INVESTIGATION OF GMPLS APPLICATIONS IN OPTICAL SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BURCU GÖKEN

IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2005

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan ÖZGEN

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. İsmet ERKMEN

Head Of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Prof. Dr. Rüyal ERGÜL

Supervisor

Examining Committee Members

Prof. Dr. Yalçın TANIK	(METU,EE)	_____
Prof. Dr. Rüyal ERGÜL	(METU,EE)	_____
Prof. Dr. Mete SEVERCAN	(METU,EE)	_____
Assoc. Prof. Dr. Melek YÜCEL	(METU,EE)	_____
M.Sc. Sıdıka BENGÜR	(ASELSAN)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Burcu Göken

Signature :

ABSTRACT

INVESTIGATION OF GMPLS APPLICATIONS IN OPTICAL SYSTEMS

GÖKEN, Burcu
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Rüyal ERGÜL

August 2005, 110 pages

In this study, possible applications of label switching in large area, fully optical networks are investigated. The objective was to design a label assignment method by using Generalized Multi-Protocol Label Switching (GMPLS) concept to get an efficient optical network operation. In order to fulfill this objective, two new approaches were proposed: a label assignment method and a concatenated label structure.

Label assignment method was designed to provide an efficient utilization of resources. Concatenated label structure aimed handling the label in optical domain. Mainly, the lambda switch capable GMPLS networks were investigated. In order to verify the performance of label assignment method, a simulator was developed.

The results of simulation have clearly indicated that the proposed approaches could be beneficial in an optical network operation.

Keywords: GMPLS, RSVP-TE, label assignment, resource management, lambda switching, all-optical label switching.

ÖZ

OPTİK SİSTEMLERDE GENELLEŞTİRİLMİŞ ÇOK-PROTOKOLLÜ ETİKET ANAHTARLAMA UYGULAMALARININ ARAŞTIRILMASI

GÖKEN, Burcu
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Rüyal ERGÜL

Ağustos 2005, 110 sayfa

Bu çalışmada, geniş alan optik ağlardaki muhtemel etiket anahtarlama uygulamaları araştırılmıştır. Tezin amacı, genelleştirilmiş çok-protokollü etiket anahtarlama teknolojisini kullanarak verimli bir optik ağ iletişimi sağlayacak bir etiket atama yöntemi tasarlamaktır. Bu amacı gerçekleştirmeye yönelik iki yeni metot önerilmektedir: etiket atama metodu ve eklemeli etiket yapısı.

Ağdaki kaynakların en verimli şekilde değerlendirilmesini sağlamak için yeni bir etiket atama metodu tasarlanmıştır. Eklemeli etiket yapısı ise etiketin optik ortamda işlenebilmesini amaçlamaktadır. Temel olarak dalga boyu anahtarlayabilen genelleştirilmiş çok-protokollü etiket anahtarlamalı ağlar ele alınmıştır. Önerilen etiket atama metodunun doğruluğunu göstermek için bir simulasyon programı geliştirilmiştir.

Simulasyon sonuçları, önerilen yöntemlerin bütünüyle optik ağların işleyişinde yararlı olabileceğini açıkca göstermiştir.

Anahtar Kelimeler: GMPLS, RSVP-TE, etiket atama, kaynak yönetimi, bütünüyle optik iletişim.

To My Family

ACKNOWLEDGEMENTS

I would like to thank Prof. Dr. Rüyal Ergül for his valuable supervision and constructive criticism in the development of this thesis. I had the pleasure of working with him.

I give my special thanks to my friend Murat Önder for his generous support in writing the C++ code of the simulation software.

I would like to thank my colleagues for their continuous encouragement and support.

I wish to express my deep gratitude to my parents for their support throughout my life. I would like to thank my cousin İnci for her support. I also thank my brother Çağrı for his understanding, and especially for sharing his PC with me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FİGURE S

LIST OF ABBREVIATIONS

AP	Available Pool
CoS	Class of Service
CR-LDP	Constraint-based Routing LDP
FEC	Forwarding Equivalence Class
FP	Flagged Pool
FSC	Fiber Switch Capable
GMPLS	Generalized Multi-Protocol Label Switching
IGP	Interior Gateway Protocol
IETF	Internet Engineering Task Force
IS-IS	Intermediate System to Intermediate System
IS-IS-TE	IS-IS with Traffic Engineering Extensions
LSC	Lambda Switch Capable
LDP	Label Distribution Protocol
LER	Label Edge Router
LMP	Link Management Protocol
LSR	Label Switch Router
LSP	Label Switched Path
MPLS	Multi-Protocol Label Switching
OOC	Optical Orthogonal Codes
OSC	Optical Switch Controller

OSPF	Open Shortest Path First
OSPF-TE	OSPF with Traffic Engineering Extensions
OXC	Optical Cross-Connect
PSC	Packet Switch Capable
QoS	Quality of Service
RFC	Request For Comment
RSVP	ReSource reserVation Protocol
RSVP-TE	RSVP with Traffic Engineering Extensions
SDH	Synchronous Digital Hierarchy
SONET	Synchronous Optical Networks
TDM	Time Division Multiplexing
TE	Traffic Engineering
TOT	Time Out Timer
TTL	Time To Live
UP	Used Pool
WDM	Wavelength Division Multiplexing

CHAPTER 1

INTRODUCTION

The optical networks have become the most important networking technology in communication because of their bandwidth capacity, which can meet the enormous traffic demand of today's networks. Since the growth in traffic seems to continue, the research and developments on optical networking will keep on and optical networks will constitute a serious part of the future networks.

Two concepts of optical networks, namely all-optical networking and GMPLS, are important research topics to have an efficient optical network. Designing the network all-optically eliminates the electro-optical conversions, which slow high-bandwidth data transmissions in optical networks.

Besides the developments on the all-optical networks technology, the Internet Engineering Task Force (IETF) carried out a lot of researches to make the control plane of the optical networks more flexible and controllable. As a result of these efforts, Multi-Protocol Label Switching (MPLS) [1], and more recently Generalized Multi-Protocol Label Switching (GMPLS) [2] have been proposed by the IETF for the dynamic provision of control plane and restoration of lightpaths in optical networks [3]. GMPLS provide the control plane for devices that switch in any of the following domains: packet, time, wavelength or fiber. This common control plane simplifies the network operation, resource management and Quality-of-Service (QoS) provision expected in new applications [4]. Moreover, GMPLS

ensures fast path setup, which enables fast resource allocation of high-speed networks [5].

The thesis study was focused on the GMPLS which could bring new features to optical networks. GMPLS applications in optical networks has been studied and as a result of these studies, the following concepts were proposed: a label assignment method, which aims to improve resource assignment and management of lambda switch capable GMPLS networks, and a concatenated label structure, which could be useful in all-optical label switching.

Resource assignment and resource management in a network are very critical issues, which affect the network performance directly. If the resource assignment procedure is not based on a reliable method, then the blocking of transmission will be unavoidable. Hence, proposed label assignment method aims to minimize the blocking probabilities in network links.

While investigating GMPLS signaling for label assignment and management, the idea of concatenated label structure has been observed as a useful concept. The concatenated label is a label structure containing the information of all labels for an entire path. So, label concatenation may be a method to carry the resource assignment information of the entire path, from source to destination throughout the network. Besides that, it is thought that, if the labels used through the entire path can be formed at once at the signaling step, this could help to realize all-optical label switching during the data transmission. To make a data transmission in an optical network, firstly the route has to be selected, and then the resource reservation has to be performed. If a route and resources to be used for the transmission are known, the data is sent using the reserved label(s) through the determined route. If the information of all reserved labels can be used to form a concatenated label structure, representing the labels of the entire path, then this concatenated label will be converted to optical domain only once, and information will be conveyed all-optically for the rest of transmission.

Since the objective of the thesis is explained briefly, the outline of the thesis and the content of this study will be presented below.

The first chapter is an introduction chapter in which some introductory materials are given together with the objective and outline of the thesis.

The second chapter gives general information about GMPLS to form a background. Before GMPLS is explained, general concepts of MPLS are given to make the operation of GMPLS more understandable.

The resource assignment and management problem in a lambda switch capable GMPLS network is investigated in the third chapter. A label assignment method and concatenated label structure are proposed. The label assignment method is explained in detail, which has a different algorithm on label flagging [6]. Lastly, the blocking probability concept is given.

In Chapter 4, the simulator that is designed to verify the label assignment method is given. Firstly, the simulator structure and algorithm are explained. Then, the simulation results are discussed.

Finally, in the last chapter, conclusions and future work are presented.

CHAPTER 2

OVERVIEW OF GMPLS

Generalized Multi-Protocol Label Switching (GMPLS) is designed for optical networks to dynamically provision resources and to provide network survivability [2]. GMPLS is a control plane technology proposed by the IETF to support multiple types of switching technologies, not only packet switching but also fiber switching, wavelength switching or time slot switching [3].

In this chapter, general concepts of GMPLS technology will be explained to construct a background for the thesis subject. Since GMPLS is a generalized version of MPLS, firstly MPLS and its operation will be discussed to form a basis for GMPLS.

2.1. What is MPLS?

MPLS is an IETF developed network protocol, which uses a technique known as label switching to forward data through the network. MPLS technology is intended to improve the performance and Quality of Service (QoS) requirements of IP networks [3]. MPLS stands for "Multiprotocol" Label Switching, multiprotocol because its techniques are applicable to any network layer protocol, where it integrates Layer 2 information about network links into Layer 3 [8].

2.1.1. Background Information

In a connectionless packet switched network, each router in the network selects the next hop for the packet independently, based on the results of the packet's header analysis and the results of running the routing algorithm [1]. Firstly, the incoming packet's header is examined and then the packet is assigned to a particular Forwarding Equivalence Class (FEC). The objective here is to increase the payload as much as possible so that resources are used more efficiently. Afterwards each FEC at the router is mapped to the next hop [7].

All packets belonging to the same FEC travel the same path in the network. In MPLS, there is a term called Label Switch Path (LSP). LSP is a data forwarding path determined by labels attached to each data packet, where the data is forwarded at each hop according to the value of the labels [8].

In conventional IP routing, longest prefix address match look-up on the destination address of the received IP packet header [9].As a packet is traveling in the IP network, each hop reexamines the packet and assigns it to a FEC, where packet headers contain more information than is needed. Instead of examining the packet headers, Multi-Protocol Label Switching (MPLS) proposes the method of labeling the packets to be able represent a FEC with a single label, where this label is short, fixed length, locally significant identifier [1].

In MPLS, the assignment of a packet to a particular FEC is done only once, as the packet enters the network. As the packet goes forward in the network, the next hops do not make any further analysis; the packet is switched according to its label. At each hop, the label is replaced with a new label and the packet is forwarded to downstream [1].

2.1.2. Basic Operation of MPLS

In an MPLS network, the incoming packets are assigned with a short fixed-length label by a Label Edge Router (LER). Subsequent routing decisions made by Label Switched Routers (LSR) are based on the MPLS label without having to examine the packet and its header [1]. At each LSR, the label is replaced with a new label, called label swapping, which tells the next router how to forward the packet. The label is removed when the packet leaves the MPLS network [1]. Most of the work is done by LERs, which are responsible for label assignment and label removal; whereas the LSRs do the label swapping.

In Figure 2.1., an example MPLS network is given. Two transmission requests arrive to LER X at the same time, whose destination nodes are different. LER X assigns different labels to the packets. According to the assigned label, each packet traverses the related LSRs and reaches its destination node without any confusion.

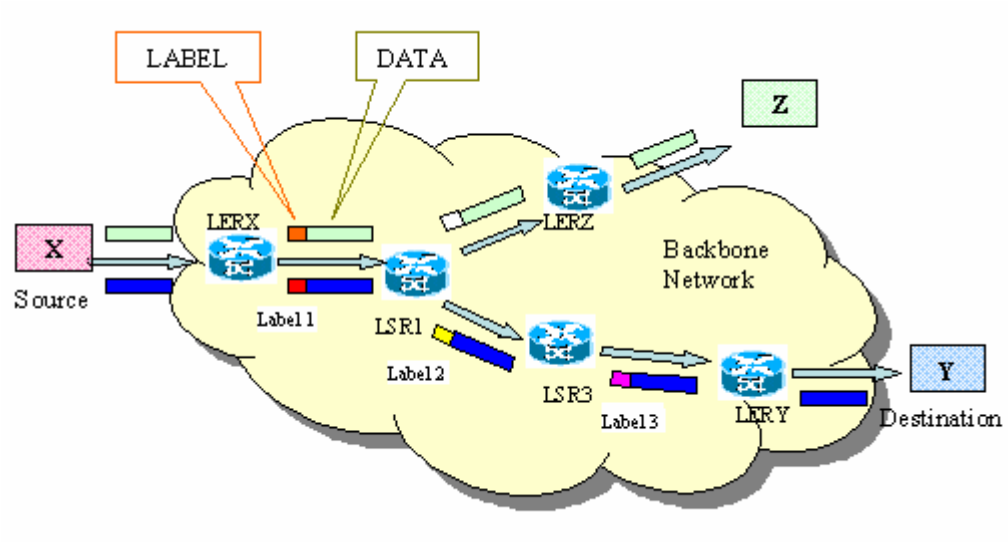

Figure 2.1. Basic operation of an MPLS network

2.1.2.1. MPLS Label

MPLS label is a short fixed length identifier used to represent a FEC. It has local significance only and changes from hop to hop.MPLS stands for "Multiprotocol" Label Switching, multiprotocol because its techniques are applicable to any network layer protocol, where it integrates Layer 2 information about network links into Layer 3 [8].

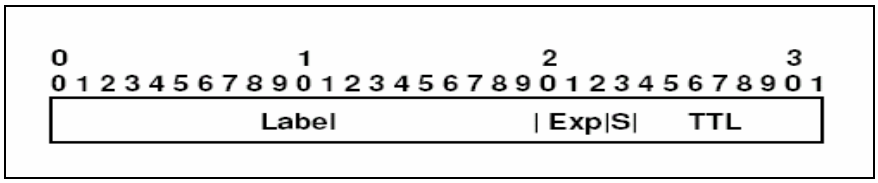

Figure 2.2. MPLS generic label structure [10]

MPLS generic label, which is also called shim header, is given in Figure 2.2. The label contains 32 bits and it is composed of four parts. The contents of the label parts are explained in Table 2.1.

Table 2.1. MPLS generic label contents [9]

Label Part	Content	Length
Label	contains the actual value of the label	20 bits
Experimental (EXP)	reserved for experimental use [e.g. Class of Service (CoS)]	3 bits
Stack (S)	indicates if the label is the last in the stack	1 bit
Time to Live (TTL)	provides conventional IP TTL functionality	8 bits

2.1.3. Label Switch Path Establishment

Before a data packet can be transferred in an MPLS network, an LSP must be established. Establishing an LSP means that the labels have to be assigned to the specified transmission at each LSR. First of all, the network topology and path have to be determined. Then, then labels have to be assigned and distributed. Lastly the data is forwarded through the established LSP.

2.1.3.1. MPLS Routing

MPLS network consists of routers, namely LSRs. The source and destination LSRs are named as LERs. In an MPLS network, routers use routing protocols to discover the network topology. Routing can be done in two ways, either on a hop-by-hop basis or in an explicit manner [3].

In hop-by-hop routing, each LSR decides the next hop independently by running a shortest-path algorithm [3]. The routing decision is made using the data at the routing table, which contains the network topology information about the nodes and the links. The routing table data can be obtained by an Interior Gateway Protocol (IGP), such as Open-Shortest-Path-First (OSPF) [11] or Intermediate System to Intermediate System (IS-IS) [12].

In explicit routing, the ingress LSR decides the entire route for the LSP [3]. The routing decision is made using the network topology and link state information. The ingress LSR can determine an explicit route by running a constraint-based routing algorithm [3]. Explicit routing brings traffic engineering capability, since many constraints such as network resources and link state information are taken into account while deciding the entire route for an LSP [3].

2.1.3.1. MPLS Signaling

After the route decision, a signaling protocol is used to establish the LSP. The signaling protocols to be used for the LSP setup depend on the type of the routing.

If hop-by-hop routing is realized, then Label Distribution Protocol (LDP) [13] is used as the signaling protocol and by use of LDP a label assignment is done at each LSR [3].

If explicit routing is used, Resource Reservation Protocol with traffic engineering extensions (RSVP-TE) [14] or Constraint-based Routing LDP (CR-LDP) [15] may be used as the signaling protocol to distribute the label information between LSRs [3].

2.2. GMPLS Concepts and Operation

In this section, general concepts of GMPLS will be introduced to form a base for the thesis study explained in the following chapters.

2.2.1. GMPLS Evolution

GMPLS extends MPLS to provide the control plane for switching in any of the following domains: packet, time, wavelength or fiber. To adapt the MPLS routing and signaling protocols to the optical domain some modifications and extensions are required to. These requirements are being standardized by the IETF [2], [3], which can be summarized as follows:

1- Enhancements to the OSPF and IS-IS routing protocols to advertise availability of optical resources in the network (e.g. interface types, link types, bandwidth information).

2- Enhancements to the two signaling protocols defined for MPLS-TE signaling, namely RSVP-TE [14] and CR-LDP [15], to allow LSPs to be explicitly established across the optical core.

3- A new Link Management Protocol (LMP) [16] designed to deal with the problems related to link management in optical networks.

2.2.2. Generalized Labels

MPLS was introduced for a Packet Switch Capable (PSC) network, so the shim header (generic label) is the only label structure which meets the requirements of a PSC interface. Since GMPLS support multiple types of switching interfaces, the generic label structure defined by MPLS has to be also enhanced. In this section, supported switching types and label formats related to the supported switching types are explained.

1- *Packet Switch Capable (PSC) Interfaces:* PSC interface can switch the data based on the packet or cell header [17]. An LSR that forwards the data based on the shim header can be given as an example. The label related to PSC interface is called *Packet Label*. It has the same format as the MPLS generic label, which is given in Figure 2.2.

2- *Time Division Multiplex Capable (TDM) Interfaces:* TDM interface can switch the data based on the time slots that carry the data [17]. SONET/SDH cross connect can be given as an example. The label related to TDM interface is called *Timeslot Label*.

3- *Lambda Switch Capable (LSC) Interfaces:* LSC interface can switch the data based on the wavelengths that carry the data, e.g. an optical cross connect (OXC) that operates at the level of single wavelength or group of wavelengths [17]. The label related to LSC interface is called *Wavelength*

Label or *Waveband Label* depending on the operating level of the OXC. The label formats are given in Figure 2.3 and 2.4.

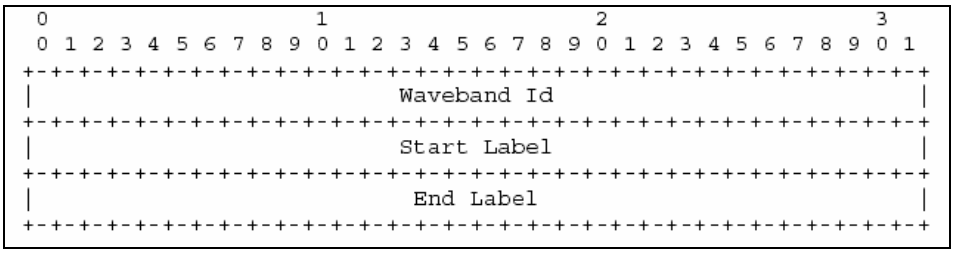

Figure 2.3. Waveband Label Structure

4- *Fiber Switch Capable (FSC) Interfaces:* FSC interface can switch the data based on the physical ports, namely fiber ports [17]. OXCs that operate at the level of single fiber or multiple fibers can be given as an example. The label related to FSC interface is called *Fiber Label*. Its format is given below figure.

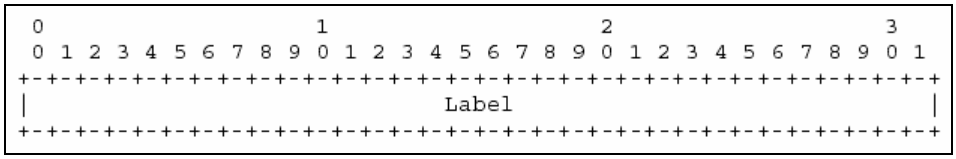

Figure 2.4. Fiber/Wavelength label structure

As mentioned before, the format of a GMPLS label depends completely to the switching interface. It can be complicated such as a SONET/SDH label or as simple as an integer value such as a wavelength label [2].

A GMPLS label is non-hierarchical, which means that it only carries a single level of label. When a hierarchical LSP is wanted to be established, where multiple levels of labels are required, each LSP setup must be occur separately [2].

2.2.3. Out-Of-Band Signaling

In a GMPLS network, to be able to transfer data, control information has to be distributed through the entire network. Control information, which includes routing, signaling and management, forms a base for a transmission. So, in a GMPLS network there must to be at least one bidirectional control channel to exchange the control information between the adjacent nodes [3].

Control information can be transmitted in-band, which means control traffic is carried on the same optical channel, where data is transmitted. This is simple for channel management. However, the control channel must be demultiplexed at each node, which will increase the system complexity and also time consumption [3]. Moreover, it would be wasteful to use a whole bandwidth of an optical network as a signaling channel [8]. These can be avoided by carrying the control information out-of-band, since the control channel does not have to use the same physical medium as the data channel. Control traffic may be transmitted via another fiber, wavelength or even using gigabit ethernet link [3].

Using out-of-band signaling has the advantages that are given in the previous paragraph, but it requires control channel establishment, maintenance and management. All of them can be provided by LMP [16].

2.2.5. Extended Protocols for GMPLS Networks

Since GMPLS extends the MPLS control plane, it has to extend all things related with the control plane. Control plane has two building blocks, which are routing and signaling. GMPLS extends “OSPF with Traffic Engineering Extensions (OSPF-TE)” [18] and “IS-IS with Traffic Engineering Extensions (IS-IS-TE)”

[19] for routing. “OSPF Extensions in Support for GMPLS” [20] and “IS-IS Extensions in Support for GMPLS” [21] are the extended versions of routing protocols used in a GMPLS network. The signaling protocols RSVP-TE and CR-LDP used in MPLS networks are also extended and the required extensions are given in [22] for RSVP-TE and in [23] for CR-LDP. GMPLS also brings a new protocol, Link Management Protocol (LMP), which is used to define and manage both control channels and data channels between GMPLS nodes.

LMP provides the following functions in the network:

1- Control channel management,

2- Link connectivity verification,

3- Link property correlation,

4- Fault isolation [16].

In a GMPLS network, an LSP must be established similar to the case in an MPLS network mentioned in Section 2.1.3. To establish an LSP, the route has to be determined which is calculated by using a routing algorithm. First the route is decided, then the signaling session begins and an LSP is established via a signaling protocol. Once the LSP setup is complete, the data channel is established to carry data traffic.

2.2.6 GMPLS Signaling

The signaling is composed of two parts: label request and label reservation. Label request is send by the source node to downstream. On reception of label request signal by the destination node, the destination decides the label(s) used for the transmission sends the label reservation signal to upstream. When the source receives the reservation message, it prepares the data and sends is to the

destination with the assigned label. The basic operation of signaling session is given in the below diagram.

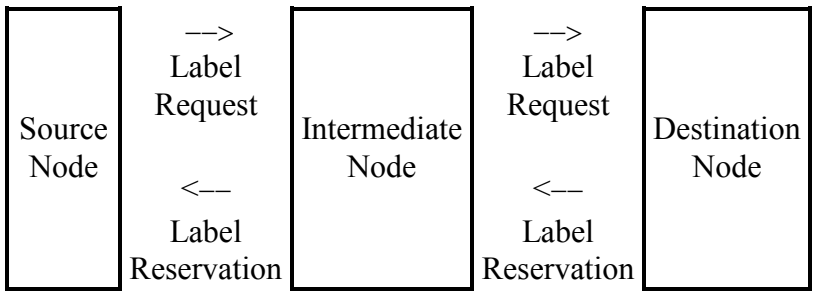

Figure 2.5. GMPLS signaling messages

Both RSVP-TE and CR-LDP can be used as the signaling protocol. These two protocols have different terminology. Label request message sent in the downstream direction is called Path message in the RSVP-TE, whereas it is called Label Request message in the CR-LDP. Similarly, Label reservation message sent in the downstream direction is called Resv message in the RSVP-TE, whereas it is called Label Mapping message in the CR-LDP.

Since RSVP-TE is used for the signaling part of the thesis study, in the next section the evolution of RSVP-TE is given briefly.

2.2.5.1. RSVP-TE Evolution

To understand the use of RSVP signaling in GMPLS the evolution of RSVP has to be considered firstly:

1- RSVP is a resource reservation setup protocol designed for an integrated services Internet. The functional specifications of RSVP are described in [24].

2- The RSVP-TE protocol adds necessary extensions to the RSVP protocol for establishing LSPs in MPLS networks. RSVP-TE supports the instantiation of explicitly routed LSPs, with or without resource reservations. The extensions to the RSVP protocol are given in [14].

3- To be able to use the RSVP-TE protocol in GMPLS networks some extensions are required to MPLS-RSVP-TE signaling to support GMPLS. Required extensions to RSVP-TE for GMPLS networks are given in [22].

CHAPTER 3

RESOURCE ASSIGNMENT AND MANAGEMENT IN GMPLS NETWORKS

Resource assignment and management in a network is a very critical issue, which affects the network performance directly. If the resource assignment is not based on a reliable method, then the blocking of transmissions will be unavoidable and as a result many connection failures will occur in the network. Hence, in order to increase the performance of a network, resource assignment techniques have to be improved continuously.

In this chapter, the resource assignment and management in a GMPLS network will be investigated. It will be based on the RSVP-TE protocol, one of the defined signaling protocols for an explicitly routed GMPLS network. The use of label restriction by the upstream to support optical constraints will be examined. A method called label flagging will be given in detail. Then, we will propose a new label assignment method, which uses a modified version of label set structure based on the label flagging, and we will discuss the advantages of this method and its effects on the network performance.

In GMPLS, the labels are directly related to the network resources. We will restrict the resource assignment and labels to WDM networks, where a label will represent a specific wavelength to be used, symbolized by lambda (λ). From now

on, fiber switch capable, time switch capable and packet switch capable GMPLS networks will not be considered in this study.

3.1. GMPLS Signaling for Label Assignment

In the previous chapter, signaling in GMPLS networks is introduced. Here, we will summarize the signaling procedures briefly and will focus on the parts of the signaling which are related to our study.

In an explicitly routed GMPLS network, the ingress LSR decides the entire route for the LSP by using any constrained shortest path first algorithm. After a route for the LSP is decided, a signaling session begins to distribute the label information between LSRs. RSVP-TE or CR-LDP may be used as the signaling protocol. From this point, the study will be based on RSVP-TE protocol and RSVP-TE messages will be mentioned while examining the network signaling.

Since GMPLS networks are optical networks and multiple types of switching are supported in the network, some hardware limitations come out during the data transfer, e.g. limitations caused by wavelength conversion incapable nodes. As mentioned in Chapter 2, the signaling protocols used in MPLS networks have to be improved to overcome these limitations. Therefore, GMPLS brings some extensions to signaling protocols. One of the important extensions is label restriction by the upstream to support optical constraints. RSVP-TE protocol realizes the label restriction by introducing the *Label Set* object in the label request message.

Before transferring the data in the optical domain, the signaling session has to be completed to decide the label, which will be used to guide the data throughout the network. In the signaling session, the source node initiates a label request addressed to the destination node. The label request is forwarded from source to destination via intermediate nodes and each node in the network forms a Label Set according to its hardware capabilities and idle resources. When the label request

message reaches to the destination, destination node prepares a label assignment message and sends it in the backward direction. While the label assignment message travels in the network, the nodes will learn the label to be used. When the label assignment message reaches to the source, source node prepares the data and sends it with the assigned label. The signaling messages are carried out-of-band.

In RSVP-TE protocol, the label request message sent in the downstream direction is named as *Path* message. The label assignment message sent in the upstream direction is named as *Resv* message. Since GMPLS labels are directly related to network resources, dictating the label choice by downstream node can lead to conflicts during LSP setup. So, to restrict the label choice of the downstream node, the upstream node includes Label Set object in the Path message.

3.1.1. Label Set

As mentioned in Section 3.1, the Label Set is used to limit label choices of a downstream node to a set of acceptable labels for a particular LSP setup. The receiver of a Label Set must restrict its label selection to one of the label within the set. Thus, the labels included in the Label Set restrict the labels that can be selected by the egress node.

While a Path message including Label Set is traveling in the network, the general procedures to be applied are described below:

1- The source node initiates the Path message including a Label Set object. Label Set is formed based on the node's hardware capabilities and idle resources for the specified link.

2- As Path message is traveling through the specified path, each intermediate node generates its own outgoing Label Set, based on the incoming Label Set, the node's hardware capabilities and the idle resources. When a Path message is received, the Label Set represented in the message is compared

against the set of available labels at the downstream interface and the resulting intersecting Label Set is forwarded in a Path message. When the resulting Label Set is empty, the Path is terminated, and an error message is generated [22].

3- When the destination node receives the Path message, it decides the label to be used according to the Label Set and prepares the Resv message.

The use of Label Set is optional. The absence of Label Set implies that all labels are acceptable [22]. However, to reduce the chances of transmission to be failed because of the hardware incapability or label collisions, Label Set usage is suggested. The advantages of using Label Set are explained in the following section.

3.1.2. Advantages of Label Set Application

The Label Set is useful in the optical domain when, for example,

1- the LSR is only capable of transmitting or receiving a subset of wavelengths that can be switched by neighboring LSRs

2- there is a sequence of interfaces which cannot support wavelength conversion and require the same wavelength be used end-to-end over an entire path

3- it is desired to limit the amount of wavelength conversion to reduce the distortion on the optical signals

4- two ends of a link support different sets of wavelengths [17].

Besides overcoming the optical constraints, Label Set is useful for reducing the blocking probability due to label collisions when establishing lightpaths in a lambda switched GMPLS network. An LSP setup request may fail due to one of two blocking events, namely *Forward-link Blocking* and *Backward-link Blocking*.

Forward-link Blocking happens while the label request signaling travels in the downstream direction. It is due to insufficient label resources on the forward link and non-load balancing routing algorithms. Backward-link blocking occurs while the label response travels towards the source node. It is due to the conflict of label reservations, which means that the same label over the same link is selected for more than one connection request [6].

When Label Set is included in the Path message, the labels which can be selected throughout the network will be restricted, which will decrease the probability of reserving the same label, i.e. same wavelength. So, Label Set application will decrease the wavelength conflicts and therefore the failures due to the blocking probability of LSP setups in the network.

3.1.3. Process of Label Set Object at the Network Nodes

On reception of a Path message, the Label Set object in the Path message is processed by the node. The receiving node compares the Label Set represented in the message against the set of available labels at the downstream interface and the resulting intersecting Label Set is forwarded in a Path message.

In a WDM network, where the labels are wavelengths, there are optical crossconnects (OXCs) at the nodes to do the lambda switching. An OXC has incoming and outgoing lambda ports, connected to neighboring OXCs, and incoming and outgoing data ports attached to a controlling router. An OXC is composed of OXC Switching Controller (OSC) and OXC switch fabric [25]. The OSC converts the received messages from the control channel to the proper control command, and sends this command to the OXC fabric, which will do the lambda switching job [25]. When the Path message is received by a node, the Label Set has to be processed by OSC at that node.

To process a Label Set at the node, optical switch controllers (OSCs) have two pools of wavelengths, namely *Used Pool (UP)* and *Available Pool* (AP) [6]. The

idle resources are placed at the Available Pool and the busy resources are placed at the Used Pool. While generating the outgoing Label Set, the OSC compares the received Label Set with the wavelength's state information obtained from the pools. If the wavelength is in the AP, it can be suggested to downstream, so it remains at the Label Set. If any wavelength within the received Label Set is in the UP, this wavelength is deleted from the Label Set before forwarding it to a downstream node.

Although Label Set plays an important role to reduce the failure of LSP setups, this is not sufficient to eliminate the backward-link blocking completely. When two or more destination nodes select the same wavelength for LSPs that share the same links, the occurrence of backward-link blocking is unavoidable. To provide an efficient solution to the label collision problem and in turn decrease the wavelength blocking probability effectively, a third pool namely *Flagged Pool (FP)* is proposed by IETF working group. The proposed method FP represents the wavelengths that have been suggested by an upstream node but that have not yet been reserved by the destination node. This method, which is called Label Flagging Method, aims the probability of selecting the same wavelength for different LSPs that share the same links. The proposed method is published by [6] and [26], in Section 3.2 this method is given as to build a background before proposing our new solution [26].

3.2. Label Flagging Method

In the Label Set application, OSC controls the pools to process the Label Set. In addition to the Available and Used Pools, Flagged Pool is proposed to reduce the blocking in GMPLS-centric all-optical networks. The labels in the UP are reserved and selected, where the labels in the AP are neither reserved nor selected. When an available label selected from the AP is suggested to downstream, it is marked as flagged and it is placed to the Flagged Pool (FP). FP provides a gray area for the labels, which are subject to collision, and thus being in an intermediate state

between the labels belonging to the Used Pool and the ones belonging to the Available Pool. Because of the added pool FP, the signaling mechanism and Label Set process at the OSCs has to be modified. The modifications are given in the following sections.

3.2.1. Required RSVP-TE Extensions for Label Flagging

While proposing the new pool FP, new GMPLS Signaling mechanisms referred to as generalized label flagging method and RSVP-TE specific signaling extensions formats are introduced by IETF working group [26].

The *Flagged Set* object is added to the Path message in order to suggest wavelengths from the FP. The Flagged Set contains wavelengths that are in the FP of at least one node on the network traversed by its Path message, and excludes wavelengths that are in the UP of any nodes [26].

While the Path message travels through the network, different cases may occur. The cases can be listed as follows:

1- If a wavelength is in the Label Set of a Path message, and the wavelength is also in the AP of a node, then the wavelength will remain in the Label Set of the Path message.

2- If a wavelength is either in the Label Set or Flagged Set of a Path message and the wavelength is in the FP of a node, then the wavelength will be placed in the Flagged Set of the Path message.

3- If a wavelength is either in the Label Set or Flagged set of a Path message, and the wavelength is in the UP of a node, then the wavelength will be removed from the Label Set or Flagged Set of the Path message [6].

3.2.2.1 Label Prioritization

The labels are inserted into the Flagged Pool by using a local timestamp to be able to create different priority levels, which provides differentiation when selecting a wavelength at the destination node. The priority levels are determined according to the local timestamp information and the flagged labels with different priorities are placed in different Flagged Sets while preparing a Path message.

The wavelengths, which are placed at the FP earlier, will have a higher priority, since collision possibility will decrease for the wavelength at the FP as time passes. The suggested labels within the Path message are placed from a high priority to a low priority. The locations of Label Set and Flagged Sets in a Path message are arranged in the order that is given in Figure 3.1. Here, N represents the lowest priority level.

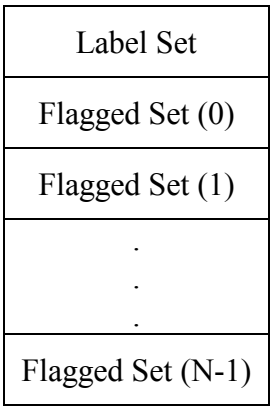

Figure 3.1. The order of Label Set and Flagged Sets in a Path message

Each downstream node in the network updates the Label Set and Flagged Sets, which provides global information about priorities of the network resources [6].

At the destination node, wavelengths in the Label Set will be preferentially selected over wavelengths in the Flagged Sets, and the selection of flagged wavelengths depends on the priority level of the specified Flagged Set.

3.2.2.2 Flagged Set Operation

When the source node prepares a Path message, it places the wavelengths belonging to the AP into the Label Set. The wavelengths belonging to the FP are inserted into the Flagged Sets according to their local timestamps.

When a Path message arrives to an intermediate node, the OSC examines the Label Set and the Flagged Sets of the Path message, and locally updates the local timestamp for the wavelengths which have not been reserved yet. Based on the local timestamps, OSC assign a priority to the flagged wavelengths and place them into the appropriate Flagged Set of the Path message. The wavelengths which are placed to the FP earlier will have a higher priority, since collision possibility decreases as the value of time passing for the wavelength at the FP increases.

On reception of Path message, the destination node selects a wavelength from the Label Set. If there are no wavelengths in the Label Set, then it selects a wavelength from the Flagged Set with next highest priority, and so on. When destination node selects the label, it encapsulates the label into Resv message, and forwards the Resv message to its upstream node [6].

For Flagged Set operation, an expiration threshold is defined to remove a label from FP to the AP for the cases of connection failures in the network. So, the local timestamp is also used to transit a wavelength from the FP to the AP when the threshold time is expired [26].

To express the label flagging operation and label prioritization given in [6] better, the below example can be given. This example examines only the label suggestion procedure.

The following conditions are assumed:

1- There are 8 wavelengths per link.

2- The number of priority level is 4.

3- $t_i < t_j$, if $i < j$.

4- $t_j - t_i = t$, if $j=i+1$, where $t > 0$.

5- The expiration threshold is equal to 4t.

6- Node 1 is the source node, Node 2 is the intermediate node, and Node 3 is the destination node. Node 1 sends the Path message at time t_4 .

The state diagrams of resource pools are given in the below figures, and then the operation steps are explained.

Node 1		
AP	**FP**	**UP**
λ_1		λ_2
λ_3		
λ_4		
λ_5		
λ_6		
λ_7		
λ_8		

Node 2		
AP	**FP**	**UP**
λ_1	λ_5, t_1	λ_3
λ_2	λ_6, t_1	
λ_7	λ_4, t_2	
λ_8		

Node 3		
AP	**FP**	**UP**
λ_3	λ_8, t_1	λ_1
λ_5		λ_2
λ_6		λ_7
λ_8		

Figure 3.2. Initial states of the resource pools

Node 1		
AP	**FP**	**UP**
	λ_1, t_4	λ_2
	λ_3, t_4	
	λ_4, t_4	
	λ_5, t_4	
	λ_6, t_4	
	λ_7, t_4	
	λ_8, t_4	

−−> Path

Node 2		
AP	**FP**	**UP**
λ_1	λ_5, t_2	λ_3
λ_2	λ_6, t_2	
λ_7	λ_4, t_3	
λ_8		

Node 3		
AP	**FP**	**UP**
λ_3	λ_8, t_1	λ_1
λ_4		λ_2
λ_5		λ_7
λ_6		

Figure 3.3. States of resource pools at time t_4

Node 1			Node 2				Node 3		
AP	FP	UP	AP	FP	UP		AP	FP	UP
	λ_1, t_4	λ_2	λ_2	λ_5, t_5	λ_3		λ_3		λ_1
	λ_3, t_4			λ_6, t_5			λ_4		λ_2
	λ_4, t_4			λ_4, t_5			λ_5		λ_7
	λ_5, t_4			λ_1, t_5		–––>	λ_6		
	λ_6, t_4			λ_7, t_5		Path	λ_8		
	λ_7, t_4			λ_8, t_5					
	λ_8, t_4								

Figure 3.4. States of resource pools at time t_5

Operation Steps of label request signaling and flagging operation:

1- At time t_4, Node 1 initiates a label request signaling by sending the Path message. Since there is no wavelength in the FP, there is only Label Set in the Path message and no Flagged Sets. Label Set is formed by the wavelengths belonging to the AP are they are put into the FP with local timestamp t_4. The Path message contains the following Label Set and Flagged Sets:

Label Set: $[\lambda_1, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7, \lambda_8]$

Figure 3.5. Suggested Labels in the Path message sent at time t_4

2- At time t_5,

a- Node 2 updates and sends the Path message. Firstly, it extracts the labels at the UP from the received Label Set and Flagged Sets. Since there are wavelengths both in the AP and FP, the Path message contains both Label Set and Flagged Sets. Label Set is formed by the wavelengths belonging to the AP are they are put into the FP with local timestamp t_4. The wavelengths belonging to the FP are inserted

into the Flagged Sets by assigning priority levels according to their local timestamps t_2, t_3. The Label Set and Flagged Sets contained in the Path message are:

Label Set:	$[\lambda_1, \lambda_7, \lambda_8]$
Flagged Set (0):	$[\lambda_5, \lambda_6]$
Flagged Set (1):	$[\lambda_4]$

Figure 3.6. Suggested Labels in the Path message sent at time t_5

b- Node 3 updates its FP pool and transfers λ_8 from FP to AP, since the threshold time for label staying in FP is expired, $t_5 - t_1 = 4t$.

3- On reception of Path message, Node 3 (destination node) will select a label from the Label Set. If the Label Set were empty, then the labels in the Flagged Set (0) would be selected.

Both the several priority levels and the threshold expiration applied for the labels in the FP enables relaxing the usage of the label set and minimize the number of label collisions during the label selection [26]. We propose a new label assignment method by modifying the label flagging operation to be able to further minimize the backward blocking probability. The proposed method is given in the following sections in detail.

3.3. Proposed Label Assignment Method

Based on the given idea in [6] and [26], and using the GMPLS signaling concepts given in Chapter 2, we propose a new label assignment method by using the AP, FP and UP concepts and modifying the Label Set and Label structures. As mentioned in the previous sections, the label choice of the downstream node may

be restricted by forwarding only Label Set or both Label Set and Flagged Sets in the Path message. In our method, the OSCs will use the information in the AP, FP and UP to restrict the downstream node's label selection. However, the flagged labels will not be suggested and therefore Flagged Set will not be used. We make use of label flagging method in a different manner and we select the labels being inserted in the Label Set according to the new algorithm which will be declared in the following sections.

Our study is based on an all-optical GMPLS network, where a label will represent directly a specific wavelength. So, the contents of the network resource pools, namely AP, FP and UP, will be the wavelengths. Also we have to mention that according to GMPLS terminology, label assignment term may be used in the meaning of resource assignment.

3.3.1. Network Resource Pools

In an all-optical GMPLS network, where the network resources are the wavelengths, the resource information is kept by means of the pools in the OSCs. According to the topology of the network, a node may be connected to more than one link, where the wavelengths on a link are independent from the ones of another link. Therefore, the resource information of each link connected to a node has to be kept by the OSCs.

To manage the network resources according to some procedures, the wavelengths are placed into the pools with some parameters, e.g. local timestamp. The definitions of resource pools and the required parameters, namely resource state parameters, are given below.

1- *Resource Pools*: Three types of pools are used for resource assignment and management.

a- *Available Pool (AP)*: AP keeps idle resources. Idle resources are the wavelengths which are not used or not suggested to downstream.

b- *Flagged Pool (FP)*: FP keeps flagged resources. Flagged resources are the wavelengths which are suggested to downstream.

c- *Used Pool (UP)*: UP keeps busy resources. Busy resources are wavelengths which are reserved or currently in use.

2- *Resource State Parameters*: These parameters are independent for each state of the resources. If a resource changes its state, for example idle to busy, resource state parameters are updated according to this change. Three types of resource state parameters are used.

a- *Message ID (m_i)*: Message ID is used to distinguish the data transmissions from each other. It is a global parameter valid in the whole network. Message ID is assigned at the beginning of the signaling session and it is used until the data transmission is completed. This parameter is used for the resource states of belonging to FP or UP. When a resource is in the AP, then the resource has not any currently assigned Message ID. Message ID is denoted by m_i, where i represents the number of the specified transmission.

b- *Local Timestamp (t_i)*: Local timestamp is used to mark the local time when the resource changes its state, which means that the resource is put into another pool. This parameter is used for the resource states of belonging to FP or UP. When a resource is in the AP, then assigning a local timestamp to that resource is not necessary. Timestamp is a local parameter, which is valid only for the specified link of the node. Local Timestamp is denoted by t_i.

c- *Time-Out-Timer (TOT*): Time-Out-Timer is the maximum wait duration of a resource at a pool. When TOT is over, then the resource

has to change its state. This parameter is used for the resource states of belonging to FP or UP. For the resources is in the AP, there is not any TOT value. Maximum wait duration of a resource at the FP is denoted by TOT_{FP}. When TOT_{FP} is over, then the resource is transported from FP to AP. Maximum wait duration of a resource at the UP is denoted by TOT_{UP}. When TOT_{UP} is over, then the label is transported from UP to AP. TOT_{FP} and $TOT_{UP\ are}$ different parameters, their values may be same or different depending on the network conditions. The values of TOT_{FP} and $TOT_{UP\ are}$ set to constant values according to the network performance.

In this section, we have provided some information on the resource pools and the resource state parameters. Before explaining the proposed label assignment method, we have to focus on the structures of Label Set and Label objects used in the RSVP-TE Signaling. While presenting our proposal, we will use a modified version of these objects. In the next section, the modifications which have to be applied to the Label Set and Label objects are given.

3.3.2. Modified GMPLS Signaling Components

As mentioned at the beginning of Section 3.3, we propose a new label assignment method by using the AP, FP and UP concepts and modifying the Label Set and Label structures. In our method, the OSCs will use the information in the AP, FP and UP to restrict the downstream node's label selection. However, the flagged labels will not be suggested and therefore Flagged Set will not be used. We make use of label flagging method in a different manner.

When a node is source node, then the node prepares and forwards the Path message by putting its available resources to the Label Set. In the standard use of Label Set, each intermediate node receiving the Path message compares the available wavelengths with the incoming Label Set and takes the intersection, and

then the Label Set content is updated by the outcome of the intersection. In this way, the resource state information is carried starting from the source node up to the destination node. For the wavelength conversion incapable networks when the intersection results in an empty set, then the label request process is stopped and a failure message returns to the source node. If the node is capable of making wavelength conversion, then the Label Set can be removed from the Path message.

Instead of using the standard structure of the Label Set, we propose a different method to form the Label Set. According to our method, each node takes the intersection of the available wavelengths with the incoming Label Set, but it forms its own Label Set independent from the incoming Label Set, where the intermediate nodes behave like a source node. Therefore, they are not responsible for updating the Label Set. The proposed structure can be called as *Concatenated Label Set*. Although the intermediate node suggests its own Label Set, it takes the intersection of its available wavelengths with the last Label Set in the incoming Path message. Since a concatenated version of Label Set, the last Label Set in the concatenated structure has to be compared. The node makes the comparison only to get information whether there will be a wavelength conversion during the transmission. When the intersection is an empty set, this shows that during the data transmission a wavelength conversion will occur at this node.

In the concatenated Label Set structure method, all intermediate nodes will behave identical, and will suggest their available resources to downstream by adding their own Label Set into the Path message. While adding the Label Set, they have put also their Node IDs. The Label Sets of consecutive nodes are placed into the concatenated Label Set in an ordered way. The structure of concatenated Label Set is given in Figure 3.7.

Node ID (N_i)	Label Set of Source Node
Node ID (N_j)	Label Set of the First Intermediate Node
Node ID (N_k)	Label Set of the Second Intermediate Node
. . .	. . .
Node ID (N_n)	Label Set of the Last Intermediate Node

Figure 3.7. The structure of concatenated Label Set

Traffic engineering capability of a network will be much better, when many constraints such as network resources and link state information are taken into account while establishing an LSP. Since the concatenated Label Set provides the whole information of the network resources along the LSP, it will have an important function while preparing the Resv message at the destination node. Similar to the concatenated Label Set structure, the Resv message will have a concatenated Label structure including all of the labels which will be used through the specified LSP. The concatenated Label will provide an efficient solution while transferring the data in the optical domain. The structure of concatenated Label will be as follows:

Node ID (N_i)	Label assigned to Source Node
Node ID (N_j)	Label assigned to First Intermediate Node
Node ID (N_k)	Label assigned to Second Intermediate Node
. . .	. . .
Node ID (N_n)	Label assigned to Last Intermediate Node

Figure 3.8. The structure of concatenated Label Set

Actually, the label assigned to a node means that this label (resource) will be used when the data is being transmitted on the link between this node and the next one. For example, if the transmission will occur from Node 1 to Node 4, then the label will be assigned to Node 1.

3.3.3. Label Assignment Procedures

Before starting a data transmission in the optical domain, the signaling session has to be completed to decide the label, which will be used to guide the data throughout the network. This is called the resource assignment or label assignment for the concerned data transmission. The signaling session has two steps: Label Request (by sending a Path message to downstream) and Label Reservation (by sending a Resv message to upstream). After the signaling session is completed the data transmission in the optical domain starts. While the data is traveling through the network, there is a thing to be done, which is to set the used resources for the transmission free. This step is called Removal of Label Reservation. As mentioned in Section 3.3.1, two TOTs are defined in the network, namely TOT_{FP} and $TOT_{UP,}$ which are used to set the resources free, when the signaling or data transmission is not completed in the expected time. The use of these TOTs will be also explained in the scope of this section.

3.3.3.1 Label Request Signaling

At the label request step of signaling, the source node prepares a Path message by placing the wavelengths belonging to the AP into the Label Set. If a wavelength is suggested to downstream, then it is marked as flagged and put into the FP. At this moment, a local timestamp and a Message ID is assigned to that wavelength, where the local timestamp indicates the time, when the wavelength is transferred from AP to FP and Message ID shows the number of the specified transmission. To be able to initiate a Path message, the AP of the source node should not be empty. If the source node does not have any available resources at the moment of

data transmission request, then the label request will be delayed until a resource becomes idle.

When a Path message arrives to an intermediate node, the node controls its available resources. The node puts the available resources into the Label Set and adds the Label Set to the Path message. Then, the Path message including the concatenated Label Set is forwarded to downstream. The wavelengths are assigned with the Message ID of the related transmission and local timestamps, when they are put in the FP. If the node does not have any available resources at the moment of Path message arrival, then the label request will be delayed until a resource becomes idle.

3.3.3.2 Label Reservation Signaling

In the standard case of label reservation scheme, the destination node selects a wavelength from the Label Set on reception of a Path message. When destination node selects the label, it encapsulates the label into Resv message, and forwards the Resv message to its upstream node. In the proposed label assignment method, the function of the destination node changes a little bit, because in this case there is not only one Label Set in the received Path message, but a concatenated Label Set is provided. On reception of the Path message the destination node obtains the whole information of the network resources along the LSP. Using this information the destination node will decide the label(s) to be used for the specified transmission and inform the nodes by a Resv message including concatenated Label.

Firstly, the destination node examines the concatenated Label Set, and cuts it into parts of independent Label Sets. Then, it takes the intersection of all Label Sets included. If the intersection results in a wavelength, then destination node prepares the Resv message in a concatenated manner, where all labels assigned to each node will be the same. If the intersection is an empty set, then the destination

knows that wavelength conversion will occur during this data transmission, and it tries to reserve labels in a way that minimum number of wavelength conversion will occur. Therefore, it searches the wavelength which is suggested mostly and so on. After the labels are determined, the destination prepares the Resv message including concatenated Label and sends it to upstream. While the concatenated Label is being prepared, the destination also put the Node IDs in front of the labels. Node IDs are obtained from the concatenated Label Set.

On reception of Resv message, each node searches its Node ID in the concatenated Label and gets the label information bound with its Node ID. Then, the node will arrange the resource pools according to the assigned label. The wavelengths suggested to downstream were waiting in the FP. The node transfers the assigned reserved label from FP to the UP to reserve it for the specified data transmission. At this moment, a new local timestamp is assigned to that wavelength, where the local timestamp indicates the time, when the wavelength is transferred from FP to UP. Message ID of the wavelength stays unchanged, which shows the number of the specified transmission. The other wavelengths in the FP, which have the same Message ID as the one in the Resv Message, are not reserved, so they are put into the AP. The attached local timestamps and Message IDs are removed from them.

When the source node receives the Resv message, like the intermediate nodes it does the same job with the concatenated Label and resource pools. Afterwards, it prepares the data and sends it using the assigned wavelength to downstream.

3.3.3.3 Removal of Label Reservation

With the start of data transmission, the nodes passing the data have to set the used resources free, which means transferring the used wavelength from UP to AP and arranging the parameters assigned to this wavelength.

After the source node sends the data, it transfers the wavelength used for this transmission from UP to AP, and the local timestamp and Message ID assigned to this wavelength are cleared.

An intermediate node receiving a data from upstream examines the Message ID of the incoming data and forwards it with the reserved label assigned for this Message ID. After the node sends the data to downstream, it transfers the wavelength used for this transmission from UP to AP, and the local timestamp and Message ID assigned to this wavelength are cleared.

When the destination node receives the data, it takes the data makes the required job, e.g. converting the optical data into the electrical domain.

3.3.3.4 Use of Time-Out-Timers

If there is no problem in the network, and the signaling session and the data transmission are completed in the expected time, then the resources are set free with the natural procedures. As explained in previous sections, a flagged label is set to idle, when this label is not reserved with the received the Resv message. Similarly, a reserved label is set to idle, when the data passes the node, where this label is reserved for the specified data transmission. However, in the real life the things may not be well as explained here. There may be unexpected collisions, corrupted transmissions, problems with the network devices etc. So, there should be a mechanism to reset the network resources, which means making the flagged or reserved resources acceptable. This can be realized by using expiration thresholds in the network. So, there are two Time-Out-Timers (TOTs) defined in the network, namely TOT_{FP} and TOT_{UP}.

When a label is flagged, it is put into the FP with a local timestamp assigned to it. If the difference between the current time and the local timestamp of the label is greater or equal to TOT_{FP}, then it is assumed that a problem has been occurred with the concerned network signaling and the label is transferred from FP to AP.

So, the resource is set to free for other data transmissions. Similarly, when a label is reserved, it is put into the UP with a local timestamp assigned to it. If the difference between the current time and the local timestamp of the label is greater or equal to TOT_{UP}, then it is assumed that a problem has been occurred with the concerned network signaling or data transmission and the label is transferred from UP to AP. So, the resource is set to free for other data transmissions.

3.3.4 Considerations at Label Suggestion

In the proposed label assignment method, the available resources suggested to downstream are marked as flagged and put into the FP. To get rid of the probability of assigning the same label for different transmission requests, we do not suggest the labels in the FP and we only use the labels in the AP when forming the Label Set. In the case of consecutive transmission requests for the same LSP, the first request will suggest all available resources, which means all acceptable resources will be flagged and the other transmissions will be automatically delayed until the resource reservation is done. So, this will result in an inefficient network performance, since the resources are not used, although they are idle.

To deal with the situation given in the above paragraph, we propose to limit the maximum number of suggestible resources. This means that the maximum number of suggested wavelengths may be equal or less than the maximum number of the network resources. The maximum number of suggestible wavelengths will refer to the maximum size of the Label Set. For example, if there are 8 wavelengths per link, then the maximum size of the Label Set may be adjusted from 1 to 8. Decreasing the number of suggested wavelengths increases the number of transmissions per time, but the probability of wavelength conversion increases, since the probability of suggesting the same Label Set at all nodes along the LSP will decrease.

There are two concerns: the delay due to inefficient use of available resources and the delay due to the wavelength conversions. By decreasing the number of suggestible resources, the delay due to inefficient use of available resources decreases, but the delay due to wavelength conversions increases. So, the best network performance can be achieved with the trade-off between these concerns. According to the network resources and traffic intensity of the network, the number of suggestible wavelengths can be adjusted to obtain an efficient network resource usage.

3.3.4 Label Assignment Example

In Section 3.3 and in its subsections, the related information on the proposed label assignment method is given and the assignment procedure is explained in detail. In this section, we will give an example to consolidate the fundamentals of the introduced method.

The following conditions are assumed for the example:

1- There are 8 wavelengths per link.

2- The sum of message processing time at a node and transmission time between two nodes is equal to t. This means that $t_j - t_i = t$, if $j=i+1$, where $t > 0$.

3- Both TOT_{FP} and TOT_{UP} are equal to 30t.

4- The number of suggested wavelengths at a time is equal to 2.

5- Node 1 is the source node, Node 2 is the intermediate node, and Node 3 is the destination node. Node 1 sends the Path message at time t_6.

The state diagrams of resource pools are given in the below figures, and then the operation steps are explained.

Node 1		
AP	**FP**	**UP**
λ_1	λ_7, t_1	λ_2, t_1
λ_3	λ_8, t_1	
λ_4		
λ_5		
λ_6		

Node 2		
AP	**FP**	**UP**
λ_1	λ_5, t_1	λ_3, t_1
λ_2	λ_6, t_1	
λ_7	λ_4, t_2	
λ_8		

Node 3		
AP	**FP**	**UP**
λ_1	λ_4, t_1	λ_3, t_2
λ_5		λ_2, t_3
λ_6		λ_7, t_3
λ_8		

Figure 3.9. Initial states of the resource pools

Node 1		
AP	**FP**	**UP**
λ_4	λ_7, t_1	λ_2, t_1
λ_5	λ_8, t_1	
λ_6	λ_1, t_6	
	λ_3, t_6	

–––> Path

Node 2		
AP	**FP**	**UP**
λ_1	λ_5, t_2	λ_3, t_1
λ_2	λ_6, t_2	
λ_7	λ_4, t_3	
λ_8		

Node 3		
AP	**FP**	**UP**
λ_1	λ_4, t_1	λ_3, t_2
λ_5		λ_2, t_3
λ_6		λ_7, t_3
λ_8		

Figure 3.10. States of resource pools at time t_6

Node 1		
AP	**FP**	**UP**
λ_4	λ_7, t_1	λ_2, t_1
λ_5	λ_8, t_1	
λ_6	λ_1, t_6	
	λ_3, t_6	

Node 2		
AP	**FP**	**UP**
λ_7	λ_5, t_2	λ_3, t_1
λ_8	λ_6, t_2	
	λ_4, t_3	
	λ_1, t_7	
	λ_2, t_7	

–––> Path

Node 3		
AP	**FP**	**UP**
λ_1	λ_4, t_1	λ_3, t_2
λ_5		λ_2, t_3
λ_6		λ_7, t_3
λ_8		

Figure 3.11. States of resource pools at time t_7

Node 1				Node 2				Node 3		
AP	**FP**	**UP**		**AP**	**FP**	**UP**		**AP**	**FP**	**UP**
λ_4	λ_7, t_1	λ_2, t_1		λ_7	λ_5, t_2	λ_3, t_1		λ_5	λ_4, t_1	λ_3, t_2
λ_5	λ_8, t_1			λ_8	λ_6, t_2			λ_6		λ_2, t_3
λ_6	λ_1, t_6				λ_4, t_3			λ_8		λ_7, t_3
	λ_3, t_6				λ_1, t_7		<––			λ_1, t_8
					λ_2, t_7		Resv			

Figure 3.12. States of resource pools at time t_8

Node 1				Node 2				Node 3		
AP	**FP**	**UP**		**AP**	**FP**	**UP**		**AP**	**FP**	**UP**
λ_4	λ_7, t_1	λ_2, t_1		λ_2	λ_5, t_2	λ_3, t_1		λ_5	λ_4, t_1	λ_3, t_2
λ_5	λ_8, t_1			λ_7	λ_6, t_2	λ_1, t_9		λ_6		λ_2, t_3
λ_6	λ_1, t_6			λ_8	λ_4, t_3			λ_8		λ_7, t_3
	λ_3, t_6		<––							λ_1, t_8
			Resv							

Figure 3.13. States of resource pools at time t_9

Node 1			Node 2			Node 3		
AP	**FP**	**UP**	**AP**	**FP**	**UP**	**AP**	**FP**	**UP**
λ_3	λ_7, t_1	λ_2, t_1	λ_2	λ_5, t_2	λ_3, t_1	λ_5	λ_4, t_1	λ_3, t_2
λ_4	λ_8, t_1	λ_1, t_{10}	λ_7	λ_6, t_2	λ_1, t_9	λ_6		λ_2, t_3
λ_5			λ_8	λ_4, t_3		λ_8		λ_7, t_3
λ_6								λ_1, t_8

Figure 3.14. States of resource pools after t_9

Operation Steps of label request signaling and flagging operation:

1- At time t_6, Node 1 initiates a label request signaling by sending the Path message. The two wavelengths having the smallest index numbers are suggested and they are transferred from AP to FP. The Path message contains the following concatenated Label Set:

Node 1	$[\lambda_1, \lambda_3]$

Figure 3.15. Concatenated Label Set sent at time t_6

2- At time t_7, Node 2 adds its Label Set to the Path message and sends it to downstream. The Path message contains the following concatenated Label Set:

Node 1	$[\lambda_1, \lambda_3]$
Node 2	$[\lambda_1, \lambda_2]$

Figure 3.16. Concatenated Label Set sent at time t_7

3- At time t_8, Node 3 (destination node) sends the prepared Resv message to upstream. The Resv message is prepared by intersecting the Label Sets of Node 1 and Node 2, which results in λ_1. The Resv message contains the following concatenated Label:

Node 1	$[\lambda_1]$
Node 2	$[\lambda_1]$

Figure 3.17. Concatenated Label sent at time t_8

4- At time t_9, Node 2 forwards the Resv message to upstream without any change. According to the received Resv message, Node 2 updates the pools; it puts the reserved wavelength λ_1 to UP, and λ_2 back to AP.

5- On reception of the Resv message, Node 1 will prepare the data and send it to downstream using the assigned wavelength λ_1.

3.4. Blocking Types in a GMPLS Network

As mentioned in Section 3.1.2, an LSP setup request may fail due to one of two blocking events, namely Forward-link Blocking and Backward-link Blocking. Forward-link Blocking happens while the label request signaling travels in the downstream direction. It is due to insufficient label resources and non-load balancing routing algorithms. Backward-link blocking occurs while the label response travels towards the source node. It is due to the conflict of label reservations, which means that the same label over the same link is selected for more than one connection request [6].

Label Set and label flagging are useful for reducing the blocking probability due to label collisions during lightpath establishment. In our developed label assignment method, we propose to not suggest the flagged labels, which will provide to completely get rid of the backward-link blocking. However, this may increase the forward-link blocking. By limiting the number of suggestible labels, the forward-link blocking may be decreased; but this method does not prevent the forward-link blocking completely. So, we have proposed to give some delay for the blocked label request signaling until the resources are set to be free. This means that the label request signaling will be waited at the node, until there is an available resource that can be suggested. So, we will not cancel or block the label request signaling. The forward-link blocking will be eliminated by postponing the signaling. This delay method is necessary for the networks where the data has to

be certainly transmitted, since the label request will not be blocked, but the transmission will be realized with some delay.

In the next chapter, we will give a model to simulate the proposed label assignment method. By means of the simulation, we will be able to examine the effect of label flagging operation on preventing the backward-link blocking and the effect of limiting the number of suggestible label on forward-link blocking. As mentioned in the previous paragraph, we do not block the label request signaling, but we give some delay, until there is an available resource that can be suggested. Forward-link blocking will increase because of the proposed method to prevent backward-link blocking. Since forward-link blocking is get rid of by postponing the label request signaling, there will be a trade-off between the total delay amount of label request signaling and backward-link blocking. Therefore, in the simulation the network performance will be evaluated according to signaling delays and backward-link blocking probability.

Before passing the next chapter, in Section 3.5 we want to give brief information on the use of concatenated Label Set/Label to provide integrity between the signaling and data transmission parts.

3.5. Use of Concatenated Label Structure

As mentioned in Chapters 1 and 3, we propose a concatenated structure for both Label Set and Label. This proposed structure has two important aspects. First one is related with resource assignment and management, while the second one is completely related with all-optical label switching. It is thought that such a concatenation may provide the following functions:

1- If concatenated Label Set is used in the label request signaling, the state information of all network resources will be transmitted up to the destination without having any information loss at the intermediate nodes. This has many advantages for resource assignment and management.

2- If the labels used through the entire path can be formed at once at the signaling step, this could help to realize all-optical label switching during the data transmission. Since the concatenated label will include all reserved labels, this concatenated label will be converted to optical domain only once, and will be conveyed all-optically at the rest of transmission.

First advantage of concatenation, which is related with resource assignment and management, has been explained in detail in the previous sections of this chapter. Moreover, a simulator has been designed to verify the proposed label assignment method, which uses concatenated Label Set and Label structures. Second advantage of concatenation is also very important, since it may provide to transport the data fully in the optical domain. The importance of all-optical transmission and how this concatenated label can be used to provide all-optical label switching is explained in the following section.

3.5.1 All-Optical Label Switching

To meet the enormous traffic demand of today's networks, optical networks are preferred because of their high capacity. As well as providing large capacity, optical networks has the advantage of fast transmission provisioning [5]. If the label recognition can be done in optical domain without making any optical to electrical conversion, switching speed will increase considerably.

After the signaling session is completed, data transmission starts. This step is composed of three different parts:

1- Initiating a data transmission by the source node,

2- Transmitting an incoming data attached with a label to downstream by making the required processes for label switching,

3- Ending the data transmission by the destination node and obtaining the data by removing the label.

All-optical label switching is required at the second part of data transmission. When a node receives the data attached with a label, the node has to make the switching, this means that the node will take the incoming optical data and will send it with the label assigned to itself to downstream. One way of realizing all-optical label switching may be using Optical Orthogonal Codes (OOCs) to encode/decode label information in the optical domain. If concatenated label structure can be represented with a sequence of OOCs and the nodes have the capability of decoding the incoming label structure, then the node will be able to learn the related part of the concatenated label in the optical domain. The node only will decode the incoming label by the help of optical correlators and get the label assigned to itself. Since the label belonging to a node will be coded with the specific code for that node, the node will be able only to decode the related part of the concatenated label.

In Table 3.1., the concatenated label structure represented by OOCs is given. Each part in the concatenated structure, which includes Node ID of a specific node and the label assigned to that node, will be coded separately with different OOCs. So, each OOC will symbolize a node.

Table 3.1. Match Table of OOCs with labels

OOC	Content represented by the OOC	
OOCj	Node ID (N_j)	Label assigned to First Intermediate Node
OOCk	Node ID (N_k)	Label assigned to Second Intermediate Node
	. . .	. . .
OOCn	Node ID (N_n)	Label assigned to Last Intermediate Node

Most common method of designing optical encoders and decoders is based on power splitting and combining. An optical encoder/decoder can be constructed by 1xw power splitters, wx1 combiners and fiber delay lines where w is the weight of the OOCs [27]. The weight of an OOC is the number of 1's used in the codeword. The encoders/decoders, which are designed using this approach, are shown in Figure 3.17. [27].

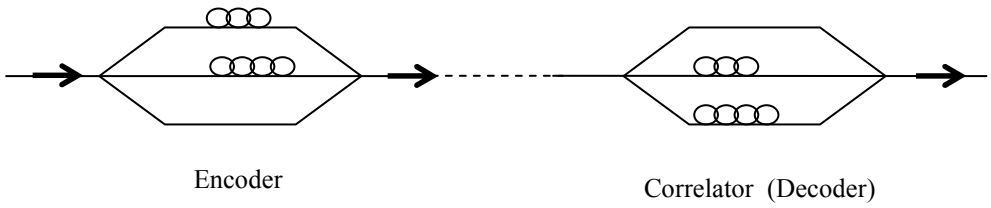

Figure 3.18. Optical encoder and decoder

Optical orthogonal code design and the implementation of such a network working with OOCs is out of scope of this thesis study. Related information on OOC design concept can be found in [27].

Although some techniques are developing for all-optical label switching, there exist still other challenges, e.g. optical data storage at intermediate nodes while switching is being done. Studies continue to developed buffers enabling optical data storage. Optical data storage is another concept that needs further investigations. Optical data storage subject is out of scope of this thesis. We may assume that we can buffer the optical data while the switching is done. If the network is not capable of storing the optical data, then the data will be converted to electrical domain and stored electrically until the switching job is completed. Especially for the cases of lambda conversion, the processing at an intermediate

node will take more time. Hence, optical to electrical conversion of the data seems unavoidable with today's technology. Optical buffering and other existing challenges of optical networks will assist in the realization of next generation of optical networks [29].

As a conclusion, label concatenation is proposed just to give an idea for an applicable all-optical label switching technique, the network structure and the techniques required for all-optical networking are not studied in this thesis study. If the reader is interested in all-optical networking, the related information can be obtained from [28], [29].

CHAPTER 4

SIMULATION AND EVALUATION OF THE LABEL ASSIGNMENT METHOD

In this chapter, we will introduce the network simulator that is developed for the study involved in this thesis work to support our theoretical ideas in a simulation environment. We wanted to see whether the proposed label assignment method works properly. We also aimed to evaluate the effect of this method on the network performance parameters, such as blocking probability.

Here, a brief description of the simulator will be given, the details of the simulator and simulation results will be explained in the subsections of this chapter. The simulator has a user friendly Graphical User Interface, where the user selects some network parameters and creates the network traffic being simulated. The simulator is developed by using C++ programming language. The network topology of the simulator is fixed and it is designed to model an optical lambda switch capable GMPLS network. According to the input traffic load and the selected parameters, the simulator realizes the required signaling (label request and label assignment) and the data transmission automatically. While the signaling and data transfer occur, the contents of resource pools along the LSP will change and the user will be able to track the changes visually by the help of the bars representing AP, FP and UP for each link of the network. In addition to the visual bars, there is also an event screen and a result screen on the GUI. The signaling and data transfer steps

are displayed on the event screen in a time order. After the simulation is completed, the results of the simulation are shown on the result screen.

4.1. Simulator Overview

The simulator is designed mainly to simulate the label assignment method proposed in Section 3.3. The simulation is realized with a predefined network topology and some presumed conditions, which are described in the following sections in detail.

4.1.1. Network Topology

The lambda switch capable GMPLS network is simulated by a fixed 10-node network, which is shown in Figure 4.1.

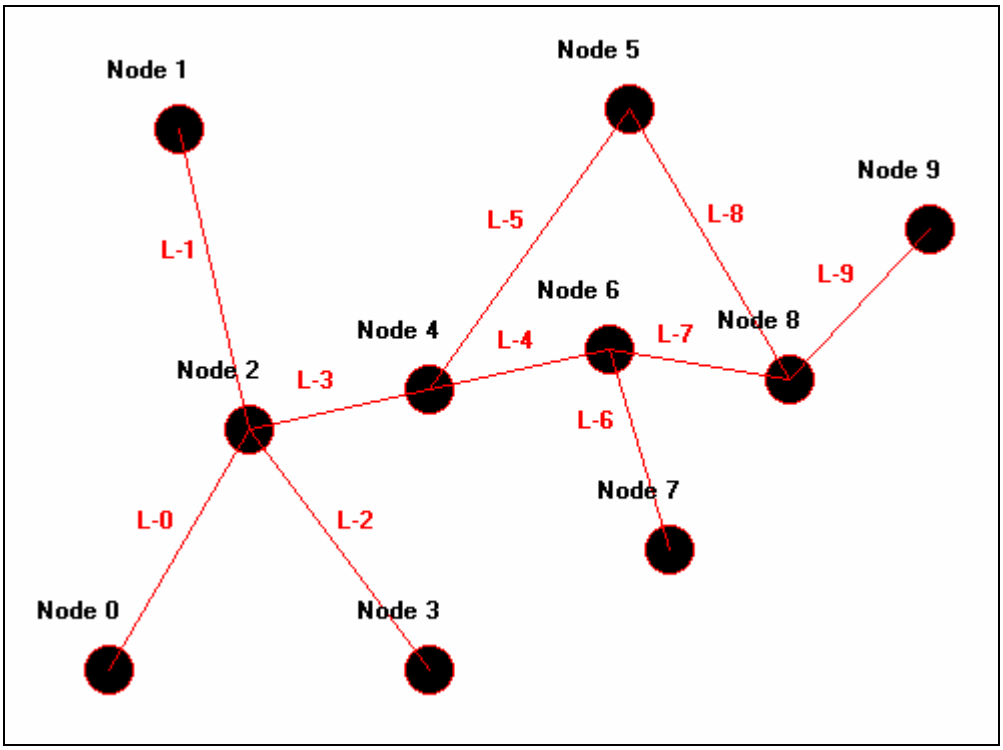

Figure 4.1. Simulator network topology

In Figure 4.1., we see that the total number of the links is 10. The nodes and links are labeled with the index numbers from 0 to 9. For the network, we assume the following conditions:

1- The link distances between the nodes are equal for the whole network.

2- There are 8 wavelengths per link.

3- Each node in the network may represent a source, an intermediate or a destination node.

4- The nodes are composed of OXCs, which are capable of wavelength conversion.

4.1.2. System Parameters

The following parameters are assumed and used to construct the simulator:

1- The sum of message processing time at a node and transmission time between two nodes is equal to τ (Tao), which is represented in the simulator by unit time 1.

2- Connection requests may be generated in two ways, namely manual or automatic:

   a- *Manual Request Generation*: The request number to be simulated is left to the user. For each event, the user selects the source-destination pair and enters the request arrival time as an integer multiple of τ.

   b- *Automatic Request Generation*: Connection requests arrive to each node according to a Poisson process with rate λ requests per unit time. The simulator calculates request rate λ according to the desired number of connection requests and the simulation time as an integer

multiple of τ. The number of requests and the simulation time are determined by the user.

3- Traffic over each source-destination node pair depends on the probability of choosing the source-destination pair. How the source-destination pair is selected is explained in Section 4.1.2.1.

4- Fixed routing is used in simulations in order to simplify the problem. Therefore, the routing table is fixed in the program. Routing table is given in Section 4.1.2.3.

5- TOT_{FP} and TOT_{UP} have constant values that are integer multiples of τ. TOTs can be adjusted in the program. Default values for TOTs are: $TOT_{FP} = 50\tau$, $TOT_{UP} = 50\tau$.

6- The timing of the simulator is such that time is incremented by τ. Hence, the local timestamps are integer multiples of τ.

7- The maximum number of suggested wavelengths can be selected by the user. It can be any number between 1 and 8.

8- All wavelengths are indexed. The wavelength assignment is done according to the label assignment method mentioned in Chapter 3. If there is more than one available wavelength, then the first available wavelength is chosen, i.e. the wavelength whose index number is the smallest.

4.1.2.1. Choosing Source-Destination Node Pair

If manual request generation is used for simulation, then the user determines the traffic density over the network by selecting the source-destination pairs. If automatic request generation is used, then the source and destination nodes are selected automatically by the software.

The source and destination are selected in a manner that the distribution of the selections can be uniform or non-uniform according to the assigned probabilities to each node. This provides flexibility while modeling the network. In some real-life networks, certain nodes have heavy traffic, which means that most of the network traffic originates from and goes to these nodes. To simulate this situation, these nodes can be assigned higher probabilities than the others, which will result in more traffic originating from and going to these nodes.

In our design, there are 10 nodes. To assign the probabilities to the nodes, we reserve a number interval for each node in an ordered way, where the first number of the first node is 1, and the last number of the last node is 100. To make the choice, a random number between 1 and 100 is generated, and the obtained number is compared with the number intervals of the nodes. We search the interval, which the number is belonging to. In this way, we determine a node. The source and destination nodes are selected separately according to this algorithm. If the destination comes out the same with the source node, then the choice of destination is tried again, until the source and destination node numbers are distinct.

Here, we provide an example for node selection. The intervals assigned to the nodes are given, where N_i denotes the node ID:

{N_0: 1-5, N_1: 6-10, N_2: 11-20, N_3: 21-40, N_4: 41-45, N_5: 46-55, N_6: 56-70, N_7: 71-75, N_8: 76-85, N_9: 85-100}.

For this input, the selection probabilities of the nodes become:

P_i = {N_0: 0.05, N_1: 0.05, N_2: 0.1, N_3: 0.2, N_4: 0.05, N_5: 0.1, N_6: 0.15, N_7: 0.05, N_8: 0.1, N_9: 0.15}.

For example, let the generated random number be 30, then the node labeled with N_3 will be selected.

4.1.2.2. Modeling Poisson Arrivals

The simulator generates Poisson distributed event requests when the user selects the automatic request option.

The number of request arrivals in a time interval is Poisson distributed with rate λ requests per unit time. The rate λ is equal to k/T, where k is the total number of occurred requests in the time interval T. Poisson distributed event generation is given in Appendix A in detail.

According to the information given in [30], the inter-arrival times of a Poisson distributed request arrivals has an exponential distribution defined by

$$f(t) = \lambda . e^{-\lambda . t} \tag{4.1}$$

Given a sequence of discrete events occurring at times t_0, t_1, t_2, t_3,..., the intervals between successive events are $\Delta t_1 = (t_1 - t_0)$, $\Delta t_2 = (t_2 - t_1)$, ..., $\Delta t_i = (t_i - t_{i-1})$ If we calculate the inter-arrival times (Δt_i), then we will find the request arrival instances based on the inter-arrival times. For example, the time instant of the first arrival will be equal to $(t_0 + \Delta t_1)$ where t_0 the initial time of the time interval T.

To produce exponentially distributed inter-arrival time values the following formula can be used [30].

$$t = -\frac{\ln u}{\lambda} \tag{4.2}$$

In (4.2), u is a random number drawn from the uniform distribution in the interval (0, 1]. *u* can be obtained by using a random number generator on a computer.

In the simulation program, we have used (4.2) to find the inter-arrival times of the requests, then we have calculated the time instances of the requests based on the initial time of the time interval T and calculated inter-arrival times.

4.1.2.3. Routing Scheme

Routing is used to find a route for connections between a source-destination pair. Fixed routing is one of the well-known routing algorithms. In fixed routing, there is a single possible route for each pair of network nodes. Any connection between a pair of source and destination nodes uses this route [3]. Fixed shortest path routing is a preferable type of fixed routing algorithm. When applied in the network, it gives the shortest path in the sense that smallest number of links from a source node to a destination node is determined. All such paths are stored in a look-up table.

In the simulation software, we have created a routing table based on the fixed shortest path routing. Since same network topology is used for all simulation runs, the following routing table, which is given in Table 4.1., is used.

Table 4.1. Fixed Routing Table

		To Destination Node									
		N_0	**N_1**	**N_2**	**N_3**	**N_4**	**N_5**	**N_6**	**N_7**	**N_8**	**N_9**
From Source Node	**N_0**	-	N_2	N_2	N_2	N_2	N_2	N_2	N_2	N_2	N_2
	N_1	N_2	-	N_2	N_2	N_2	N_2	N_2	N_2	N_2	N_2
	N_2	N_0	N_1	-	N_3	N_4	N_4	N_4	N_4	N_4	N_4
	N_3	N_2	N_2	N_2	-	N_2	N_2	N_2	N_2	N_2	N_2
	N_4	N_2	N_2	N_2	N_2	-	N_5	N_6	N_6	N_6	N_6
	N_5	N_4	N_4	N_4	N_4	N_4	-	N_8	N_8	N_8	N_8
	N_6	N_4	N_4	N_4	N_4	N_4	N_8	-	N_7	N_8	N_8
	N_7	N_6	N_6	N_6	N_6	N_6	N_6	N_6	-	N_6	N_6
	N_8	N_6	N_6	N_6	N_6	N_6	N_5	N_6	N_6	-	N_9
	N_9	N_8	N_8	N_8	N_8	N_8	N_8	N_8	N_8	N_8	-

In Table 4.1., the node IDs are denoted by N_i, e.g. Node 1 is expressed by N_1. The first column includes the source nodes and the first line includes the destination nodes. The resulting cell of the intersection of a source and a destination gives the next node in the path. So, a path for a connection from a source node to a destination node is computed as follows. For example, for a connection from source node N_0 to destination node N_9 the next node after the source node is N_2. Then the following node is computed by selecting N_2 as the source node in the table. The resulting next node becomes N_4. If continued in this way the following path is obtained for a connection from source node N_0 to destination node N_9:

$\{N_0 \text{--}\!> N_2 \text{--}\!> N_4 \text{--}\!> N_6 \text{--}\!> N_8 \text{--}\!> N_9\}$.

4.1.3. Simulation Algorithm

The simulation algorithm is constructed to implement the label assignment method given in Section 3.3. Since the resource assignment and management is aimed to be simulated for different cases, it is wanted that the routing scheme is same for all cases. Therefore, for each simulation the same routing table, which is given in Table 4.1., is used.

The simulation program is composed of three main parts: Input Block, Main Program, and Output Block.

4.1.3.1. Input Block

Before starting a simulation, the simulation inputs have to be entered. The simulation inputs can be divided into two groups.

1- *Inputs adjusted before program compilation*: This type of inputs is composed of some system parameter settings that will affect the simulation algorithm. These inputs are adjusted in the program code, before the

simulation program is compiled. The following inputs are included in this group:

a- *Resource Pools Setting*: Flagged Pool (FP) is included or excluded in the resource pool configuration. When FP is included, label flagging operation is made according to the proposed label assignment method. When FP is excluded, then label assignment is realized as declared in the GMPLS standards. FP is a simple boolean parameter that is set as true or false. If it is true, then FP is included in the resource pool configuration.

b- *Probability Setting of Nodes in Source-Destination Pair Selection*: To adjust the probabilities of the nodes, a number interval is assigned for each node in an ordered way, where the first number of the first node is 1, and the last number of the last node is 100. To implement the probability assignment a priority array of size 10 is used. To have the probability set P_i, the entries of the array are adjusted as given in the set A_i. The sets P_i and $_{Ai}$ are given below.

P_i = {N_0: 0.05, N_1: 0.05, N_2: 0.1, N_3: 0.2, N_4: 0.05, N_5: 0.1, N_6: 0.15, N_7: 0.05, N_8: 0.1, N_9: 0.15}

A_i = {5, 10, 20, 40, 45, 55, 70, 75, 85, 100}.

c- *TOT Setting*: TOT_{FP} and TOT_{UP} are defined as constants in the program. They are set to values that are integer multiples of τ.

2- *Inputs entered during program execution*: These inputs are entered via the simulator GUI during the program execution. This type of inputs determines the traffic load to be simulated and also the maximum size of the Label Set that effects the simulation results directly. The inputs included in this group can be given as follows:

a- *Request Generation:* Transmission requests may be generated in two ways, namely manual or automatic, as mentioned in Section 4.1.2.

   i- *Manual Request Generation*: To generate transmission request manually, the user selects the source-destination pair and enters the request arrival time as an integer multiple of τ. Then, the user adds this request as an event. All events to be simulated are added in this way. The list of added events is shown in the event screen.

   ii- *Automatic Request Generation*: To generate transmission requests automatically, the user enters the number of events and the time interval of event generation. The time interval can be entered as an integer multiple of τ. In the program, τ is represented by unit time. The simulator generates events according to a Poisson process with rate λ requests per unit time. The rate λ is equal to k/T, where k is the total number of events occurring in the time interval T. In the simulation program, the inter-arrival times of the events are found by using (4.2), then the time instances of the events are calculated based on the initial time of the time interval T and calculated inter-arrival times. Initial time is assumed to be 0.

b- *Maximum size of Label Set:* User can set the maximum size of the Label Set from 1 to 8. This parameter affects the simulation results directly, since it affects the number of Path messages that can be sent from a node at the same time. If all available labels are suggested for a transmission request, i.e. maximum size of Label Set is 8, then the incoming Path messages will be delayed until the suggested labels are set as available.

4.1.3.2. Main Program

In the main program, there are two main structures, which are link structure and node structure. Link structure is used for resource pool management and node structure is used for the main works including label request, label reservation and data transmission. Besides that, there is a timer in the program and all works done in the program are dependent on this timer. Timer is initially set to 0. When the program starts, the timer is started. The sum of message processing time at a node and transmission time between two nodes is assumed to be τ (Tao), which is represented in the simulator by unit time 1. So, after each operation the timer is incremented by 1.

1- *Link Structure:* To simplify the problem of resource pool management, the pool configurations are arranged and updated in the link structure. To implement the resource pools in this way is easier, since transmission requests may travel along a link in both directions, namely from left to right and from right to left. For example, let Link 1 connect Node1 and Node 2, then Link 1 will be used for the transmissions in the direction from Node 1 to Node 2 and also for the transmissions from Node 2 to Node1. So, matching the resource pools with the links will simplify obtaining the information of resource states during the program execution. The program components related with link structure are given below.

   a- *Number of Wavelengths*: The number of wavelengths supported at a link has a constant value in the program. In this simulation, there are 8 wavelengths per link.

   b- *Resource Pool Configuration:* An array structure is used to keep the resource state information. For each resource, i.e. wavelength, four parameters are assigned, which are pool type, message ID, local timestamp and TOT. The parameters have different meanings according to the pool type, i.e. AP, FP and UP. For example, if a

wavelength is in the FP then TOT parameter refers to TOT_{FP}. Detailed information on these parameters is given in Chapter 3. In Table 4.2., an example is given to show how for the resource state information is kept. Initially all resources are in the AP. During the program execution the resource parameters are updated dynamically.

Table 4.2. Resource State Table

Resources	Pool Type	Message ID	Local Timestamp	TOT
λ_1	UP	1	t_5	TOT_{UP}
λ_2	UP	2	t_6	TOT_{UP}
λ_3	FP	5	t_4	TOT_{FP}
λ_4	FP	4	t_6	TOT_{FP}
λ_5	FP	7	t_8	TOT_{FP}
λ_6	AP	-	-	-
λ_7	AP	-	-	-
λ_8	AP	-	-	-

2- *Node Structure:* The main works including label request, label reservation and data transmission are carried out in the node structure. Each node has event buffers, where the events are kept. After the input block is prepared, the simulation program has all related information on the added events. Based on this information, all events are put into the buffers of the nodes that will be source node for a transmission. When simulation starts, each node make required process and updates its buffer according to the process results. The program components related with node structure are given below.

a- *Routing Table:* For each transmission there is a path to be traveled in both signaling and data transmission sessions. Each node uses the routing table to decide the next node after making the related operation. Operation may be label request, label reservation or data transmission.

b- *Event Buffer:* Event buffers are initially empty. With the start of simulation event buffers are filled with the related information and the nodes make operation according to these information. After each operation, the buffers are updated. The number of event buffers at each node depends on the number of events that will be processed at that node. When a node have a task to do with an event, i.e. being source node, intermediate node or destination node for the specified event, then the node has to have a buffer for this event. The maximum number of the event buffers is set to 100 in the program. The structure of an event buffer is given in the Table 4.3. The size of the event buffer is 50 byte. Each field of the buffer uses a field of 1 byte. The contents of the buffer are numbered according to the order of the content fields. An event buffer has two blocks: header block and data block. Header block includes the general information for an event and its size is fixed. The data block includes the node IDs and the data belonging to that node. The size of data block depends on the node number along the specified path. Data type field is set to Path, Resv or Data according to the current operation for the event. Based on the data type, the field keeping the data of the node includes either Label Set or Label. If data type is Path, then Label Set is included in this field. If data type is Resv or Data, then Label is included. Since data block has a variable size, the empty fields in the buffer are filled with the integer –1 to indicate the end of the data block.

Table 4.3. Structure of an Event Buffer

Block Type	Field Number	Content Type
Header Block	1	Message ID
	2	Source Node ID
	3	Destination Node ID
	4	Data Type
	5	Current Time
Data Block	6	Node ID of the source node
	7	Data of this Node
	8	Node ID of the first intermediate node
	9	Data of this Node
	.	.
	.	.
	.	.
	i	Node ID of the last intermediate node
	i+1	Data of this Node
	i+2	End of the Buffer (-1)
	i+3	Empty (-1)
	.	.
	.	.
	.	.
	50	Empty (-1)

According to the information given on main program components, the main program algorithm can be explained in the following steps.

1- Before the simulation starts the initial states of the program components are as follows:

   a- Current time is equal to 0.

   b- All resources of each link are in the AP.

   c- The event buffers of the source nodes are filled with the information obtained from the Input Block.

2- When the simulation starts, the timer is started. The timer is incremented by 1, until there is an operation at any node. At each time increment, each node controls whether there is an operation to be done at this node. If there is any, then the node does the required operation and the time is incremented by 1. The operation may be

   a- Sending a Path or Data message or receiving a Resv message, if the node is the source node.

   b- Transmitting a Path, Resv or Data message, if the node is an intermediate node.

   c- Receiving a Resv or Data message or receiving a Path message if the node is the destination node.

3- At the beginning of the program, each node controls, whether a Path message will be sent or not. If a node will send a Path message, the node prepares its Label Set. Since all resources are in the AP, the node forms the Label Set according to the required number of resources. The number is equal to the maximum size of the Label Set. The node adds its Label Set into the event buffer attached with its node ID. Then the node finds the

next node using the routing table, so the link to be used is determined. When the node sends the Path message,

a- The current timer is incremented by 1.

b- The event buffer of the next node is updated with the information contained in this buffer. This means that the next node gets all information on this event and learns the operation to be done.

c- The resource state information of the used link is updated.

4- Since the simulation has started, there may be any operation. Therefore, each node controls its event buffers continuously. The operation to be done depends on both the data type field of the event buffer and the type of the node for this operation, i.e. source node, intermediate node or destination node. When the node makes the require operation, it updates its event buffer, then current timer is incremented by 1 and the event buffer of the next node is updated. At this time, the resource pool states are also updated. The resource pools are updated according to the Table 4.4., if FP is included in the pool configuration. If FP is excluded in the pool configuration, then the update of the resource pools is done according to the given information in Table 4.5.

5- During the program execution, each operation done at the nodes are shown on the event screen to inform the user. Also, the bar chart shows the resource pool states at each link. Both of these outputs are shown while the program is running.

6- When all events are processed, this means that all data transmissions are completed, the program gives the numerical results of the simulation, i.e. number of blocked events, number of lambda conversions etc.

Table 4.4. Transfer scheme between the resource pools when FP is included

	Transfer Type	Transfer Reason
1	from AP to FP	When the Path message is sent to downstream.
2	from FP to UP	On reception of the Resv message, if the specified label is reserved.
3	from UP to AP	When the data is passed to downstream or
		When the wait duration in the UP is expired ($t_{current} - t_i = TOT_{UP}$).
4	from FP to AP	On reception of the Resv message, if the specified label is not reserved or
		When the wait duration in the FP is expired ($t_{current} - t_i = TOT_{FP}$).

Table 4.5. Transfer scheme between the resource pools when FP is excluded

	Transfer Type	Transfer Reason
1	from AP to UP	On reception of the Resv message, if the specified label is reserved.
2	from UP to AP	When the data is passed to downstream or
		When the wait duration in the UP is expired ($t_{current} - t_i = TOT_{UP}$).

4.1.3.1. Output Block

There are three types of simulation outputs: Informative Output, Graphical Output, and Numerical Output.

1- *Informative Output*: During the program execution, all operations made at the nodes are shown in the event screen. Each operation is written in a different line. The user can follow the operation steps for the simulated event set on the event screen.

2- *Graphical Output*: During the program execution, the bar chart shows the resource pool states at each link. In the bar chart field there are 3 bars for each link, which represent AP, FP and UP. Initially all resources are in the AP, so AP bar is full, and the other bars are empty. During the simulation, the status of bars is changed according to the amount of the resources at the pools. When FP is excluded in the pool configuration, only AP and UP bars change their status, the bar for FP is always empty.

3- *Numerical Output*: When all events are processed, this means that all data transmissions are completed; the program gives the numerical results of the simulation. The numerical outputs are shown on the result screen. The following outputs are given as the simulation results:

   a- *Number of blocked events due to the backward-link blocking*: This output is calculated by counting the number of the events that have faced with blocking at least one time. Here, blocking means that a lambda is detected in use, as a transmission has decided to use that lambda.

   b- *Amount of delay due to the forward-link blocking*: The sum of message processing time at a node and transmission time between two nodes is assumed to be τ (Tao), which is represented in the simulator by unit time 1, and after each operation the timer is incremented by 1.

Therefore, a label request message, namely Path message is delayed at a node by unit time, when there is no available resource to be suggested. A unit time after, the available resources are controlled and if there is any available resource, the Path message is sent, if there is not then Path message wait by unit time again. This process is continued until there is any available resource to be suggested by the specified Path message. The delay amount for a Path message is calculated by summing the wait times for this message. The total delay amount is found by summing the calculated wait times of all Path messages that are delayed. Hence, the total delay amount is not directly added to the simulation time, but it includes all wait times.

c- *Number of lambda conversions done during the data transmission*: This output is calculated at the destination node. Since concatenated Label Set is used, the destination has the information on the available lambdas at each link. So, using this information the destination node decides the labels to be used for each node. Since lambda conversion means that the label will change at a node and the destination node has the information of all labels for a LSP, the destination determines automatically, whether there will be a lambda conversion or not. While deciding the labels, it is aimed that there will be minimum lambda conversion.

4.3. Simulation Results and Evaluation

The simulations are carried out:

1- To compare the proposed label assignment method with the standard method.

2- To investigate the effect of the maximum size of Label Set.

The simulations are realized under the following conditions. These are inputs that are adjusted before the program compilation.

1- Source-destination selection probabilities:

   P_i = {N_0: 0.2, N_1: 0.1, N_2: 0.1, N_3: 0.05, N_4: 0.1, N_5: 0.05, N_6: 0.15, N_7: 0.1, N_8: 0.05, N_9: 0.1}

2- $TOT_{FP} = 50\tau$, $TOT_{UP}=50\tau$.

3- Time interval of event generation is set to 100 τ.

The program has been run 400 times with different combination of the following inputs. So, each combination has been run 10 times.

1- Resource Pool Setting: Both resource pool configurations including FP and excluding FP have been simulated.

2- Number of Events: The program has been run for 20, 40, 60, 80 and 100 events. The events are generated automatically.

3- Maximum Size of Label Set: The program has been run for 2, 4, 6 and 8 suggested labels.

To see the effect of proposed label assignment method on backward-link blocking, two graphs, namely Figure 4.2. and Figure 4.3., are drawn based on the obtained simulation data. In Figure 4.2., we see the number of blocked events changing with the number of simulated events, namely transmission requests, and in Figure 4.3, we show the variation of backward-link probability according to the number of events. The simulations are done for both the case with FP and the case without FP. The software has been run for different values of suggestible lambda numbers. As seen in the graph, the backward-link is completely prevented, when FP is used. This verifies the effect of proposed label assignment method on preventing backward-link blocking. Also, it is obviously seen that the blocking probability

depends strongly on the traffic load in the network. This result verifies the reliability of our simulation tool.

As we have mentioned before, the proposed method will result in an increase of forward-link blocking. However, we get rid of forward-link blocking by delaying the label request signaling, until there is any available resource. Therefore, to show the penalty of preventing backward-link blocking, we also show the total delay amount of label request signaling, which is given in Figure 4.4. The delay distribution is only given for the case with FP, since it is a result of using proposed label assignment method.

Before Figures 4.2, 4.3 and 4.4., we give the simulation data used to draw the related graph. The simulation data is obtained by taking the average of simulation results that come out at the simulation runs of each combination. The combinations of parameters used during the simulation have been explained in the previous paragraphs.

The effect of limiting the number of suggestible label is also investigated for both backward-link blocking simulations and total delay amount of label request signaling simulations. To do this, maximum size of label set is set to the values 2, 4, 6 and 8 at different times, and the simulation results are shown in 3 graphs for each value. First graph shows the variation of backward-link blocking probability, second graph shows the number of blocked events obtained by simulations and third graph shows the total delay amount of label request signaling. The Figures 4.5, 4.6 and 4.7 show the results, when label set size is 2; the Figures 4.8, 4.9 and 4.10 show the results, when label set size is 4; the Figures 4.11, 4.12 and 4.13 show the results, when label set size is 6; and the Figures 4.14, 4.15 and 4.16 show the results, when label set size is 8. As seen in the Figures, the total delay amount increases when the number of lambdas suggested, i.e. maximum size of label set, is increased. This is an expected result as declared in Chapter 3.

Table 4.6. Simulation Data for Number of Blocked Events

		Number of Events				
		20	40	60	80	100
Without FP	Max Size of Label Set=2	2	8,9	16,7	28,7	39
	Max Size of Label Set=4	3,1	9,2	17,5	30	41,8
	Max Size of Label Set=6	3,4	10,1	18,4	31,3	47,1
	Max Size of Label Set=8	2,8	7,9	16,3	26,5	42,9
With FP	Max Size of Label Set=2	0	0	0	0	0
	Max Size of Label Set=4	0	0	0	0	0
	Max Size of Label Set=6	0	0	0	0	0
	Max Size of Label Set=8	0	0	0	0	0

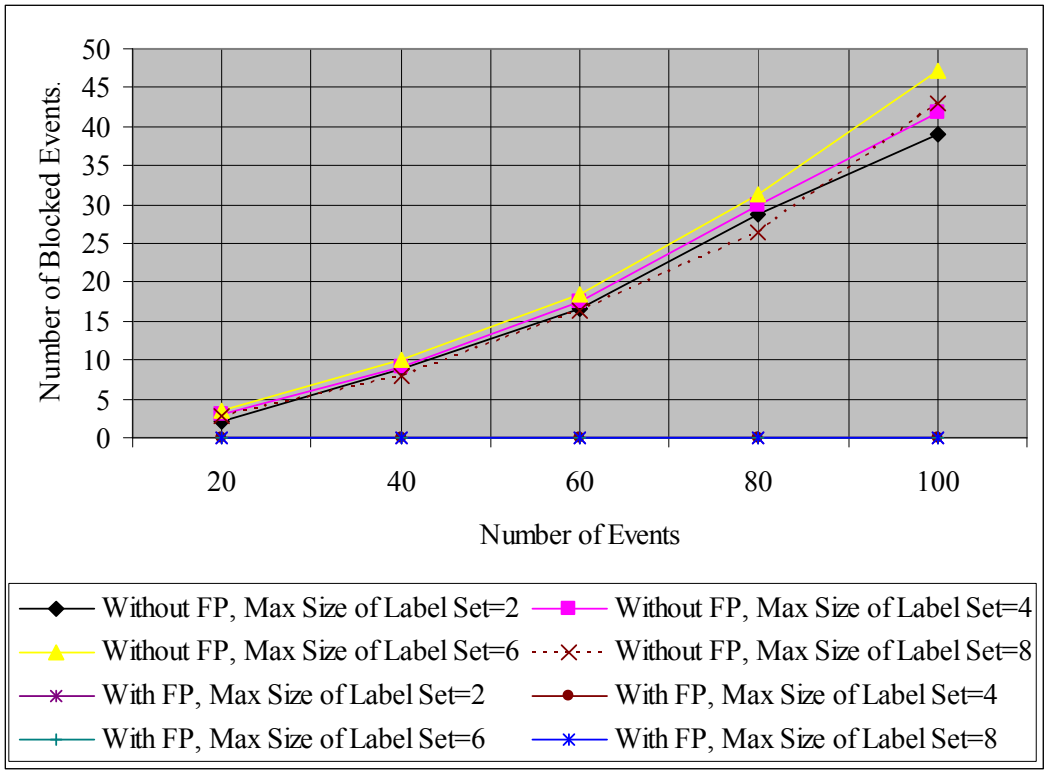

Figure 4.2. Number of Blocked Events vs. Number of Events

Table 4.7. Simulation Data for Backward-link Blocking Probability Results

		Number of Events				
		20	40	60	80	100
Without FP	Max Size of Label Set=2	0,1	0,2225	0,2784	0,35875	0,39
	Max Size of Label Set=4	0,155	0,23	0,2917	0,375	0,418
	Max Size of Label Set=6	0,17	0,2525	0,3067	0,39125	0,471
	Max Size of Label Set=8	0,14	0,1975	0,2717	0,33125	0,429
With FP	Max Size of Label Set=2	0	0	0	0	0
	Max Size of Label Set=4	0	0	0	0	0
	Max Size of Label Set=6	0	0	0	0	0
	Max Size of Label Set=8	0	0	0	0	0

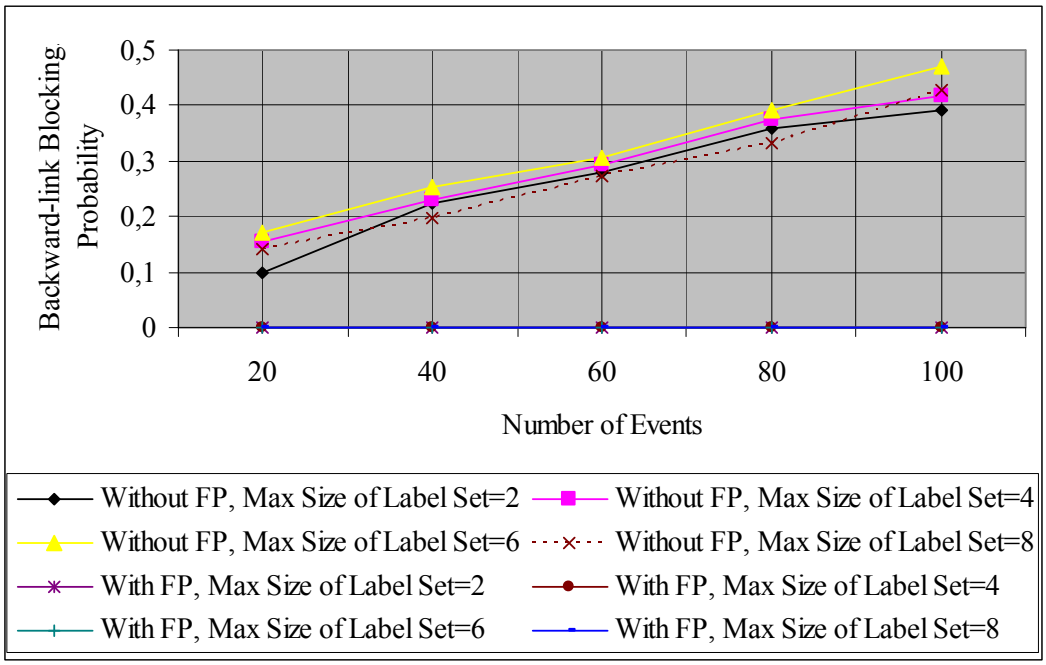

Figure 4.3. Backward-link Blocking Probability vs. Number of Events

Table 4.8. Simulation Data for Total Delay Distribution

	Number of Events				
	20	40	60	80	100
Max Size of Label Set=2	0	0,3	6,1	35,8	76,9
Max Size of Label Set=4	5,3	26,6	99,6	341,5	385,7
Max Size of Label Set=6	5,6	33	107,9	281,5	2444,8
Max Size of Label Set=8	38,9	1321,1	3269,4	4430,1	7415

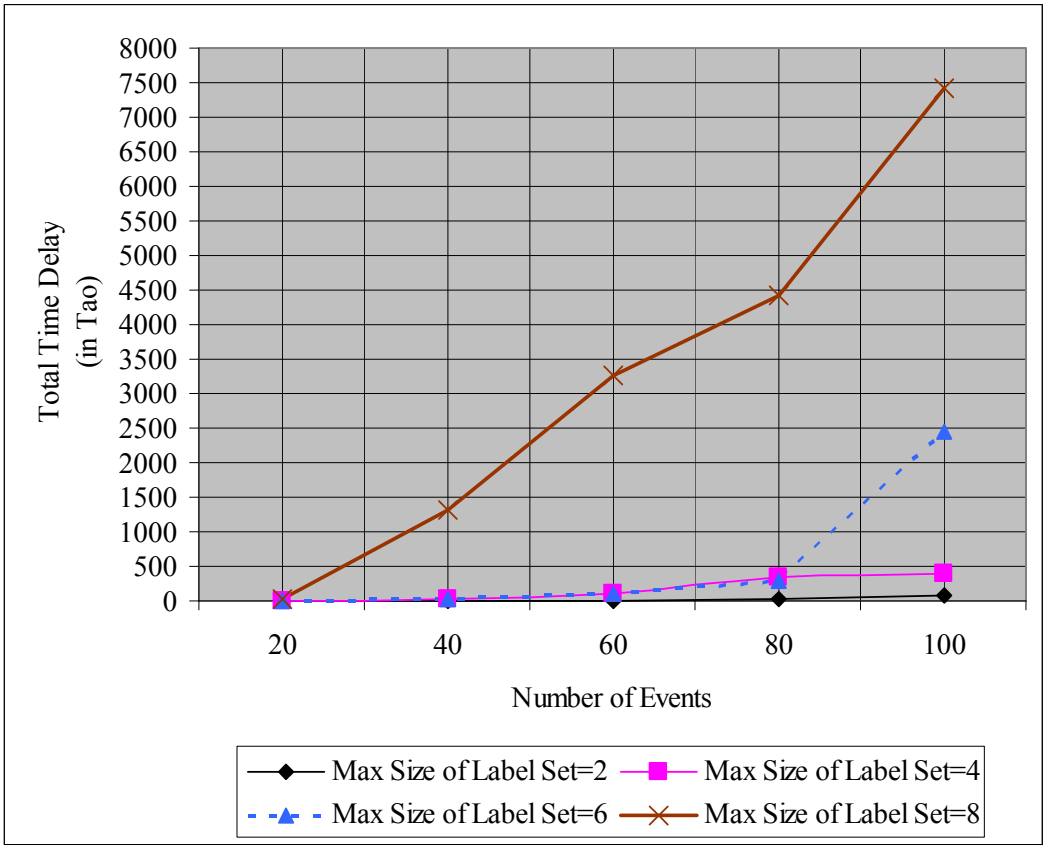

Figure 4.4. Total Time Delay vs. Number of Events

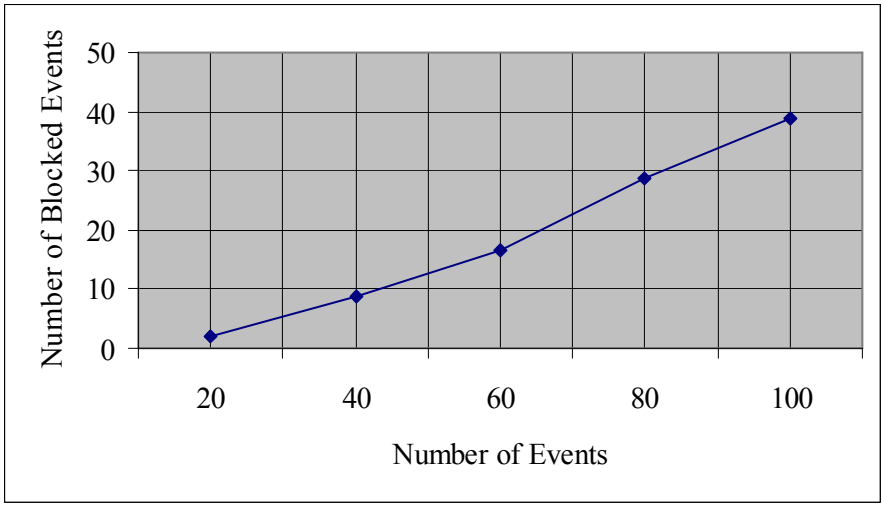

Figure 4.5. Number of Blocked Events vs. Number of Events; Maximum Size of Label Set=2; without FP

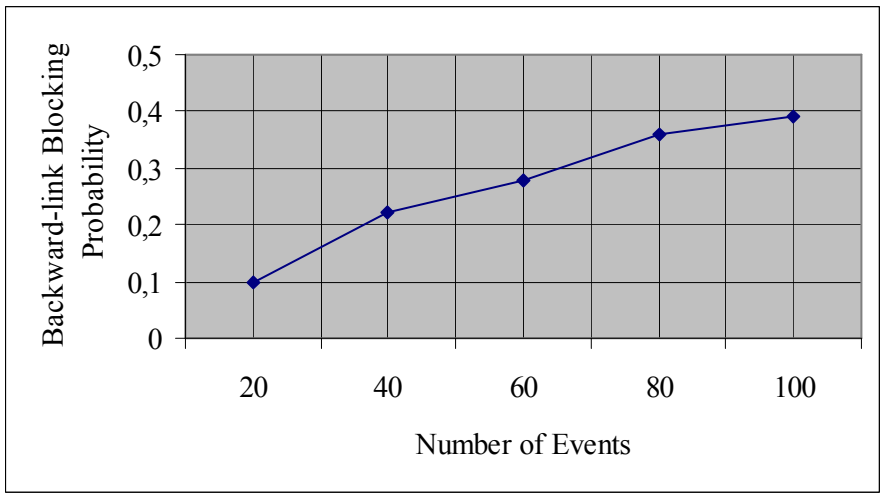

Figure 4.6. Backward-link Blocking vs. Number of Events; Maximum Size of Label Set=2; without FP

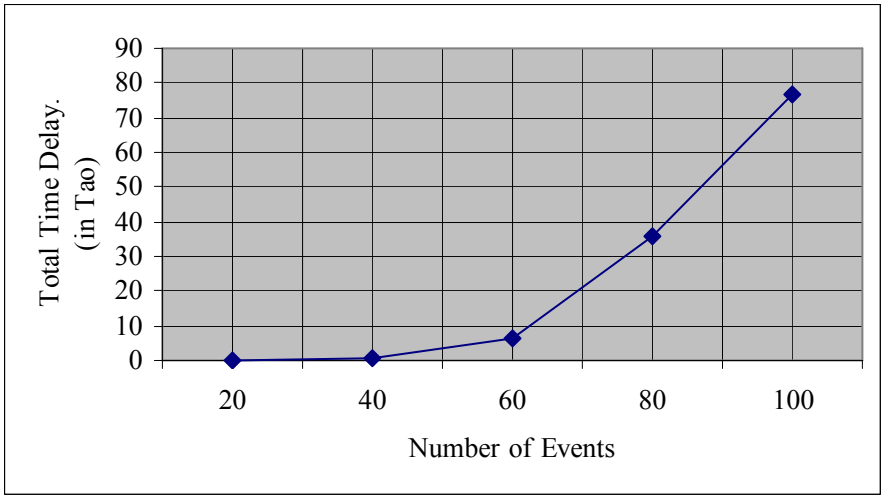

Figure 4.7. Total Time Delay vs. Number of Events; Maximum Size of Label Set=2; without FP

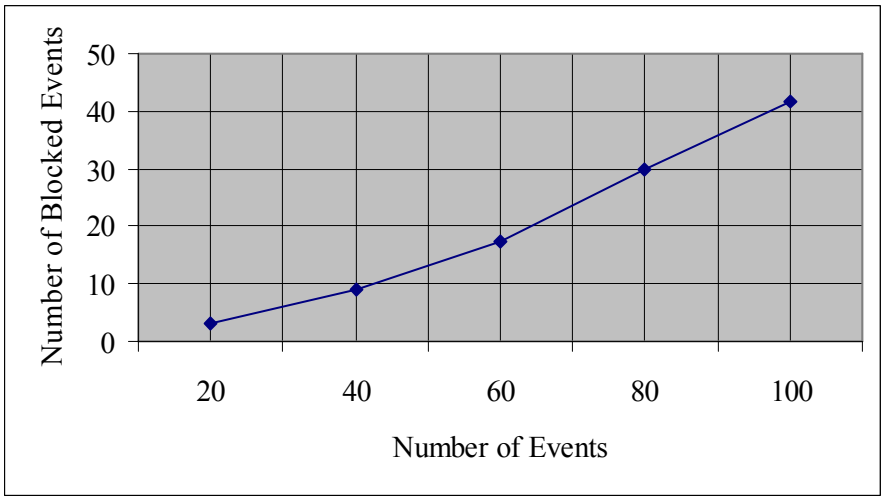

Figure 4.8. Number of Blocked Events vs. Number of Events; Maximum Size of Label Set=4; without FP

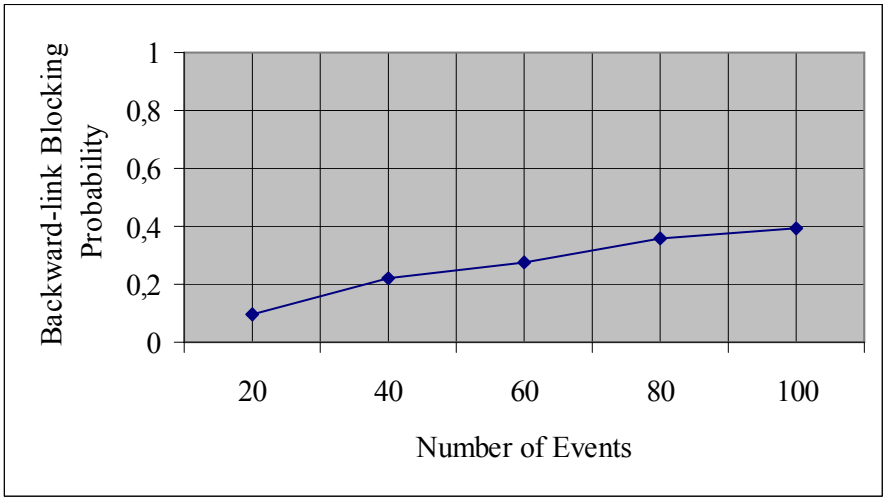

Figure 4.9. Backward-link Blocking vs. Number of Events; Maximum Size of Label Set=4; without FP

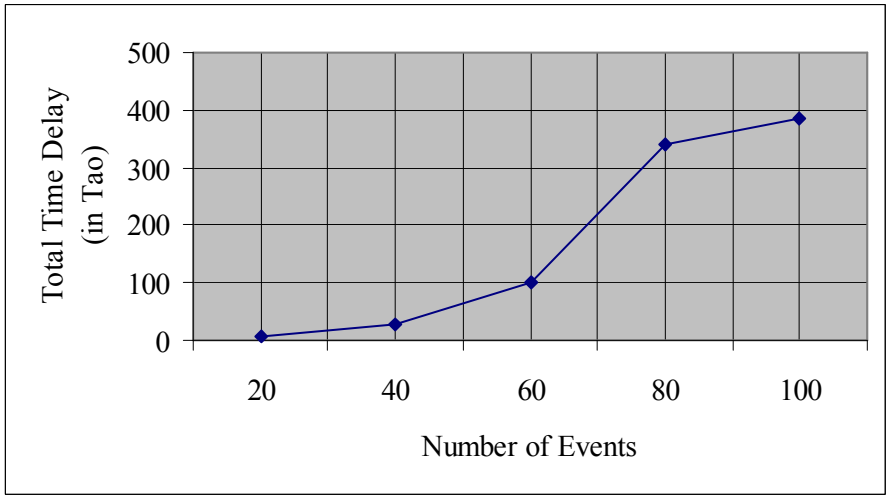

Figure 4.10. Total Time Delay vs. Number of Events; Maximum Size of Label Set=4; with FP

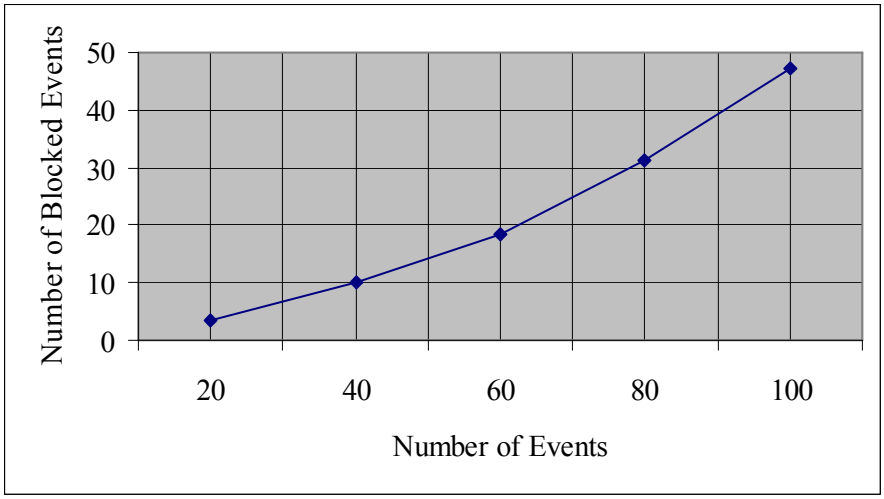

Figure 4.11. Number of Blocked Events vs. Number of Events; Maximum Size of Label Set=6; without FP

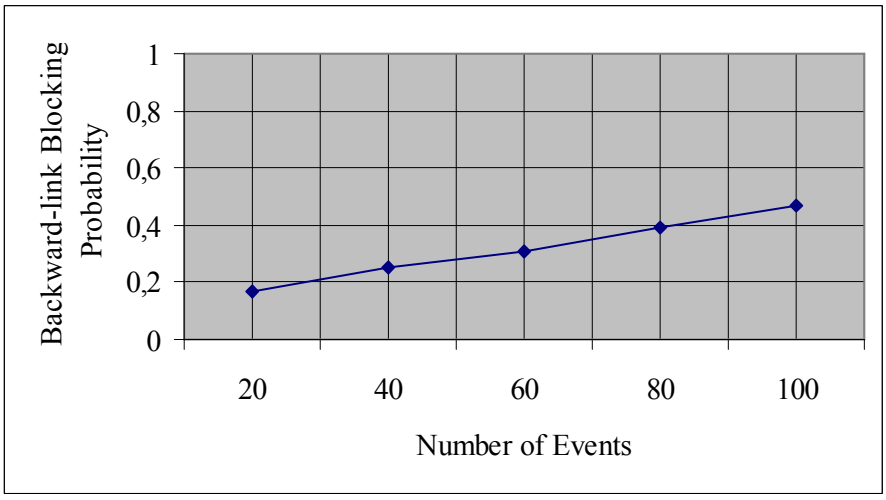

Figure 4.12. Backward-link Blocking vs. Number of Events; Maximum Size of Label Set=6; without FP

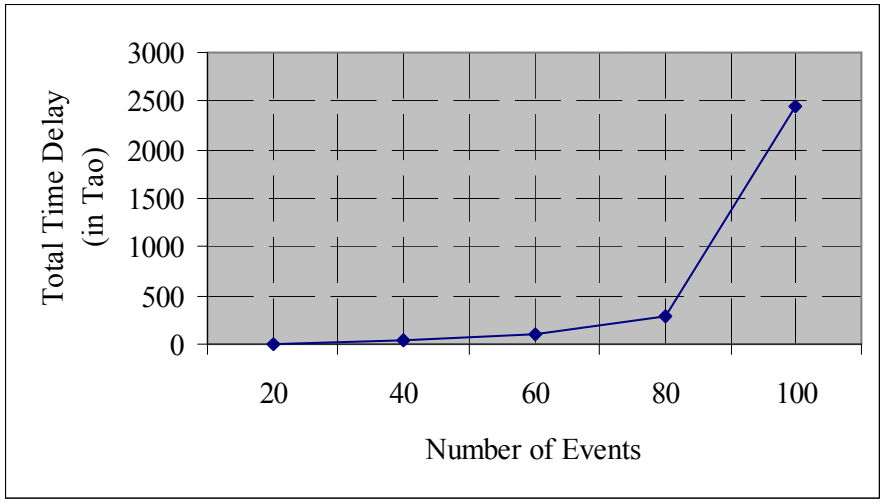

Figure 4.13. Total Time Delay vs. Number of Events; Maximum Size of Label Set=6; with FP

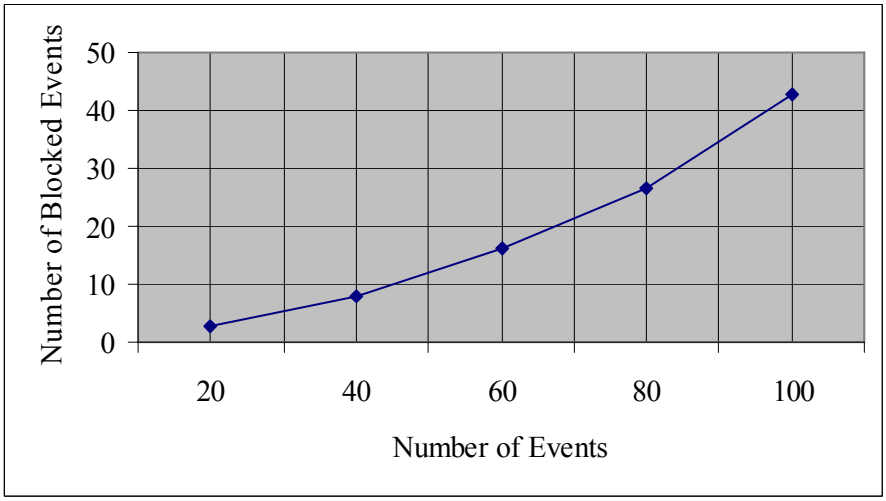

Figure 4.14. Number of Blocked Events vs. Number of Events; Maximum Size of Label Set=8; without FP

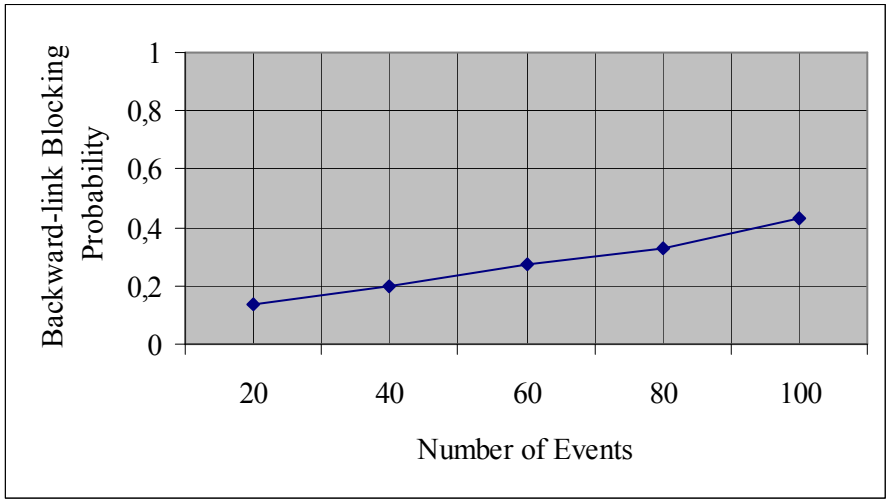

Figure 4.15. Backward-link Blocking vs. Number of Events; Maximum Size of Label Set=8; without FP

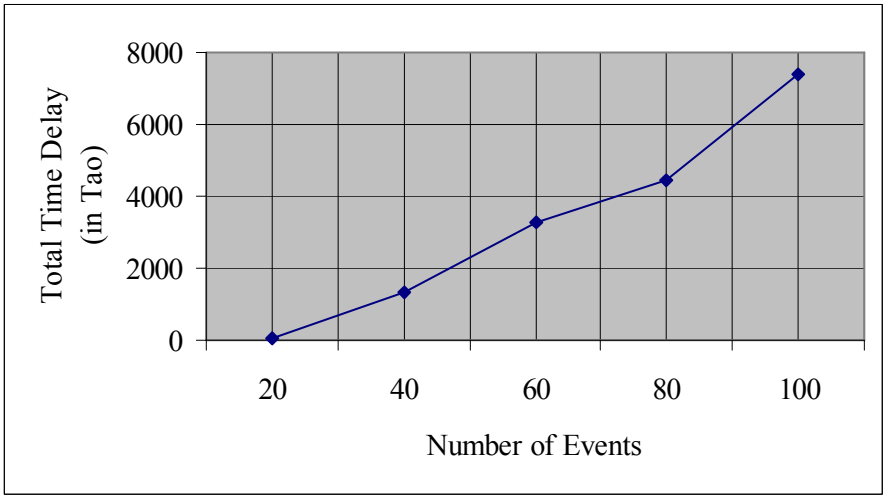

Figure 4.16. Total Time Delay vs. Number of Events; Maximum Size of Label Set=8; with FP

Lastly, we give the amount of lambda conversions faced during the transmissions in Figures 4.17. and 4.18. This is given both for the cases when FP is used or FP is not used. As it is seen, the number of lambda conversions increases as the number of transmission requests increases. An important fact is that for the operation with FP lambda conversion is reduced when all of the lambdas are suggested to downstream. The amount of lambda conversion decreases, since for each label request message all available lambdas are suggested and the other incoming label request messages are delayed. So, the probability of using different labels, i.e. lambdas, at different nodes will decrease.

The delay amounts shown in total delay distribution graphs are only delays faced in the signaling session. The delay amounts due to the lambda conversion are not given in this study. We only show the amount of lambda conversions. The delay times due to the lambda conversion process at a node is estimated to be between 10τ -20τ.

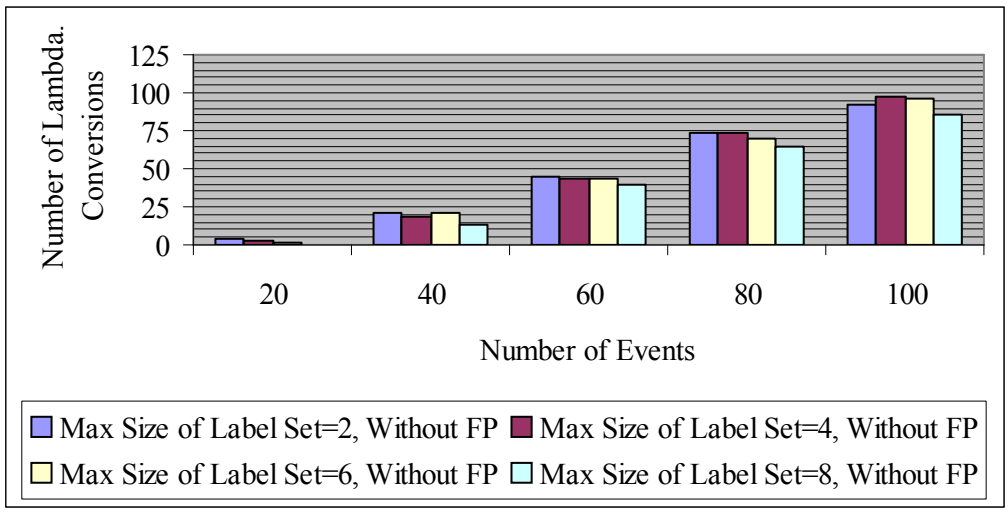

Figure 4.17. Number of Lambda Conversions vs. Number of Transmission Requests for the operation without FP

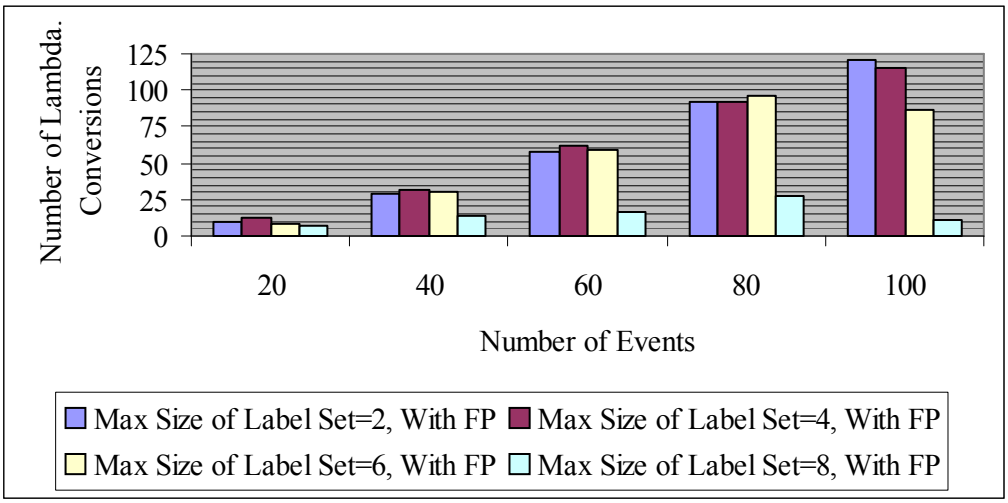

Figure 4.18. Number of Lambda Conversions vs. Number of Transmission Requests for the operation with FP

By means of these simulations, we have been able to examine the effect of label flagging operation on preventing the backward-link blocking. We have also seen the total delay amount of the label request signaling messages, when the proposed label assignment method is used. Besides that, the effect of limiting the number of suggestible label on both blocking and delay distribution is investigated. Also, we obtain the amount of lambda conversions, which is valuable statistical information for optical networks. The simulation results show that the proposed label assignment method improves the network performance by completely preventing backward-link blocking, but there is a trade-off between backward-link blocking and delaying the label request signaling. This study may be useful for the data networks, where the data lost is not acceptable.

CHAPTER 5

CONCLUSIONS

In this study, possible applications of label switching in large area, fully optical networks are investigated. Two important concepts of optical networking, namely all-optical networking and GMPLS, are covered in the thesis, which will give an idea for the future optical networks and provide the ways to achieve an efficient optical network. After GMPLS and all-optical networking are searched, two concepts, namely label assignment method and a concatenated label structure are proposed by using the background information obtained during the literature research.

Label assignment method was designed to minimize the blocking probabilities in the network and to provide an efficient utilization of resources. Mainly, the lambda switch capable GMPLS networks are investigated. The studies are based on the label flagging method introduced in [6]. A new algorithm is designed for the label assignment including label flagging and concatenated label structure. To verify the proposed method, a simulator is developed.

We run the simulation for different cases by changing some system parameters to investigate the effects of flagging method and traffic load on the blocking probability and the effects of system parameters in the network performance.

The results of simulation show that the proposed method completely prevents the backward-link blocking, while increasing the forward-link blocking, which can be

eliminated by postponing the signaling. This delay method is necessary for the networks where the data has to be certainly transmitted, since the label request will not be blocked, but the transmission will be realized with some delay. Hence, simulation outputs have clearly indicated that the proposed approaches could be beneficial in an all-optical network operation.

All-optical transmission provides high speed data transmission by eliminating the electro-optical conversions. Label concatenation is proposed to make all-optical transmission easier. It is thought that, if the information of all reserved labels can be used to form a concatenated label structure including the labels of the entire path, then this concatenated label will be converted to optical domain only once, and will be conveyed all-optically at the rest of the transmission.

For all-optical networking topic, only a concatenated label structure is proposed and the realization of the all-optical switching job at the nodes is left to out of this study. So, we assume that we will be able to switch the incoming data fully in the optical domain by the help of the concatenated label structure and we give only an idea for the further investigations in this area.

Since the optical networks have become the most important networking technology in communication because of their bandwidth capacity, which can meet the enormous traffic demand of today's networks, the research and developments on optical networking will keep on and optical networks will constitute a serious part of the future networks.

REFERENCES

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

[2] E. Mannie, "Generalized Multiprotocol Label Switching (GMPLS) Architecture", IETF RFC 3945, October 2004.

[3] J. Zheng, H. T. Mouftah, "Optical WDM Networks", IEEE Press, 2004.

[4] The International Engineering Consortium, "Generalized Multiprotocol Label Switching (GMPLS)", http://www.iec.org, October 2003.

[5] Mike J. O' Mahony, Dimitra Simeonidou, David K. Hunter, Anna Tzanakaki, "The Application of Optical Packet Switching in Future Communication Networks", The International Engineering Consortium, IEEE Communications Magazine, March 2001.

[6] T. Özügür, M. Park, J. P. Jue, "Label Prioritization in GMPLS-Cenric All-Optical Networks", IEEE, 2003.

[7] Bilkent University, "MPLS (Multiprotocol Label Switching)", http://www.ee.bilkent.edu.tr/~ee536/week_12.pdf, March 2004.

[8] N. Jerram, A. Farrel, "MPLS in Optical Networks", Data Connection Limited, October 2001.

[9] Opnet Technologies Inc., "Introduction to MPLS", 2001.

[10] The International Engineering Consortium, “Multiprotocol Label Switching (MPLS)”, http://www.iec.org, April 2003.

[11] J. Moy, “OSPF Version 2”, IETF RFC 2328, April 1998.

[12] D. Oran, “OSI IS-IS Intra-domain Routing Protocol”, IETF RFC 1142, February 1990 .

[13] L. Anderson, P. Doolan, N. Feldman, A. Fredette, B. Thomas, “LDP Specification”, IETF RFC 3036, January 2001.

[14] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, “RSVP-TE: Extensions to RSVP for LSP Tunnels”, IETF RFC 3209, December 2001.

[15] B. Jamoussi, L. Andersson, R. Callon, R. Dantu, L. Wu, P. Doolan, T. WorsterN. Feldman, A. Fredette, M. GirishE. Gray, J. Heinanen, T. Kilty, A. Malis, “Constraint Based LSP Setup Using LDP”, IETF RFC 3212, January 2002.

[16] J. Lang, “Link Management Protocol (LMP)”, IETF internet draft, October 2003.

[17] L. Berger, “Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description”, IETF RFC 3471, January 2003.

[18] K. Ishiguro, T. Takada, A.Davey, A. Lindem, “Traffic Engineering Extensions to OSPF Version 3”, IETF internet draft, March 2005.

[19] H. Smit, T. Li, “IS-IS Extensions for Traffic Engineering”, IETF RFC 3784, June 2004.

[20] K. Kompella, Y. Rekhter, “OSPF Extension in support of Generalized MPLS”, IETF internet draft, October 2003.

[21] K. Kompella, Y. Rekhter, “IS-IS Extension in support of Generalized MPLS”, IETF internet draft, October 2003.

[22] L. Berger, “Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource Reservation Protocol- Traffic Engineering (RSVP-TE) Extensions”, IETF RFC 3473, January 2003.

[23] P. Ashwood-Smith, L. Berger, “Generalized Multi-Protocol Label Switching (GMPLS) Signaling Constraint-based Routed Label Distribution Protocol (CR-LDP) Extensions”, IETF RFC 3472, January 2003.

[24] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, “Resource Reservation Protocol (RSVP)”, IETF RFC 2205, September 1997.

[25] J. H. Hahm, K. Lee, “Bandwidth Provisioning and Restoration Mechanisms in Optical Networks”, IETF internet draft, December 2000.

[26] T. Ozügür, D. Papadimitriou, “Label Set Priority and Flagging Operations”, May 2002.

[27] C. Argon, “Applications of Code Division Multiple Access in Fiber-Optic Communication Systems”, Ms. Thesis, Graduate School of Natural and Applied Sciences, Middle East Technical University, June 1994.

[28] A. Koçyiğit, D. Gökışık, S. Bilgen, “All-Optical Networking”, Turk J Elec Engin, VOL.9, NO.2, 2001.

[29] A. E. Willner, D. Gürkan, A. B. Şahin, J. E. Mcgeehan, M. C. Hauer, “All-Optical Address Recognition For Optically-Assisted Routing in Next-Generation Optical Networks”, IEEE Optical Communications, May 2003.

[30] Answers.Com. Homepage, http://www.answers.com, July 2005.

APPENDIX A

POISSON DISTRIBUTED EVENT GENERATION

Poisson distribution is used in the simulation to model the transmission request arrivals in the network. Below, some information on Poisson distribution is given.

The Poisson distribution is discovered by Siméon-Denis Poisson. It is a discrete probability distribution belonging to certain random variables *N* that count, among other things, a number of discrete occurrences that take place during a time-interval of given length [30]. The probability of having exactly *k* occurrences before time *t* is calculated by the below formula, where λ is the *rate*, i.e., the average number of occurrences per unit time.

$$P(N_t = k) = \frac{e^{-\lambda t}(\lambda.t)^k}{k!} \quad \text{(A.1)}$$

Let E_i be an event set that is Poisson distributed with rate λ requests per unit time. The rate λ is equal to k/T, where k is the total number of occurred requests in the time interval T.

$$E_i = \{E_1, E_2, E_3, \ldots, Ei, \ldots, E_{k-1}, E_k\} \quad \text{(A.2)}$$

There is another task, which is finding the arrival times of the events. The set of arrival times is represented by t_i.

$$t_i = \{ t_1, t_2, t_3, \ldots, t_i, \ldots, t_{k-1}, t_k \} \quad \text{(A.3)}$$

If initial time of event generation is considered to be t_0, then the intervals between successive events will be $\Delta t_1 = (t_1 - t_0)$, $\Delta t_2 = (t_2 - t_1)$, ..., $\Delta t_i = (t_i - t_{i-1})$. If the inter-arrival times (Δt_i) can be calculated, then the request arrival instances will be found based on the inter-arrival times. For example, the time instant of the second arrival will be equal to $(t_1 + \Delta t_2)$.

According to the information given in [30], the inter-arrival times of a Poisson distributed request arrival scheme have an exponential distribution defined by

$$f(t) = \lambda . e^{-\lambda . t} \quad \text{(A.4)}$$

To produce exponentially distributed inter-arrival time values the following formula can be used [30].

$$t = -\frac{\ln u}{\lambda} \quad \text{(A.5)}$$

In (A.5), u is a random number drawn from the uniform distribution in the interval (0, 1]. u can be obtained by using a random number generator on a computer.

When Poisson distributed events are generated, exponentially distributed inter-arrival times will be obtained. If inter-arrival times are calculated, the time

instances of the events will be obtained by using the inter-arrival times and initial time of the event generation process.

APPENDIX B

C++ CODE OF THE SIMULATOR

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <windows.h>
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <dos.h>

#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

#define MAX_NUM_OF_NODES  10
#define MAX_NUM_OF_LINKS   10
#define NODE_DIAMETER          25
#define MAX_NUM_LAMBDA     8
#define TOTFP              50
#define TOTUP              50
#define TAU                 1

#define DATATYPE_PATH      0
#define DATATYPE_NOTIF     1
#define DATATYPE_RESV      2
#define DATATYPE_DATA      3

#define POOL_AP            0
#define POOL_FP            1
#define POOL_UP            2
```

```
#define POOL_PRP         3
#define POOL_NOLAMBDA      4



//node->link->pool[][x].. x = ..
#define POOLSTATE_OFFSET   0
#define MESSAGEID_OFFSET   1
#define TIME_OFFSET        2
#define TOT_OFFSET         3

#define BUFFERLENGTH       50
#define NUMOFBUFFERS       100

#define LAMBDA_CONVERSION_TIME 1

//#define MAX_TIME          100
#define EXTRA_TIME         50

TForm1 *Form1;

int  FindAvailableBufferNo(int nodeNo);
void PrepareRoutingTable(void);
void PreparePacket(int source, int destination, int time);
void GetOrderedIndexes(int input[], int output[], int length);
void DrawSimulationBars_And_CheckTOT(void);

int currentTime   = 0;
int messageNumber = 1; //This will used as message ID, and it will be incremented by 1 with a new message
int maxLambdaToOffer;
int MAX_TIME = 100;

bool Flagged = false;

struct links
{
  int   numOfLambda;
  int     pool[MAX_NUM_LAMBDA][4]; //pool[0 for lambda1][0] = state of lambda1: AP, FP, UP, NO LAMBDA1
  int   numOfAP;
  int   numOfFP;
  int   numOfUP;
  //int   linkID;
```

```
}link[MAX_NUM_OF_LINKS];

struct nodes
{
 int    x; //x coordinate of left upper corner
 int    y; //y coordinate of left upper corner
 int    routingTable[MAX_NUM_OF_NODES];
 int     numOfLinks; //The number of links can be maximum 4, in the network the
number of links connected to a node is maximum 4
 int    data[NUMOFBUFFERS][BUFFERLENGTH];

}node[MAX_NUM_OF_NODES];


int linkTable[MAX_NUM_OF_NODES][MAX_NUM_OF_NODES] = {{-1,-1,
0,-1,-1,-1,-1,-1,-1,-1},
                                    {-1,-1, 1,-1,-1,-1,-1,-1,-1,-1},
                                    { 0, 1,-1, 2, 3,-1,-1,-1,-1,-1},
                                    {-1,-1, 2,-1,-1,-1,-1,-1,-1,-1},
                                    {-1,-1, 3,-1,-1, 5, 4,-1,-1,-1},
                                    {-1,-1,-1,-1, 5,-1,-1,-1, 8,-1},
                                    {-1,-1,-1,-1, 4,-1,-1, 6, 7,-1},
                                    {-1,-1,-1,-1,-1,-1, 6,-1,-1,-1},
                                    {-1,-1,-1,-1,-1, 8, 7,-1,-1, 9},
                                    {-1,-1,-1,-1,-1,-1,-1,-1, 9,-1,}};

int routTable[MAX_NUM_OF_NODES][MAX_NUM_OF_NODES] = {{-1, 2,
2, 2, 2, 2, 2, 2, 2, 2},
                                    { 2,-1, 2, 2, 2, 2, 2, 2, 2, 2},
                                    { 0, 1,-1, 3, 4, 4, 4, 4, 4, 4},
                                    { 2, 2, 2,-1, 2, 2, 2, 2, 2, 2},
                                    { 2, 2, 2, 2,-1, 5, 6, 6, 6, 6},
                                    { 4, 4, 4, 4, 4,-1, 8, 8, 8, 8},
                                    { 4, 4, 4, 4, 4, 8,-1, 7, 8, 8},
                                    { 6, 6, 6, 6, 6, 6, 6,-1, 6, 6},
                                    { 6, 6, 6, 6, 6, 5, 6, 6,-1, 9},
                                    { 8, 8, 8, 8, 8, 8, 8, 8, 8,-1}};
/*
int routTable[MAX_NUM_OF_NODES][MAX_NUM_OF_NODES] = {{-1, 2,
2, 2, 2, 2, 2, 2, 2, 2},
                                    { 2,-1, 2, 2, 2, 2, 2, 2, 2, 2},
                                    { 0, 1,-1, 3, 4, 4, 4, 4, 4, 4},
                                    { 2, 2, 2,-1, 2, 2, 2, 2, 2, 2},
                                    { 2, 2, 2, 2,-1, 5, 6, 6,56,56},
                                    { 4, 4, 4, 4, 4,-1,48,48, 8, 8},
```

```
                                    { 4, 4, 4, 4, 4,48,-1, 7, 8, 8},
                                    { 6, 6, 6, 6, 6, 6, 6,-1, 6, 6},
                                    {56,56,56,56,56, 5, 6, 6,-1, 9},
                                    { 8, 8, 8, 8, 8, 8, 8, 8, 8,-1}};
*/
//Program starts with this function
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int k;

  node[0].x = 40;   node[0].y = 320;
  node[1].x = 75;   node[1].y = 50;
  node[2].x = 110;  node[2].y = 200;
  node[3].x = 200;  node[3].y = 320;
  node[4].x = 200;  node[4].y = 180;
  node[5].x = 300;  node[5].y = 40;
  node[6].x = 290;  node[6].y = 160;
  node[7].x = 320;  node[7].y = 260;
  node[8].x = 380;  node[8].y = 175;
  node[9].x = 450;  node[9].y = 100;

  networkImage->Canvas->MoveTo(0,0);
  simImage->Canvas->MoveTo(0,0);

  networkImage->Canvas->Pen->Color = clRed;
  for (int i=0; i<MAX_NUM_OF_NODES; i++)
  {
    networkImage->Canvas->Brush->Color = clBlack;
    networkImage->Canvas->Ellipse(node[i].x,                                    node[i].y,
node[i].x+NODE_DIAMETER, node[i].y+NODE_DIAMETER);

    for (int j=0; j<NUMOFBUFFERS; j++)
        for (k=0; k<BUFFERLENGTH; k++)
       node[i].data[j][k] = -1;

    for (int m=0; m<MAX_NUM_OF_LINKS; m++)
      for (int j=0; j<MAX_NUM_LAMBDA; j++)
        link[m].pool[j][POOLSTATE_OFFSET] = POOL_AP; //initial ap
```

```
//start of debug code
/*
   link[0].pool[0][POOLSTATE_OFFSET] = POOL_NOLAMBDA;
   link[0].pool[1][POOLSTATE_OFFSET] = POOL_NOLAMBDA;
   link[0].pool[2][POOLSTATE_OFFSET] = POOL_NOLAMBDA;
   link[0].pool[3][POOLSTATE_OFFSET] = POOL_NOLAMBDA;

   link[3].pool[4][POOLSTATE_OFFSET] = POOL_NOLAMBDA;
   link[3].pool[5][POOLSTATE_OFFSET] = POOL_NOLAMBDA;
   link[3].pool[6][POOLSTATE_OFFSET] = POOL_NOLAMBDA;
   link[3].pool[7][POOLSTATE_OFFSET] = POOL_NOLAMBDA;
*/
/*
   int inArray[8] = {2, 0, 1, 2, 3, 2, 3, 0};
   int outArray[8];

   GetOrderedIndexes(inArray, outArray, 8);

   for (int j=0; j<8; j++)
    Memo1->Lines->Add(outArray[j]);
*/
//end of debug code
   node[i].numOfLinks = 0;
   for (int j=0; j<MAX_NUM_OF_NODES; j++)
   {
    node[i].routingTable[j] = routTable[i][j];

    if ((node[i].routingTable[j]>-1) && (node[i].routingTable[j]<10))
    {
     //draw links
     networkImage->Canvas->MoveTo(node[i].x                                    +
NODE_DIAMETER/2,node[i].y+NODE_DIAMETER/2);
     networkImage->Canvas->LineTo(node[node[i].routingTable[j]].x              +
NODE_DIAMETER/2,node[node[i].routingTable[j]].y+NODE_DIAMETER/2);
/*
     for (k=0; k<node[i].numOfLinks; k++)
      if (node[i].link[k].linkID == (i*10) + node[i].routingTable[j])
       break;
     if (k == node[i].numOfLinks)
     {
      node[i].link[node[i].numOfLinks].linkID          =          (i*10)          +
(node[i].routingTable[j]);
      node[i].numOfLinks++;
     }
```

```
*/
     }//if
   }//for

/*
   Memo1->Lines->Add("Node " + IntToStr(i));
   Memo1->Lines->Add(IntToStr(node[i].link[0].linkID)      +      "      "      +
IntToStr(node[i].link[1].linkID) + " " + IntToStr(node[i].link[2].linkID) + " " +
IntToStr(node[i].link[3].linkID));
   Memo1->Lines->Add(IntToStr(node[i].numOfLinks));
*/
  }//for

  //Enter number of lambda for every link
  link[0].numOfLambda = 8;
  link[1].numOfLambda = 8;
  link[2].numOfLambda = 8;
  link[3].numOfLambda = 8;
  link[4].numOfLambda = 8;
  link[5].numOfLambda = 8;
  link[6].numOfLambda = 8;
  link[7].numOfLambda = 8;
  link[8].numOfLambda = 8;
  link[9].numOfLambda = 8;

  node0->Left = node[0].x;  node0->Top  = node[0].y;
  node1->Left = node[1].x;  node1->Top  = node[1].y;
  node2->Left = node[2].x;  node2->Top  = node[2].y;
  node3->Left = node[3].x;  node3->Top  = node[3].y;
  node4->Left = node[4].x;  node4->Top  = node[4].y;
  node5->Left = node[5].x;  node5->Top  = node[5].y;
  node6->Left = node[6].x;  node6->Top  = node[6].y;
  node7->Left = node[7].x;  node7->Top  = node[7].y;
  node8->Left = node[8].x;  node8->Top  = node[8].y;
  node9->Left = node[9].x;  node9->Top  = node[9].y;

  link0->Left = (node[0].x + node[2].x + 30) / 2;   link0->Top   = (node[0].y +
node[2].y + 30) / 2;
  link1->Left = (node[1].x + node[2].x + 20) / 2;   link1->Top   = (node[1].y +
node[2].y + 30) / 2;
  link2->Left = (node[3].x + node[2].x + 70) / 2;   link2->Top   = (node[3].y +
node[2].y + 30) / 2;
  link3->Left = (node[4].x + node[2].x + 30) / 2;   link3->Top   = (node[4].y +
node[2].y + 30) / 2;
```

```
  link4->Left = (node[4].x + node[6].x + 50) / 2;   link4->Top  = (node[4].y +
node[6].y + 30) / 2;
  link5->Left = (node[4].x + node[5].x + 30) / 2;   link5->Top  = (node[4].y +
node[5].y + 30) / 2;
  link6->Left = (node[6].x + node[7].x + 10) / 2;   link6->Top  = (node[6].y +
node[7].y + 30) / 2;
  link7->Left = (node[6].x + node[8].x + 30) / 2;   link7->Top  = (node[6].y +
node[8].y + 30) / 2;
  link8->Left = (node[5].x + node[8].x + 70) / 2;   link8->Top  = (node[5].y +
node[8].y + 30) / 2;
  link9->Left = (node[8].x + node[9].x + 30) / 2;   link9->Top  = (node[8].y +
node[9].y + 30) / 2;

  for (int i=0; i<MAX_TIME; i++)
    timeCombo->Items->Add(IntToStr(i));

  Memo1->Lines->Add("");

  //Seed for random function rand()
  time_t t1 = time(NULL);
  srand((long)t1);

  DrawSimulationBars_And_CheckTOT();
}
//---------------------------------------------------------------------------



//---------------------------------------------------------------------------
void __fastcall TForm1::addEventBtnClick(TObject *Sender)
{
  if (sourceCombo->ItemIndex < 0) {Application->MessageBox("Choose a source
node !!!", "ERROR", MB_OK); return;}
  if (destCombo->ItemIndex < 0) {Application->MessageBox("Choose a
destination node !!!", "ERROR", MB_OK); return;}
  if (timeCombo->ItemIndex < 0) {Application->MessageBox("Choose the
starting time of the event !!!", "ERROR", MB_OK); return;}
  if (sourceCombo->ItemIndex == destCombo->ItemIndex) {Application-
>MessageBox("Source and destination cannot be the same node !!!", "ERROR",
MB_OK); return;}

  Memo1->Lines->Add(sourceCombo->Text + " -> " + destCombo->Text + " at
t=" + timeCombo->Text);
  PreparePacket(sourceCombo->ItemIndex, destCombo->ItemIndex, timeCombo-
>ItemIndex);
```

```
}
//---------------------------------------------------------------------------

class MainLoopThread:public TThread
{
 public:
  MainLoopThread():TThread(false){}

 void __fastcall Execute()
 {
  int lambdaIntersection = 0xFF;
  int foundLambda = 0;
  int nextNode = 0;
  int lambda;
  int temp,k,x;
  int linkNo;
  int availBufferNo;
  int sumsOfAvailableLambdas[8];
  int orderedLambdaArray[8];
  int tempNextNode;
  int tempSource;
  int numOfFoundLambda;
  //For both flagged and non-flagged cases; if there is no available lambda, then
delay by 1 unit time
  int bostaLambdaYok = 0;
  //For non-flagged case only; if the lambda to be suggested is in use by another
event, then delay by 1 unit time
  int onerilenLambdaKullanimda = 0;
  //For both flagged and non-flagged cases; if there is a lambda conversion, then
delay by 1 unit time
  int lambdaConversionYapiliyor = 0;

  int bostaLambdaYokArray[200] = {0,0,0};
  int onerilenLambdaKullanimdaArray[200] = {0,0,0};

  Form1->addEventBtn->Enabled = false;
  Form1->addEventsBtn->Enabled = false;
  Form1->startSimBtn->Enabled = false;

  currentTime = 0;
  while (currentTime < MAX_TIME + EXTRA_TIME)
  {
   for (int i=0; i<MAX_NUM_OF_NODES; i++)
   {
    for (int j=0; j<NUMOFBUFFERS; j++)
```

```
{
 if ((node[i].data[j][0] > 0) && (node[i].data[j][4] == currentTime))
 {
  switch (node[i].data[j][3])
  {
   case DATATYPE_PATH:
    ////////////////////////////////////////////////////////////////
    ////                                                        ////
    ////                  P A T H                           ////
    ////                                                        ////
    ////////////////////////////////////////////////////////////////
    if (node[i].data[j][2] == i) //if this node is destination
    {
     Form1->Memo1->Lines->Add("Node " + IntToStr(i) + " has received
PATH message sent to itself, t=" + IntToStr(currentTime));

    nextNode = node[i].routingTable[node[i].data[j][1]];
    availBufferNo = FindAvailableBufferNo(nextNode);
                node[nextNode].data[availBufferNo][0] = node[i].data[j][0];
    node[nextNode].data[availBufferNo][1] = node[i].data[j][1];
    node[nextNode].data[availBufferNo][2] = node[i].data[j][2];
    node[nextNode].data[availBufferNo][3] = DATATYPE_RESV;
    node[nextNode].data[availBufferNo][4] = currentTime + 1;

                //determine lambda to use
    k = 5;
          while (node[i].data[j][k] > -1)
    {
     lambdaIntersection &= node[i].data[j][k+1];
     k += 2;
    }

    if (lambdaIntersection != 0)
    {
     k = 0;
     while (!(lambdaIntersection & 0x80)) {
      lambdaIntersection <<= 1;
      k ++;
     }
     foundLambda = k;
     //The lambda is determined, prepare Resv message and sent it
upstream
     k = 5;
     while (node[i].data[j][k] > -1)
     {
```

```
                node[nextNode].data[availBufferNo][k]   = node[i].data[j][k];
                node[nextNode].data[availBufferNo][k+1] = foundLambda;
                k+=2;
               }
               Form1->Memo1->Lines->Add("Node " + IntToStr(i) + " has started
reservation session by sending RESV message that will end at Node " +
IntToStr(node[i].data[j][1]) + ", NO Lambda Conversion, t=" +
IntToStr(currentTime));
                              }
             else //Here, the intersection of Label Sets is empty set, therefore there
will be LAMBDA CONVERSION
               {
                //Find the number of repeated available lambdas, do this for each
available lambda
                k = 5;
                for (int s=0; s<8; s++)
                 sumsOfAvailableLambdas[s] = 0;
                while (node[i].data[j][k] > -1)
                {
                 for (int s=0; s<8; s++) {
                   temp = node[i].data[j][k+1];
                   if (temp & 0x80)
                     sumsOfAvailableLambdas[s] ++;
                   temp <<= 1;
                 }
                 k+=2;
                }
                //Find the index order of available lambdas

                GetOrderedIndexes(sumsOfAvailableLambdas,   orderedLambdaArray,
8);

                k = 5;
                while (node[i].data[j][k] > -1)
                {
                 tempNextNode = node[i].data[j][k+2];
                 if (tempNextNode < 0) //Since destination node is not written in the
message packet
                   tempNextNode = node[i].data[j][2];

                 tempSource = node[i].data[j][k];
                                //To control the pools find link ID
                 linkNo = linkTable[tempSource][tempNextNode];

                 if (Flagged) {
```

```
                for (x=0; x<8; x++)
                  if
(link[linkNo].pool[orderedLambdaArray[x]][POOLSTATE_OFFSET]          ==
POOL_FP)
                    if
(link[linkNo].pool[orderedLambdaArray[x]][MESSAGEID_OFFSET]          ==
node[i].data[j][0])  //check message id
                      break;
               }
               else {
                 for (x=0; x<8; x++)
                   if
(link[linkNo].pool[orderedLambdaArray[x]][POOLSTATE_OFFSET]          ==
POOL_AP)
                     break;
               }

               node[nextNode].data[availBufferNo][k]   = node[i].data[j][k];
               node[nextNode].data[availBufferNo][k+1] = x;
               k+=2;
              }
              Form1->Memo1->Lines->Add("Node " + IntToStr(i) + " has started
reservation session by sending RESV message that will end at Node " +
IntToStr(node[i].data[j][1]) + ", LAMBDA CONVERSION, t=" +
IntToStr(currentTime));
            }
          }
          else //This is an intermediate node, so the node adds its message and
sends the message to the next node
          {
                        //Find the next node, to which message will be transmitted
                        nextNode = node[i].routingTable[node[i].data[j][2]];

                        //Find the link to be used for the transmission
            linkNo = linkTable[i][nextNode];

                        //Find the lambda vector that will be added to the end of
the message packet; and arrange the pools
            numOfFoundLambda = 0;
                        foundLambda = 0;
                        for (int s=0; s<MAX_NUM_LAMBDA; s++) {
                           if   (link[linkNo].pool[s][POOLSTATE_OFFSET]   ==
POOL_AP) {
               numOfFoundLambda ++;
               if (numOfFoundLambda <= maxLambdaToOffer)
```

```
            {
             foundLambda  |= 0x01;
             if (Flagged) {
              //transfer lambda from AP to FP, adjust the pool parameters
                                link[linkNo].pool[s][POOLSTATE_OFFSET]    =
POOL_FP;
                                link[linkNo].pool[s][MESSAGEID_OFFSET]    =
node[i].data[j][0];
               link[linkNo].pool[s][TIME_OFFSET]      = currentTime;
               link[linkNo].pool[s][TOT_OFFSET]     = TOTFP;
             }
            }
                           }
                           if (s != MAX_NUM_LAMBDA-1)
    foundLambda <<= 1;
                           }

        if (foundLambda == 0) //There is no available lambda, delay by 1 unit
time
        {
         if (node[i].data[j][0] < 200)
           bostaLambdaYokArray[node[i].data[j][0]] ++;
         bostaLambdaYok ++;
         node[i].data[j][4] += 1;
         break;
        }

        Form1->Memo1->Lines->Add("Node " + IntToStr(i) + " has transmitted
PATH message sent from Node "+ IntToStr(node[i].data[j][1]) + " to Node " +
IntToStr(node[i].data[j][2]) + ", t=" + IntToStr(currentTime));
                           availBufferNo = FindAvailableBufferNo(nextNode);
                           //prepare the message
        node[nextNode].data[availBufferNo][0] = node[i].data[j][0];
        node[nextNode].data[availBufferNo][1] = node[i].data[j][1];
        node[nextNode].data[availBufferNo][2] = node[i].data[j][2];
        node[nextNode].data[availBufferNo][3] = node[i].data[j][3];
        node[nextNode].data[availBufferNo][4] = currentTime + 1;
        k = 5;
        while (node[i].data[j][k] > -1)
        {
         node[nextNode].data[availBufferNo][k]   = node[i].data[j][k];
         node[nextNode].data[availBufferNo][k+1] = node[i].data[j][k+1];
         k+=2;
        }
                           //Add your message to the end
```

```
                        node[nextNode].data[availBufferNo][k]   = i;
                        node[nextNode].data[availBufferNo][k+1]                    =
foundLambda;
                    }

          //the array is processed, empty array
          for (k=0; k<BUFFERLENGTH; k++)
            node[i].data[j][k] = -1;
        break;
        case DATATYPE_NOTIF:
        break;
        case DATATYPE_RESV:
          ////////////////////////////////////////////////////////////
          ////                                                    ////
          ////                R E S V                             ////
          ////                                                    ////
          ////////////////////////////////////////////////////////////
          //get the lambda we are reserved to use
                    k = 5;
          while (node[i].data[j][k] > -1)
          {
            if (node[i].data[j][k] == i)
                        {
                            foundLambda = node[i].data[j][k+1];
              break;
                        }
            k+=2;
          }

                        //arrange the pools
                    nextNode = node[i].routingTable[node[i].data[j][2]];
                    //find the link to be used for the transmission
          linkNo = linkTable[i][nextNode];

          if (Flagged) {
                    for (int s=0; s<MAX_NUM_LAMBDA; s++) {
                        if    (link[linkNo].pool[s][MESSAGEID_OFFSET]    ==
node[i].data[j][0]) {//message IDs are same
                if (link[linkNo].pool[s][POOLSTATE_OFFSET] == POOL_FP) {
                                    if (s == foundLambda) {
                                        link[linkNo].pool[s][POOLSTATE_OFFSET] =
POOL_UP;
                                    link[linkNo].pool[s][MESSAGEID_OFFSET]    =
node[i].data[j][0];
                    link[linkNo].pool[s][TIME_OFFSET]     = currentTime;
```

```
                link[linkNo].pool[s][TOT_OFFSET]      = TOTUP;
              }
                                  else
                                     link[linkNo].pool[s][POOLSTATE_OFFSET] =
POOL_AP;
              }
                                }
                             }
          }
          else {
            if     (link[linkNo].pool[foundLambda][POOLSTATE_OFFSET]     ==
POOL_AP) {
             link[linkNo].pool[foundLambda][POOLSTATE_OFFSET]              =
POOL_UP;
                              link[linkNo].pool[foundLambda][MESSAGEID_OFFSET]
= node[i].data[j][0];
             link[linkNo].pool[foundLambda][TIME_OFFSET]     = currentTime;
             link[linkNo].pool[foundLambda][TOT_OFFSET]      = TOTUP;
            }
            else {//offered lambda is used by another event, so delay by 1 unit time
             if (node[i].data[j][0] < 200)
               onerilenLambdaKullanimdaArray[node[i].data[j][0]] ++;
             onerilenLambdaKullanimda ++;
             node[i].data[j][4] += 1;
             break;
            }
          }

                          if (node[i].data[j][1] == i) //if this node is destination
(actually source, since RESV moves reverse)
                          {
          Form1->Memo1->Lines->Add("Node " + IntToStr(i) + " has received
RESV message as a response of its PATH message, and has started DATA
transmision session to destination Node " + IntToStr(node[i].data[j][2]) + ", t=" +
IntToStr(currentTime));

                             availBufferNo = FindAvailableBufferNo(nextNode);

                             node[nextNode].data[availBufferNo][0]              =
node[i].data[j][0];
          node[nextNode].data[availBufferNo][1] = node[i].data[j][1];
          node[nextNode].data[availBufferNo][2] = node[i].data[j][2];
          node[nextNode].data[availBufferNo][3] = DATATYPE_DATA;
          node[nextNode].data[availBufferNo][4] = currentTime + 1;
          k = 5;
```

```
            while (node[i].data[j][k] > -1)
            {
              node[nextNode].data[availBufferNo][k]   = node[i].data[j][k];
              node[nextNode].data[availBufferNo][k+1] = node[i].data[j][k+1];
              k+=2;
            }

            if   (link[linkNo].pool[node[i].data[j][6]][POOLSTATE_OFFSET]   ==
POOL_UP)
              link[linkNo].pool[node[i].data[j][6]][POOLSTATE_OFFSET]          =
POOL_AP;
            else {
              Form1->Memo1->Lines->Add("WARNING NO_1: Used lambda is
not in the UP, because the threshold time for UP has expired!!!");
              Form1->Memo2->Lines->Add("WARNING NO_1: Used lambda is
not in the UP, because the threshold time for UP has expired!!!");
            }
                        }
                        else //intermediate node, transmit the same message
                        {
            Form1->Memo1->Lines->Add("Node " + IntToStr(i) + " has transmitted
RESV message sent from destination Node " + IntToStr(node[i].data[j][2]) + " to
source Node " + IntToStr(node[i].data[j][1]) + ", t=" + IntToStr(currentTime));

                            //If this is an intermediate node, we sent the message in the
backward direction, the next node has to be found
                            nextNode = node[i].routingTable[node[i].data[j][1]];

                            availBufferNo = FindAvailableBufferNo(nextNode);

            node[nextNode].data[availBufferNo][0] = node[i].data[j][0];
            node[nextNode].data[availBufferNo][1] = node[i].data[j][1];
            node[nextNode].data[availBufferNo][2] = node[i].data[j][2];
            node[nextNode].data[availBufferNo][3] = node[i].data[j][3];
            node[nextNode].data[availBufferNo][4] = currentTime + 1;
            k = 5;
            while (node[i].data[j][k] > -1)
            {
              node[nextNode].data[availBufferNo][k]   = node[i].data[j][k];
              node[nextNode].data[availBufferNo][k+1] = node[i].data[j][k+1];
              k+=2;
            }
                        }
                        //array is processed,empty array
          for (k=0; k<BUFFERLENGTH; k++)
```

```
            node[i].data[j][k] = -1;

        break;
        case DATATYPE_DATA:
          ////////////////////////////////////////////////////////////
          ////                                                    ////
          ////                  D A T A                           ////
          ////                                                    ////
          ////////////////////////////////////////////////////////////
                        if (node[i].data[j][2] == i) //if this node is destination: do
nothing
                        {
                           Form1->Memo1->Lines->Add("Node " + IntToStr(i) + "
has received DATA sent by source Node " + IntToStr(node[i].data[j][1]) + ", t=" +
IntToStr(currentTime));
                           Form1->Memo1->Lines->Add(" ");
                        }
                        else //intermediate node, transmit the same message
                        {
            k = 5;
            while (node[i].data[j][k] > -1)
            {
             if (node[i].data[j][k] == i)
               break;
             k+=2;
            }

            if     (node[i].data[j][k-1]     !=     node[i].data[j][k+1])     //LAMBDA
CONVERSION
            {
             lambdaConversionYapiliyor ++;
             node[i].data[j][k-1] = node[i].data[j][k+1]; //1 unit time after, do not
make lambda conversion
             node[i].data[j][4] += LAMBDA_CONVERSION_TIME; //Process 1
unit time after
             break;
            }

                         Form1->Memo1->Lines->Add("Node " + IntToStr(i) + " has
transmitted DATA sent from source Node " + IntToStr(node[i].data[j][1]) + " to
destination    Node    "    +    IntToStr(node[i].data[j][2])    +    ",    t="    +
IntToStr(currentTime));
                           //arrange the pools
                      nextNode = node[i].routingTable[node[i].data[j][2]];
```

```
            linkNo = linkTable[i][nextNode];

                                for (int s=0; s<MAX_NUM_LAMBDA; s++) {
                                  if     (link[linkNo].pool[s][MESSAGEID_OFFSET]    ==
node[i].data[j][0]) //message IDs are same
                if (link[linkNo].pool[s][POOLSTATE_OFFSET] == POOL_UP)
                                        link[linkNo].pool[s][POOLSTATE_OFFSET]     =
POOL_AP;
            }

                                availBufferNo = FindAvailableBufferNo(nextNode);

            node[nextNode].data[availBufferNo][0] = node[i].data[j][0];
            node[nextNode].data[availBufferNo][1] = node[i].data[j][1];
            node[nextNode].data[availBufferNo][2] = node[i].data[j][2];
            node[nextNode].data[availBufferNo][3] = node[i].data[j][3];
            node[nextNode].data[availBufferNo][4] = currentTime + 1;
            k = 5;
            while (node[i].data[j][k] > -1)
            {
              node[nextNode].data[availBufferNo][k]   = node[i].data[j][k];
              node[nextNode].data[availBufferNo][k+1] = node[i].data[j][k+1];
              k+=2;
            }
                        }
                        //array is processed, empty array
          for (k=0; k<BUFFERLENGTH; k++)
             node[i].data[j][k] = -1;

                    break;
        }
       }
     }
   }
   currentTime ++;

   Synchronize(SimDraw);

   Sleep(TAU);
  } //while(1)

  Form1->addEventBtn->Enabled = true;
  Form1->addEventsBtn->Enabled = true;
  Form1->startSimBtn->Enabled = true;
```

```
  Form1->Memo2->Lines->Add("SIMULATION RESULTS:");
  Form1->Memo2->Lines->Add("Number of unavailable lambdas: " +
IntToStr(bostaLambdaYok));
  Form1->Memo2->Lines->Add("Number of offered lambdas in use: " +
IntToStr(onerilenLambdaKullanimda));
  Form1->Memo2->Lines->Add("Number of lambda conversions: " +
IntToStr(lambdaConversionYapiliyor));

  bostaLambdaYok = 0;
  onerilenLambdaKullanimda = 0;

  for (int i=0; i<200; i++) {
    if (bostaLambdaYokArray[i] != 0)
      bostaLambdaYok ++;
    if (onerilenLambdaKullanimdaArray[i] != 0)
      onerilenLambdaKullanimda ++;
  }
  Form1->Memo2->Lines->Add("Number of unavailable lambdas in distinct
events: " + IntToStr(bostaLambdaYok));
  Form1->Memo2->Lines->Add("Number of offered lambdas in use for distinct
events: " + IntToStr(onerilenLambdaKullanimda));

  return;
 }
 //---------------------------------------------------------------------------
 void __fastcall SimDraw()
 {
   DrawSimulationBars_And_CheckTOT();
 }
}; //class
//---------------------------------------------------------------------------
void __fastcall TForm1::startSimBtnClick(TObject *Sender)
{
  int addedAnyEvent = 0;

  for (int i=0; i<MAX_NUM_OF_NODES; i++)
    for (int j=0; j<NUMOFBUFFERS; j++)
      if (node[i].data[j][0] > 0) {
        addedAnyEvent = 1;
        break;
      }

  if (addedAnyEvent == 0) {Application->MessageBox("You must enter at least
one event before running the simulation !!!", "ERROR", MB_OK); return;}
```

```
  if (maxLambdaCombo->ItemIndex < 0) {Application->MessageBox("Enter
maximum number of lambdas to offer !!!", "ERROR", MB_OK); return;}
  maxLambdaToOffer = maxLambdaCombo->ItemIndex + 1;

  MainLoopThread *mainClass = new MainLoopThread;
  mainClass->Resume();
}
//---------------------------------------------------------------------------
int FindAvailableBufferNo(int nodeNo)
{
  int a = 0;

    while (a<NUMOFBUFFERS)
    {
   if (node[nodeNo].data[a][0] < 0)
          break;
   a++;
    }

  if (a == NUMOFBUFFERS)
   Form1->Memo2->Lines->Add("BUFFER IS FULL!!!!");

    return (a);
}
//---------------------------------------------------------------------------
void PreparePacket(int source, int destination, int time)
{
  int bufferNo;

  if ((source<0) || (source>=MAX_NUM_OF_NODES) || (destination<0) ||
(destination>=MAX_NUM_OF_NODES)) {
       Form1->Memo1->Lines->Add("Entered data has source-destination
problem");
       return;
    }

  bufferNo = FindAvailableBufferNo(source);

  node[source].data[bufferNo][0] = messageNumber ++;
  node[source].data[bufferNo][1] = source;
  node[source].data[bufferNo][2] = destination;
  node[source].data[bufferNo][3] = DATATYPE_PATH;
  node[source].data[bufferNo][4] = time;
}
//---------------------------------------------------------------------------
```

```
void GetOrderedIndexes(int input[], int output[], int length)
{
 int a,temp;
 int change;

 for (a=0; a<length; a++)
  output[a] = a;

 do
 {
  change = 0;
  for (a=0; a<length-1; a++)
  {
   if (input[a] < input[a+1])
   {
    temp       = input[a];
    input[a]   = input[a+1];
    input[a+1] = temp;

    temp       = output[a];
    output[a]  = output[a+1];
    output[a+1] = temp;

    change++;
   }
  }
 } while (change > 0);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::addEventsBtnClick(TObject *Sender)
{
 int numOfTotalEvent;
 int node1, node2, deltaT;
 int myTime = 0;
 double temp;
 int j;
 int priorities[MAX_NUM_OF_NODES] = {20,30,40,45,55,60,75,85,90,100};

 try {
  numOfTotalEvent = StrToInt(Edit1->Text);
 } catch (EConvertError &e) {Application->MessageBox("Please enter an integer
for the number of events!!!", "ERROR", MB_OK); return;}

 for (int i=0; i<numOfTotalEvent; i++)
```

```
  {
    node1 = (int)(rand()*100/RAND_MAX);
    j = 0;
    while (node1 >= priorities[j]) j++;
    node1 = j;
    //Find node2 that is different than node1
    do {
      node2 = (int)(rand()*100/RAND_MAX);
      j = 0;
      while (node2 >= priorities[j]) j++;
      node2 = j;
    } while (node2 == node1);

    temp = (-log((float)rand()/RAND_MAX) /
((float)numOfTotalEvent/MAX_TIME));
    if ((temp - floor(temp)) > 0.5) deltaT = ceil(temp); else deltaT = floor(temp);
    myTime += deltaT;

    Memo1->Lines->Add(IntToStr(node1) + " ---> " + IntToStr(node2) + " at t=" +
IntToStr(myTime));
    PreparePacket(node1, node2, myTime);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::resetBtnClick(TObject *Sender)
{
  for (int i=0; i<MAX_NUM_OF_NODES; i++)
    for (int j=0; j<NUMOFBUFFERS; j++)
        for (int k=0; k<BUFFERLENGTH; k++)
       node[i].data[j][k] = -1;

  for (int m=0; m<MAX_NUM_OF_LINKS; m++)
    for (int j=0; j<MAX_NUM_LAMBDA; j++)
      link[m].pool[j][POOLSTATE_OFFSET] = POOL_AP; //initial ap

  DrawSimulationBars_And_CheckTOT();
}
//---------------------------------------------------------------------------
void DrawSimulationBars_And_CheckTOT()
{
  //Draw simulation bars
  for (int ss=0,kk=0; ss<MAX_NUM_OF_LINKS; ss++) {
    link[ss].numOfAP = 0;
    link[ss].numOfFP = 0;
    link[ss].numOfUP = 0;
```

```
   for (int m=0; m<8; m++) {
     if (link[ss].pool[m][POOLSTATE_OFFSET] == POOL_AP)
link[ss].numOfAP++;
     else if (link[ss].pool[m][POOLSTATE_OFFSET] == POOL_FP) {
       if (link[ss].pool[m][TOT_OFFSET] + link[ss].pool[m][TIME_OFFSET] <=
currentTime) {
         link[ss].pool[m][POOLSTATE_OFFSET] = POOL_AP;
         link[ss].numOfAP++;
       }
       else
         link[ss].numOfFP++;
     }
     else if (link[ss].pool[m][POOLSTATE_OFFSET] == POOL_UP) {
       if (link[ss].pool[m][TOT_OFFSET] + link[ss].pool[m][TIME_OFFSET] <=
currentTime) {
         link[ss].pool[m][POOLSTATE_OFFSET] = POOL_AP;
         link[ss].numOfAP++;
       }
       else
         link[ss].numOfUP++;
     }
   }

   Form1->simImage->Canvas->Brush->Color = clWhite;
   Form1->simImage->Canvas->Rectangle(kk*23, 150, (kk+1)*23, 0);

   Form1->simImage->Canvas->Brush->Color = clGreen;
   Form1->simImage->Canvas->Rectangle(kk*23, 150, (kk+1)*23, 150 -
link[ss].numOfAP*150/8);

   Form1->simImage->Canvas->Brush->Color = clWhite;
   Form1->simImage->Canvas->Rectangle((kk+1)*23, 150, (kk+2)*23, 0);
   Form1->simImage->Canvas->Brush->Color = clBlack;
   Form1->simImage->Canvas->Rectangle((kk+1)*23, 150, (kk+2)*23, 150 -
link[ss].numOfFP*150/8);

   Form1->simImage->Canvas->Brush->Color = clWhite;
   Form1->simImage->Canvas->Rectangle((kk+2)*23, 150, (kk+3)*23, 0);
   Form1->simImage->Canvas->Brush->Color = clRed;
   Form1->simImage->Canvas->Rectangle((kk+2)*23, 150, (kk+3)*23, 150 -
link[ss].numOfUP*150/8);
   kk+=4;
  }
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::clearBtnClick(TObject *Sender)
{
  Memo1->Text = "";
}
//---------------------------------------------------------------------------
void __fastcall TForm1::maxTimeBtnClick(TObject *Sender)
{
  try {
    MAX_TIME = StrToInt(Edit2->Text);
  } catch (EConvertError &e) {Application->MessageBox("Please enter an integer
for the max time!!!", "ERROR", MB_OK); return;}

  timeCombo->Items->Clear();
  for (int i=0; i<MAX_TIME; i++)
    timeCombo->Items->Add(IntToStr(i));
}
//---------------------------------------------------------------------------

void __fastcall TForm1::clearbtn2Click(TObject *Sender)
{
  Memo2->Text = "";
}
//---------------------------------------------------------------------------
```