned objective in the following manner.

1. Individual agents may be created to reside at every node in the MPLS/Diffserv AS so as to monitor individual QoS-metrics (e.g. delay, bandwidth, jitter, etc).
2. When a threshold violation of a particular QoS-parameter in a given node occurs, an agent can update the weight of the affected inter-node link belonging to a previously defined QoS-KN. This is achieved by modifying the link (L) value defined by Wave, which belongs to individual tracks created when the mobile-agent-based QoS-architecture is initially launched
3. The QoS-KN pertaining to a specific constraint is now updated and ready to be used by other colonies of agents involved in the routing discovery procedure.

The proposed procedure is graphically depicted in fig. 4.3. It can also be seen then that this section of the QoS support architecture depends only on the information directly retrieved from the network nodes; no direct interaction with other agent colonies is required. The updated QoS-KN is then indirectly used by the routing discovery agents' colony, as will now be explained.



a) QoS Information retrieved from the switch

b) QoS Information is used to update a link in the BW-based KN

c) QoS Information updated at a BW-based KN link

Figure 4.3 QoS link information update upon detection of a bandwidth change

Finding the Constraint Shortest Path Tree

Based on the proposed method for keeping the QoS-KN up-to-date, a separate colony of agents is defined for the discovery of mp2p routes. Such agents are thus launched for the searching of suitable paths over the QoS-KN links, whose values have been previously mapped to a

predefined QoS availability premise, which depends on the needs of the user. An example of this mapping procedure is presented in the next figure:

Link Value	*QoS Attribute*
1	High-Bandwidth Low-Delay
2	Medium-Bandwidth Medium-Delay
3	Low-Bandwidth High-Delay

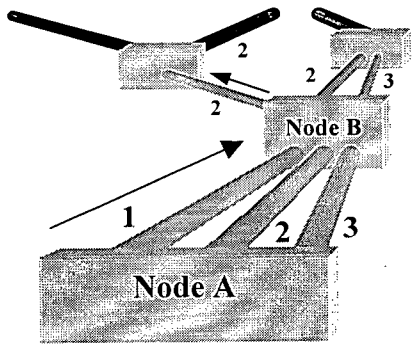


Figure 4.4 Agents follow virtual pathways according to their QoS availability

A set of foundations can now be described for the final design of the routing discovery agents as follows:

First: *Mobile agents will be designed to find SPTs in a spatially pruned network depending on the QoS parameters being considered at the time of the mp2p route discovery.* This formulation raises a second issue that is an importance consequence of working within the mobile agent paradigm: *no network state information is ever updated at remote places throughout the network.* Instead, each agent collects pertinent information regarding resource availability to perform some computation before migrating to other nodes, while carrying its own agent's state, as shown in figure 4.5. This is better known as strong migration [75].

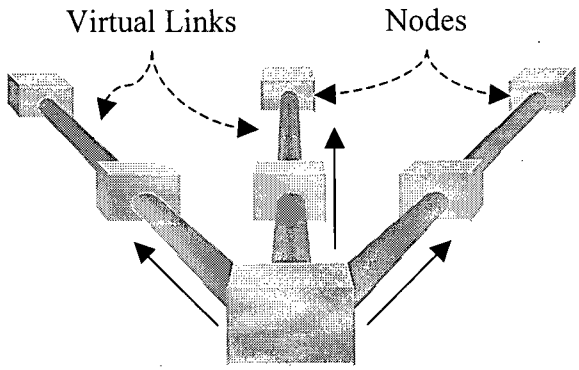


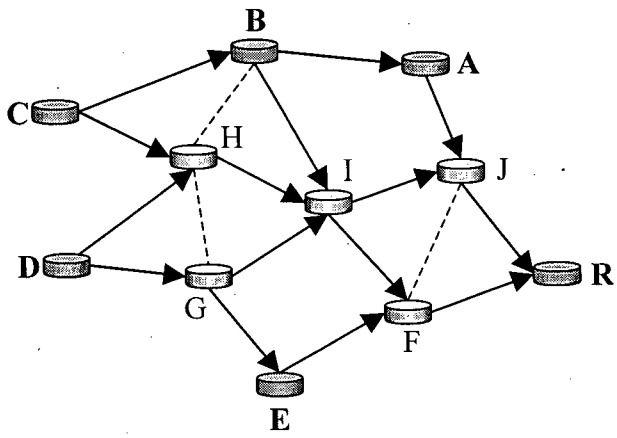
Figure 4.5 Mobile agents clone and migrate to other nodes while searching for SPTs

The third aspect, which is a consequence of the second aspect, is defined as follows: *Although individual agents are created to perform specific tasks autonomously, there exists a mobile agents' colony whose objective will be clearly defined and accomplished as a whole.* By implementing an algorithm that meets the criteria proposed, two important features are met, which are worth mentioning:

1. Mobile agents searching for the shortest route will always traverse paths able to comply with the requirements of the request. This guarantees that the QoS constraint will always be met.
2. Preference is first given to complying with the requirements of the route, and optimization of network resources is performed on the run as a secondary priority.

As recently mentioned, in the scheme presented here, mobile agents are deployed to find not one, but the entire set of existing SPTs that meet the QoS criteria between the sources and the common destination (the root of the mp2p) in the pruned network. Figure 4.6 depicts graphically this process. The motivation behind acquiring all the constraint shortest paths in between the m sources and a single root node in network will become clear shortly.

Source	*SPTs*
A	A-J-R
B	B-A-J-R B-I-J-R B-I-F-R
C	C-B-A-J-R C-B-I-J-R C-B-I-F-R C-H-I-J-R C-H-I-F-R
D	D-G-E-F-R D-G-I-J-R D-G-I-F-R D-H-I-J-R D-H-I-F-R
E	E-F-R



Legend:
Edge Nodes: A, B, C, D, E & R
Core Nodes: F, G, H, I & J

Figure 4.6 First stage of mp2p algorithm, finding SPTs from all sources to root node

Spotting Common Nodes in SPTs

The second step is aimed at minimizing network resources in order to realize the best possible network utilization. The procedure to follow is simple: a second colony of agents is deployed so that they traverse the very same paths the previous agents discovered; that is, each source node sends a wave that clones itself and travels through the SPTs found for that specific node. While traveling across such SPTs, they mark those nodes in an attempt to identify them as possible data-merge nodes in the mp2p tree.

The objective is to find nodes traversed by waves originating in different sources so that a future set of nodes can use this information in choosing a path towards the optimization of network resources (i.e. both the number of nodes and edges used). Figure 4.7 provides an illustration for a better understanding of this concept. Note that if a wave finds a node that is common to another STP being traversed by waves coming from the same node, that node is not marked again. This implies that nodes are marked only once per wave originated in a common source node.

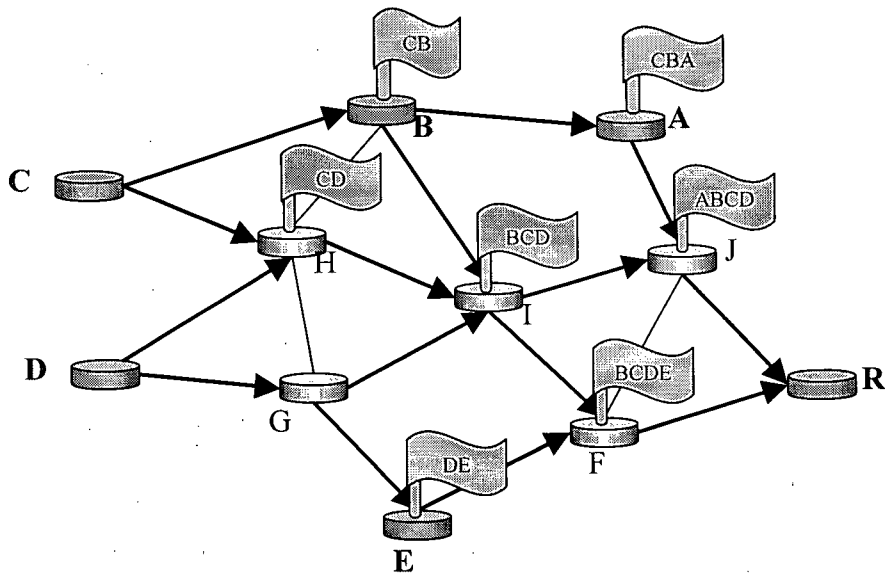


Figure 4.7 Common nodes in shortest path trees are marked as possible merge points

Defining merge nodes for the final mp2p tree

After the nodes that coincide with other SPTs routes are identified, a last colony of agents is launched to determine the final individual routes that will constitute the mp2p tree. Again, the waves launched traverse the same SPTs previously found and marked. Each wave is assigned a weight whose magnitude increases as it travels a path with more mutual nodes of paths found by other source nodes. This means that the more joint nodes found during their traversal across individual SPTs, the more weight is given to them. Upon reaching the destination, the agent records the path traversed and the weight brought along. Subsequent agents arriving from the same node will attempt to perform the same operation at the root node, but they will only succeed in doing so if the weight brought is greater than that of a path previously recorded. Agents traversing paths that contain a larger number of nodes included in alternate routes from other source nodes determine the final path to follow.

While determining mp2p routes, the network nodes act as passive entities providing a service by means of a Wave interpreter. Each agent reaching a node joins a first-in first-out queue and awaits service by the Wave interpreter residing in such a node before making further mobility decisions. To provide better procedural insights into the routing algorithm, details on the wave program are now presented.

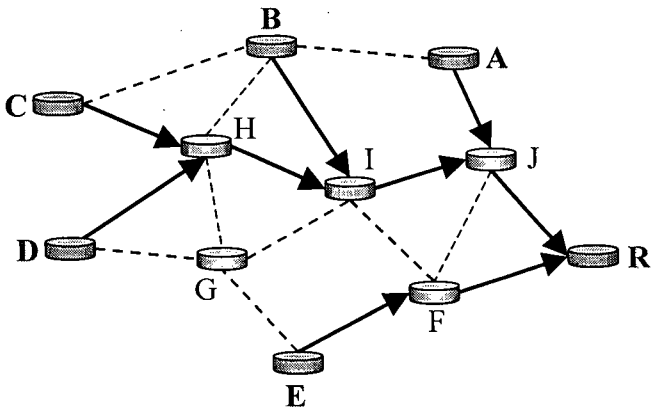


Figure 4.8 Cooperating among mobile agents to determine the final mp2p tree

4.3 Multipoint-to-Point Routing With Wave

After giving a conceptual presentation of the algorithm's operation, a more detailed explanation of the actual wave program to construct the mp2p trees now follows. The reader is referred to the reference section for extensive details of the Wave paradigm and programming features.

4.3.1 Discovering of Multiple Shortest Paths

In order to best describe the inner workings of the algorithm for finding mp2p routes, the following explanation makes direct reference to figures showing the actual wave code. Figure 4.9 shows the algorithm of the mobile agent used for finding all possible QoS-compliant SPTs from a given source to a destination, as explained previously.

The algorithm itself is divided into two sub-parts enclosed by a SQ rule, which means that each sub-part will be activated in sequence. In the first sub-part of the program (lines 1 through 5) the wave hops across the network using links marked with a predefined identity, which is directly related to the QoS availability of the physical link through the Wave "L" environmental variable.

```
1       Findspt=`
2       Fcollect=A.Fsource=C.
3       SQ(
4               RP(
5                       Flinktype#.Flength+L.
6                       OS(
7                               ID(Fsource/~Nsource.Nsource&Fsource.Ndistance&Flength),
8                               ID(
9                                       Fsourceindex=Nsource.Fsourceindex::Fsource.
10                                      Fdistindex=Ndistance.Fdistindex:Fsourceindex.
          11                            Fdistindex==NONE,OS((Fdistindex<=Flength.!3),).
12                                      Fsourceindex2=Fsourceindex.Fsourceindex2&@.
13                                      Fsourceindex2&Flength.Ndistance:Fsourceindex2.
14                              )
15                      )
16              ),
17              RP(
18                      Fpath&C.
19                      (C==Fdestination.Fmarkpath.Fcompete.!3),
20                      (
21                       Flinktype#.Flength+L.
22                       ID(Fsourceindex=Nsource.Fsourceindex::Fsource.
23                       Fdistindex=Ndistance).Fdistindex:Fsourceindex.
24                       Flength<=Fdistindex
25                      )
26              ),
27      )'
```

Figure 4.9 Wave code for finding multiple SPTs from several sources to a common destination

The frontal variable "Flinktype" defines the QoS-KN searched, according to the QoS metric being considered. For simplicity, before to an actual simulation run of this algorithm, sample networks are pre-defined and run to serve as the target instances for the agents' deployment. While travelling from the source node to the intended destination, agents hop through the links marked as able to meet the QoS guarantees sought, and increment by one a frontal variable as part of the distance travelled so far. Upon reaching an intermediate node, a wave checks whether others coming from the same source have already been there. If no previous visits are recorded,

two nodal variables are created: one for holding the identity of the arriving wave, and another for keeping the sum of the weights from the source node up to the one reached so far (see line 7).

If a wave reaching a node encounters previous records by a wave originating in the same source node, it compares the distance recorded with the one being brought in its own frontal variable (lines 9 –11). If the distance brought is smaller, the wave is allowed to proceed, thus replacing the distance previously recorded by the one it brings (lines 12 - 16) before hopping (and possibly cloning itself) to adjacent nodes. Otherwise, the wave dies. The code is enclosed inside a RP (repeat) rule, meaning that this sequence will repeat for as long as the conditions for further execution are met as previously explained, or until the agent reaches the destination node. It can be seen that indexing techniques are used at the intermediate nodes to enable the recording of multiple source-distance pairs. Refer to figure 4.10 for a graphical description of the process.

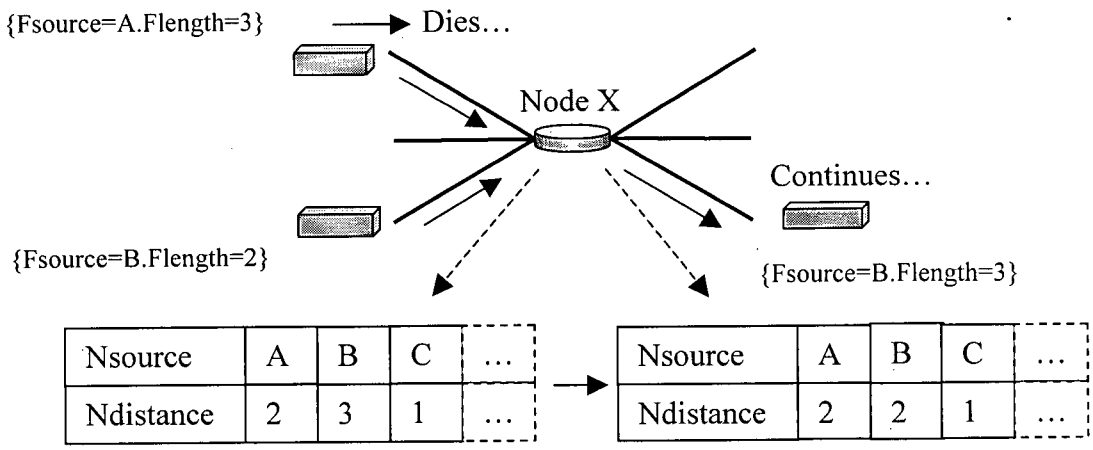


Figure 4.10 Hopping procedure of waves reaching an intermediate node

The second RP rule (lines 17-26) is built as a companion of the first RP rule for finding multiple SPTs in the mp2p tree connection. After executing the wave embraced within the first RP rule, each node searched holds a vector containing the identity of the nodes participating in the tree's construction, and the shortest distance found. Then, a second set of waves is launched to identify and collect the paths found. The frontal variable Fpath (line 18) is defined to collect such a path

as the wave traverses through the network, but first, the agent verifies whether the destination node has been reached (line 19), in which case, the wave is ready to launch subsequent sections of the mp2p tree finding program. If the destination has not been reached, the agent is cloned and dispersed through QoS compliant links, while also recording the traveled distance so far (line 21). Upon reaching an intermediate node, the agent is only allowed to continue its execution if the distance brought is less than or equal to the one previously recorded by the wave agents in the first RP rule (lines 22-24). Thus, the agents embraced by the second RP rule perform the forward collect of the SPTs found. This procedure has two remarkable features. First, navigating though nodes containing distances that are equivalent to the ones found previously allows the collecting of all possible SPTs in between the multiple sources and the common destination (the root of the mp2p tree). Second, the nature of the algorithm ensures that the paths found are cycle-less. This results in an attractive feature for MPLS, since it releases it from having to launch a separate procedure to verify routes without cycles.

4.3.2 Determining Joint Routes

As previously explained, in the second stage of the algorithm a group of agents is launched to spot and stamp all the nodes that might be considered merge nodes in the mp2p tree. The number of SPTs per origin-node found after the first stage of the routing algorithm is finalized determines the number of waves to launch. Therefore, each wave is mapped to one of the SPTs found. Figure 4.11 shows the Wave program that accomplishes this function.

```
28      Fmarkpath=`
29      Fcount=-2.
30      RP(
31          OS(
32              ID(Nvisits==NONE.Nvisits=1.Nvisitedby&Fsource),
33              ID(Fsource/~Nvisitedby.Nvisitedby&Fsource.Nvisits+1),
34          ).
35          C/=Fsource.Ftemp=Fpath.Ftemp:Fcount.Flinktype#Ftemp.Fcount-1
37      )
```

Figure 4.11 Wave function that finds possible merge nodes in SPTs

Notice that the frontal variable Fmarkpath has been previously assigned data that actually represents the code for this part of the algorithm. Thus method is known in Wave as code injection. Each wave inherits an STP for individual navigation from the previous code injection by the "Findspt" function. The wave simply traverses the network following the inherited path until it reaches its destination.

When a wave reaches an intermediate node, it verifies the existence of previous records by the same kind of waves generated in other nodes. If the wave finds that no previous visits have been recorded, it creates two nodal variables: one that contains the identity of the originating node, and another to hold a counter value (line 32). If a wave finds that previous records exist, the wave searches for evidence of another wave originating from the same node (with the same identity) having visited the node before (line 33). A wave will always append its identity to the proper nodal variable if no previous records from similar waves exist, and will also increment the counter. Should the arriving wave encounter existing records with the identity of the originating node, the wave merely hops to the next node in the SPT, without modifying any record.

From the previous explanation, it can be inferred that identity nodal variables at intermediate nodes will only hold unique values, corresponding to the identities of waves that visit the node. No repetitions are allowed. As explained in a previous section, the objective of this procedure is to assign a "weight" to each node in the network that is an instance of an individual SPT found by each originating node.

4.3.3 Defining the Final Routes of the Tree

The last part of the mp2p tree routing discovery is designed to define final individual paths in the mp2p tree. Figure 4.12 shows the Wave program used to accomplish this objective.

```
37      Fcompete=`
38      Fnum=2.
39      RP(OS((ID(1<Nvisits).Fweight+1),).C/=Fdestination.
40         Ftemp=Fpath.Ftemp:Fnum.Flinktype#Ftemp.Fnum+1).
41      OS(
42          (
43           ID(Fsource/~Nedges.Nedges&Fsource.Nweight&Fweight).
44              Fpath%`'.CR(Fsource#Fpath).#P
45          ),
46          ID(
47             Fsourceindex=Nedges.Fsourceindex::Fsource.
48             Fweightindex=Nweight.Fweightindex:Fsourceindex.
49             Fweightindex<Fweight.Fsourceindex2=Fsourceindex.
50             Fsourceindex2&@.Fsourceindex2&Fweight.
51             Nweight:Fsourceindex2.Fsource#.Fpath%`'.C=Fpath.#P
52          )
53      )'.
```

Figure 4.12 Wave program for final definition of the mp2p tree

The first RP rule of the algorithm causes the Wave code to repeat the navigation procedure through the SPTs available, but this time, no records are modified (line 39 & 40). Instead, each wave individually navigating the SPTs collects the weights assigned to each intermediate node on their way to the root node. Each time the wave reaches an intermediate node, it collects a weight (if it exists), and adds it to the frontal variable that holds the sum weights found to this point. The first agent to arrive at the destination proceeds to record the path traveled along with its weight (line 43). An indexing scheme is used by the waves creating virtual nodes at the destination. This procedure ensures that waves arriving from distinct origin nodes do not interfere with records from other waves (line 44).

An agent arriving at its destination always checks the existence of previous records. In such a case, a wave initially checks whether the weight being brought is larger than the one already recorded. If so, the records are modified to reflect both the new path and its corresponding weight. On the other hand, if the weight being brought is less than or equal to the one recorded, the wave simply dies (lines 46-52).

As a consequence of this procedure, the end of the algorithm will produce a set of vectors, each containing a list of IP addresses that comprise the final shortest path to be used in the

construction of the mp2p tree. These vectors can be passed on to a label distribution protocol being implemented in the MPLS network to finalize the setting up of the tree. A complete printout of the Wave algorithm described in this chapter is shown in Appendix B.

4.4 Implementation and Practical Results

In this section, the results presented belong to simulations of the algorithm for discovering *static* mp2p routes with networks of different sizes and under different circumstances. This is in accordance with the assumptions made for working with the Diffserv architecture to provide best-effort and probabilistic QoS SLAs.

The algorithm for finding mp2p routes is slightly modified to measure the amount of agents processed by the Wave interpreter at a given node during specific time periods. Several experiments were conducted with experimental network sizes of 12, 20 and 30 nodes. The topologies of the experimental networks used can be found in Appendix C.

12-Network Node Simulation

First, the 12-node knowledge network was deployed, and the number of processed agents was measured in all of the nodes. The destinations chosen were nodes *d* and *g*, because they have one and two incident nodes respectively. Accordingly, the remaining nodes were used as ingress points participating in the mp2p tree. The results can be observed in figure 4.13 for destination node *d*, and figure 4.16 for destination node *g*. To measure the number of agents processed at the nodes, a Wave program is launched at different nodes at the same time that the mp2p routing procedure commences. For example, if the node occupancy were to be measured at node 'k', the structure of the wave simulation would be as follows:

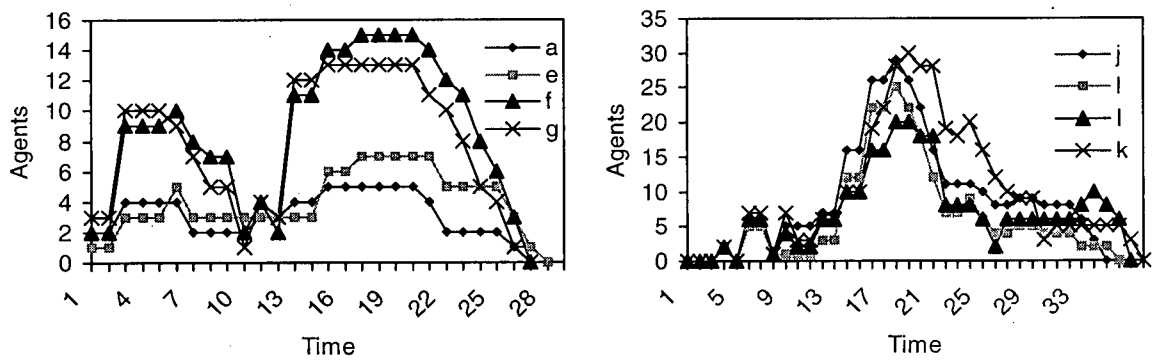
{Monitor occupancy at node 'k'},{find mp2p tree with root node 'd'}

The way this monitoring program works is quite simple. A single nodal variable is created to reflect the number of agents that either arrive or leave a given node, and every time a wave arrives, this nodal variable is incremented. Conversely, when an agent leaves the node (or if it dies at the node), the variable is decremented. In the mean time, a free-running counter is run while monitoring the nodal variable's value. The variable's value is appended to a different variable every time a single cycle is completes, thus providing the finest level of value screening. At the end of the routing procedure, a variable containing the desired result is obtained.

A number of observations can be readily made from the results observed in fig. 4.13. For instance, it is evident that the number of processed agents in all of the nodes follows a bursty nature. On the nodes with smaller numbers of incident edges (i.e. lower degrees), the intensity of wave arrivals appear to occur in two phases, whereas at the nodes where the number of incident edges is larger (i.e. of higher degree), the arrival of waves occurs in a single burst right in the middle of the process. The reason for such results can be explained through intuitive thinking. At the beginning of the process, the edge nodes receive either waves from nearby nodes or 'echoes' of waves launched by themselves. This causes a slight burst of agents to arrive during the same short period of time. In the mean time, the nodes with higher degrees (located in the centre of the network) see less activity. The second burst of agents at the edge nodes occurs when waves from more distant nodes start arriving. There is also a period when agents coming from all the edge nodes in the network converge at the core nodes while individually searching for SPTs. It is then that such nodes see the burst of agents processed.

Another evident result is the fact that core nodes see more activity in regards to the number of agents processed. It can be readily seen in figure 4.14 that, for this particular case, the core nodes experience higher occupancies, which also happen to have higher node-degrees. The plots also indicate that the time in which the core nodes remain active during the mp2p routing procedure is

longer than for the remaining nodes, except for the destination node. The reason for this is that after the stage where the mobile agents conclude the search for multiple SPTs, only those nodes identified as candidates for final routes remain active during the following two stages of the process. To corroborate the observations made during this first simulation, another run made is presented in which the destination is now chosen to be node 'g'. This is done with the explicit intent of observing whether the behaviour of the system is any different when the degree of the destination node is higher. It can be seen in figure 4.15 that the plots show remarkable similarity in regards to the level of node occupancy experienced. Besides, figure 4.16 displays that the executing time and the number of serviced agents per node remains consistent with that of the first simulation. Finally, figure 4.17 graphically shows the final topology of the tree found.



a) Occupancy in 1 & 2 degree nodes

b) Occupancy in 5 & 6 degree nodes

Figure 4.13 Occupancy of agents in a 12-node network with a 1-degree destination node {d}

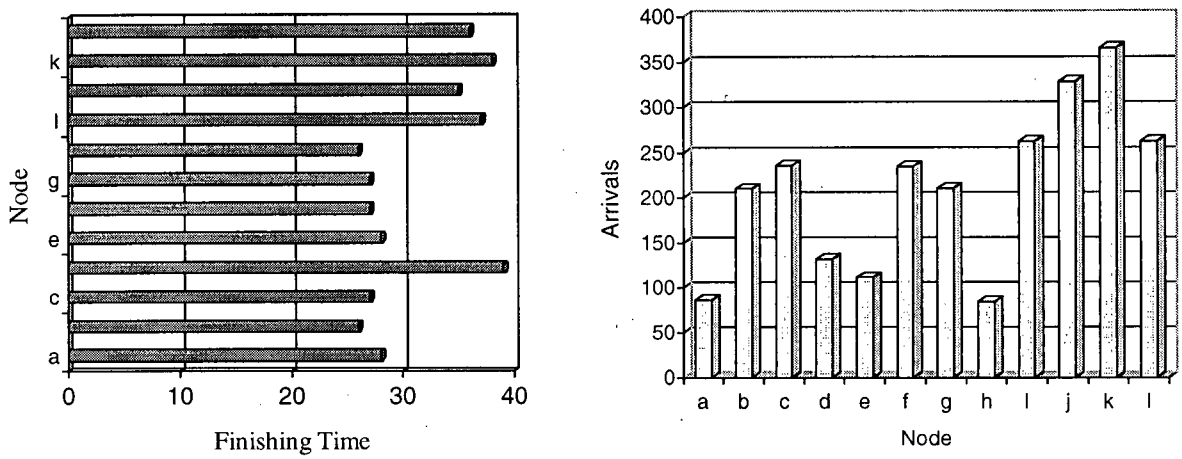


Figure 4.14 Finishing time and arrivals per node in a 12-node network with a 1-degree destination node {d}

longer than for the remaining nodes, except for the destination node. The reason for this is that after the stage where the mobile agents conclude the search for multiple SPTs, only those nodes identified as candidates for final routes remain active during the following two stages of the process. To corroborate the observations made during this first simulation, another run made is presented in which the destination is now chosen to be node 'g'. This is done with the explicit intent of observing whether the behaviour of the system is any different when the degree of the destination node is higher. It can be seen in figure 4.15 that the plots show remarkable similarity in regards to the level of node occupancy experienced. Besides, figure 4.16 displays that the executing time and the number of serviced agents per node remains consistent with that of the first simulation. Finally, figure 4.17 graphically shows the final topology of the tree found.

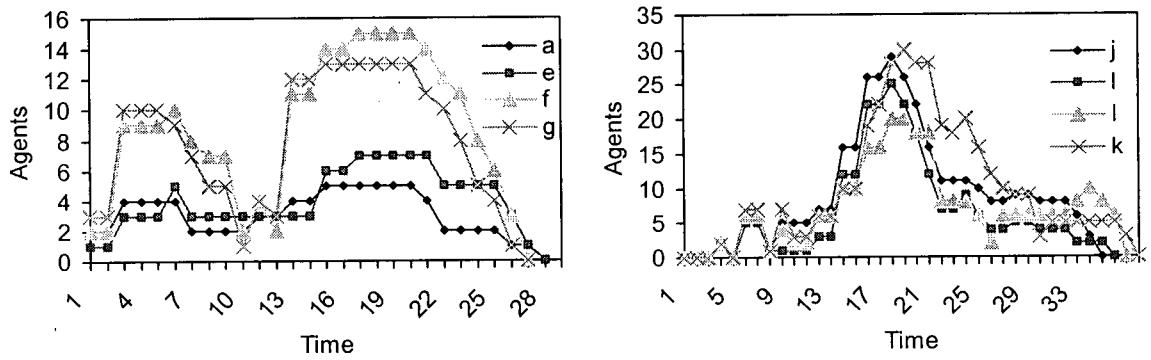


Figure 4.13 Occupancy of agents in a 12-node network with a 1-degree destination node {d}

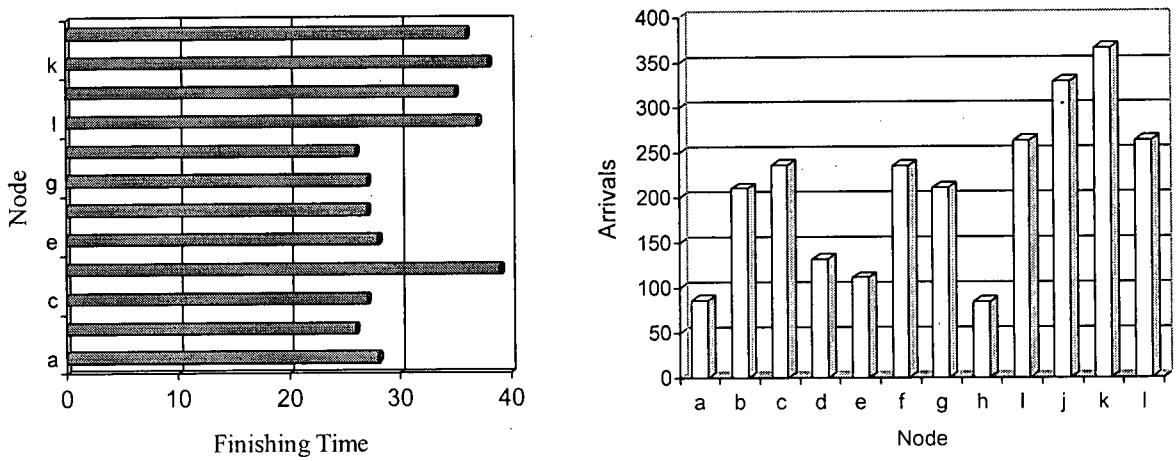
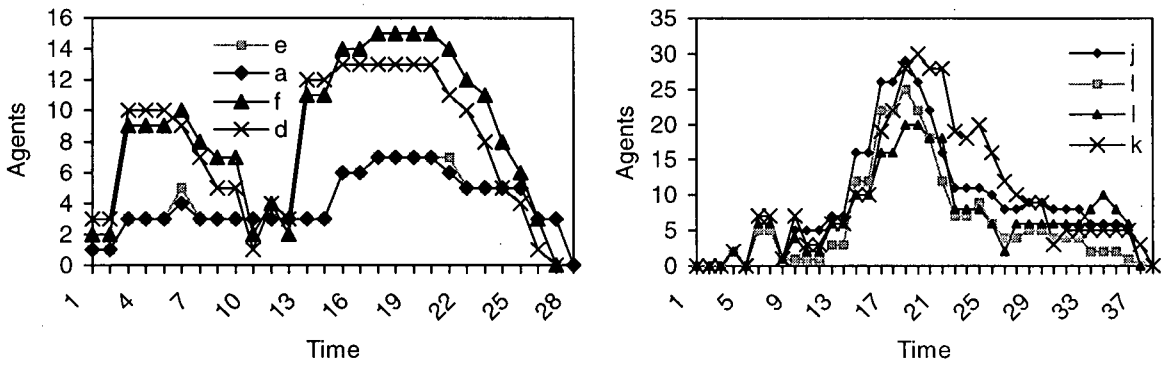


Figure 4.14 Finishing time and arrivals per node in a 12-node network with a 1-degree destination node {d}



a) Occupancy in 1& 2 degree nodes

b) Occupancy in nodes of 5 degrees

Figure 4.15 Occupancy of agents in a 12-node network with a 2-degree destination node {g}

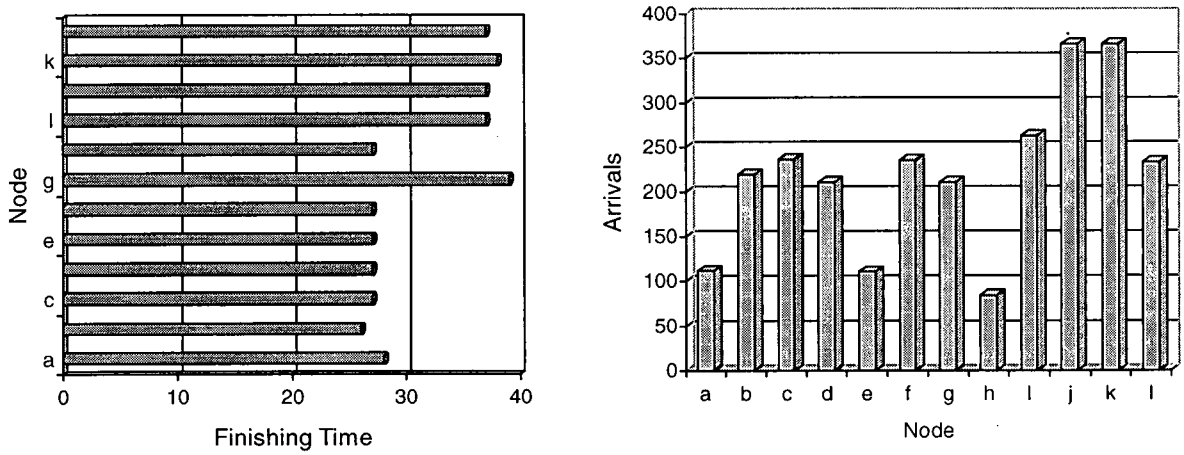
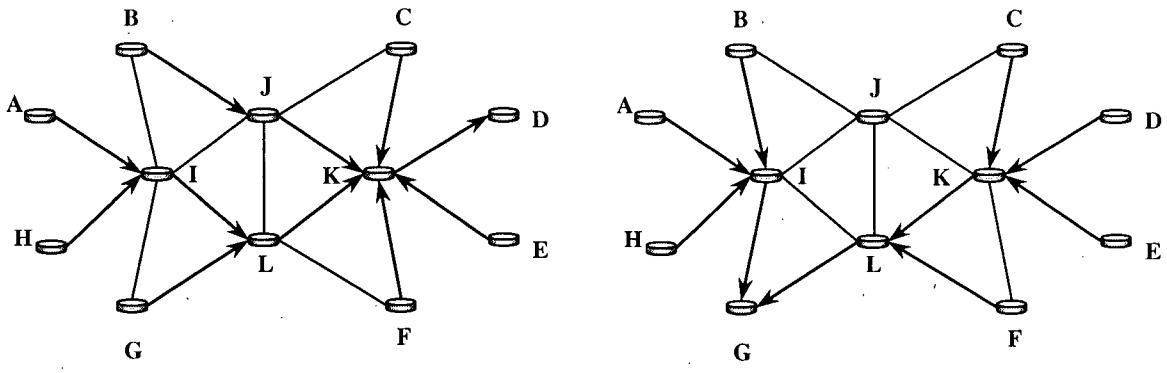


Figure 4.16 Finishing time and arrivals per node in a 12-node network with a 2-degrees destination node {g}



a) Tree constructed with destination node 'd'

b) Tree constructed with destination node 'g'

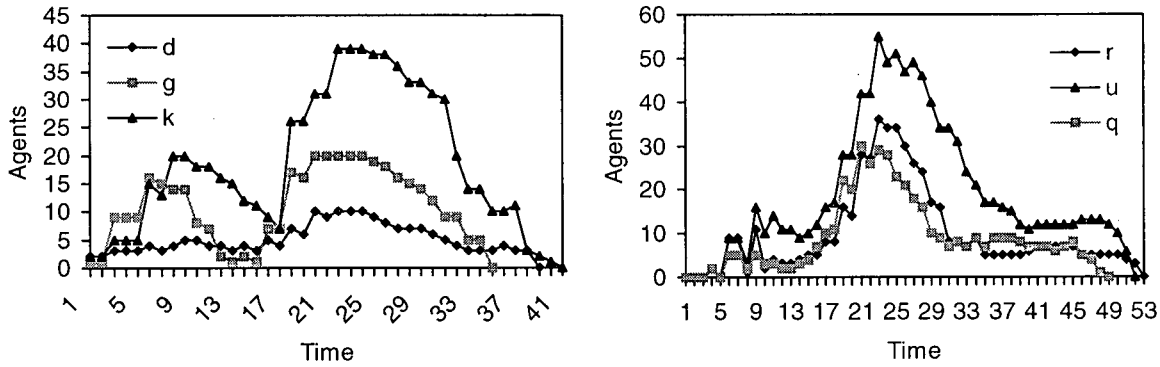
Figure 4.17 Final routes found during the mp2p tree construction in the 12-node network

20-Network Node Simulation

The results obtained during the mp2p tree routing simulation for the 20-network node reveal the behaviour of the mobile agent system when the topology becomes arbitrary (instead of symmetrical, as with the 12-node network), and has more nodes. The first simulation run was conducted with node 'a' serving as the destination node, and the resulting plots were grouped in two sets, one for nodes with a 1-3 degree, and a second plot for nodes with a 4-6 degree.

Figure 4.18/a shows that the node occupancy behaviour follows the same pattern as in the previous simulations, suggesting that the number of agents reaching a given node increases linearly with the degree of an edge node. The shapes of plots in figure 4.18/b show consistency with previously obtained results, suggesting that the core nodes exhibit a behaviour similar to that observed in the 12-node network simulation. Figure 4.19 also confirms the fact that the finishing-time behaviour of nodes involved in the mp2p connection remains consistent, meaning that the active time of those nodes involved in the final mp2p tree topology is longer than the ones who do not participate in the later part of the process.

There is, however, an unexpected outcome: although node 'k' is of smaller degree than node 'r', it experiences a higher number of agent arrivals. This result rules out the option that the occupancy of a given node depends on the node's degree as a sole factor. This is evident because even though the degree of node 'k' is smaller than that of node 'r', it underwent higher arrivals' incidences. On the other hand, node 'r' experiences arrivals of waves in a more pronounced burst, whereas the bursts at node 'k' are less intense, but more extended time-wise. To confirm this observation, note that node 'q', which has the highest degree among the nodes monitored, experiences fewer arrivals of agents than do nodes 'k', 'r' and 'u'.



a) Occupancy at 1-3 degree nodes

b) Occupancy at 4-6 degree nodes

Figure 4.18 Occupancy of agents in a 20-node network with a 1-degree destination node {a}

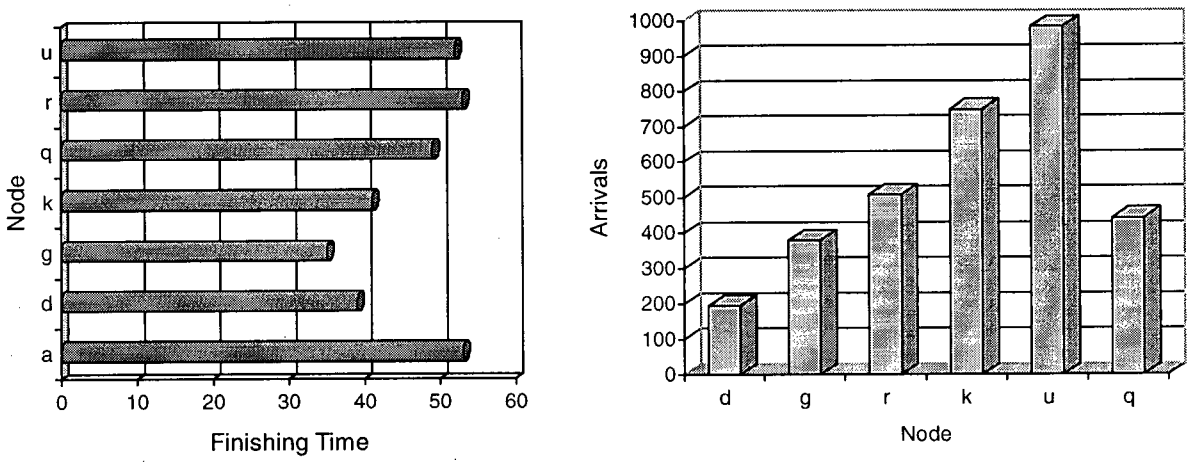


Figure 4.19 Finishing time and arrivals per node in a 20-node network with a 1-degree destination node {a}

An additional run using the 20-node network was made, where node 'g' was defined as the new root node of the mp2p tree. It can be seen that the occupancy behaviour obtained from the edge nodes remains consistent; nonetheless, the core nodes experience a different outcome, in which the occupancies are now comparable to one another in the monitored nodes. The data presented in figure 4.22 shows that the final routes obtained by each node participating in the mp2p connection result in one of the other possible trees' optimal configurations. Another interesting feature observed is that, while some of the core nodes see decreased activity due to lesser wave arrivals, other core nodes experience an increased number of arrivals.

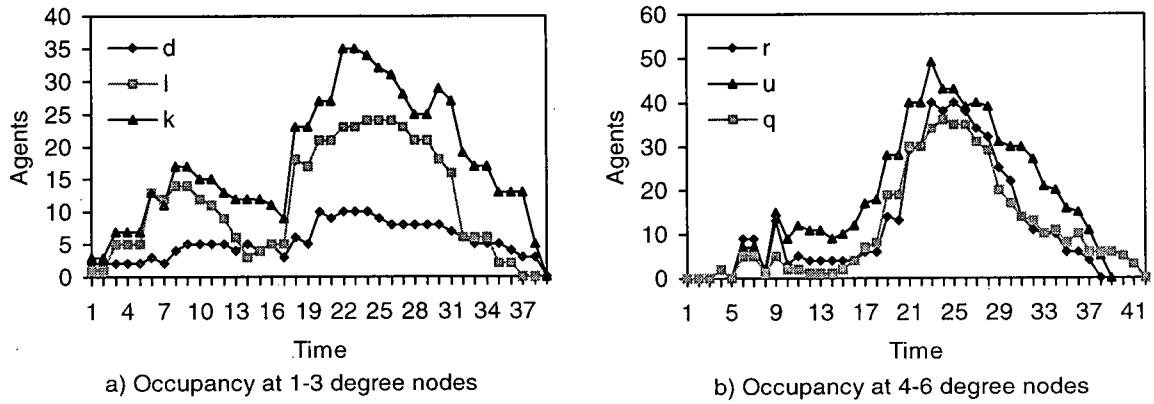


Figure 4.20 Occupancy of agents in a 20-node network with a 2-degree destination node (g)

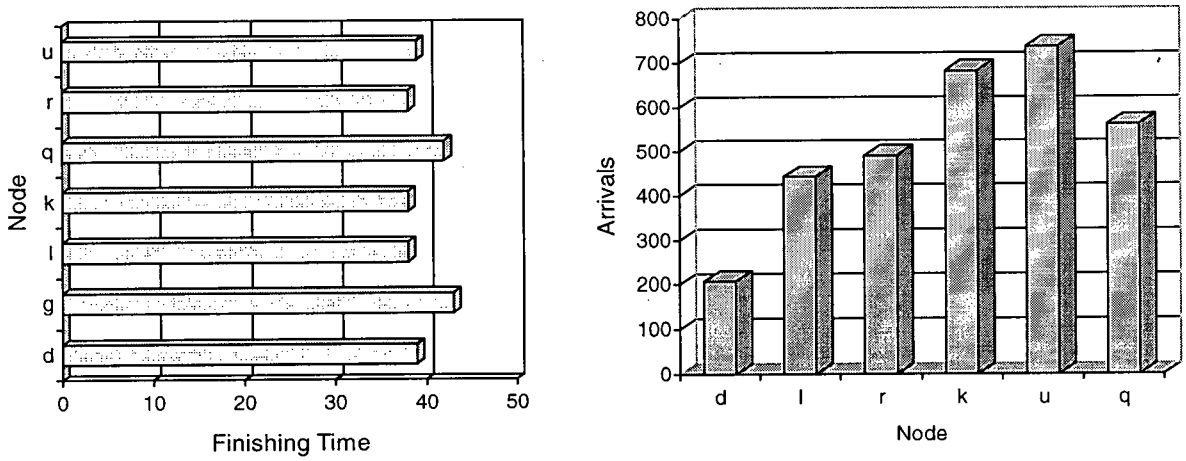


Figure 4.21 Finishing time and arrivals per node in a 20-node network with a 2-degrees destination node {g}

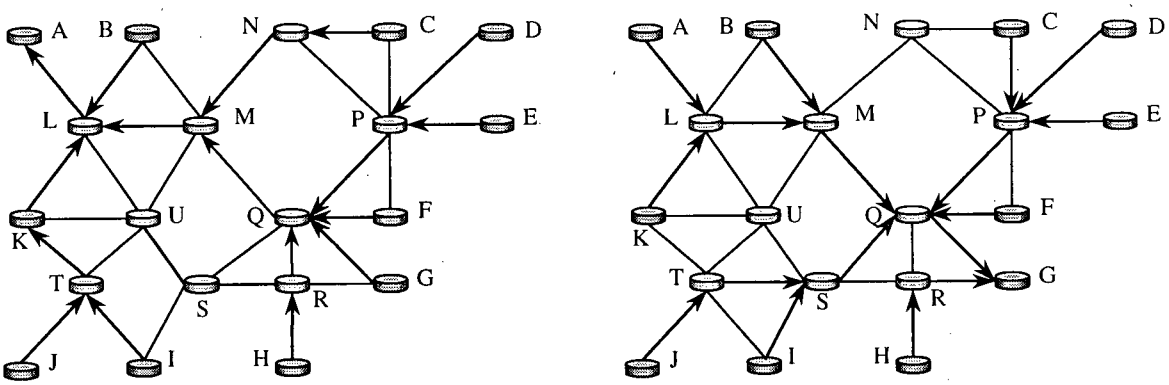


Figure 4.22 Final routes found during the mp2p tree construction in the 20-node network

30-Network Node Simulation

To complete the simulations performed for finding static mp2p trees, a final round of experiments are performed using the 30-node network. To obtain more conclusive results, four simulations with distinct destination nodes are conducted.

It can be observed that the occupancy at edge nodes follows a slightly different pattern than previously seen. In comparison to previous behaviour seen at the edge nodes in the 12-node network, the split-like burst of agent arrivals seems to fade away as the topology of the network becomes even more arbitrary, and larger in the number of nodes. This can be observed throughout the four different runs presented here with destination nodes 'r', 'd', 'g' and 's'. Further, the earlier observation made on the linear-like increase of the node's occupancy due to their higher degree disappears. Therefore, the previous assumption with respect to the relationship between arrivals of agents and node degree is false. On the other hand, the bursty behaviour of arrivals experienced at the core nodes remains invariant, although the 'spikiness' of the plots appears to be less narrow.

To observe the variability of the agents' occupancy during runs with different destination nodes (as in past experiments), a comparison of agent occupancy per monitored node is shown in figure 4.31. It can be observed that the variability of the occupancy at nodes of lesser degree is smaller compared to nodes of greater degree. Alternatively, a compilation of the total number of agents' arrivals per destination-node in the monitored nodes is also presented in figure 4.31. Although the monitored nodes see an increase in the number of arrivals when the mp2p routes for destination node 's' are computed, the overall results are still comparable.

Finally, figure 4.32 graphically depicts the final configuration of the mp2p trees found for each destination-node case.

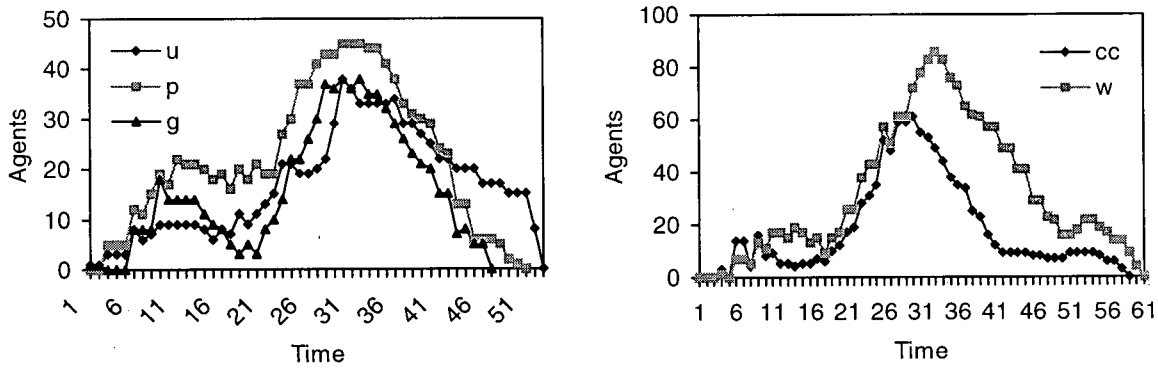


Figure 4.23 Occupancy of agents in a 30-node network with a 1-degree destination node {r}

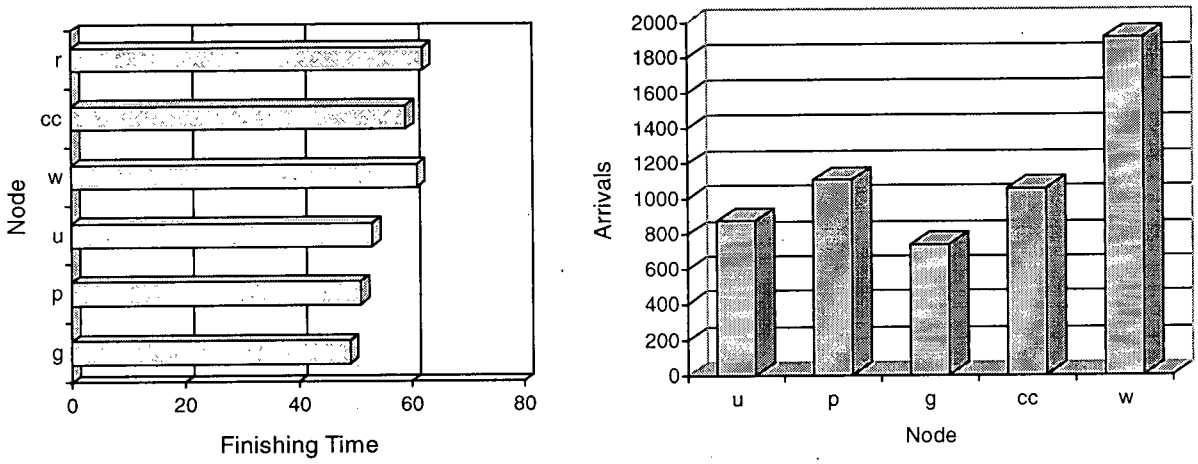


Figure 4.24 Finishing time and arrivals per node in a 30-node network with a 1-degree destination node {r}

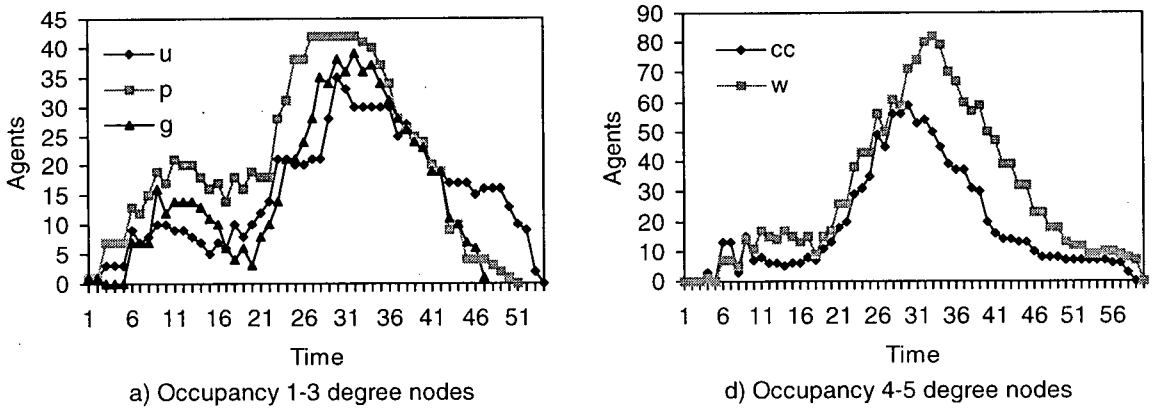


Figure 4.25 Occupancy of agents in a 30-node network with a 2-degrees destination node {d}

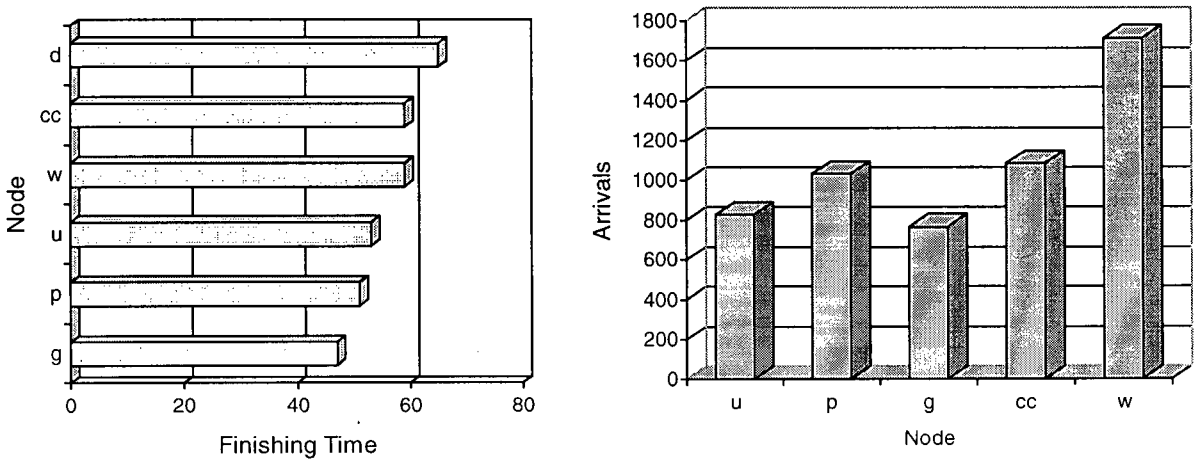


Figure 4.26 Finishing time and arrivals per node in a 30-node network with a 2-degrees destination node {d}

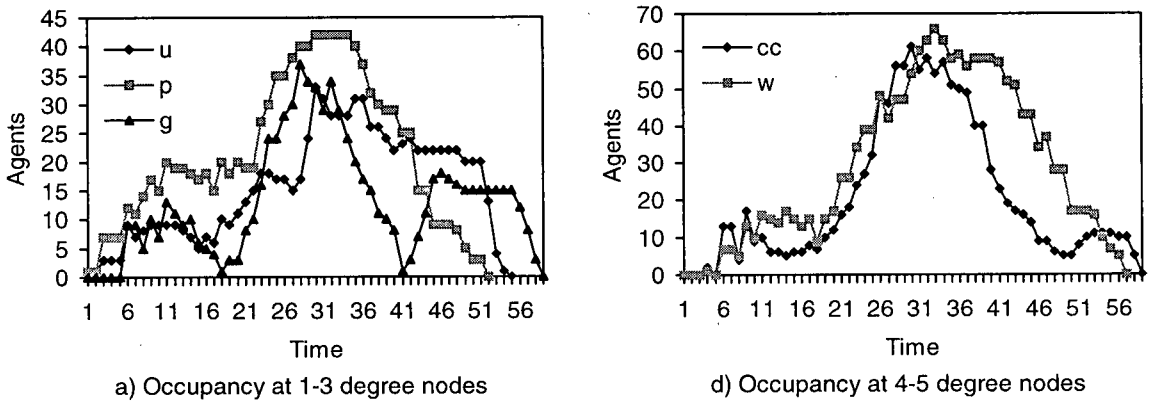


Figure 4.27 Occupancy of agents in a 30-node network with a 3-degrees destination node (g)

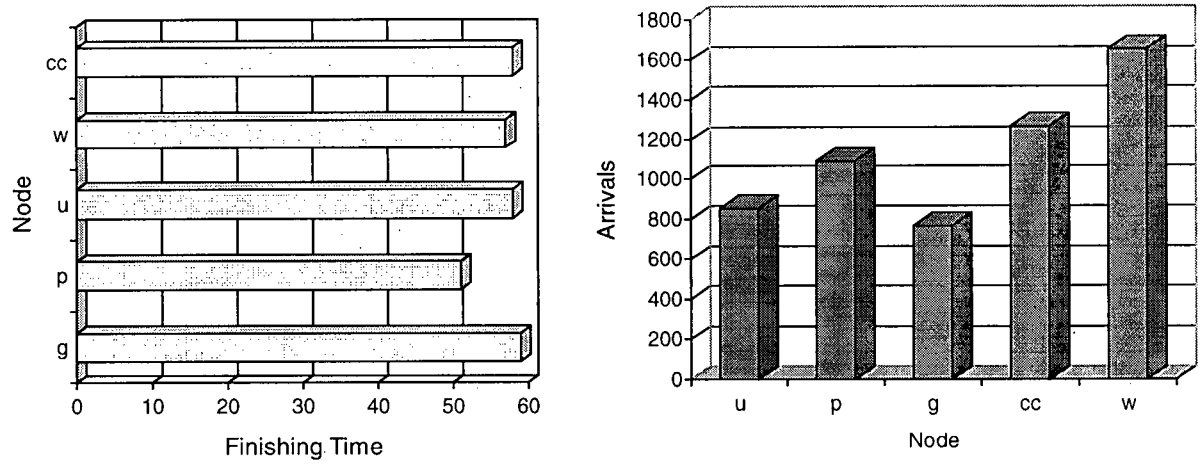


Figure 4.28 Finishing time and arrivals per node in a 30-node network with a 3-degrees destination node {g}

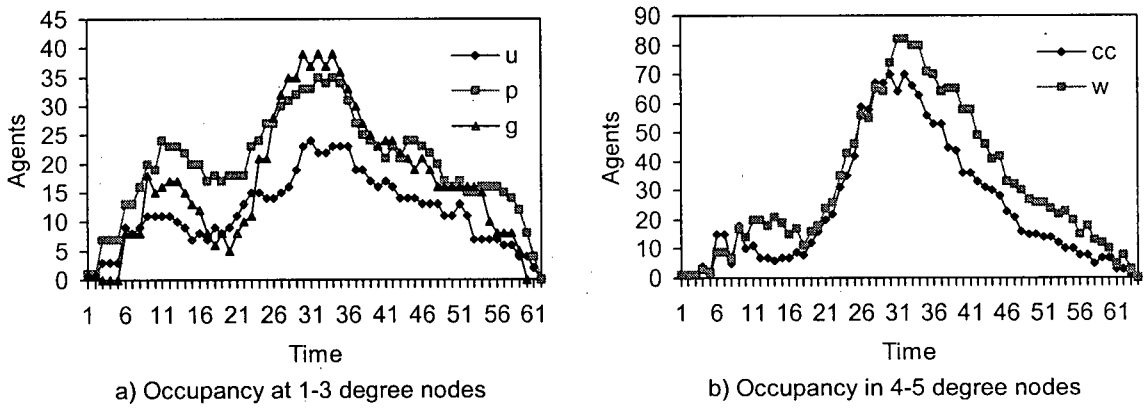


Figure 4.29 Occupancy of agents in a 30-node network with a 4-degrees destination node {s}

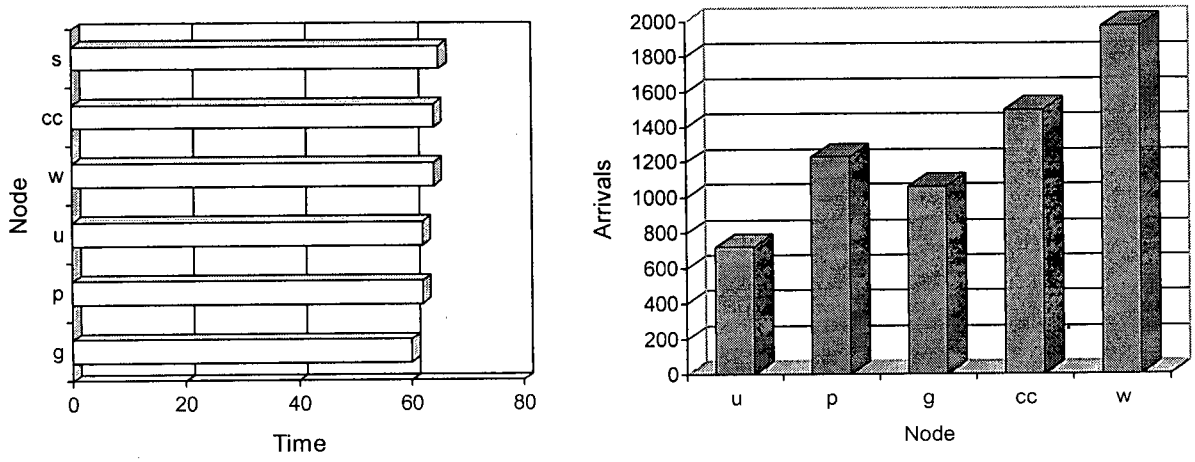


Figure 4.30 Finishing time and arrivals per node in a 30-node network with a 4-degrees destination node {s}

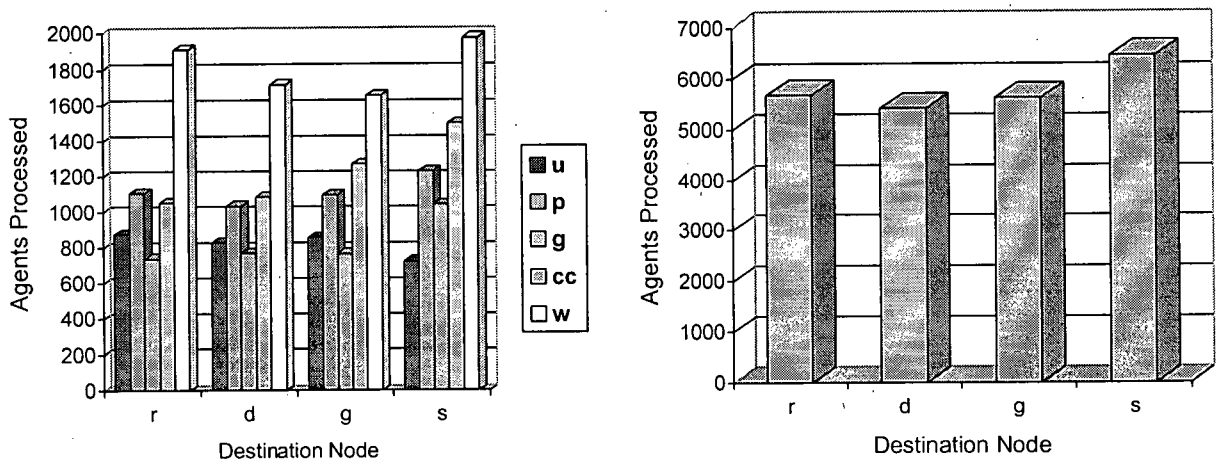
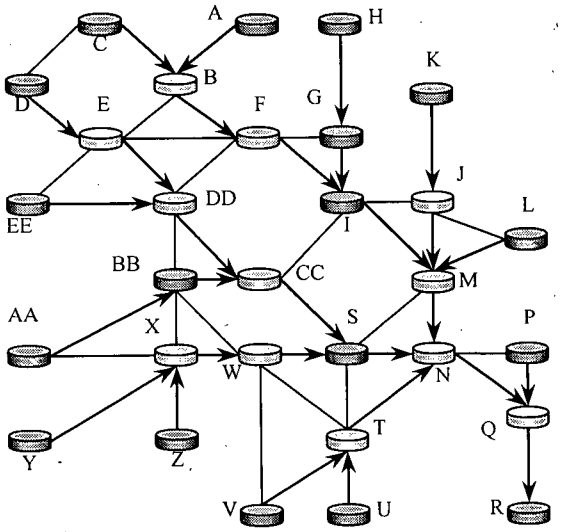
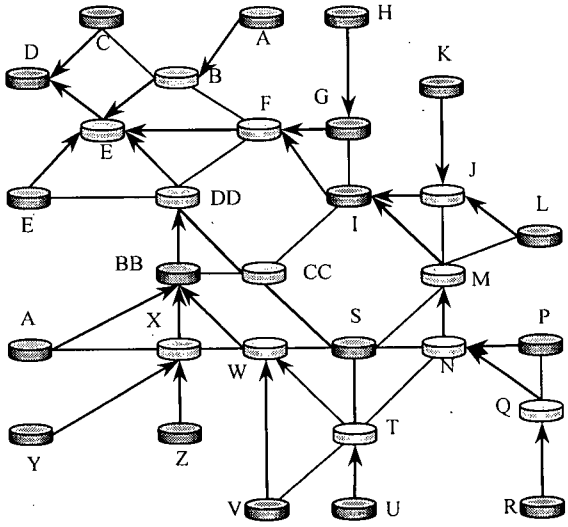


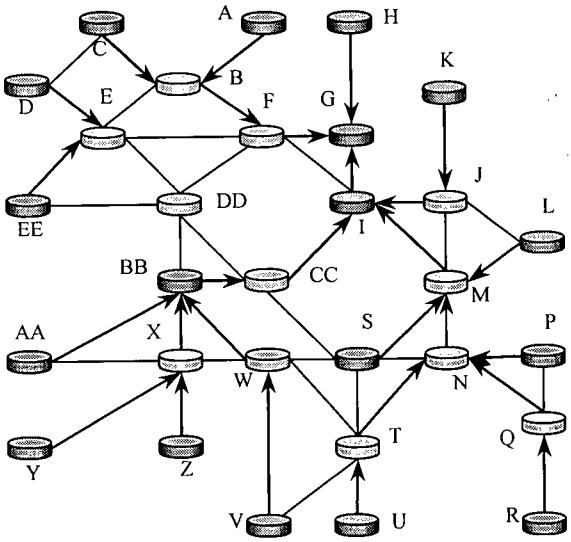
Figure 4.31 Variability in node occupancy and number of agent arrivals per destination node



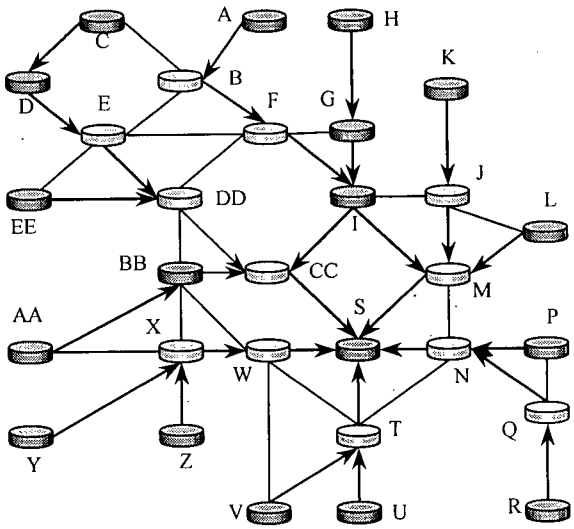
a) Tree constructed with destination node 'r'



b) Tree constructed with destination node 'd'



c) Tree constructed with destination node 'g'



d) Tree constructed with destination node 's'

Figure 4.32 Final routes found during the mp2p tree construction in the 30-node network

One final set of measurements is made with the intention of observing the time performance of the mp2p routing algorithms under different loads. The program runs with an increasing number of source nodes participating in a single connection. Consider the following example, where a

first run includes only one node source (e.g. node 'a'), with destination node 'd'. The second run of the experiment would include two source nodes (e.g. nodes 'a' and 'b'), and so on until all of the edge nodes in the network are included. This experiment is conducted for each of the networks used so far. The results of the experiments are presented in the next figure:

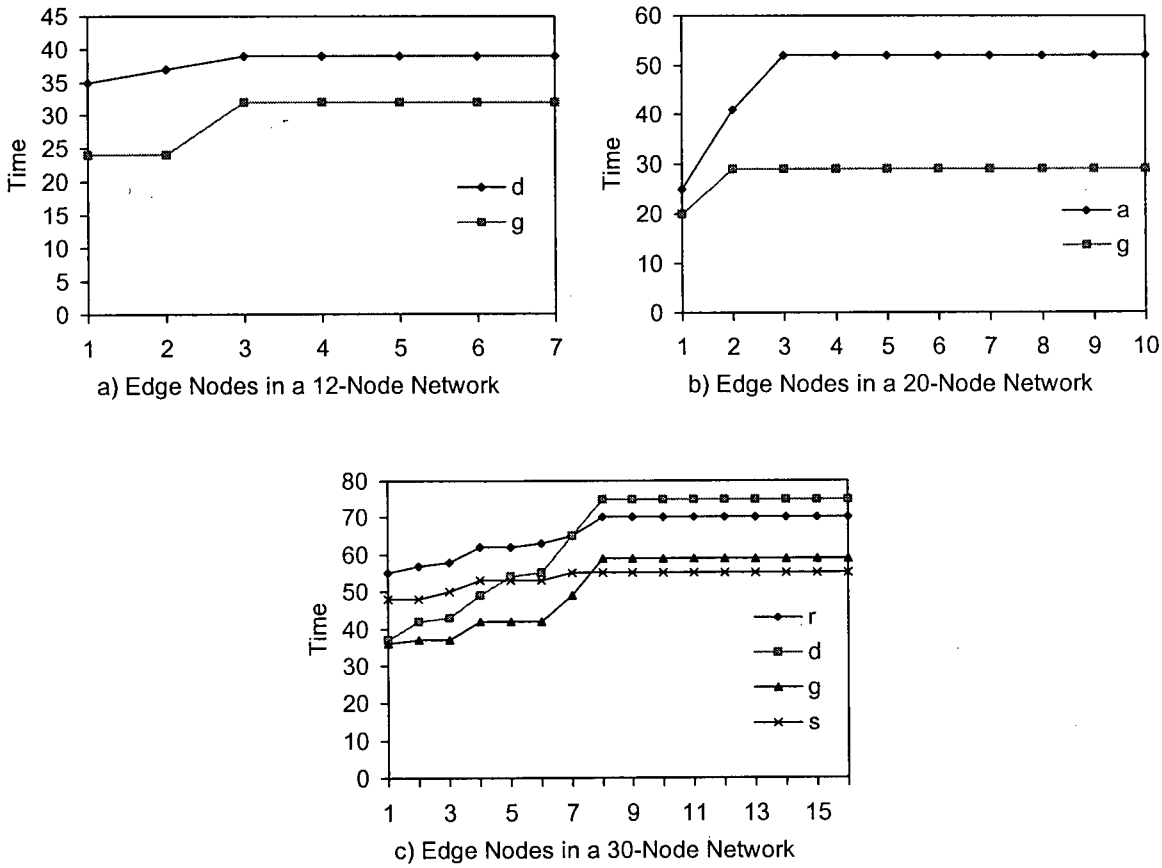


Figure 4.33 Time performance of the mp2p routing algorithm

It is interesting to note that for all the simulations performed, the time to complete the mp2p tree remains constant after a determined number of source nodes are incorporated into the procedure. Although there is little or no difference between the results of the 12-node and the 20-node network, it takes longer until the completion time of the mp2p routing procedure in the 30-node network becomes constant. It can also be observed that the degree of the final destination has no

significant influence in either the total number of agents required to obtain the mp2p tree or the time it takes to compute it.

Explaining the variability of agent behaviour observed during the experiments presented throughout this chapter is difficult, because no model has been formally established for studying and defining the migration behaviour of mobile agents in a communications network. However, the experiments carried out for the creation of static mp2p trees enable us to make some important observations, which are now described:

- The time it takes to discover a mp2p tree seems to converge to an upper bound when the number of origin nodes is increased
- The degree of a node has a direct impact in the number of agents processed; however, this does not seem to play a deterministic role in the overall results, as nodes with higher degree may see fewer agent arrivals, as well as the contrary
- The topology of the network hardly affects the traffic behaviour of the agents
- The arrival of agents at the network nodes follow a bursty nature

These preliminary results provide useful information that can be used in studying the behaviour of the routing system for the dynamic creation of mp2p trees. A deeper approach is presented in the next chapter to understand better the traffic characteristics of this cooperative mobile agent system.

Chapter 5 Dynamic Re-routing with Mobile Agents

In the preceding chapter, a number of results are presented after running simulations for building static mp2p trees. However, these results provided only partial knowledge of the characteristics of a routing system using mobile agents, since the behaviour of the agents cannot be completely studied during single-run routing simulations. Therefore, additional results are necessary to better comprehend the behaviour of a system that performs routing within the mobile agent paradigm. Furthermore, according to the assumptions made when working with the Diffserv QoS-framework, the need for building dynamic mp2p trees arises as requests from users to either join or leave specific sessions, pertaining to premium services, arrive in a random fashion. Therefore, this chapter serves a twofold objective: explaining how the routing algorithm can be modified to dynamically reconfigure an mp2p tree, while preserving network resources and maintaining the QoS needs of the existing connections. It will also help in studying the dynamic behaviour of mobile agent flows during the simulations.

The first section explains how the routing algorithm is enhanced to support dynamic optimization of an mp2p tree. The second section explains how the dynamic routing simulations were conducted, including the generation of random requests for leaving or joining the mp2p tree. In the third section, a simple delay model of the mp2p routing scheme is presented. Both the arrival and departure characteristics of the agents at the network nodes are considered, as well as the behaviour of the Wave interpreter under such circumstances.

5.1 Redefinition of the Multipoint-to-Point Routing Algorithm

The structure in which the algorithm is run to obtain the mp2p routes is re-arranged to allow portability for use in dynamic re-routing. The following sections discuss the need for modifying the mp2p wave algorithm to accommodate a number of routing sessions, while also discussing in detail the changes made to the program.

5.1.1 Supporting Multiple Routing Flows in the Same QoS-KN

The first aspect to consider is that the algorithm should be able to find individual mp2p routes that meet the QoS guarantees in an environment where other agents or waves might also be present during the same tree-construction process. This means that, in a realistic scenario, more than one set of agents in search of individual mp2p trees might be found, each of them with individual premises. For this reason, an agent colony performing a specific task should be able to function in the same environment without interfering or being interfered with by other waves.

As an example, consider the case of two separate agent colonies in search of two different mp2p routes with destinations X and Y. Each colony creates pertinent nodal variables, holding both identity and distance values in vectors to later perform indexed look-ups. These references rely on source-based identities to determine partial distances. This in turn means that the distances recorded in the vector variables are valid for the agent colony that has access to this data, depending on the wave that creates it according to its source node. A wave originating in the same node but with a different destination (belonging to a different mp2p routing session) has no way of knowing whether the records found represent valid distances measured with respect to the same source-destination premise. Therefore, a way of organizing such information must be created so that waves belonging to separate colonies do not interfere with one another.

```
1       Findspt=`
2       Fcollect=A.Fsource=C.
3       SQ(
4          RP(
5             Flinktype#.Flength+L.
6             OS(C/~Fedges,OS(goto#destinations,CR(goto#destinations))).
7             ID(
8                OS(
9                   (
10                   Fdestination/~Ndestination.
11                   Ndestination&Fdestination.
12                   CR(Fdestination#satellite)
13                  ),
14                  Fdestination#satellite
15               )
16            ).
17            OS(
18               ID(Fsource/~Nsource.Nsource&Fsource.Ndistance&Flength),
19               ID(
20                  Fsourceindex=Nsource.Fsourceindex::Fsource.
21                  Fdistindex=Ndistance.Fdistindex:Fsourceindex.
22                  Fdistindex==NONE,Flength<Fdistindex.
23                  Fsourceindex2=Fsourceindex.Fsourceindex2&@.
24                  Fsourceindex2&Flength.Ndistance:Fsourceindex2.
25               ).
26               #P.OS(goto#,)
27            )
28         ),
29         RP(
30            Fpath&C.
31            (
32             C==Fdestination.goto#destinations.Fpath%`'.
33             CR(Fsource#Fpath).Fpass=1.!3
34            ),
35            (
36             Flinktype#.Flength+L.OS(goto#hell,).Fdestination#.
37             ID(Fsourceindex=Nsource.Fsourceindex::Fsource.
38                Fdistindex=Ndistance).
39             Fdistindex:Fsourceindex.Flength==Fdistindex.
40             #P.OS(goto#,)
41            )
42         )
43      )'.
```

Figure 5.1 Modified Wave code to find multiple SPTs and support multiple routing mp2p sessions

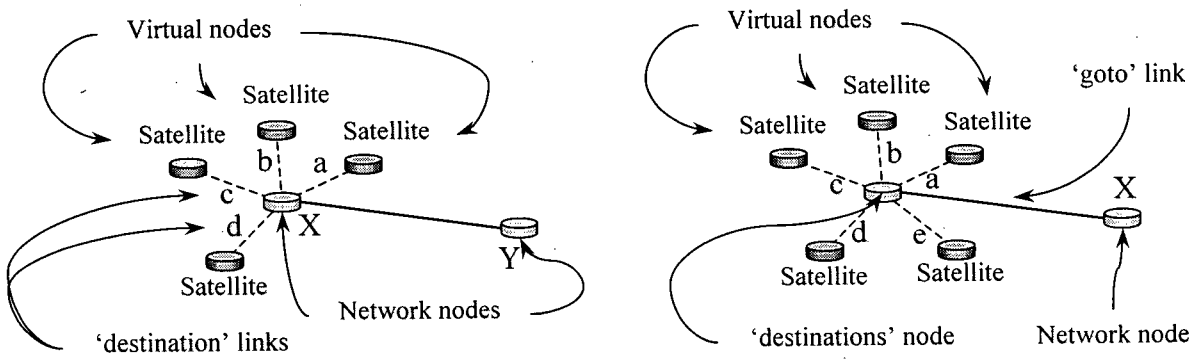
5.1.2 Modifications to the Multipoint-to-Point Routing Algorithm

In order to overcome the problem mentioned, it is necessary to modify the algorithm so that waves belonging to separate routing sessions may create vector variables that are meaningful only to the colony of agents that create them. This problem can be overcome by creating virtual nodes at intermediate nodes in the network to store information pertaining to individual routing sessions; thus, granting or restricting access to virtual nodes can be controlled easier. This means

that every part in the algorithm of the agent that either creates or accesses a vector variable needs to be modified. Figure 5.1 reflects the changes made to the 'Findspt' procedure for supporting dynamic re-routing. It can be seen that the overall process for finding multiple SPTs remains the same. However, additional wave instructions restrict access to specific vector variables, depending on the destination of the agents reaching an intermediate node:

Since there is no way of implementing multidimensional arrays of variables in Wave, the partial distances found by the 'Findspt' algorithm are stored in vector variables created in virtual nodes. Upon reaching an intermediate core node in search of a given SPT, a wave either jumps to an already existing virtual node, or creates one if it does not exist as shown in line 12 of figure 5.1. A virtual node is created with the name '*satellite*' to distinguish it from a regular network node. The virtual link tying the real core node and the virtual node is designated with the value carried in the variable 'Fdestination'. By doing this, vector variables containing specific distances valid for individual destination-bounded agents, can be stored in virtual 'satellite' nodes. Thus, when a wave arrives at an intermediate core node, it will jump through the link whose identity matches the value stored in the frontal variable Fdestination. Therefore, any access or modification to the pertinent variable holding partial distance values does not interfere with those of other destinations, since they are individually stored in different 'satellite' nodes. Should the agents reach not a core node but an edge node, the same procedure is followed; however, another virtual node is previously created with the name 'destinations' before creating the 'satellite' node. The purpose of this is to differentiate an agent already reaching its destination node from one reaching an intermediate node. This feature will be explained in more detail shortly. Figure 5.2 provides a clearer picture of the process just mentioned. The hops made to the created virtual nodes only take place inside the Wave interpreter; no actual hops are required in the physical network to access the nodes created, and no propagation time is incurred in, only processing time. There can be as many virtual 'satellite' nodes as source nodes exist in the network.

Concurrently, there can be as many virtual 'destination' nodes as exit points in the network. It is worth noting that any subsequent access to individual contents of vector variables is done through the 'satellite' virtual nodes, plus the 'destination' nodes in case of egress nodes. When a wave finishes its intended task in a satellite node, it returns to the actual physical node that contains it by 'jumping back' through the links it came from, as shown in lines 26 and 40 of figure 5.1. After performing this local hop, the agents resume their navigation task through the network.



a) Access to 'satellites' though core nodes in the network

a) Access to 'satellites' though 'destination' nodes from egress nodes the network

Figure 5.2 Creation of virtual nodes to restrict access to variables

In the case of the 'Fmarkpath' function used in the mp2p routing algorithm, the same enhancements apply as shown in figure 5.3. Lines 4 and 9 show the additional steps included in an algorithm for achieving the same goal of supporting several mp2p routing sessions.

```
1       Fmarkpath=`
2       Fcount=-2.
3       RP(
4          OS(goto#destinations,).Fdestination#.
5          OS(
6             ID(Nvisits==NONE.Nvisits=1.Nvisitedby&Fsource),
7             ID(Fsource/~Nvisitedby.Nvisitedby&Fsource.Nvisits+1),
8          ).
9          #P.OS(goto#,).
10         C/=Fsource.Ftemp=Fpath.Ftemp:Fcount.Flinktype#Ftemp.Fcount-1
11      ).ID(Nroutes&Fpath)'.
```

Figure 5.3 Wave code that finds possible merge nodes in SPTs while supporting multiple routing mp2p sessions

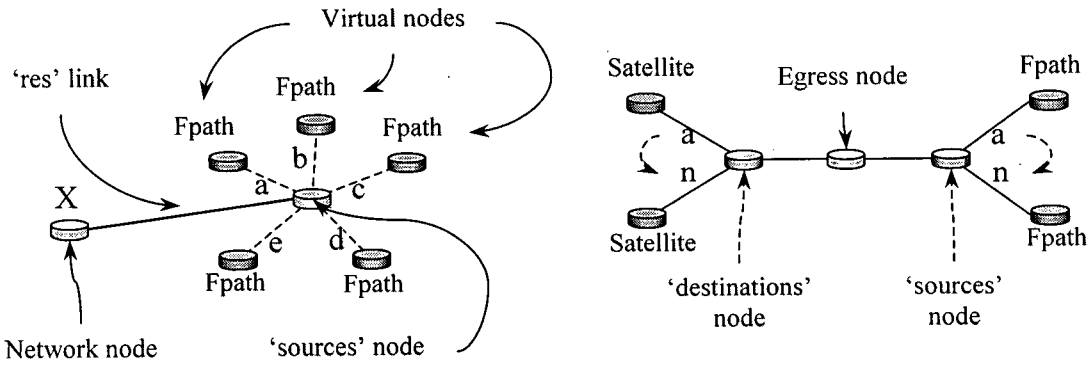
In the case of the 'Fcompete' function, similar steps are followed during the beginning of the algorithm as shown in lines 4 and 6 of figure 5.4. However, a slight variation is implemented in the last section for the definition of the final mp2p route. Notice that after the weight collecting procedure (lines 3 through 8), a second virtual node named 'sources' is created in all egress nodes. Returning to the pending explanation, the purpose of this extra node is to provide the means for differentiating waves in search of SPTs from those already in the last part of the mp2p tree set-up process. Thus, such waves are able to perform a simple operation for defining the final result, while also being prevented from interfering with other waves.

Upon reaching the intended destination node, the agents make an internal hop to the 'sources' node before performing their final operation; nevertheless, the individual branches of the mp2p tree are stored in the same manner as in the original algorithm. Therefore, it can be seen that the Wave interpreter at egress nodes in the MPLS network possesses the ability to process both agents in search of the mp2p tree, and agents in the final stage of the mp2p routing procedure.

```
1     Fcompete=`
2     Fnum=2.Fase=3.
3     RP(
4        OS(goto#destinations,).Fdestination#.
5        OS((ID(1<Nvisits).Fweight+1),).
6        #P.OS(goto#,).C/=Fdestination.
7        Ftemp=Fpath.Ftemp:Fnum.Flinktype#Ftemp.Fnum+1
8     ).
9     OS(res#sources,CR(res#sources)).
10    OS(
11       (
12        ID(Fsource/~Nedges.Nedges&Fsource.Nweight&Fweight).
13           Fpath%`'.OS(Fsource#,CR(Fsource#Fpath))
14       ),
15       ID(
16          Fsourceindex=Nedges.Fsourceindex::Fsource.
17          Fweightindex=Nweight.Fweightindex:Fsourceindex.
18          Fweightindex<Fweight.Fsourceindex2=Fsourceindex.
19          Fsourceindex2&@.Fsourceindex2&Fweight.
20          Nweight:Fsourceindex2.Fsource#.Fpath%`'.C=Fpath
21       )
22    )'.
```

Figure 5.4 Wave program for final definition of the mp2p tree and support of multiple mp2p routing sessions

Finally, figure 5.5 shows a graphic representation of how the 'destinations' virtual nodes are linked to the egress nodes to facilitate the processing of information towards the definition of the final mp2p tree.



a) Access to 'Fpath' though 'source' nodes from egress nodes the network

b) Access to virtual nodes at the network egress nodes

Figure 5.5 Graphical representation of virtual nodes at egress nodes

5.2 Simulating Dynamic Routing of Multipoint-to-Point Trees

Before carrying out the simulation runs for the dynamic mp2p tree construction algorithm, it is necessary to determine two important factors:

a) Define a way in which the mp2p routing algorithm can be invoked by a separate program

b) Build a separate program to generate random requests for mp2p routing sessions

It is then necessary to define the circumstances under which the mp2p routing algorithm is intended to run, which should reflect realistic scenarios if possible. Further, it is also important to consider how the Diffserv-over-MPLS QoS-framework would alter how routing updates are performed, as shown in the next sections.

5.2.1 Invoking the mp2p Routing Algorithm

Even though the building blocks of the mp2p routing algorithm have been defined, the original algorithm invokes the sub-functions 'Fmarkpath' and 'Fcompete' within the 'Findspt' procedure. This has the advantage of having a more compact code, but when adapted to run in dynamic re-routing situations it deletes tracks that were necessary for other SPT search processes, thus interfering in separate mp2p routing sessions, and causing erroneous results. Therefore, modifying the way in which the mp2p algorithm is invoked becomes primary. An improved function is then created to strategically call upon the building functions of the overall routing procedure, which is shown in the next figure.

```
1   Fmp2p=`
2   SQ(
3      WT(@#Fnodes.Findspt.OS((Fpass/=1.!3),)),
4      WT(
5         goto#destinations.Fnodes#.OS((C==satellite.!3),).Fpath=C.Fsource=L.#P.
6          (#P.C=NONE.!3),(goto#.Fpath|`'.WT(Fmarkpath).WT(Fcompete).!3)
7      ),\
8      (res#sources.Fnodes#.Nout=C.Nout|`'.T=Nout.C=NONE.!3),
9   )'.
```

Figure 5.6 Modified structure of the mp2p routing process

It can be observed that both the Fmarkpath and the Fcompete functions are not invoked inside the Findspt function anymore, but individually instead, which enables the execution of every section of the program in a more autonomous manner. Another enhancement is that the structure of this new design encloses each function in separate WAIT rules. This ensures that the final decision of the mp2p route is performed only after the preceding function ends. Finally, since there is now a specific function that calls upon the individual blocks that encompass the mp2p routing algorithm, a separate routine can be constructed to perform the experiments pertaining to the dynamic re-definition of the mp2p routes as depicted in figure 5.7. A simple analysis of the computational time complexity for this program is presented in Appendix D, resulting in a logarithmically bounded expression.

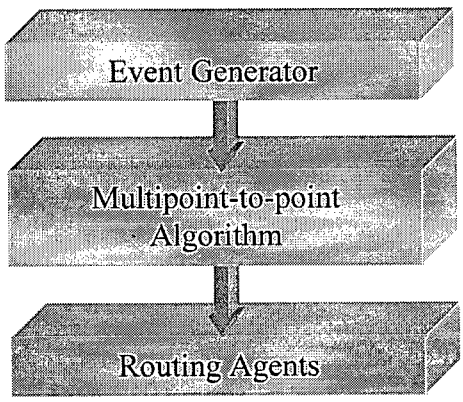


Figure 5.7 Simulation Model for Dynamic Routing

5.2.2 Design of the Routing Session Request Program

As previously mentioned, no system based on mobile agent technology was found that could be used as a benchmark for making concrete assumptions on the nature of the requests for mp2p connections. However, it is still possible to make fairly realistic assumptions for building a simple program that assists in the study of the traffic generated, when mobile agents are used for performing mp2p routing. For instance, it is observed that some classical routing schemes perform periodic updates of routing tables in 30-second intervals to keep network state information up to date [52]. It would not be realistic to expect that requests for connections triggering mobile agent traffic could actually follow this periodic pattern, so a scenario where such requests are received in a non-deterministic manner is considered, where the requests arrive in a more random fashion. Similarly, it is considered that a bounded number of ingress nodes make requests to dynamically join an mp2p connection, where the maximum number of simultaneous requests is determined by the number of ingress nodes in the AS, minus one (the egress node). It is, therefore, decided that a program be built to generate mp2p requests for connections with a bounded number of edge nodes participating in the request, while randomly generating their occurrence. The program is designed to generate a random number of batched requests within a predefined 60 second period, with a bounded number of participating edge

nodes. During this interval, the nodes making the requests follow a non-predetermined pattern, meaning that any node in particular may participate in the request. To achieve this, an additional wave program is also designed to make use of a program written in C language, which generates sequences of random numbers by using utilities provided by the Linux Operating System. The next figure shows the listing for the wave program used.

```
1     Fsimulate=`
2     @#Fedges.Fin=Fedges.Fin::C.Nstop=0.
3     RP(
4        Nstop/=1.ID(Fin?ran).Fin|``.Frandom=Fin.
5        Frandom:1.Frandom*6.Frandom?sleep.Frandom=Fin.
6        Frandom:2.Frandom+1.Fcountx=Frandom.Fcountx+3.Fnext=3.
7        RP(
8           Fnext<Fcountx.Frandom=Fin.Frandom:Fnext.
9           OS((Frandom==NONE.!1),).Frandom+1.
10          OS((Frandom/~Fsource.Fsource&Frandom),).
11          Fnext+1
12       ).
13       Frandom=Fedges.Frandom:Fsource.Fnodes=Frandom.Fdestination=C.
14       Fin%``.OS((Fnodes/=NONE.Fmp2p),).Fsource=NONE
15    )'.
```

Figure 5.8 A random connection request generation program

In accordance with the simulation model presented in figure 5.7 and the overall routing process previously described, the connection request generation program works as follows. First, a copy of the program is distributed to all the edge nodes in the AS by cloning the wave and hopping to the nodes contained in the 'Fedges' variable, which holds parameters that the simulation program receives before running. If the program is to be run in a different network, the 'Fedges' parameter should be updated accordingly. Line 2 of the program performs the pertinent operations to map single integer values to the true identity of the edge nodes, such as 1→ A, 2 → B, and so on. Lines 3 through 15 enclose the code used to call upon the mp2p routing program to perform the actual simulation. The wave program then uses its interfacing feature to obtain a random number, the actual value of which is obtained by an external C written program. The program makes use of readily available functions that interact with the operating system to

obtaining a pseudo-random number generating sequence with a period of 2^{31}-1. A statistical goodness-to-fit (GOF) test was performed to verify the pseudo-randomness of the values obtained, by using commercially available software [64]. Figure 5.9 shows the resulting graphic for the GOF tests performed using Chi-square, Kolmogorov-Smirnov, and Anderson-Darling tests, which yield a uniformly distributed probability function. This result means that the chance of obtaining any given value between the lower and upper bounds is fairly even.

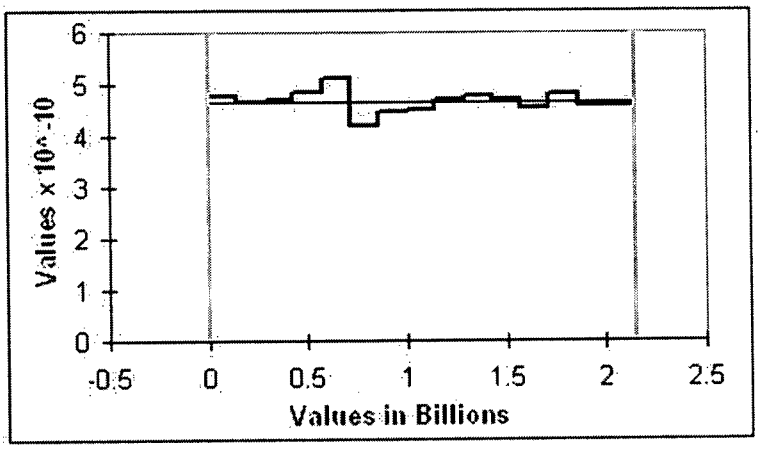


Figure 5.9 Distribution of the pseudo-random number sequence generated

The number obtained is decomposed in single integer scalars ranging from 0 to 9, which are used for specific simulation purposes. Figure 5.10 shows the use of each scalar obtained. The first number considered from left to right is used as a delay value by multiplying it by 6 (line 5 of program shown in figure 5.8). This means that the simulation program will delay its execution within a period of 6 seconds as a minimum and 54 as a maximum.

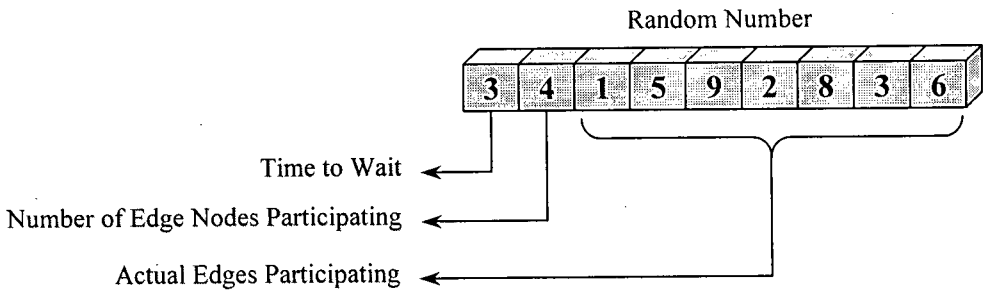


Figure 5.10 Use of individual integers in the events generation

Once the waiting period is finished, the integer values following the one just discussed are used to determine the nodes that ultimately participate in the request for the mp2p tree creation.

Using the mapping definition explained previously, the second integer from left to right is used to indicate how many nodes will participate in the connection (line 6), while the rest of the numbers are used according to the previous value (lines 7-12). Using figure 5.10 as an example, if the number being under considered has a value of 4, then the next four scalars are considered for the creation of the tree as origin nodes. If those obtained integers were 1, 5, 9 and 2, then the nodes A, E, I and B are be used as origin nodes, according to the mapping distribution defined. All the remaining numbers, if any, are discarded. On the other side, if the number of numbers required for the creation of the tree is insufficient, then as many scalars as available are used. After this mapping is performed, the rest of the event generation program is ready to call upon the mp2p routing algorithm to obtain the desired routes. The outcomes obtained by using the scheme just described in the experiments conducted are presented in the next section.

5.3 Delay Properties of the Routing System With Mobile Agents

When designing a new routing algorithm it is important to study the delay characteristics of the proposed scheme to predict its performance under actual circumstances. To accomplish this, a routing delay model must be obtained, that is, a mathematical expression to formally define the factors that play an essential role in the delay incurred by the routing algorithm to produce a final result.

5.3.1 The Routing Delay Model

As a first step to define the time it takes for the algorithm to find an mp2p tree as requested, it is necessary to consider the course followed by the mobile agents during their search task. In this regard, figure 5.11 illustrates the direction taken by the agents during each stage of the routing process, followed by figure 5.12, which defines the amount of delay incurred by any given agent while traversing the network.

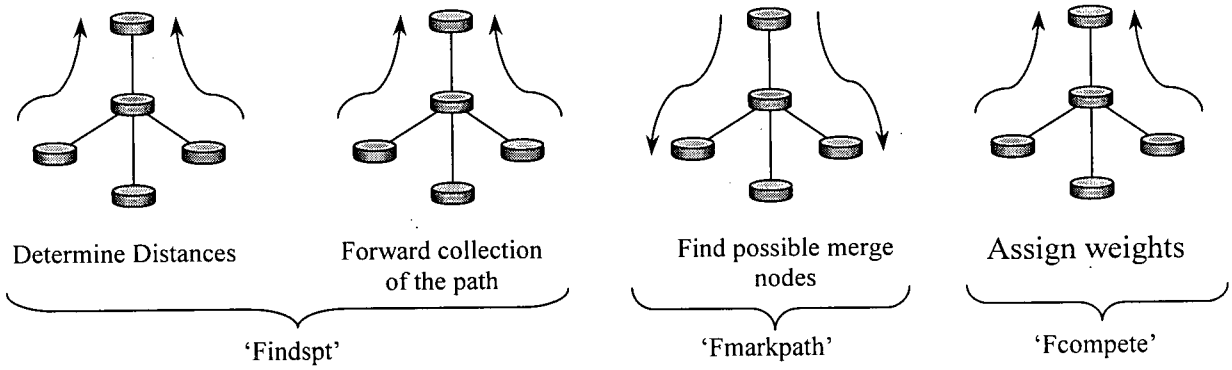


Figure 5.11 Direction followed by mobile agents during each step of the routing process

From the above figure it can be seen that the delay incurred by every agent adds up, according to both the number of nodes in the network and the propagation time between each node. The maximum number of nodes that an agent has to traverse is bounded by the diameter of the QoS-KN. Every stage in the algorithm experiences the same latency, where the total delay from one end node to the other is depicted by the sum of propagation delays, plus the sum of queuing delays of the nodes traversed as shown in figure 5.12.

Considering every stage in particular, the delay incurred by every agent can be formally expressed as follows:

- For the 'Findspt' section

$$Total\ Delay = 2(Processing\ time)(diameter) + 2(diameter-1)(Propagation\ delay)$$

Or

$$D_{TI} = 2LDp + 2Tp(L-1)$$

$$D_{TI} = 2[LDp + Tp\ (L\text{-}1)]$$

$$D_{TI} = 2[L\ (Dp+Tp) + Tp] \qquad (1)$$

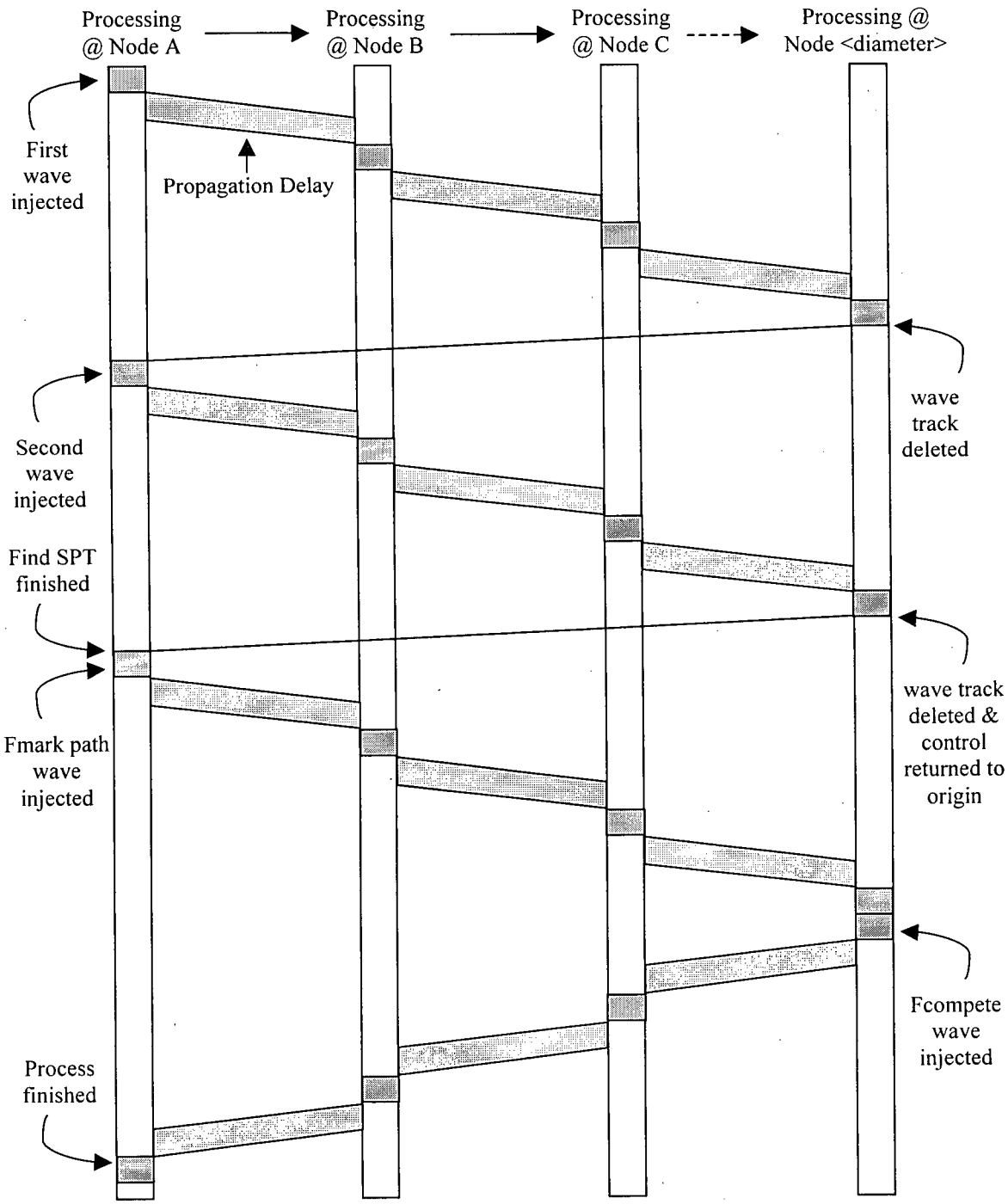


Figure 5.12 Delay diagram for the mp2p routing algorithm

- For the 'Fmarkpath' and the 'Fcompete' stages the procedure is analogous, then

$$D_{T2} = 2[L\ (Dp+Tp) + Tp] \quad (2)$$

Therefore combining (1) and (2), the total delay for the mp2p routing algorithm is defined thus:

$$D_T = 4[L\ (Dp+Tp) + Tp] \quad (3)$$

For simulation purposes, the propagation time can be assumed to be either zero or near zero, and the above formula can be simplified:

$$D_T = 4LDp \quad (4)$$

The total delay is then reduced to two factors: the diameter of the QoS-KN, and the processing delay at the nodes. The actual value of the network's diameter is not a constant but a variable, since the QoS-KN is to be updated upon changes in the availability of resources in the network. In the worst case, the longest diameter is found when all of the individual network links are able to offer the required QoS; in other words, the QoS-KN matches the physical communications network. In the case of a processing delay at the nodes, the analysis becomes more complicated, as it is necessary to determine the queuing characteristics of the system being considered. Such characteristics are addressed next.

5.3.2 Arrival Characteristics of Agents

To determine the type of queuing delay experienced in the Wave interpreter at each intermediate node, it is helpful to examine in more detail the process in which the agents are launched into the routing process. It is observed that every edge node waits a pseudo-random time period before receiving a request for finding an mp2p tree. Once the mp2p routing algorithm receives a request for performing routing, not one, but a number of agents are launched into the network. This means that a batch (bulk) of agents is indeed generated for routing purposes. As explained in previous sections, the number of agents searching for the mp2p tree is further increased, as

cloning of such agents takes place during the routing process. A consequence of this is that, during the routing process, any given agent reaching a node will most likely have to wait for service by the single Wave interpreter at an intermediate node, until other queued waves in his batch receive service first (as in a First Come First Served based queue). However, during the simulations conducted for the dynamic routing case, it can be seen that individual ingress nodes might eventually receive routing requests simultaneously. It should then be evident that the time spent by an agent waiting for service at a given node depends not only on the number of waves in front of the agent belonging to the same routing batch, but also on the agents from other routing batches also in front of the batch, as shown in figure 5.13.

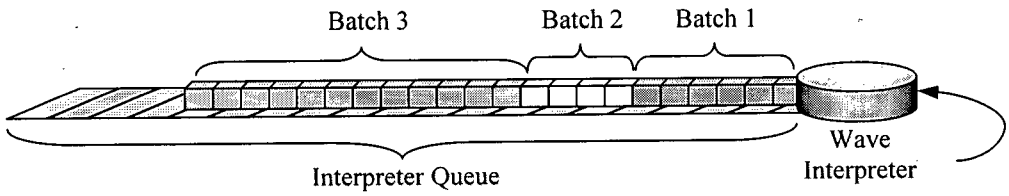


Figure 5.13 Batch arrivals of agents during the routing process

In light of this, the arrival of agents at any network node is expected to follow a fairly bursty pattern, which is indeed the case. An example of a 2000-second interval of agent arrival is shown in figure 5.14.

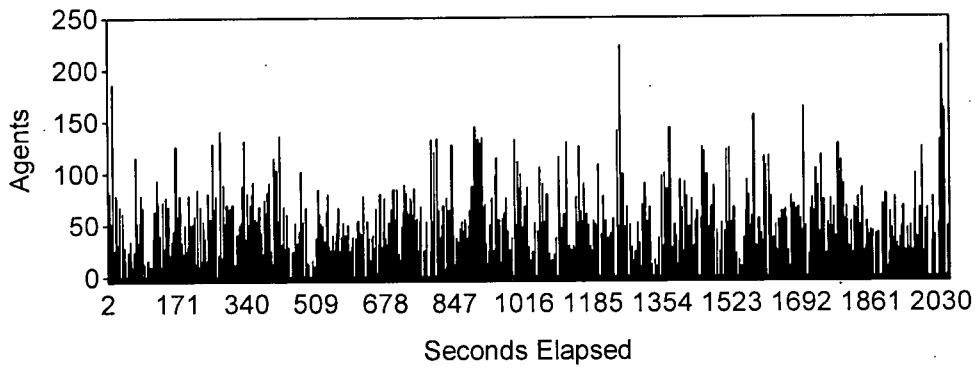


Figure 5.14 Snapshot of agent arrival at a random node

Figure 5.15 provides an illustration of the measurements made in the monitoring of arrival of the waves during the experiments conducted, which include inter-arrival times between batches reaching a node, arrival rate of the batches, and the size of the batches.

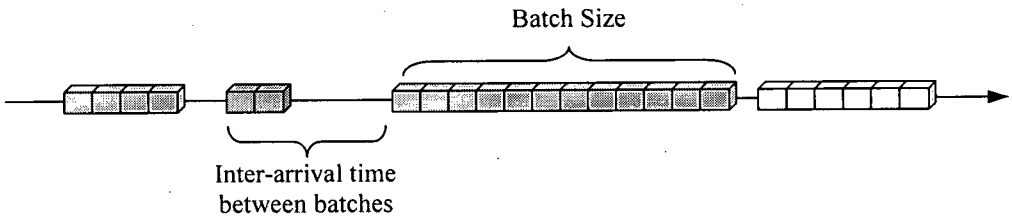


Figure 5.15 Batch-related measurements performed on the arrival of agents

The first experiments are aimed at determining the arrival pattern of blocks of agents being sent off, as requests for routing sessions are being honoured. According to the previous rationale, batches of agents are received at intermediate nodes in the network shortly after the routing sessions begin. Recalling results observed during the previous chapter, intermediate nodes are observed to experience a higher number of agent arrivals throughout the network. Therefore, measurements are performed at randomly chosen intermediate node to determine the worst-case arrival pattern that a bulk of agents displays at intermediate nodes. This is the only case where actual time measurements are conducted. The outcome of the measurement is shown in the next figure, which is a direct result of a GOF test performed over data on batch arrivals.

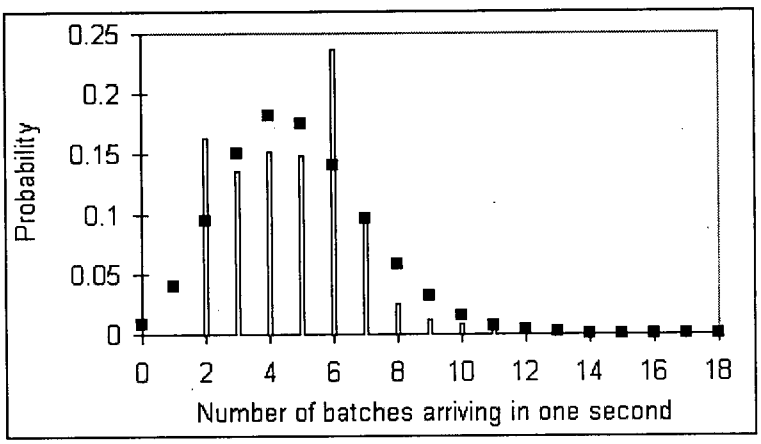


Figure 5.16 Distribution of batch arrivals

According to the results obtained, the bulk arrivals best fit a Poisson-distributed pattern, with a rate of 4.8 batches per second. In figure 5.16, the squared dots represent the Poisson discrete-plot with the specified rate, whereas the bars represent actual measured data in the number of bulk arrivals per second. The probability density function of the Poisson random variable is described by this equation:

$$f(x) = \frac{e^{-\lambda}\lambda^{x}}{x!}$$

A second set of measurements is performed to determine the distribution function that best fits the inter-arrival times of batches at any given node. According to the statistical GOF test performed, the best approximation results in an exponentially distributed function, with density:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$$

The resulting plot for the GOF test performed is shown in figure 5.17, in which the bars represent the actual binned data and the curve is the estimated plot.

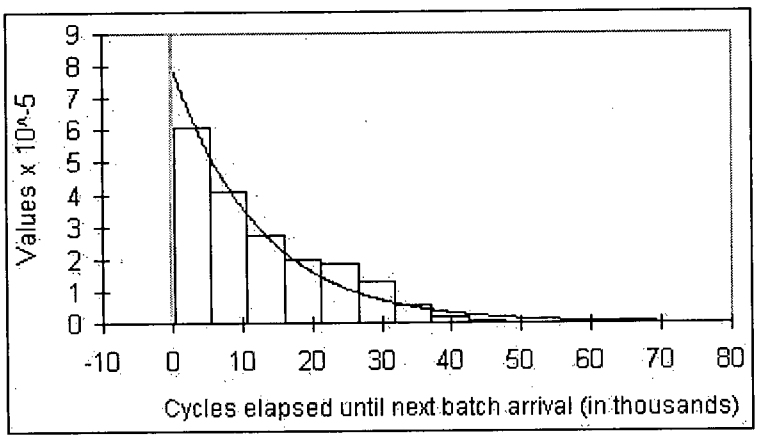


Figure 5.17 Probabilistic pattern in the arrival of agents' batches

A third measurement was performed to determine the length of the batches observed arriving at the intermediate nodes during the simulations. According to the GOF test carried out, the length of the batches is characterized by a gamma-distributed random variable, as shown in figure 5.18.

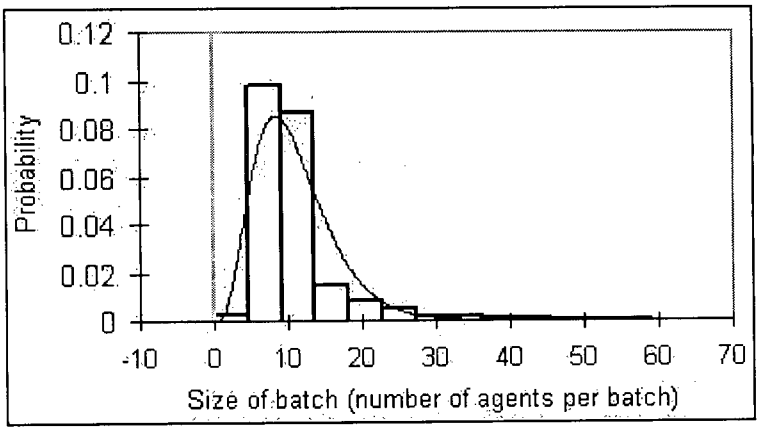


Figure 5.18 GOF test for the agents' batch size depicting a gamma distribution

Again, the bars represent the actual binned data and the single curve stands for the estimated probability plot. The probability density function of the gamma random variable is depicted thus:

$$f(x) = \frac{\lambda(\lambda x)^{\alpha-1} e^{-\lambda x}}{\Gamma(\alpha)} \qquad where \quad \Gamma(\alpha) = \int_0^\infty e^{-x} x^{\alpha-1} dx$$

The estimated values for the mean and variance of the batch size are 10.8728, and 27.0861 agents respectively. According to the observations made in the experiments carried out, it is clear that the arrival pattern of the agents performing routing follows a Markovian (Poisson) process with batch arrivals with gamma-distributed length. Such results confirm the original assumptions made at the beginning of this section.

5.3.3 Departure Characteristics of Agents

Characterizing the departure rate of mobile agents leaving a Wave Interpreter is much simpler. It is evident that the size of the agents does not change in time, and the execution of the code evolves depending on the stage of the general routing process. A quick look at every procedure of the mp2p routing algorithm reveals that during the routing process simple information retrieval and storage procedure take place, along with simple assigning operations. In essence, there is only a minimal difference in the type of computation performed by a Wave Interpreter every time that an agent comes into service. This simple reasoning leads to the assumption that the service time of every agent may be considered deterministic. This assumption is believed to be accurate enough, since the variability on service time among different waves is minimal, even though the current executing stage may be different. Having defined both the arrival and departure characteristics of the agents at the network nodes, observations with reference to the queuing delay at the Wave Interpreters are examined in the next section.

5.3.4 Queuing Delay at the Wave Interpreter

Recalling the analysis shown in section 5.3.1, a delay model is defined to establish the latency experienced by an agent in the routing procedure as asserted in equation (4). After reviewing the results presented during the past two sub-sections, the before mentioned expression can now be addressed in more detail to finally determine the processing delay (Dp) value. To determine an equation that accurately reflects the processing delay experienced by an agent at a single node, it is necessary to determine the type of queuing delay taking place at the intermediate nodes in the network. This necessary information is obtained by observing the properties of the arrival process, followed by the wave agents performing routing, as well as by the nature of the service delay experienced by the same agents at the nodes. Within the context of queuing theory, the

results obtained clearly reflect the behaviour of a $M^{[X]} / D / 1$ queue. As defined in the literature [12], this notation can be used to represent a Poisson-distributed batch-arrival pattern, where $M^{[X]}$ indicates that the inter-arrival time of the batches are exponentially distributed, with a batch size of [X], and agents being processed and dispatched in a deterministic (D) manner by a single Wave Interpreter. The equation for obtaining a steady state expected number of customers (waves) in the system (in queue and being served) has been already obtained and presented in [16], and is defined by the following expression:

$$L = \frac{\rho\left(\sigma_a^2 + \bar{a}^2\right)}{2\bar{a}(1-\rho)} + \frac{\rho}{2} \qquad \text{where} \qquad \begin{array}{l} \sigma_a^2 = \textit{Variance of batch size} \\ \bar{a} = \textit{Mean value of batch size} \\ \rho = \textit{Utilization factor} \\ L = \textit{Expected number of customers in queue} \end{array} \tag{5}$$

This expression can be used to find the mean time that a wave has to wait at each intermediate node while travelling towards its final destination. In particular, Little's theorem [12] can be used to obtain the waiting time in the system as follows:

*Number of customers in the system = Arrival rate * Time spent in the system by i^{th} customer*

Or

$$L = \lambda Dp$$

Thus,

$$Dp = L/\lambda \tag{6}$$

It is important to notice that the arrival rate depicted here represents that of batches, and not of actual agents. Considering the relationship between the number of edge nodes and the diameter in a tree as shown in Appendix D, and combining equations (4), (5) and (6), the following expression results:

$$D_T = 4\,Log_2 N\left[\left[\frac{\rho\left(\sigma_a^2 + \bar{a}^{-2}\right)}{2\bar{a}(1-\rho)}\right] + \frac{\rho}{2}\right] \Big/ \lambda \qquad (7)$$

Having established this, it is now possible to determine an upper bound for the processing delay experienced by agents reaching an intermediate node towards the destination. It is seen that the arrival of the batches follows a Poisson distribution, and the length of the batches follow a gamma distribution. Considering the practical results obtained in the simulations, table 5.1 shows some numerical values obtained when using equation (7), while considering a range of utilization factors (ut), and edge nodes (N). Figure 5.19 shows the resulting plots for the tabulated values.

N	L	Dt (ut=.1)	Dt (ut=.2)	Dt (ut=.3)	Dt (ut=.4)	Dt (ut=.5)	Dt (ut=.6)	Dt (ut=.7)	Dt (ut=.8)	Dt (ut=.9)
1	0	0	0	0	0	0	0	0	0	0
2	1	0.660369	1.475415	2.511425	3.878883	5.776658	8.602487	13.28442	22.60663	50.48992
3	1.585	1.046661	2.338477	3.980514	6.147885	9.155787	13.63462	21.05532	35.83067	80.02464
4	2	1.320739	2.950829	5.02285	7.757767	11.55332	17.20497	26.56885	45.21327	100.9798
5	2.322	1.53333	3.425807	5.831348	9.006488	13.41299	19.97436	30.84548	52.49098	117.234
6	2.585	1.70703	3.813891	6.491939	10.02677	14.93244	22.23711	34.33974	58.4373	130.5146
7	2.807	1.853891	4.142012	7.050461	10.8894	16.21713	24.15024	37.2941	63.46484	141.7431
8	3	1.981108	4.426244	7.534275	11.63665	17.32997	25.80746	39.85327	67.8199	151.4698

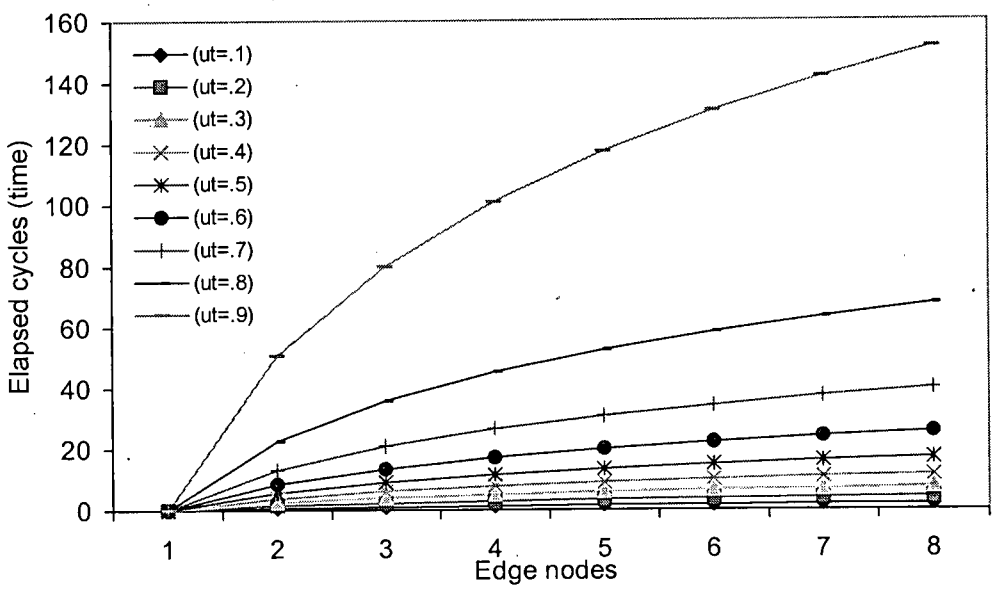


Figure 5.19 Estimated delay of agents during the routing process

On the other hand, figure 5.20 shows the actual routing delay obtained after performing simulations. It can be seen that the plot resembles the one presented in the previous figure, which obeys a logarithmic shape. However, it can be seen that the processing delay scales of both figures do not match. This can be explained by the fact that the analysis presented previously considers that all the computations are performed in a distributed manner; that is, several Wave interpreters on top of every hardware switch perform computations across the network. However, for the simulations presented here, all the processing is performed in a single computer; therefore, the Wave interpreter has to divide its processing time among all the agents waiting to be served. In an actual implementation, the processing time should reflect a decreased processing value, as the routing computations are distributed among all the Wave interpreters in the QoS-KN.

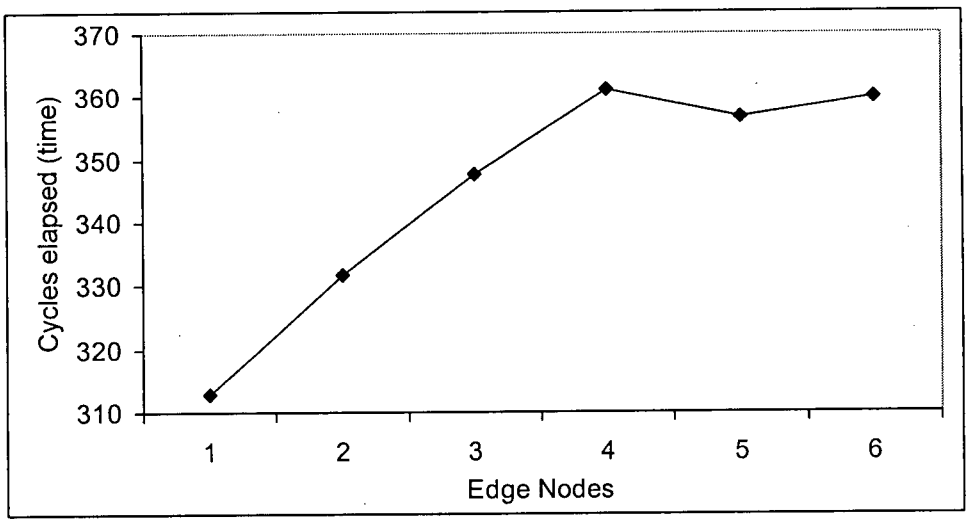
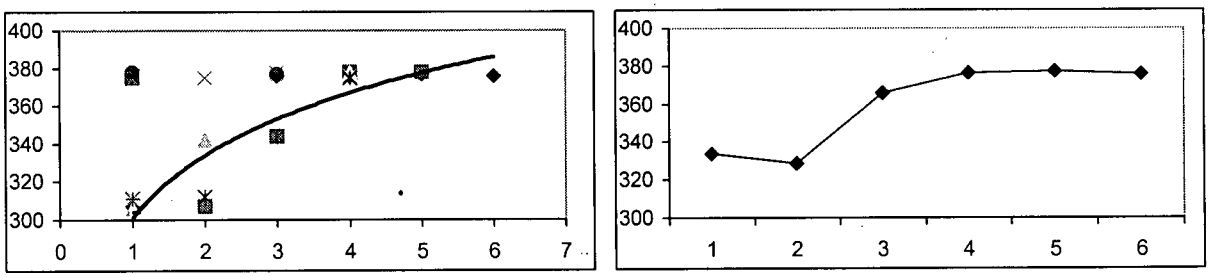
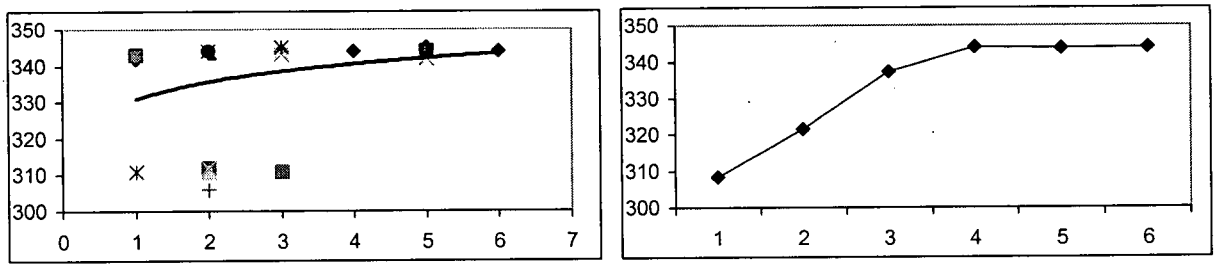


Figure 5.20 Overall average routing delay at the edge

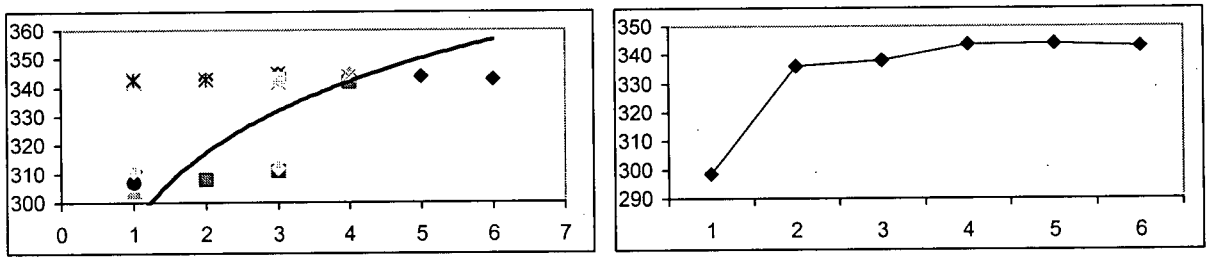
In addition, figures 5.21 and 5.22 show the results for each edge node participating in an mp2p connection as the root of the tree within the network used for the simulations. The figures on the left depict actual readings on scattered-type charts with trend lines depicting the average estimated delay, whereas the plots on the left show the actual routing delay time experienced at each respective node.



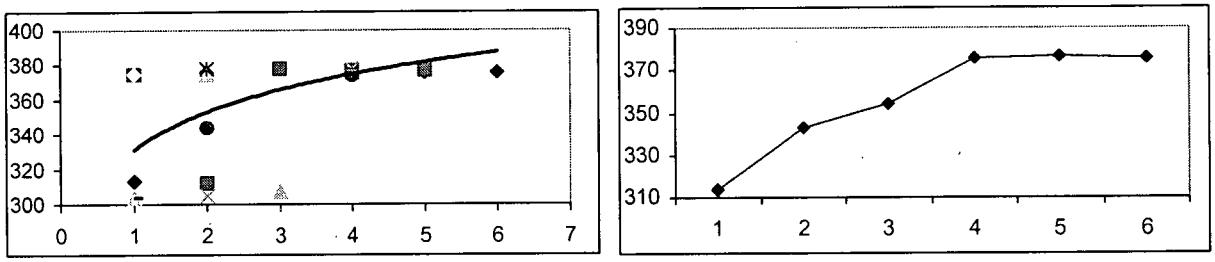
(a) Delay results for node 'a'



(b) Delay results for node 'b'

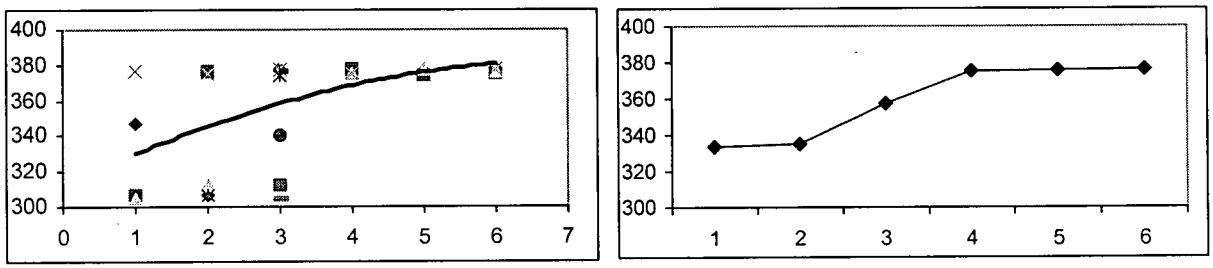


(c) Delay results for node 'c'

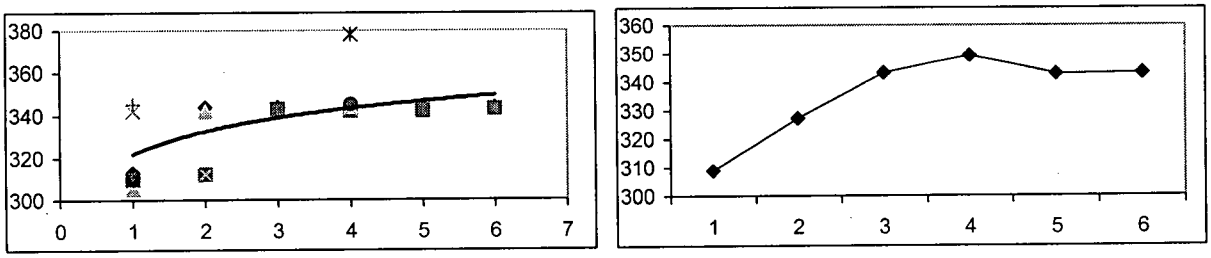


(d) Delay results for node 'd'

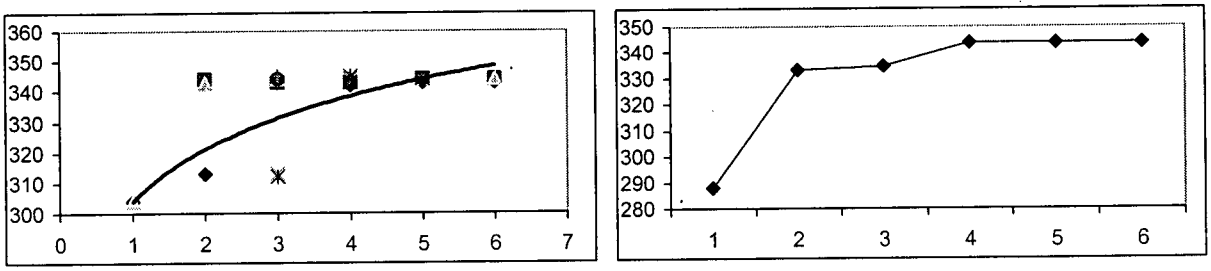
Figure 5.21 Estimated and actual routing delay at the edge nodes a-d



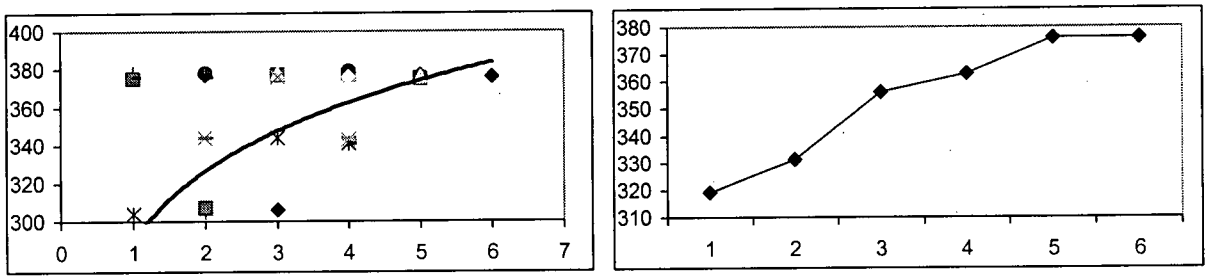
(e) Delay results for node 'e'



(f) Delay results for node 'f'



(g) Delay results for node 'g'



(h) Delay results for node 'h'

Figure 5.22 Estimated and actual average routing delay at the edge nodes e-h

It is important to consider that a number of assumptions have been made in this work, which help explicate the behaviour of a routing system based on mobile agent technology. Routing is a complex task that requires close interaction with other networking tasks. However, the assumptions made are believed to be an accurate basis for the building of a simulation that can illuminate the behaviour of mobile agents programmed for this specific task.

Recalling the objectives formulated for this work, the previous analysis accomplishes two purposes. First, not only is an efficient algorithm for finding mp2p routes using a novel methodology presented, but a brief analysis is also provided to give an insight into the real behaviour of the agent-based routing system. The information obtained may prove useful if an actual mobile agent system is to be designed to support networking tasks. In addition, the previous analysis also assists in reinforcing the analysis presented in Appendix D, which suggested that the time complexity of the designed algorithm is bounded by a logarithmic expression. It can be seen that both the simulation results and the theoretical analysis, along with the time complexity examination, provide a close match that validates results obtained individually. Further, important considerations are pointed out in the next chapter, which concludes this thesis work.

Chapter 6 Conclusions

This thesis work concludes with comments on both the results obtained and presented during the last two chapters, and on the importance of taking into account other factors directly involved with the proposed routing scheme. A number of final considerations and remarks are presented next.

6.1 Interaction With Other Technologies

Although routing is important in the overall consideration of how to achieve QoS at the network layer in the Internet backbone, there are other factors also. It is important to determine the way in which the proposed routing scheme should consider the mutual interaction between itself and other networking tasks, such as resource reservation, connection admission control and traffic engineering. A brief discussion of each individual interaction now follows:

- *Traffic Engineering*: The interaction between routing with mobile agents and traffic engineering is one of the most important points to consider. During the simulations performed, it is seen that the agents attempt to find optimal routes on behalf of the user, while minimizing network resources. However, an uncontrolled selection of routes leads to inappropriate distribution of resources throughout the network, in turn leading to network congestion. For instance, mobile agents might converge at finding specific paths, which individual data transfer sessions would follow while traversing the same network region. Although the objective of finding adequate QoS routes and minimizing network resources may be achieved, the distribution of such resources in the global picture might not be optimal. Therefore, it is necessary to determine a way in which the agents could ultimately

find optimal routes throughout the network, while achieving a scheme where network resources could be evenly distributed as well. One way to realize this is to provide mobile agents with a smart navigation mechanism to prevent future network congestion. However, this solution might become troublesome since the inherent simplicity of the agents would be lost by providing them with complex network navigation algorithms. A better solution can be achieved by designing, implementing and deploying a different colony of agents to roam the network and disseminate information regarding the distribution of network resources. The existing static agent colony residing at individual network nodes might use such information to advertise, not only the availability of resources in a specific link of the network, but also the level of occupancy of the link.

- *Resource Reservation*: Another important issue to consider is that of resource reservation in the network. While in the process of finding optimal routes, a particular intermediate node might advertise that enough resources exist to meet certain QoS constraints. However, the node might also be in the process of already reserving resources for other connections before updating the QoS-KN. Therefore, later requests to reserve resources at that node (as part of the path found), might find them unavailable. However, the solution recently mentioned would also apply here, in which both the information regarding current network availability, and the degree of utilization of a particular link could assist the mobile agents in their routing decision.
- *Connection Admission Control*: Once additional network resources have been reserved to honour requests for connections, the need a rise to advertise the availability of network resources at the entry nodes. It is important to recall that an ingress node in an AS represents a particular leaf in an mp2p tree connection. Therefore, any significant changes in the availability of resources at any upstream node in the mp2p tree need to be advertised to all the nodes that belong to the branches affected by that change, including network entry nodes.

Such information would be of prime importance for a connection admission control scheme operating at the ingress nodes in the Diffserv/MPLS network to properly manage the number of incoming requests for connections from outside the network.

- *MPLS*: Although previous chapters have already addressed a number of issues involving interfacing of the proposed routing technology and MPLS, comments can still be made in regards to the manner in which MPLS might actually establish the mp2p tree. The most obvious solution is to let the involved nodes of the intended LSP assign proper label values, while also relying on a label distribution protocol to complete the task. In this case, the list of participating switches can be passed from the Wave interpreter to the MPLS node serving as the root of the tree. Then, a downstream-distribution of labels follows as previously explained in Chapter 2, by moving to the next host on the list in a sequential fashion once a label has been assigned in the current switch. The other solution would be to make use of the NHLFE feature provided by MPLS, as also explained in Chapter 2. Then, the list of IP addresses for the LSP can be passed to the switch, so that every data-packet being forwarded may use it when travelling across the network. However, the disadvantage of this forwarding method is that the number of labels used might not be controlled as in the previous solution. Here every data-packet travels individually towards its destination, losing the shared-label feature attained by the alternative method. In other words, only an mp2p tree is formed by the individual connections, but no actual setting up of the shared LSPs takes place.

6.2 QoS-Routing: Mobile Agent based vs. Classical Schemes

Up until now, no commercially available routing protocol that performs the mp2p task has been found. Therefore, carrying out a trade-off analysis to determine the superiority of one routing

scheme over the other might lead to inaccurate results. For instance, the IETF has made available an interesting presentation in [38], where a performance evaluation analysis shows the viability of a particular routing protocol aimed at finding QoS routes. The document considers the groundwork found in [4], which also takes into account a classical routing approach. According to the particular specifications made there, it can be seen that performing QoS routing with current technology is a viable solution. However, the analysis fails to take into account a number of considerations that become crucial issues. First, the analysis considers only point-to-point routes; no multipoint scheme is ever envisioned. Second, no support for explicit routing is taken into account either. Moreover, the introduction of a QoS-framework such as Diffserv or ISA, along with MPLS as the foundation for support in a future network infrastructure, should definitely be considered. Consequently, it is important to consider the before mentioned aspects to be decisive factors when analyzing the viability of the routing solutions, while also taking into account additional factors, such as the introduction of the Internet Protocol version 6 (IPv6), which clearly affects both the size of the routing tables and the computational complexity of the proposed solutions.

On the other hand, mobile agent technology is still in its early stages. Further investigation is required to determine its feasibility for complex networking tasks [79]. This work aims at contributing to this global effort.

6.3 Final Remarks

As previously stressed, classical schemes rely on the local gathering of network state information to later disseminate it across the network. The information is then employed to compute and update routing tables, and later used in the decision process when a data packet arrives at a node

and needs to be forwarded towards its final destination. The introduction of quality of services in the transmission of data across the Internet backbone introduces new challenges for current routing schemes. Therefore, current routing protocols require a number of enhancements that will certainly place a heavier burden on network infrastructure. Increased complexity in the computation of routing tables, increased size of memory banks to accommodate such tables, and augmented network control traffic to transmit them are the most relevant disadvantages that the current routing schemes ultimately have to confront.

This work has presented an alternative approach for finding routes that comply with QoS requirements, by making use of the mobile software agent paradigm. Specifically, a number of small but effective procedures are created and put together to create an algorithm that finds multipoint-to-point routes. As argued previously, this type of connection will play an important role in the Internet backbone, as new technologies such as Multi-Protocol Label Switching and Differentiated Services Architecture are proposed for support of differentiation of classes of services, and guarantees of QoS. The mobile agents approach can be seen as a viable and promising solution to solving routing problems. The inherent philosophy of this paradigm offers a new strategy for solving inconveniences found not only in routing, but also in other networking tasks. In this case, the Wave paradigm is used as a tool for implementing routing algorithms based on mobile agent technology. The Wave architecture provides a way of creating a virtual system on top of actual communications network, which can be used to represent specific knowledge on the current state of the network in a fully distributed manner. This implies that instead of moving large amounts of data across the network for performing specific computations, small software programs can be transferred from one place to another for processing distributed information in a mobile fashion. In other words, this solution contemplates moving the tasks computation to the location of data, instead of the other way around, as current paradigms do. This approach introduces the possibility of designing mobile code (agents) so as

to enable a collective interaction among them. Colonies of mobile agents can be deployed throughout the network for performing specific tasks in a fully parallel, asynchronous and distributed manner. Such is the case for the algorithm designed to perform multipoint-to-point routing.

In designing an experimental solution to deal with routing issues, several key aspects are also observed. These aspects need to be carefully reviewed, should a solution based on mobile agent technology actually be contemplated for implementation to perform either routing or any other task in a network, as outlined next:

- Mobile agents are programs designed to carry out specific tasks at a remote network location. This implies that any system intending to use this technology must provide a way of hosting the software environment in which such agents work; otherwise the agents are unable to actually perform any kind of computation. Currently, the only infrastructure readily available for supporting implementations using mobile agent technology is found at end systems. There is no deployed infrastructure for supporting mobile agent technology at intermediate locations in any network. This means that any scheme using mobile agent technology would have to implement a system for hosting mobile agents within a specific operating system at the intermediate nodes. This assertion would call for the enhancement of the actual networking hardware to accommodate the necessary software infrastructure for implementing any mobile agent platform. This in turn introduces a monetary cost, which calls for a trade-off analysis to determine whether this enhancement is viable or not.
- Mobile agent technology intended for use on networking tasks should enable the implementation of agents, be compact but powerful in nature. A mobile agent platform using agents that require long pieces of code to perform simple tasks can become rather inefficient. This would only lead to the creation of significant network control traffic and increased computation time.

- Controlling the number of agents operating in a network is also important. An excessive number of agents might lead to congestion at the agent-hosting systems, which is of course undesirable. This would create delays in the processing of a specific network task relying on this technology.
- While there seems to be consistent behaviour in the traffic generated by the routing scheme presented here, these results might vary if the scheme is implemented in a real network. The propagation delay experienced by the agents while travelling through the network links would contribute to this variation. In addition, enabling a network to host a mobile agent system would likely introduce traffic by agents performing different networking tasks besides routing. In that case, the traffic characteristics of the flows created by the agents might also vary as a consequence of the environment being shared by multiple colonies of agents.

Finally, it can be said that the application of mobile agents for telecommunication applications is still in an early stage of research. Further investigation is required to validate the contribution that such technology brings to networking infrastructure. Additional studies are necessary to investigate the applicability of mobile agent schemes for other networking tasks, such as traffic balancing, resource reservation and connection admission control to understand the benefits of using this novel technology to solve complex networking problems.

Bibliography

[1] Acharya A., et al. "Mobility Support for IP over Wireless ATM." IEEE Communications Magazine. April 1998.

[2] Andersson, L. et. al. "LDP specification." Internet Draft, draft-ietf-mpls-ldp-11.txt, Internet Engineering Task Force, August 2000. Work in progress.

[3] Andrikopoulos, I. and Pavlou, G. "Supporting Differentiated Services in MPLS Networks." In Proceeding of 7th International Workshop on Quality of Service (IWQOS'99), London, May 1999.

[4] Apostolopoulos, G. et. al. "QoS Routing Mechanisms and OSPF Extensions". RFC2676, August 1999.

[5] Armitage G. "MPLS: The Magic Behind the Myths," IEEE Communications Magazine, January 2000.

[6] Ash J. et. al. "Applicability Statement for CR-LDP." Internet Draft, draft-ietf-mpls-crldp-applic-01.txt, work in progress, IETF, January 2001.

[7] Aukia P. et al, "Rates: A Server for MPLS Traffic Engineering," IEEE Network, March 2000.

[8] Aurrecoechea, C, et. at. "A survey of QoS architectures." In 4th IFIP International Conference on Quality of Service, Paris, France, March 1996.

[9] Awduche, D et al. "Requirements for Traffic Engineering Over MPLS", RFC2702, September 1999.

[10] Awduche, D. "MPLS and traffic engineering in IP networks," IEEE Communication Magazine, pp. 42--47, December 1999.

[11] Beckers, R. et. al. "Trails and U-turns in the Selection of the Shortest Path by the Ant Lasius Niger". Journal of Theoretical Biology, 1992.

[12] Bertsekas, D. and Gallager, R. "Data Networks". Prentice Hall, 2nd Edition, 1992.

[13] Blake, S. et. al. "An Architecture for Differentiated Services." IETF-Network Transport WG, RFC2475, December 1998

[14] Braden, R. et. al. "Integrated Services in the Internet Architecture: an Overview". IETF RFC1633, June 1994.

[15] Buckley, F. and Harary, F. "Distance in Graphs". Addison-Wesley, 1990.

[16] Chaudry, M.L. and Templeton, J.G.C. "A First Course in Bulk Queues". John Willey and Sons, 1983.

[17] Chavez A., Moukas A., Maes P. "Challenger: A Multiagent System for Distributed Resource Allocation." Proceedings of the First International Conference on Autonomous Agents '97, Marina Del Ray, California, 1997.

[18] Cieslik, D. "Steiner Minimal Trees". Kluwer Academic Publishers, 1998.

[19] Colburn, J. and Xue, G. "Grade of Service Steiner Trees in Series-Parallel Networks". Appeared in "Advances in Steiner Trees", Du, D.-Z et. al. (editors), Kluwer Academic Publishers, Netherlands, 2000.

[20] Crawley, E. et. al. "A Framework for QoS-based Routing in the Internet." IETF Network WG, RFC2386, August 1998.

[21] Davie, B. et. al. "MPLS using LDP and ATM VC Switching." IETF MPLS WG, RFC3035. January 2001.

[22] De Meer, H. "Management of QoS with Software Agents." Cybernetics and Systems: An International Journal, 27(5), 1998.

[23] De Meer, H. et. al. "Programmable Agents for Flexible QoS Management in IP Networks." IEEE-JSAC, Vol. 18, No. 2, February 2000.

[24] De Meer, H. et. al. "Tunnel Agents for Enhanced Internet QoS." IEEE Concurrency, 6(2), April-June 1998.

[25] Di Caro, G and Dorigo, M. "Mobile agents for adaptive routing". In Proceedings of the 31st International Conference on System Sciences (HICSS-31), volume 7, pages 74--83. IEEE Computer Society Press, 1998.

[26] Dijkstra, E. "A Note on Two Problems in Connection with Graphs". Numerical Mathematics, October 1959.

[27] Eric W. M. Wong, T.S. Yum and A. K. M. Chan, "A Taxonomy of Rerouting in Circuit-Switched Networks," IEEE Communications Magazine. November, 1999, pages: 116-122.

[28] Fischer S. et. al. "Application Design for Cooperative QoS Management", Proc. 5th International Workshop on Quality of Service (IWQOS'97), Columbia University, New York, USA

[29] Fischer, S and H de Meer. "Decision Support in Cooperative QoS Management." In Proc. of the 6th Int. Workshop on Quality of Service (IWQoS'98), San Francisco, USA, May 1998.

[30] Fischer, S et. al. "Cooperative QoS Management for Multimedia Applications." 4th IEEE Int. Conf. on Multimedia Computing and Systems, June 1997.

[31] Ford, L. and Fulkerson, D. "Flows In Networks". Princeton, NJ. Princeton University press, 1962.

[32] Ghanea-Hercock R., Collis J. & Ndumu D. "Co-operating Mobile Agents for distributed Parallel Processing." Autonomous Agents Conference, Seattle, U.S. May 1999. 398-399

[33] Grassé, P. "La Théorie de la Stigmergie: Essay d'interptétation du Comportement des Termites Constructeurs". Insectes Sociaux, 1959.

[34] Guedes, L.A. et. al. "An Agent-based Approach for Supporting Quality of Service in Distributed Multimedia Systems". Computer Communications Journal, 21(14), September 1998. Elsevier Science

[35] Hafid, A and Fischer, S. "A multi-agent architecture for cooperative quality of service Management." Proceedings of IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS'97), Montreal, Canada

[36] Hiroyuki Saito et al. "Traffic Engineering Using Multiple Multipoint-to-Point LSPs". Presented at the IEEE Infocom 2000. March 26 - 30, 2000 Tel-Aviv, Israel.

[37] Hummel, H. and Loke, S. "Explicit Tree Routing". Internet Draft, draft-hummel-mpls-explicit-tree-01.txt, work in progress, IETF, December 1999.

[38] IETF http://www.ietf.org/proceedings/98dec/slides/ospf-moy-orla-98dec/sld001.htm, as of October, 2001.

[39] Jamoussi, Bilel et al. "Constraint-Based LSP Setup Using LDP". Draft-ietf-mpls-cr-ldp-04. internet draft, work in progress, January 2001.

[40] Karnik N. M. and Tripathi A. R. "Design Issues in Mobile-Agent Programming Systems", IEEE Concurrency, Vol. 6, 1998.

[41] Kawaguchi, N. et. al. "MAGNET: Ad-Hoc Network System based on Mobile Agents," Computer Communications, Vol.23, No. 8, pp.761-768 (2000).

[42] Kendall, E. et. al. "Patterns of Intelligent and Mobile Agents." Proceedings of the Second International Conference on Intelligent Agents, 1998, ACM press.

[43] Lange, Danny and Oshima, Mitsuru. "Seven Good Reasons for Mobile Agents". Communications of the ACM, March 1999.

[44] Le Faucheur, F. et. al. "MPLS Support of Differentiated Services." Internet draft, draft-ietf-mpls-diff-ext-07.txt, IETF. August 2000.

[45] Le Pocher, H., Leung, V.C.M., and D. W. Gillies,. "Real-time Multimedia Scheduling Policies for End-to-End Delay Jitter and Loss Guarantees Across ATM Satellite Systems," IEEE Transactions Selected Areas in Communications, vol. 17, no. 2, pp. 314-325, Feb. 1999

[46] Li, R. Yates and D. Raychaudhuri. "Performance Analysis of Path Rerouting Algorithms for Handoff Control in Mobile ATM Networks," IEEE Journal on selected areas in communications, Vol. 18, No. 3, March 2000.

[47] Lin, C. and Liu, J. "QoS routing in ad hoc wireless networks," IEEE J. Selected Areas in Communications, 17, 8, Aug. 1999, 1426-1438.

[48] Magedanz, T. and Karmouch, A. "Mobile Software Agents for Telecommunication Applications". Computer Communications 23 (2000), 705-707, Elsevier Science.

[49] Mamadou T. and Nakajima, T. "An Architecture for a QoS-based Mobile Agent System." Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications (RTCSA'98), Hiroshima, Japan, October 1998

[50] Manvi, SS and Venkataram, P. "QoS Management by Mobile Agents in Multimedia Communication." IEEE Computer Society, January 2000.

[51] McDysan, D. and Spohn, D. "Hands on ATM". McGraw Hill Series on Computer Communications, 1998.

[52] McQuillan, J.M. and Walden, D.C. "The ARPANET Design Decisions". Networks, 1.

[53] Mikler, A. et. al. "Analysis of Utility-Theoretic Heuristics for Intelligent Adaptive Network Routing," in Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI96) . 1996, vol. 1, pp. 96--101, AAAI Press.

[54] Minar, N. et al. "Cooperating Mobile Agents for Dynamic Network Routing". Software Agents for Future Communications Systems, Springer-Verlag, 1999.

[55] Minar, N. et al. "Cooperating Mobile Agents For Mapping Networks." MIT Media Group. In Proceedings of First Hungarian National Conference on Agent Based Computing, May 1998.

[56] Minar, N. et. al. "Mobile Software Agents for Dynamic Routing, Mobile Computing and Communications" Review, Vol. 3, No. 2.

[57] Moy, J. "Multicast extensions to OSPF." IETF Network Routing WG, RFC1584, March 1994.

[58] Moy, J. "OSPF Version 2". IETF RFC2328, Network WG, April 1998.

[59] Newman, P. et. al. "Ipsilon Flow Management Protocol Specification for IPv4", IETF RFC1953, Network WG, May 1996.

[60] Newman, P. et. al. "Transmission of Flow Labeled IPv4 on ATM Data Links", IETF RFC1954, Network WG, May 1996.

[61] Ng, K.W. and Leung, V. "Host mobility support for mobile computing over wide-area wireless data networks", in Proc. IEEE VTC'00(Spring), Tokyo, Japan, May 2000.

[62] Oida, K and Sekido, M. "ARS: an efficient agend-based routing system for QoS guarantees." Computer Communications 23 (2000), 1437-1447, Elsevier Science.

[63] Orda, A. "Routing with end to end QoS guarantees in broadband networks". In Proc. IEEE Infocom, vol. 1, pp. 27--34, 1998.

[64] Palisade Corporation, "Bestfit", http://www.palisade.com/html/bestfit.html as of October 2001.

[65] Pronavalai, C. et. al. "QoS Based Routing Algorithm in Integrated Services Packet Networks". In Proceedings of the IEEE International Conference on Network Protocols (ICNP '97), 1997.

[66] Prycker, M. "Asyncronous Transfer Mode". Prentice Hall, 3rd Edition, 1995

[67] Puliafito, A. et. al. "An Agent-based Framework for QoS Management." In 4th Int. Conference on Analytical and Numerical Modeling Tech. – QoS modeling, Singapore, September 1997.

[68] Raychaudihuri, D. "Current topics in wireless & mobile ATM networks: QoS control, IP support and legacy service integration." In IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'98), pages Vol (1)38-44, 1998.

[69] Rayes, A. and Mohsen, G. "Designing ATM switching networks". McGraw-Hill, 1999.

[70] Rekhter, Y. et. al. "Cisco Systems' Tag Switching Architecture Overview". RFC2105, IETF Network WG, February 1997.

[71] Rosen, E. et. al. "Multi-protocol Label Switching Architecture". IETF MPLS WG, RFC3031, January 2001.

[72] Rosen, K. (editor), "Handbook of Discrete and Combinatorial Mathematics". CRC Press, 2000.

[73] Sahai, A, and Morin, C. "Mobile Agents for Enabling Mobile User Aware Applications." Proceedings of the Second ACM International Conference on Autonomous Agents (Agents' 98), Minneapolis/St.Paul, USA, May 1998

[74] Sahasrabuddhe L. and Mukherjee B. "Multicast Routing Algorithms and Protocols: A Tutorial," IEEE Network Magazine, January 2000.

[75] Sapaty, P. "Mobile Processing in Distributed and Open Environments". John Willey & Sons, 2000.

[76] Sapaty, P. and Borst, P. "Wave: Mobile Intelligence in Open Networks". Eta-COM '96, Portland, Oregon.

[77] Schelen, O and Pink, S. "Resource Sharing In Advance Reservation Agents," Journal of High-Speed Networks: Special issue on Multimedia Networking, vol 7, no. 3-4, 1998.

[78] Schelén, O. and Pink, S. "Resource Reservation Agents in the Internet." Proceedings of 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98), July 1998.

[79] Schoder, D. "The Real Challenges of Mobile Agents." Communications of the ACM, June 2000, Vol. 43, No. 6, S. 111-112

[80] Stallings, W. "Data and Computer Communications." Prentice Hall, 6th Edition, January 2000.

[81] Stallings, W. "High-Speed Networks: TCP/IP and ATM Design Principles". Prentice Hall, 1998.

[82] Stavrakakis, I. and S. Iatrou, "A Dynamic Regulation and Scheduling Scheme for Real Time Traffic Management", IEEE/ACM Transactions on Networking Vol. 8, No. 1, February 2000.

[83] Swallow G. "MPLS Advantages for Traffic Engineering," IEEE Communications Magazine, December 1999, pages: 54-57.

[84] Tennenhouse, D. "A Survey of Active Network Research." IEEE Communications Magazine, pages 80--86, January 1997.

[85] Uhrmacher, A. et. al. "Modeling and Simulation of Mobile Agents." Future Generation Computer Systems, page (to appear) , 2000.

[86] Varshney U. "Connection Routing Schemes for Wireless ATM". HICSS-32 Proceedings of the 32nd Hawaii International Conference on Systems Sciences, January, 1999.

[87] Vuong S. and Ivanov, I. "Mobile Intelligent Agent Systems: Wave Vs. Java." IEEE Computer Society, March 1996

[88] Wang, B and Hou, J. "Multicast Routing and its QoS Extension: Problems, Algorithms, and Protocols." IEEE Network, pages 22-36, January 2000.

[89] Wen-Shyen Chen, S. "Mobility and Management Support for Mobile Agents", Proc. of the 2nd International Conference on Autonomous Agents, May 1998.

[90] White, T., Pagurek, B. and Oppacher, F. (1998). "Connection management using adaptive mobile agents". Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '98) (Arabnia, H. R., ed), pp. 802-809.

[91] Widjaja, I. and Elwalid, A. "Performance Issues in VC-Merge Capable Switches for Multiprotocol Label Switching." IEEE-JSAC, June 1999.

[92] Wong E., Chan A. and Yum T. "Analysis of Rerouting in Circuit-Switched Networks", IEEE/ACM Transactions of Networking, Vol. 8, No. 3, June 2000

[93] Wong P., Leung V. and Nasiopoulos P. "An MPEG2-to-ATM Converter to Optimize Performance of VBR Video Broadcast over ATM Networks", IEEE Trans. on Consumer Electronics, Vol. 44, No. 3, 1998

[94] Wong, S. and Mikler, A., "Intelligent Mobile Agents in Large Distributed Autonomous Cooperative Systems." Journal of Systems and Software 47 (2000), 75-87, Elsevier Science.

[95] Wong, V. and Leung, V. "A Path Optimization Signaling Protocol for Inter-Switch Handoff in Wireless ATM Networks," Computer Networks, vol. 31, no. 9-10, pp. 975-984, May 1999. Elsevier Science

[96] Xiao, X and Lionel M. Ni. "Internet QoS: A Big Picture". IEEE Network, March/April 1999

[97] Xiao, X. et. al. "Traffic engineering with MPLS in the Internet," IEEE Network Magazine, March 2000.

[98] Zhang, P. and Kantola, R. "Building MPLS VPNs with QoS Routing Capability". Fifth International Symposium on Interworking (Interworking'2000). October 3-6, 2000, Bergen, Norway.

Appendix A. Wave Language Basics

The purpose of this appendix is to present an introduction to the basic features provided by the Wave language. Such features can be divided into three main categories: variables, acts and rules. Each category will be briefly explained. For a thorough study on the Wave paradigm and its rich semantics, the reader is encouraged to consult the main reference [75].

Task variables:

There are different types of variables that waves can use to accomplish predefined tasks. Such variables are physically distributed throughout a KN, and they can be explicitly created by waves or provided by the Wave environment as explained next.

- *Task Variables:* There are two types of task variables in Wave: Nodal and Frontal variables. Nodal variables start with the letter N (e.g. Ncounter). They are created when non-empty values are assigned to them, and have significance only in the node where they were created. Access to these types of variables is granted to all waves visiting that node in the KN. Frontal variables start with an F (e.g. Fchange), and they can only be accessed by the very wave that creates it. Frontal variables do not belong to specific nodes, but rather, they travel along with the wave that created such variable. When a wave splits in different branches, a copy of the frontal variables so far created is inherited by each of the newly waves created.
- *Environmental Variables*: These types of variables do not belong to the waves traversing the KN, but rather to the KN in which they are accessed from. They give information in the characteristics of the KN environment, and they can be accessed from every node visited by a wave in the KN. A *Content* (C) variable encloses the identification name of

the current node. The *Address* (A) variable holds the full network address of the node it belongs to in octal notation. The *Predecessor* (P) variable returns the network address of the node previously visited by a wave, if applicable. The *Link* (L) variable returns the identification name of a link traversed by a wave between two nodes. The *Terminal* (T) variable is a way of accessing a graphical text interface, and can be used each time a wave needs to display a message.

Acts:

Wave defines a series of *acts* that can take up to two operators on the left and right side of the act symbols, and they are defined as follows:

- *Filters:* The type of filters defined in the WAVE language are similar to other operators found in other programming tools. They are less than (<), less than or equal to (<=), exactly equal to (==), greater than (>), greater than or equal to (>=), and, different from (/=). Two additional filters are added to the list, and they are defined as: belongs to (~), and does not belong to (/~).
- *Assignment*: This is a standard assignment operator for copying a value into a variable (=).
- *State generator*: This operator can be used to explicitly generate states during the execution of a wave agent, and is defined as '!'.
- *Fusion:* These acts are used as standard arithmetic integer operations between variables: sum (+), difference (-), multiplication (*), and division (/).
- *Special*: The '&' act appends the value in the right side of the act to a variable in the left side. The colon (:) act returns an index value in a vector, and the double-colon (::) act returns the content of a vector's index.

- *Vector-string conversions*: The | act can be used to split a string into a vector by a specified delimiter, while the % act is used when merging a vector to get rid of the current delimiter.
- *External calls*: One handy act that can be used to interface the Wave interpreter with an external program is the ? act, which enables to access commands or executable programs using the host operating system
- *Direct*: The @ is used for indicating a given wave to perform a direct jump to a specified node regardless of the current location. The @ act can be also used for storing data by vector indexing.

Rules:

The Wave language defines a number of constructs that define constraints under which the waves that embrace them can operate. Such rules are necessary to control the navigation mechanisms that the waves follow, which may be independent, asynchronous and parallel.

- *Branching*: There are five branching rules provided by the Wave language. The first one is used to indicate that a series of individual wave moves (or branches) will be performed in strict sequence and is represented as SQ. The second branching rule is used to also activate branches in sequence until one of the enclosed branches results in a TRUE state and is known as the or-sequential rule (OS). The and-sequential or AS rule operates in a similar manner as the SQ rule, except that the resulting state will be TRUE only if all the states of the individual branches result in a TRUE state after their execution. The fourth rule is known as the or-parallel rule (OP) wherein all the branches are activated in parallel, resulting in the selection of that branch that replies first with a TRUE state. Finally, the AP or and-parallel rule activates all the branches enclosed in parallel, and results in a final TRUE state only if all the branches executed also produce a TRUE state.

- *Repetition:* Wave provides the means of executing code in a loop fashion by means of the repeat (RP) rule. As in other conventional languages, the code enclosed by the RP rule will execute as long as certain condition is met, otherwise the RP cycle is broken.
- *Wait:* The wait (WT) rule can be used to suspend the remainder of a wave execution until all the embraced waves terminate, which helps in synchronizing of processes that were asynchronously made active.
- *Protecting:* The indivisible (ID) rule is used in the Wave language as a way of explicitly defining a set of individually executed wave moves to become an indivisible operation. This is a helpful feature when the need arises to restrict access to a portion of the wave program being executed.
- *Create:* The create (CR) rule empowers the embraced wave to extend or create an additional link, which will be now part of the KN being traversed.

Appendix B. Multipoint-to-Point Routing Programs

B.1 Wave program for finding mp2p static trees

```
Fcompete=`
Fnum=2.
RP(OS((ID(1<Nvisits).Fweight+1),).C/=Fdestination.
   Ftemp=Fpath.Ftemp:Fnum.Flinktype#Ftemp.Fnum+1).
OS(
   (
    ID(Fsource/~Nedges.Nedges&Fsource.Nweight&Fweight).
       Fpath%`'.CR(Fsource#Fpath).#P
   ),
   ID(
      Fsourceindex=Nedges.Fsourceindex::Fsource.Fweightindex=Nweight.
      Fweightindex:Fsourceindex.Fweightindex<Fweight.
      Fsourceindex2=Fsourceindex.Fsourceindex2&@.Fsourceindex2&Fweight.
      Nweight:Fsourceindex2.Fsource#.Fpath%`'.C=Fpath.#P
   )
)'.
Fmarkpath=`
Fcount=-2.
RP(
   OS(
      ID(Nvisits==NONE.Nvisits=1.Nvisitedby&Fsource),
      ID(Fsource/~Nvisitedby.Nvisitedby&Fsource.Nvisits+1),
   ).
   C/=Fsource.Ftemp=Fpath.Ftemp:Fcount.Flinktype#Ftemp.Fcount-1
).ID(Nroutes&Fpath)'.
Findspt=`
Fcollect=A.Fsource=C.
SQ(
   RP(
      Flinktype#.Flength+L.
      OS(
         ID(Fsource/~Nsource.Nsource&Fsource.Ndistance&Flength),
         ID(
            Fsourceindex=Nsource.Fsourceindex::Fsource.Fdistindex=Ndistance.
            Fdistindex:Fsourceindex.Fdistindex==NONE,OS((Fdistindex<=Flength.!3),).
            Fsourceindex2=Fsourceindex.Fsourceindex2&@.
            Fsourceindex2&Flength.Ndistance:Fsourceindex2.
         )
      )
   ),
   RP(
      Fpath&C.
      (C==Fdestination.Fmarkpath.Fcompete.!3),
      (
       Flinktype#.Flength+L.
       ID(Fsourceindex=Nsource.Fsourceindex::Fsource.
          Fdistindex=Ndistance).Fdistindex:Fsourceindex.Flength<=Fdistindex
      )
   ),
   !3.
)'.
Fnodes=a;b;c;d;f;g;h.Fdestination=e.Flinktype=1.
SQ(@#Fdestination,WT(@#Fnodes.Findspt.!3),).
Fnodes#.Fpath=C.Fpath|`'.T=Fpath.C=NONE.
```

B.2 Wave program for finding mp2p dynamic trees

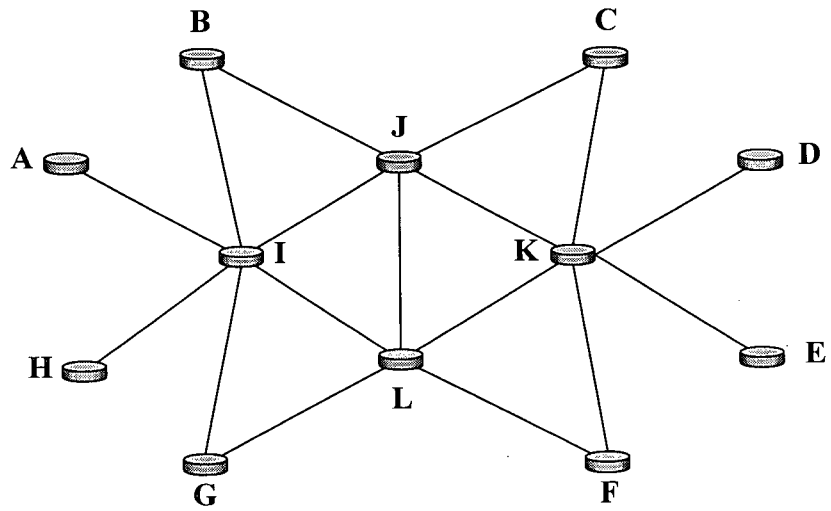
```
Fcompete=`
Fnum=2.Fase=3.
RP(

OS(goto#destinations,).Fdestination#.OS((ID(1<Nvisits).Fweight+1),).
   #P.OS(goto#,).C/=Fdestination.Ftemp=Fpath.Ftemp:Fnum.
   Flinktype#Ftemp.Fnum+1
).
OS(res#sources,CR(res#sources)).
OS(
   (
    ID(Fsource/~Nedges.Nedges&Fsource.Nweight&Fweight).
       Fpath%`'.OS(Fsource#,CR(Fsource#Fpath))
   ),
   ID(
      Fsourceindex=Nedges.Fsourceindex::Fsource.
      Fweightindex=Nweight.Fweightindex:Fsourceindex.
      Fweightindex<Fweight.Fsourceindex2=Fsourceindex.
      Fsourceindex2&@.Fsourceindex2&Fweight.
      Nweight:Fsourceindex2.Fsource#.Fpath%`'.C=Fpath
   )
)'.
Fmarkpath=`
Fcount=-2.
RP(
   OS(goto#destinations,).Fdestination#.
   OS(
      ID(Nvisits==NONE.Nvisits=1.Nvisitedby&Fsource),
      ID(Fsource/~Nvisitedby.Nvisitedby&Fsource.Nvisits+1),
   ).
   #P.OS(goto#,).C/=Fsource.Ftemp=Fpath.Ftemp:Fcount.
   Flinktype#Ftemp.Fcount-1.
).ID(Nroutes&Fpath)'.
Findspt=`
Fcollect=A.Fsource=C.
SQ(
   RP(
      Flinktype#.Flength+L.ID(OS((C==Fwhich.Narr+1),)).
      OS(C/~Fedges,OS(goto#destinations,CR(goto#destinations))).
      ID(
         OS(
            (
             Fdestination/~Ndestination.
             Ndestination&Fdestination.
             CR(Fdestination#satellite)
            ),
            Fdestination#satellite
         )
      ).
      OS(
         ID(Fsource/~Nsource.Nsource&Fsource.Ndistance&Flength),
         ID(
            Fsourceindex=Nsource.Fsourceindex::Fsource.
            Fdistindex=Ndistance.Fdistindex:Fsourceindex.
            Fdistindex==NONE,Flength<Fdistindex.
            Fsourceindex2=Fsourceindex.Fsourceindex2&@.
            Fsourceindex2&Flength.Ndistance:Fsourceindex2.
         ).
         #P.OS(goto#,)
      )
   ),
```

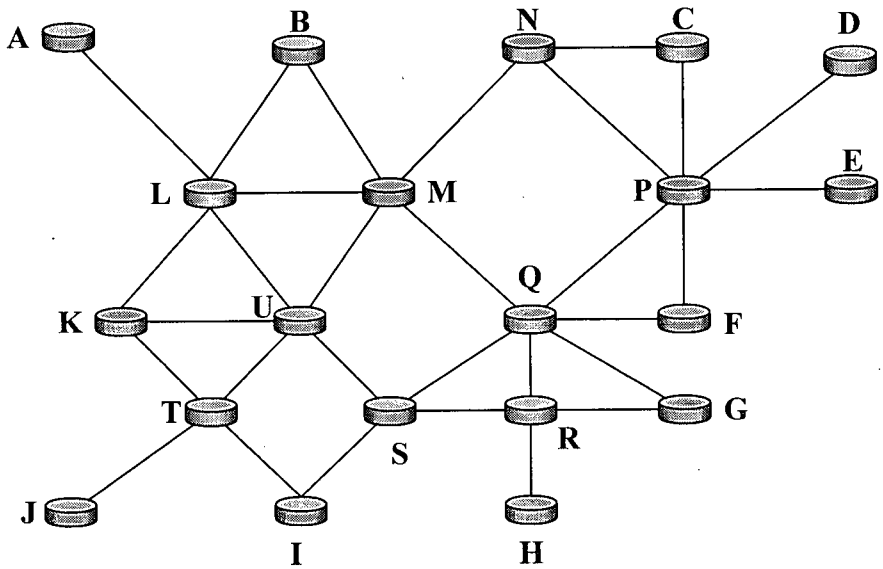
```
RP(
      Fpath&C.
      (
       C==Fdestination.goto#destinations.Fpath%`'.
       CR(Fsource#Fpath).Fpass=1.!3
      ),
      (
       Flinktype#.Flength+L.
       OS(goto#destinations,).Fdestination#.
       ID(Fsourceindex=Nsource.Fsourceindex::Fsource.
          Fdistindex=Ndistance).
       Fdistindex:Fsourceindex.Flength==Fdistindex.
       #P.OS(goto#,)
      )
   )
)'.
Fmp2p=`
SQ(
   WT(@#Fnodes.Findspt.OS((Fpass/=1.!3),)),
   WT(
      goto#destinations.Fnodes#.
       OS((C==satellite.!3),).Fpath=C.Fsource=L.#P.
      (#P.C=NONE.!3),(goto#.Fpath|`'.WT(Fmarkpath).WT(Fcompete).!3)
   ),
   (res#sources.Fnodes#.Nout=C.Nout|`'.T=Nout.C=NONE.!3),
)'.
Fsimulate=`
Flinktype=1.@#Fedges.Fin=Fedges.Fin::C.Nstop=0.
RP(
   Nstop/=1.ID(Fin?ran).Fin|`'.Frandom=Fin.
   Frandom:1.Frandom*6.Frandom?sleep.Frandom=Fin.
   Frandom:2.Frandom+1.Fcountx=Frandom.Fcountx+3.Fnext=3.
   RP(
      Fnext<Fcountx.Frandom=Fin.Frandom:Fnext.
      OS((Frandom==NONE.!1),).Frandom+1.
      OS((Frandom/~Fsource.Fsource&Frandom),).
      Fnext+1
   ).
   Frandom=Fedges.Frandom:Fsource.Fnodes=Frandom.Fdestination=C.
   Fin%`'.OS((Fnodes/=NONE.Fmp2p),).Fsource=NONE
)'.
Fedges=a;b;c;d;e;f;g;h.Fsimulate
```

Appendix C. Network Topologies Used

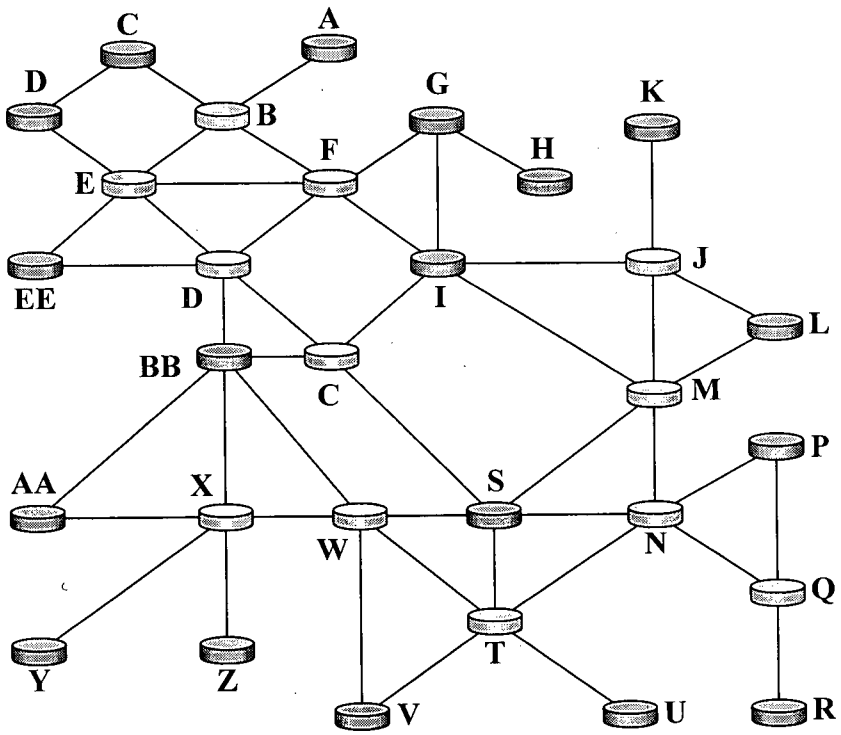
C.1 12-Node Topology



C.2 20-Node Topology



C.3 30-Node Topology



Appendix D. Time Complexity of the Routing Algorithm

A simple computational time complexity analysis of the routing algorithm presented in chapter 5 is presented in this appendix. The goal is to predict the time latency that can be expected considering the circumstances under which this algorithm is run. The following analysis only considers the procedures directly related to routing ('Findspt', 'Fmarkpath', 'Fcompete', and 'Fmp2p'). The remaining procedure is not taken into account, since it is only used for creating events for the simulation. Each section of the individual procedures is assigned a letter for easier reference. Assuming that the reader possesses some basic knowledge on computational complexity, the analysis now follows:

- ***The 'Fmarkpath' procedure:***

```
          Fmarkpath=`
D       { Fcount=-2.
          RP(
            OS(goto#destinations,).Fdestination#.
            OS(
A         {     ID(Nvisits==NONE.Nvisits=1.Nvisitedby&Fsource),
C         {     ID(Fsource/~Nvisitedby.Nvisitedby&Fsource.Nvisits+1),
            ).
            #P.OS(goto#,).
B         { C/=Fsource.Ftemp=Fpath.Ftemp:Fcount.Flinktype#Ftemp.Fcount-1
          ).ID(Nroutes&Fpath)'.
```

A) The first OS rule contains two simple jump operations, while the second is comprised by two ID rules, each containing simple variable assigning, comparisons and append operations. Therefore A=O(1).

B) No special rules, only comparisons, indexing, assigning and a jump. B=O(1).

C) The RP rule embraces A and B, which are O(1), but the execution of the cycle will continue until the destination is reached. The worst case occurs when a wave's origin and

the destination node are separated by a distance equal to the diameter of the QoS-KN. Therefore, the constraint is set by a <diameter> factor. Therefore C=O(diameter).

D) Single assign operation. D=O(1).

Therefore, for 'Fmarkpath', the time complexity is O(diameter).

- The 'Findspt' procedure:

```
Findspt=`
Q  {  Fcollect=A.Fsource=C.
      SQ(
        RP(
          Flinktype#.Flength+L.
          OS(C/~Fedges,OS(goto#destinations,CR(goto#destinations))).
          ID(
            OS(
              (
               Fdestination/~Ndestination.
               Ndestination&Fdestination.
               CR(Fdestination#satellite)
              ),
              Fdestination#satellite
            )
          ).
        ),
        RP(
          OS(
            ID(Fsource/~Nsource.Nsource&Fsource.Ndistance&Flength),
            ID(
              Fsourceindex=Nsource.Fsourceindex::Fsource.
              Fdistindex=Ndistance.Fdistindex:Fsourceindex.
              Fdistindex==NONE,Flength<Fdistindex.
              Fsourceindex2=Fsourceindex.Fsourceindex2&@.
              Fsourceindex2&Flength.Ndistance:Fsourceindex2.
            ).
            #P.OS(goto#,)
          )
        ),
        RP(
          Fpath&C.
          (
           C==Fdestination.goto#destinations.Fpath%`'.
           CR(Fsource#Fpath).Fpass=1.!3
          ),
          (
           Flinktype#.Flength+L.OS(goto#hell,).Fdestination#.
           ID(Fsourceindex=Nsource.Fsourceindex::Fsource.
           Fdistindex=Ndistance).
           Fdistindex:Fsourceindex.Flength==Fdistindex.
           #P.OS(goto#,)
          )
        )
      )'.
```

E) This block is composed of simple assigning operations and two hops. E=O(1).

F) The first two lines also contain simple operations and hops, and the next statement contains the previous OS block. F=O(1).

G) Two ID rules, containing all simple assigning operations. G=O(1).

H) Tow single-hop operations. H=O(1).

I) Comprised by blocks G and H. I = Max[G,H] = O(1).

J) The RP rule will keep running statements in blocks F and I until the destination is reached. Again, the worst case occurs when the distance traversed is as long as the QoS-KN diameter. Therefore, J=O(diameter).

K) A single assigning operation. K=O(1).

L) Simple hops and assigning operations. L=O(1).

M) Simple hops and assigning operations. M=O(1).

N) Again an RP rule executing statements as many times as <diameter> hops are. N=O(diameter).

P) Comprised by the SQ rule containing two sequential RP blocks. P = Max[J,N] = Max[O(diameter),O(diameter)]. P=O(diameter)

Q) Two simple variable assigning. Q=O(1).

The time complexity is then bounded by Max = [P,Q] = [O(1),O(diameter)]. *Therefore, 'Findspt'=O(diameter).*

- The 'Fcompete' procedure:

```
          Fmarkpath=`
R       -{Fcount=-2.
         ⎧RP(
         ⎪   OS(goto#destinations,).Fdestination#.
         ⎪   OS(
S        ⎨      ID(Nvisits==NONE.Nvisits=1.Nvisitedby&Fsource),
         ⎪      ID(Fsource/~Nvisitedby.Nvisitedby&Fsource.Nvisits+1),
         ⎪   ).
         ⎪   #P.OS(goto#,).
         ⎪   C/=Fsource.Ftemp=Fpath.Ftemp:Fcount.Flinktype#Ftemp.Fcount-1
         ⎩).ID(Nroutes&Fpath)'.
```

R) A simple operation. R=O(1).

S) A RP rule working in the same way as explained before. S=O(diameter)

For 'Fcompete', the time complexity is = Max[R,S] = Max[O(1), O(diameter)]. Then, *'Fcompete'=O(diameter).*

- The 'Fmp2p' procedure:

```
   Fmp2p=`
   SQ(
T   -[WT(@#Fnodes.Findspt.OS((Fpass/=1.!3),)),
     ⎧WT(
U    ⎨   goto#destinations.Fnodes#.OS((C==satellite.!3),).Fpath=C.Fsource=L.#P.
     ⎪   (#P.C=NONE.!3),(goto#.Fpath|`'.WT(Fmarkpath).WT(Fcompete).!3)
     ⎩),\
V   -[res#sources.Fnodes#.Nout=C.Nout|`'.T=Nout.C=NONE.!3),
   )'.
```

T) Comprised of a WT rule, embracing a simple hop and the 'Findspt' procedure. Then, T=Max[O(1),O(diameter)] = O(diameter).

U) Another WT rule, embracing hops, simple assigning and the 'Fmarkpath' and 'Fcompete' procedures. U=O(diameter).

V) Simple hops, and variables assigning. V=O(1).

Therefore, for 'Fmp2p', the time complexity is O(diameter).

The whole algorithm is comprised by the sequential execution of its individual procedures, each of them having the same complexity. Then, the complexity of the routing algorithm is determined by O(diameter).

However, when considering a k-ary tree configuration, the diameter D of a network has been shown to be logarithmic [15], and is bounded by:

$$D = Log_K N$$

Consider, for example, a binary tree with 16 edge nodes (leaves). Then D = Log_2 16, or 4. Therefore, the diameter of a network grows in a logarithmic fashion, when compared to the number of nodes in the growing network.

When honouring a routing request, if all of the edge nodes were to participate in the mp2p connection, and taking into account Q trees each belonging to an individual QoS class, the overall time complexity of the mp2p routing algorithm with mobile agents is:

$$O(N*D*Q)$$

Or

$$O(N*Log_K N*Q)$$

An upper bound the time complexity of this algorithm can be established by defining the worst case for *K*, which would occur if K equals to 2, representing a binary tree. In such regard, the highest numerical value in the expression Log_K N is obtained when K=2. Therefore the upper bound can be defined as:

$$O(N*log_2 N*Q)$$

Appendix E. Abbreviations and Acronyms

AAL	ATM Adaptation Layer
AS	Autonomous System
ATM	Asynchronous Transfer Mode
Diffserv	Differentiated Services Architecture
FEC	Forward Equivalence Classes
GOF	Goodness Of Fit
IP	Internet Protocol
ISA	Integrated Services Architecture
OSPF	Open Shortest Path First
KN	Knowledge Network
L2	Data Link Control Layer
L3	Network Layer
LAN	Local Area Network
LSP	Label Switched Path
LSR	Label Switching Router
MAN	Metropolitan Area Network
MPLS	Multi-Protocol Label Switching
NHFLE	Next Hop Forwarding Label Entry
OSI	Open Systems Interconnections
QoS	Quality of Services
SLA	Service Level Agreement
TCP	Transmission Control Protocol
TE	Traffic Engineering

VCC	Virtual Channel Connections
VCI	Virtual Channel Identifier
VPC	Virtual Path Connection
VPI	Virtual Path Identifier
VPN	Virtual Private Network
WAN	Wide Area Network