Department of Electrical
and
Computer Systems Engineering

Technical Report
MECSE-20-2003

Routing and Wavelength Assignment in GMPLS-baased DWDM Optical Networks:An OMNeT++ modelling Platform

LC. Cieutat and L.N. Binh

MONASH
UNIVERSITY

MONASH UIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER SYSTEMS ENGINEERING

MONASH UNIVERSITY

TECHNICAL REPORT

Routing and Wavelength Assignment in GMPLS-based Optical Networks:

An OMNeT++ modelling platform

By

L.N. Binh and Charles CIEUTAT

Groupe de Communications Fibre Optique et photonics Applique'

Department of Electrical and Computer Systems Engineering

Monash University

October 2003

SUMMARY

Given a set of connection requests in an all-optical Dense Wavelength Division Multiplexing network, this work aimed to investigate how to set up lightpaths and assign wavelengths in a manner which minimizes on average the blocking for the current and the future requests.

First, a thorough analysis of the research studies and different approaches in the literature to solve the routing and wavelength assignment problem in all-optical networks was conducted. This survey allowed to identify pertinent research studies and determinate the use of the Generalized Multi Protocol Label Switching framework for lightpath set-up in all-optical networks.

Based on this literature survey, a Routing and Wavelength Assignment scheme including a routing, a wavelength assignment and a reservation scheme was proposed. This scheme is composed of three parts: routing, wavelength assignment and reservation. First, the routing scheme was based on the implementation of OSPF and different associated drafts that extend OSPF capabilities for routing in all-optical networks. A proposal for an extension of the ISCD field in type 10 opaque LSA was given, allowing to consider two different metrics for the shortest path calculation based on Dijkstra algorithm. Secondly, two different wavelength assignment schemes, that is first-fit and random ,were implemented. Finally, a parallel reservation scheme was developed in order to examine the performance of the routing and wavelength assignment schemes previously mentioned.

The proposed routing and wavelength assignment scheme was developed using an object-oriented framework in order to facilitate further simulations and software reuse. The model was developed in C++ according to the object-oriented framework using the free simulation software OMNeT++.

Once the model developed, it was tested under different parameters, including total number of wavelengths per fibre and different routing and wavelength assignment schemes. The model was tested on the Abilene network, the research network linking U.S. universities that is composed of 12 gigaPoPs with an average degree of 3. The main test was to examine how the blocking of lightpath requests varies with the average link utilisation in the network. The main finding were:

- The probability of a request to be blocked is very low for link utilisation between 0 % and up to a threshold link utilisation varying between 33 % and 47 % depending on which scheme and parameters are used. At higher link utilisations, blocking increases very fastly when link utilisation increases.
- The enhanced TAW metric implemented performs better at link utilisations higher than the simple TAW metric particularly when the number of wavelengths per fibre is low. Conversely, at low link utilisation, both metrics have very similar results whatever the number of wavelengths are used.
- The wavelengths assignment schemes implemented, that is first-fit and random, yield extremely similar performance when associated with any routing and reservation schemes implemented in this work.

As a conclusion, the survey, the simulation and its key findings definitively open the way for further work on simulation of optical networking, such as taking into account wavelength conversion and more advanced reservation schemes and modelling of restorations capabilities of an optical network. Furthermore the transmission impairments of different paths will be taken into account in the routing assignment in our near future works.

NOMENCLATURE

ASE	Amplified Spontaneous Emission
CR-LDP	Constraint Routing Label Distribution Protocol
DWDM	Dense Wavelength Division Multiplexing
GMPLS	Generalized Multiprotocol Label Switching
IGP	Interior Gateway Protocol
ION	Intelligent Optical Network
ISCD	Interface Switching Capability Descriptor
IS-IS	Intermediate System to Intermediate System
LDP	Label Distribution Protocol
LSA	Link State advertisement
LSP	Label Switched Path
MPLS	Multiprotocol Label Switching
OSPF	Open Shortest Path First
OXC	Optical Cross Connect
PMD	Polarization Mode Dispersion
RSVP	Resource Reservation Protocol
RWA	Routing and Wavelength Assignment
TAW	Total and Available Wavelength
TE	Traffic Engineering
TED	Traffic Engineering Database
TLV	Type Length Value
WA	Wavelength Assignment
WCC	Wavelength Continuity Constraint
WDM	Wavelength Division Multiplexing

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

1.1. MOTIVATION

The Internet is growing extremely fast and demands always much more bandwidth capacity. In order to respond to the current and future demand with services such as, for instance, video on demand and voice over IP, operators have used the numerous advantages of DWDM in their optical long-haul transport backbone networks, bringing hundreds of Megabits of capacity on a single fibre between the main GigaPops of their network.

Nevertheless, operators are currently facing the problems of “thin” dumb optics. Dumb optics, such as DWDM point to point links, do not provide any channel networking. That means that up to now, configuration, management, protection and restoration of connections are still done mostly manually, adding a non negligible cost to the exploitation of an optical transport network and making the management of it a real burden.

Several concepts such as GMPLS and ION have emerged. Such approaches try to bring full transport functionalities with “smart” optics to an all-optical transport network. Especially, one key aspect of ION is its capability to manage end-to-end channels between two nodes.

Over the time, this complex issue was referred to as the RWA problem. The RWA problem is actually one of the major and more complex problem that researchers are faced within an ION, and its resolution is critical in order to respond to performance and quality of service issues.

1.2. AIMS

Up to now, there is no practical and efficient routing and wavelength assignment scheme that has been defined as a reference in order to find an optimal lightpath in multi-wavelengths DWDM network. On the other hand, to the author’s knowledge, there has not been any real simulation developed yet that allowed to test several schemes together, those schemes performing different functions, such as routing, wavelength assignment and reservation. Usually, each scheme is tested separately.

After a survey of the literature on RWA, the aim of this project was to simulate and model different RWA schemes that allow to efficiently establish end-to-end connections in an all-optical network. Especially, this project is expected to build the roots for a simulation platform of a GMPLS optical network. Such a platform would allow to study the influence of different parameters and different schemes onto the performance of connection set-up and further.

1.3. SUMMARY OF CONTRIBUTIONS

The most important contributions of this project are:

- An extensive literature survey on the RWA problem in all-optical networks and its implications and latest trends in the GMPLS framework

- The implementation of several schemes that appear at different stages of the RWA problem, such as routing algorithms based on different metrics, first-fit and random wavelength assignment and parallel reservation
- The design of an object-oriented framework for implementing these schemes, that will ease simulation software re-use and further work based on this model
- The implementation of a simulation model of a link-state routing algorithm in an all-optical network, giving near-optimal explicit route and wavelength assignments
- A demonstration of typical results which can be obtained with the model, such as the evolution of the blocking with the average link utilisation for different number of channels used in a fibre. This includes performance comparisons between the different schemes implemented under various network conditions.

1.4. REPORT ORGANIZATION

Section 2 – Introduces the project and its initial aims. Defines the RWA problem and its different parts. Gives an overview of the common approaches to solve the problem.

Section 3 – Introduces the theory of graphs and especially explains the principle of the Dijkstra algorithm, one of most well-known shortest path algorithm.

Section 4 – This is a very thorough survey of approaches to solve the RWA problem. First, a perspective of MPLS and its analogy in optical networks is given. This is followed by a survey of research studies on the RWA problem, which on the whole concentrates upon Dynamic Lightpath Establishment problem, which can be itself subdivided into three problems: routing, wavelength assignment and reservation. Finally, a comprehensive introduction of how GMPLS issues take place into the RWA problem is given.

SEction 5 – Based on the survey of Section 4, describes the principles of several schemes that have been chosen to be implemented. This includes a description of the experimental methodology and the hypotheses used. Finally, based on the remarks and analysis on the first part of this chapter, the second part concentrates on giving the guidelines of the implementation model that has been developed.

Section 6 – Describes the results of the implementation and discusses them. Especially, the results give an estimation of the blocking in the network for different average link utilisation in the ION and for different total number of channels per fibre.

Section 7 – Based on the literature survey of Section 4 and the conclusions of Section 6, gives recommendations for further work.

2. PROJECT OVERVIEW

In the following, an overview of the project is given. This includes an accurate definition of the RWA problem, a separation of the RWA problem in two distinct problems and a brief overview for common approaches used to solve the problem.

2.1. DEFINITION OF THE RWA PROBLEM

Over the last few years, DWDM has become the dominant technology for next generation optical networks. Using DWDM, multiple channels, distinguished by their wavelengths, can be transmitted on a single fibre, with each channel operating at its peak speed. Each wavelength of each fibre of a link is then a sub-channel that is completely independent of the other wavelengths of the same fibre. Such a concept is usually referred to as optical networking and ION, that is where the physical optical layer becomes aware of connections by identifying them thanks to their wavelengths. A typical topology of a DWDM network and its different associated channels, known as wavelengths, is given in Figure 1.

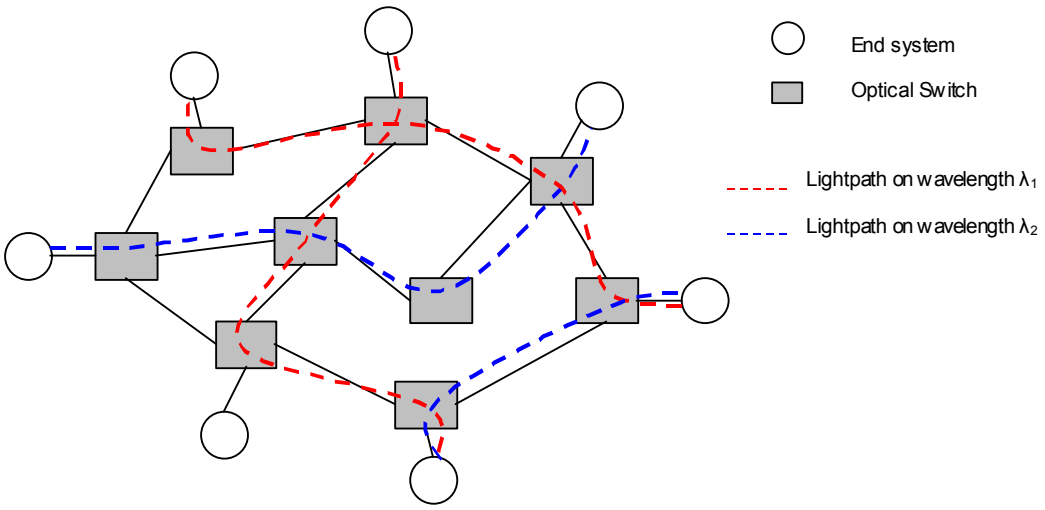


Figure 1 A wavelength-routed DWDM network

A route (a set of links) traversed by data between a source and a destination pair forms an all-optical path with a wavelength assigned on each link. Such a route is called a lightpath. Given a set of connection requests, how to set up lightpaths for them is called the RWA problem. Basically, the objective of an RWA algorithm is to set up lightpaths and assign wavelengths in a manner which minimizes on average the blocking for the current and future requests.

In the absence of wavelength conversion, a lightpath must use the same wavelength on all fibre links that it spans, which is known as the wavelength-continuity constraint. This constraint is unique to WDM networks, which may lead to inefficient wavelength utilization and degraded network performance.

To overcome this constraint, wavelength converters can be introduced at network nodes, which allow a wavelength to be optically converted to another wavelength. However, wavelength conversion technology is not mature yet. The cost of wavelength converters is still considerably high and is likely to remain as such in a short term. In this work, the use of wavelength conversion is not considered.

2.2. TYPES OF RWA PROBLEMS

Typically, there are two types of network traffic, either static or dynamic. Thus, the RWA problem is not one but double. This entails two kinds of lightpath establishment: the Static Ligthpath Establishment and the Dynamic Lightpath Establishment problems.

- **Static Lightpath Establishment (SLE)**

All connection requests are known in advance and do not change. That is, a request for setting up a set of optical paths is first given. These optical paths are not released once they are set up. The optical paths are assumed to be lightpaths, namely following the wavelength continuity constraint. The criterion of determining the best RWA is to minimize the number of wavelengths for a given network topology, the numbers of fibres, and the set of optical paths demanded.

- **Dynamic Lightpath Establishment (DLE)**

All connection requests arrive dynamically. The optimisation problem is to minimize the request blocking probability for a given number of wavelengths and/or to minimize the network cost.

The static lightpath establishment problem is more achievable with the current technology and would be a short term solution in an ION. But when the traffic in the core of the network will become too dynamic, dynamic lightpath establishment will have to be implemented in the ION. This work is to examine the DLE problem.

2.3. DIFFERENT APPROACHES FOR SOLVING THE PROBLEM

The RWA problem involves different parts, usually solved separately to simplify the problem. In the routing aspect, there are three basic types of routing approaches: fixed routing, fixed-alternate routing, and adaptive routing.

- In fixed routing, there is only one fixed route (e.g. the shortest path) between a pair of source and destination nodes.
- In fixed-alternate routing, each node maintains a routing table that contains an ordered list of fixed routes to each destination node. For example, these routes may include the first-shortest-path route, the second-shortest-path route, the third-shortest-path route, etc. The actual route for a connection request can only be chosen from this set of routes.
- In adaptive routing, routing is based on the current wavelength availability on each link. Any feasible route from the source node to the destination node can be a candidate as the actual route for a connection request. The choice of a route depends on the network policy used, such as the shortest-cost path first or the least-congested path first.

In general, fixed routing is the simplest while adaptive routing yields the best performance in terms of the request blocking probability. Fixed-alternate routing offers a trade-off between computing overhead and network performance.

The WA problem is the other part of the RWA problem. It is generally much easier that the routing problem, but depends also on the actual result of the routing solution. Nevertheless, it has usually repercussions on the performance results of the RWA algorithm when it is considered as a whole.

3. BACKGROUND THEORY

3.1.1. INTRODUCTION TO THE GRAPH THEORY

In the graph theory, a network topology can be represented as a graph G(V,E), where V denotes the set of vertices (network nodes) and E the set of edges (network links). |V| represents the number of nodes in the graph and is usually referred to as the magnitude of the graph. Each link $(i, j) \in E$ can be associated with a weight function $w_{i,j}$ which represents in a certain manner (to be defined) the cost of using the link (i,j). The degree of a node $i \in V$ is the number of neighbours of the node i.

Of special interest in this work is the problem of the shortest path in a graph. Given a graph G(V,E), the problem is to find the path in the graph that minimizes the sum of the weights of all the links taken between a two vertices. The graph theory has given numerous ways to solve that problem. The most commonly known algorithms are Bellman-Ford and Dijkstra algorithms [1]. In the following, an introduction to Dijkstra's algorithm is given.

3.1.1.1. Dijkstra's algorithm

Dijkstra's Algorithm, introduced in 1959 [2] provides one the most efficient algorithms for solving the shortest-path problem. It finds the shortest paths from a given source vertex s in V to each vertex v in V by developing the paths in order of increasing path lengths.

The Dijkstra algorithm can be formally described as follows. Let L(n) be the least cost path from vertex s to vertex n, with $i \in V$. The algorithm has 5 steps:

1. Set i=0, S_0= {u_0=s}, L(u_0)=0, and L(v)=infinity for v ≠ u_0. If |V| = 1 then stop, otherwise go to step 2.
2. For each v in V – S_i, replace L(v) by min(L(v), L(u_i)+ $w_{i,j}$). If L(v) is replaced, put a label (L(v), u_i) on v.
3. Find a vertex v which minimizes {L(v): v in V - S_i}, say u_{i+1}.
4. Let S_{i+1} = S_i U {u_{i+1}}.
5. Replace i by i+1. If i=|V|-1 then stop, otherwise go to step 2.

The time required by Dijkstra's algorithm is O($|V|^2$). For the beginner reader on the subject, a nice Java-based animation can explain very easily how the algorithm is practically working [3].

4. LITERATURE REVIEW

In the following, as a background information, an overview of the MPLS technology is presented. It explains its possible analogy to all-optical networking in an ION. Then, an extensive literature review of different schemes that have been elaborated to solve the RWA problem in a WDM network is depicted. Finally, how the RWA problem fits into the broader scope of the GMPLS framework is reviewed.

4.1. BACKGROUND INFORMATION

WDM networking has been launched by the concept of wavelength routing and optical networking. This is a critical milestone in the transport network evolution leading to the concept of a future third generation transport network based on all-optical networks.

There is currently a merging between the networking communities and the optics community. The Internet Engineering Task Force (IETF) is a community that provides drafts and standards (RFC) on the networking aspects of the Internet, of which the Internet Protocol (IP) is the fundamental basis. Such a merging appears because of the ever dramatically increasing ubiquity of the IP packets in next transport networks.

Since several years, one of the hot topics in IP-based backbone networks is MPLS. Taken apart the numerous value-added and brand-new services that it provides, MPLS is relevant in our context because of the label switching paradigm it provides. The label switching paradigm stems from the idea to merge the switching speed of circuit-switching technologies such as ATM and the survivability of an IP-based network.

The label switching paradigm used in MPLS is very similar to the concept of optical switching in all-optical networks issues. If the label is not any more an ordinary and abstract number that has local significance but a wavelength in an optical network, the lambda switching paradigm is born. This paradigm is now possible due to the availability of all-optical components such as OXC and Wavelength Converters that are able to manage the wavelength switching in one node of the ION. Those components do not execute any Optical / Electrical / Optical conversion, thus explaining the term “all-optical”.

If each node having this wavelength switching capability is now considered, the reunion of all nodes is referred to as an ION, where Wavelength Routing is possible. IONs acquired intelligence that do not reduce them to dumb point-to-point WDM optics any more. The ION is now an agile Optical Transport Network, able to manage and route the wavelengths, providing if necessary self healing and restoration capabilities. In the following, the key features of the MPLS technology are described. It is also explained more extensively what are the analogy with MPLS label switching and optical switching in an ION.

4.1.1. MPLS AND INTELLIGENT OPTICAL NETWORKS

4.1.1.1. A high level view of MPLS

If necessary, the novice reader can refer to a nice introduction to the MPLS main aspects found in [4]. MPLS is based on the following set of ideas:

- Forwarding information (label) separate from the content of IP header
- A single forwarding paradigm (label swapping), multiple routing paradigms

- Multiple link-specific realizations of the label swapping forwarding paradigm: “shim,” virtual connection/path identifier (VCI/VPI), frequency slot (wavelength), time slot
- The flexibility to form forwarding equivalence classes (FECs)
- A forwarding hierarchy via label stacking

The separation of forwarding information from the content of the IP header allows MPLS to be used with devices such as OXCs, which data plane cannot recognize the IP header. Label switch routers forward data using the label carried by the data. This label, combined with the port on which the data was received, is used to determine the output port and outgoing label for the data. The MPLS control plane operates in terms of the label swapping and forwarding paradigm abstraction. At the same time, the MPLS data plane allows multiple link-specific realizations of this abstraction. For example, a wavelength could be viewed as an implicit label.

Finally, the concept of a forwarding hierarchy via label stacking enables interaction with devices that can support only a small label space. This property of MPLS is essential in the context of OXCs and DWDMs since the number of wavelengths (which act as labels) is not very large.

4.1.1.2. Routing and label distribution protocols in the MPLS framework

The MPLS framework includes significant applications such as constraint-based routing. Constraint-based routing is a combination of extensions to existing IP link-state routing protocols (e.g., OSPF and IS-IS) with RSVP or CR-LDP as the MPLS control plane, and a Constrained Shortest-Path-First (CSPF) heuristic. The extensions to OSPF and IS-IS allow nodes to exchange information about network topology, resource availability and even policy information.

This information is used by the CSPF heuristic to compute paths subject to specified resource and/or policy constraints. For example, either RSVP-TE or CR-LDP is used to establish the label forwarding state along the routes computed by a CSPF-based algorithm ; this creates the LSP. The MPLS data plane is used to forward the data along the established LSPs. Constraint-based routing is used today mainly for two main purposes: traffic engineering and fast reroute.

4.1.1.3. Towards a simpler protocol stack: IP/MPLS over DWDM

With suitable network design, the constraint-based routing of IP/MPLS can replace ATM as the mechanism for traffic engineering. Likewise, fast reroute mechanisms offers an alternative to SONET as a mechanism for protection/restoration. Both traffic engineering and fast reroute are examples of how enhancements provided by MPLS to IP routing make it possible to bypass ATM and SONET/SDH by migrating functions provided by these technologies to the IP/MPLS control plane.

4.1.1.4. The analogy between MPLS and Optical Networks

Paving a path for future evolution of MPLS technologies are several emerging synergies between Label Switch Routers used in MPLS and photonic switches, and between an LSP and an optical path or lightpath. A lightpath is an end-to-end path composed exclusively of photonic elements without optical-electronic conversions. Analogous to switching labels in an LSR, a photonic switch toggles wavelengths from an input to an output port. Establishing an LSP involves configuring each intermediate LSR to map a particular input label and port to an output label and port. Similarly, the process of establishing a ligthpath involves configuring each intermediate photonic switch to map a particular input wavelength and port to an output wavelength and port.

LSRs and photonic switches need routing protocols like OSPF or IS-IS to exchange link-state topology and other optical resource availability information for path computation. They also need signalling protocols like RSVP and LDP to automate the path establishment process.

The important is that MPLS allows to separate logically the network into a control plane and a forwarding plane. The forwarding plane is responsible of switching information using a very simplified switching table (analogous to ATM but with increased capabilities). On the other hand, the control plane task is to manage the resources of the network. Especially, the control plane is to reserve resources on an end-to-end basis between the ingress and the egress nodes ; this includes the set-up and the turn-down of the optical path. A summary of the different schemes used in different layers is given in Table 1.

	Control Plane	Data Plane
IP	Routing layer: OSPF, IS-IS and BGP.	Forwarding layer: IP.
MPLS	Binding layer: CR-LDP or RSVP-TE.	Forwarding layer: MPLS.
OTN	λ Mapping layer: LMP or GMPLS or a combination of the two ?	λ Forwarding layer: wavelength

Table 1: Protocols and mechanisms in IP, MPLS, OTN

4.1.1.5. Interworking and managing the three control planes

[5] presents how it is possible to benefit from the advantages of each layer technology, i.e. the route discovery capabilities of the IP control plane, the Traffic Engineering capabilities of the MPLS control plane and the forwarding speed of the ION control plane.

A gradually more accepted idea is to couple the three control planes by implementing another layer, as a Management plane. At this stage, there are no common agreement on how the different control planes will interact with each other, that is no consensus has been reached on how the operations of extended OSPF, IS-IS, extended BGP, RSVP-TE, CR-LDP, LMP, GMPLS, the OIF UNI and NNI will be executed.

Nevertheless, the IP community and the optical network community have agreed that the control plane responsible of the management of the optical network layer will be based on the GMPLS framework. Before delving into the GMPLS framework, of which the RWA problem is only one part, albeit a big part, a review of the research studies on the RWA is conducted.

4.2. RESEARCH STUDIES ON THE RWA PROBLEM

In the following, a thorough review of the research studies concerning the SLE and DLE problems that constitute the two sub problems of the RWA problem is presented.

4.2.1. RWA FOR STATIC LIGHTPATH ESTABLISHMENT (SLE)

4.2.1.1. Dissociating the routing problem from the WA problem

A number of studies have investigated the RWA problem for setting up a static set of lightpaths [6] [7]. These studies formulate the problem using integer linear program (ILP) formulations, or rely on heuristic approaches in an attempt to minimize the number of wavelengths required to establish a given set of lightpaths. The ILP formulations are NP-complete and therefore may only be solved for very small systems. For larger systems, heuristic methods must be used. More generally, it is a fact that the RWA problem is an intricate problem that can be solved practically only with heuristics.

For instance, in [7], the routing problem is formulated as an ILP in which the objective is to minimize the number of wavelengths required to establish a fixed set of lightpaths. The search space of the problem is reduced by restricting the set of links through which a lightpath for a given source-destination pair may traverse. The resulting ILP is then solved by relaxing the integer constraint, solving the resulting non-integer linear program, and then utilizing a randomised rounding approach on the result to obtain an integer solution.

Other heuristics consider only alternate shortest-hop paths between a source-destination pair, and choose one of the paths according to a predefined policy. In [8], a shortest-hop path is randomly chosen for each source-destination pair. Each source-destination pair is then considered individually, and the route for the pair of nodes is switched to an alternate shortest-hop path if doing so results in a reduction of load on the most heavily loaded link in the original shortest-path route.

In [9], an approach similar to that in [7] is considered; however, the objective is to minimize the number of fibres in a multifibre network, and the set of alternate paths includes routes which may be longer than the shortest-hop routes. Quite satisfying solutions to the RWA problem are obtained most of the time by finding a lightpath that is not the shortest-hop path.

4.2.1.1.1 Wavelength assignment sub-problem

The wavelength-assignment sub-problem of the RWA problem can itself be formulated as a graph colouring problem, which is also NP-complete. Greedy heuristics for the wavelength-assignment problem for a static set of lightpaths typically involve ordering the wavelengths, and assigning the same wavelength to as many lightpaths as possible before moving on to the next wavelength [6]. Also, the set of lightpaths may be ordered by length, such that wavelengths are assigned to longer lightpaths before wavelengths are assigned to shorter lightpaths. All those heuristics are extremely CPU-intensive and they are unpractical solutions to be implemented because of their usual complexity.

4.2.1.2. Solving the routing problem and the WA problem simultaneously

[10] focuses on static lightpath assignment. That is, a request for setting up a set of optical paths is first given. The criterion of determining the RWA is to minimize the number of wavelengths for a given network topology, the numbers of fibres, and the set of optical paths demanded.

[10] shows that the routing problem and the wavelength assignment problem can be solved simultaneously by employing a multi-commodity flow model, which has been comprehensively studied in the literature. On the basis of this notion, a heuristic routing and wavelength assignment algorithm is proposed. Through numerical examples, the proposed algorithm is compared with conventional algorithms that run under the same criteria. It is shown to run better that the algorithm presented in [7].

Even though solutions have been found to solve the SLE problem, they mostly rely on complex heuristics and thus are not well suited for any practical implementations. Furthermore, the SLE problem does not consider any dynamic traffic. This may be suitable for today's backbones ; but in the very near future, automation of lightpath management responding to needs for dynamic traffic must be provided. This is the object of DLE schemes that is presented in the following.

4.2.2. RWA FOR DYNAMIC LIGHTPATH ESTABLISHMENT (DLE)

When lightpaths are established and taken down dynamically, RWA decisions must be made as connection requests arrive to the network. It is possible that, for a given connection request, there may be insufficient network resources to set up a lightpath, in which case the connection request will be blocked. The connection may also be blocked if there is no common wavelength available on all of the links along the chosen route (Wavelength Continuity Constraint also known as WCC).

Thus, the objective in the dynamic situation is to choose a route and a wavelength which maximizes the probability of setting up a given connection, while at the same time attempting to minimize the blocking for future connections. Similar to the case of static lightpaths, the dynamic RWA problem can also be decomposed into a routing sub problem and a corresponding wavelength assignment sub problem. Approaches to solving the routing sub problem can be categorized as being either fixed or adaptive, and as utilizing either global or local network state information.

4.2.2.1. Fixed routing

In fixed routing, a single fixed route is predetermined for each source-destination pair. When a connection request arrives, the network will attempt to establish a lightpath along the fixed route. If no common wavelength is available on every link in the route, then the connection will be blocked.

A fixed routing approach is simple to implement; however, it is very limited in terms of routing options and may lead to a high level of blocking. In order to minimize the blocking in fixed routing networks, the predetermined routes need to be selected in a manner which balances the load evenly across the network links. Fixed routing schemes do not require the maintenance of global network state information.

4.2.2.2. Adaptive Routing Based on Global Information

Adaptive routing approaches increase the likelihood of establishing a connection by taking into account network state information. For the case in which global information is available, routing decisions may be made with full information as to which wavelengths are available on each link in the network.

4.2.2.2.1 Centralized Versus Distributed Routing

Adaptive routing based on global information may be implemented in either a centralized or distributed manner. In a centralized algorithm, a single entity, such as a network manager, maintains complete network state information, and is responsible for finding routes and setting up lightpaths for connection requests. Since a centralized entity manages the entire network, there does not need to be a high degree of coordination among nodes; however, a centralized entity becomes a possible single point of failure. Furthermore, a centralized approach does not scale well, as the centralized entity would need to maintain a large database to manage all nodes, links, and connections in the network.

4.2.2.2.2 Fixed-alternate-Path Routing

One approach to adaptive routing with global information is alternate-path routing. Alternate-path routing relies on a set of predetermined fixed routes between a source node and a destination node [11] [12] [13] [14]. When a connection request arrives, a single route is chosen from a set of predetermined routes, and a lightpath is established on this route. The criteria for route selection is typically based on either path length or path congestion.

a) Route selection based on path length

An example of a routing algorithm based on path length is the K-shortest paths algorithm [11], in which the first K shortest paths are maintained for each source-destination pair, and the paths are selected in order of length, from shortest to longest. A connection is shortest attempted on the shortest path. If resources are not available on this path, the next shortest path is attempted.

b) Route selection based on path congestion

A path selection policy based on path congestion examines the available resources on each of the alternate paths, and chooses the path on which the highest amount of resources are available.

c) Comparison of path congestion and path length-based route selection

Choosing the shortest-path route consumes less network resources, but may lead to high loads on some of the links in the network, while choosing the path with the least congestion leads to longer paths, but distributes the load more evenly over the network.

4.2.2.2.3 Unconstrained Routing

Another adaptive routing approach utilizing global information is unconstrained routing which considers all possible paths between a source node and a destination node. In order to choose an optimal route, a cost is assigned to each link in the network based on current network state information, such as wavelength availability on links. A least-cost routing algorithm is then executed to find the least-cost route [15] [16] [17]. Whenever a connection is established or taken down, the network state information is updated.

Two examples of unconstrained routing approaches are link-state routing and distance-vector routing.

a) Link-state routing

In a distributed link-state routing approach, each node in the network must maintain complete network state information [16]. Each node may then find a route for a connection request in a distributed manner. Whenever the state of the network changes, all of the nodes must be informed. Therefore, the establishment or removal of a lightpath in the network may result in the broadcast of update messages to all nodes in the network. The need to broadcast update messages may result in significant control overhead. Furthermore, it is possible for a node to have outdated information, and for the node to make an incorrect routing decision based on this information.

Some efforts have been made to enhance common shortest path algorithms using link-state routing. In [18], several different enhanced links weights especially designed for WDM networks are defined and their performance are compared. It is showed that as a rule of thumb, a metric based on using lowest hop count and combination of available and total number of wavelengths results in the lowest blocking probability.

In the following, an enhanced metrics that can be used in the Dijkstra algorithm is presented [18]. Let be $\lambda^{a}_{i,j}$ the number of available (unused) wavelengths on a link and $\lambda^{T}_{i,j}$ the total number of possible wavelengths on that link.

a1). Total and available wavelengths (TAW)

Let be $w_{i,j}$ the weight of a link (i,j). In TAW, $w_{i,j}$ is defined as the following:

$$w_{i,j} = -\log\left[1-\left(1-\frac{\lambda^{a}_{i,j}}{\lambda^{T}_{i,j}}\right)^{\lambda^{a}_{i,j}}\right] \quad \forall (i,j) \in E^2 \qquad (1)$$

Let be p the probability that a wavelength is used on one link. If $\lambda^{a}_{i,j}$ and $\lambda^{T}_{i,j}$, are known, p can be estimated:

$$p = \frac{\lambda^{T}_{i,j} - \lambda^{a}_{i,j}}{\lambda^{T}_{i,j}} = 1 - \frac{\lambda^{a}_{i,j}}{\lambda^{T}_{i,j}} \qquad (2)$$

Then, the probability that all wavelengths will be used in the future can be written as $p^{\lambda^a_{i,j}}$. Then, the probability that at least one wavelength is available on the link in future is given by $1-p^{\lambda^a_{i,j}}$. For a route composed of multiple links, the goal is to maximise the probability that one wavelength will be available in future, ie maximise the product of $1-p^{\lambda^a_{i,j}}$ that constitute the possible route. Due to the additive characteristic of the Dijkstra algorithm, maximising the probability of an available wavelength is equivalent to minimizing the value because of the following relation $-\log\left[\prod_{i,j}\left(1-p^{\lambda^a_{i,j}}\right)\right]=\sum_{i,j}\left[-\log\left(1-p^{\lambda^a_{i,j}}\right)\right]$.

b) Distance-vector routing

A distance-vector approach to routing with global information is also possible [17]. This approach doesn't require that each node maintains complete link-state information at each node as in [16], but instead has each node maintain a routing table which indicates for each destination and on each wavelength, the next hop to the destination and the distance to the destination. The approach relies on a distributed Bellman-Ford algorithm to maintain the routing tables. Similar to [16], the scheme also requires nodes to update their routing table information whenever a connection is established or taken down. This update is accomplished by having each node send routing updates to their neighbours periodically or whenever the status of the node's outgoing links changes. Although each node maintains less information than in [16] and the updates are not broadcast to all nodes, the scheme may still suffer from a high degree of control overhead.

An interesting approach to distance-vector algorithms is also that, in some way, it can perform a constraint routing based on the number of hops and any other kind of metric [19]. Indeed, it should be noted that standard routing algorithms are typically single objective optimisations, i.e., they may minimize the hop-count, or minimize any other kind of metric, but not both. Double objective path optimisation is a more complex task, and, in general, it is an intractable problem.

The Bellman-Ford shortest path algorithm is adapted to compute paths of a minimum metric for all hop counts. It is a property of the BF algorithm that, at its h-th iteration, it identifies the optimal path between the source and each destination, among paths of at most h hops. However, because the BF algorithm progresses by increasing hop count, it essentially provides for free the hop count of a path as a second optimisation criteria. This can be an especially interesting feature if the objective is to find the shortest path (with a certain metric) while still finding a ‘relatively’ short path, i.e. a path that also minimizes the hop count.

c) Comparison between distance-vector and link-state routing in the RWA problem

When it is possible to handle global knowledge of the network, distance-vector and link-state routing are two possibilities. Nevertheless, the choice of either algorithm entails very different behaviours when considering the RWA problem. In [20], the performance of the two approaches for solving dynamic lightpath establishment is studied. Major results are that link-state outperforms distance-vector algorithms for shorter stabilizing delays and lower blocking at low loads. Distributed routing yields lower blocking probability under high loads.

Nevertheless, the principal drawback of distance-vector algorithms is that they are not suited as much as link-state algorithms when considering traffic engineering issues. Particularly, the major advantage of link-state algorithms is that each node has a global knowledge of the network. This makes it very easy to find explicit routes from a source node to a destination node, thus adding more fault tolerance to the network. For instance, it is possible to add restoration capacities when nodes have full knowledge of the network.

Although routing schemes based on global knowledge must deal with the task of maintaining a potentially large amount of state information which is changing constantly, these schemes often make the most optimal routing decisions if the state information is up to date. Thus, global-knowledge based schemes may be well suited for networks in which lightpaths are fairly static and do not change much with time.

4.2.2.3. Adaptive Routing Based on Local Information

While near-term emerging systems will be fairly static, with lightpaths being established for long periods of time, it is expected that, as network traffic continues to scale up and become more bursty in nature, a higher degree of multiplexing and flexibility will be required at the optical layer. Thus, lightpath establishment will become more dynamic in nature, with connection requests arriving at higher rates, and lightpaths being established for shorter time durations.

In such situations, certain people estimated that maintaining distributed global information may become infeasible. The alternative is to implement routing schemes which rely only on local information. A number of adaptive routing schemes exist which rely on local information rather than global information. The advantage of using local information is that the nodes do not have to maintain a large amount of state information; however, routing decisions tend to be less optimal than in the case of global information. Two examples of local-information-based adaptive routing schemes are alternate routing with local information, and deflection routing.

4.2.2.3.1 Alternate-Path Routing with Local Information

While alternate-path routing schemes typically rely on global information, variations exist which utilize only local information. A least-congested alternate path routing scheme is investigated in [21]. In this scheme, the choice of a route is determined by the wavelength availability along the alternate paths. Two variations of the scheme are considered: the case in which wavelength availability information is known along the entire path, and the case in which only local information is available.

a) End-to-end wavelength knowledge

In the first approach, the decision making entity is aware of the wavelength availability information for all of the links in each of the alternate paths. In this case, the chosen route is that which has the greatest number of wavelengths which are available along all of the links in its path. For example, in Figure 2, if two alternate routes from source node A to destination node D are considered, with available wavelengths as shown on each link, then two wavelengths (λ_1 and λ_3) are available along the entire length of route 1, while only one wavelength (λ_2) is available along the entire length of route 2; thus, route 1 will be chosen.

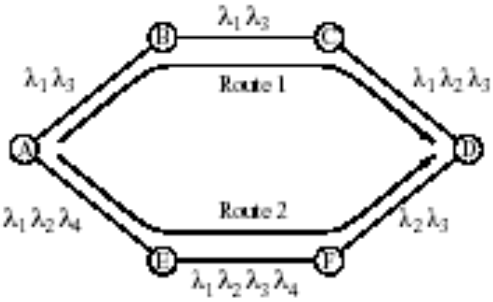


Figure 2: Alternate routing. Available wavelengths are shown on each link.

The limitation of basing the route selection decision on full path information is that the information may be difficult to maintain or difficult to obtain in a timely manner. Each node would be required to either maintain complete state information, or the information would need to be gathered in real time, as the lightpath is being established.

b) Partial wavelength knowledge

The alternative, based on local information, is to gather wavelength availability information only along the shortest k hops of each path. The route is then chosen based on which path is the least congested along its shortest k hops. In Figure 2, if k = 2, then route 2 would be chosen, since it has three wavelengths available on the shortest two links (λ_1; λ_2, and λ_4), while route 1 only has two wavelengths available on the shortest two links (λ_1 and λ_3).

Although local information may provide a good estimate of the congestion along a path, it does not guarantee that any particular wavelength will be available along the entire path; thus, it is possible that after choosing a route, the connection will still be blocked due to lack of available wavelengths.

4.2.2.3.2 Deflection Routing

Another approach to adaptive routing with limited information is deflection routing, or alternate-link routing [22]. This routing scheme chooses from alternate links on a hop-by-hop basis rather than choosing from alternate routes on an end-to-end basis. The routing is implemented by having each node maintain a routing table which indicates, for each destination, one or more alternate outgoing links to reach that destination. These alternate outgoing links may be ordered such that a connection request will preferentially choose certain links over other links as long as wavelength resources are available on those links.

Other than a static routing table, each node will only maintain information regarding the status of wavelength usage on its own outgoing links. When choosing an outgoing link for routing, the decision can be determined on either a shortest-path or least-congested basis.

a) Shortest path deflection routing scheme

Under the shortest path criteria, the routing scheme will shortest attempt to choose the outgoing link which results in the shortest path to the destination. If there is no feasible wavelength available on the link, then the routing scheme will attempt to choose an alternate outgoing link which leads to the next shortest path to the destination. The routing scheme proceeds in this manner until the destination is reached or the connection is blocked.

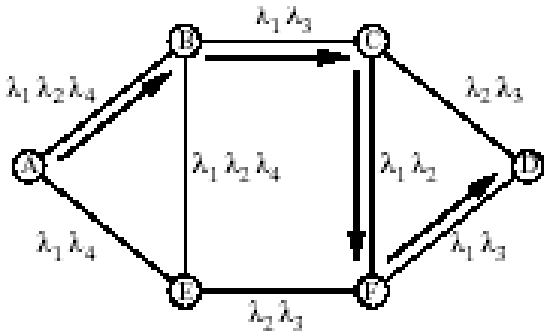


Figure 3: Shortest path deflection routing scheme

Figure 3 illustrates the deflection routing scheme for a connection request from node A to node D. The default shortest path in this example is along the path A→B→C→D. When the request reaches node C, it cannot continue over link CD, since no common wavelength is available on links AB, BC, and CD. The request is therefore deflected to node F, where it can continue to the destination along link FD. The wavelength selected for the lightpath will be λ_1.

Note that, in the absence of any deflections, the default routing for any connection will be shortest-path routing. Also, once the routing for a lightpath is deflected at a node, the default routing from the point of deflection onward will again be shortest-path routing if no further deflections take place.

b) Least-congested deflection routing scheme

In a least-congested deflection routing approach, the routing scheme chooses, from among the alternate outgoing links, the link which has the largest number of feasible wavelengths. The set of feasible wavelengths consists of the set of wavelengths which are available on all of the previous hops as well as the next outgoing link.

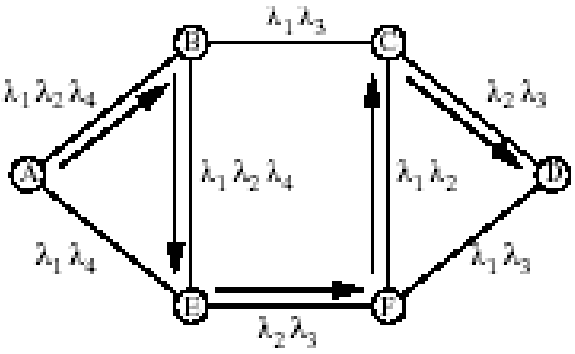


Figure 4: least-congested deflection routing scheme

Least-congested deflection routing is illustrated in Figure 4 for a connection from node A to node D. On the shortest hop, link AB is selected, since it has three available wavelengths, while link AE has only two available wavelengths. When the connection request arrives to node B, it will be routed to node E, since there are three feasible wavelengths (λ_1; λ_2; and λ_4) available on link BE, and there is only one feasible wavelength (λ_1) available on link BC.

The least-congested deflection routing approach will generally result in longer paths than the shortest-path deflection routing approach; however, least-congested deflection will allow a lightpath to be routed around congested areas in the network, balancing the load more evenly across the network. The results in [22] show that a shortest-first policy results in lower blocking at low loads, while a least-congested policy results in lower blocking at higher loads.

c) Issues in deflection routing

A number of issues arise when implementing a deflection routing scheme. One such issue is the problem of looping, in which a connection request message returns to a node which has already been visited. Loop detection may be addressed by having each connection request message maintain a path vector containing a list of visited nodes. If a node receives a connection request message which indicates that the message has already visited this node, then the connection attempt will be blocked. An alternative to maintaining a path vector is to utilize a time-to-live field, which would prevent the connection request message from looping in the network indefinitely.

Another problem which may arise is that a connection request may be deflected a large number of times, leading to an unreasonably long route for the lightpath. Possible solutions to this problem include limiting the maximum length or number of hops in a lightpath, or limiting the number of deflections that a route can take. When a connection request message reaches its limit on the maximum number of hops or deflections, the connection attempt will be blocked. Further restrictions may also be placed on the selection of possible outgoing ports in order to prevent routes from heading back towards the source node.

4.2.2.4. Summary

A summary of the different kinds of routing algorithms in the RWA problem are presented in Table 2.

Routing classification			Description	Advantages	Drawbacks
Fixed routing			A single fixed route is predetermined for each source-destination pair	Simple to implement. No state information needed.	limited in terms of routing options. High blocking.
Adaptive routing (global information).	Alternate path	Path length	A set of fixed routes are predetermined for each source node and a destination node. The criteria for route selection is typically based on either path length or path congestion. Path length may lead to high loads on some of the links in the network. Path congestion leads to longer paths, but distributes the load more evenly over the network.	Probability of blocking is smaller than for fixed routing. Simplicity. Optimal routing decision. Well suited for networks in which lightpaths are fairly static and do not change much with time.	Global information needed (high load in the network).
		Path congestion			
	Unconstrained	Link state routing	A cost is assigned to each link in the network based on current network state information, such as wavelength availability on links. A least-cost routing algorithm is then executed to find the least-cost route. Contrary to link-state routing, vector-distance doesn't require that each node maintains complete link-state information.	Probability of blocking is smaller than for fixed routing. Simplicity. Optimal routing decision. Well suited for networks in which lightpaths are fairly static and do not change much with time.	Global information heeded (high load in the network).
		Vector distance routing			
Adaptive routing (local information).	Alternate path routing. Each node maintains λ usage at least for k hops.	End-to-end λ knowledge.	The criteria for choosing routes is based on wavelength availability. Wavelength availability can be known completely or partially. In partial wavelength knowledge,	Local information. More adapted for fast changing networks with very dynamic lightpaths CAC.	Routing decision is not necessarily optimal. Possible blocking due to non λ

		Partial λ knowledge.	information is only gathered along the first k hops of each path. The route is then chosen based on which path is the least congested along its shortest k hops		avaibility.
	Deflection (or alternate-link) routing. For each node, status of wavelength usage only on its own outgoing links. Least-congested results in longer paths than the shortest-path deflection routing approach but provides better load balancing. Shortest-first policy results in lower blocking at low loads, while a least-congested policy results in lower blocking at higher loads.	Shortest path	Choose the outgoing link which results in the shortest path to the destination. If there is no feasible wavelength available on the link, choose an alternate outgoing link which leads to the next shortest path to the destination. Proceed in this manner until the destination is reached or the connection is blocked. In the absence of any deflections, the default routing for any connection will be shortest-path routing	Local information. More adapted for fast changing networks with very dynamic lightpaths CAC.	Possible looping. May return excessively long routes for the lightpath.
		Least congested	Choose the outgoing link which has the largest number of feasible wavelengths. The set of feasible wavelengths consists of the set of wavelengths which are available on all of the previous hops as well as the next outgoing link.		

Table 2: Routing algorithms in the RWA problem

4.2.3. WAVELENGTH ASSIGNMENT

In general, if there are multiple feasible wavelengths between a source node and a destination node, then a wavelength assignment algorithm is required to select a wavelength for a given lightpath. The wavelength selection may be performed either after a route has been determined, or in parallel with finding a route.

Since the same wavelength must be used on all links in a lightpath, it is important that wavelengths are chosen in a way which attempts to reduce blocking for subsequent connections. A review of wavelength-assignment approaches can be found in [23].

4.2.3.1. First-fit WA heuristic

One example of a simple but effective wavelength-assignment heuristic is first-fit. In first-fit, the wavelengths are indexed, and a lightpath will attempt to select the wavelength with the lowest index before attempting to select a wavelength with a higher index. By selecting wavelengths in this manner, existing connections will be packed into a smaller number of total wavelengths, leaving a larger number of wavelengths available for longer lightpaths.

4.2.3.2. Random WA heuristic

Another approach for choosing between different wavelengths is to simply select one of the wavelengths at random. In general, first-fit will outperform random wavelength assignment when full knowledge of the network state is available [12]. However, if the wavelength selection is done in a distributed manner, with only limited or outdated information, then random wavelength assignment may outperform first-fit assignment. The reason for this behaviour is that, in a first-fit approach, if multiple connections are attempting to set up a lightpath simultaneously, then it may be more likely that they will choose the same wavelength, leading to one or more connections being blocked.

4.2.3.3. Most-used and least-used WA heuristic

Other simple wavelength assignment heuristics include the most-used-wavelength heuristic and the least-used-wavelength heuristic. In most-used wavelength assignment, the wavelength which is the most used in the rest of the network is selected. This approach attempts to provide maximum wavelength reuse in the network. The least-used approach attempts to spread the load evenly across all wavelengths by selecting the wavelength which is the least-used throughout the network. Both most-used and least-used approaches require global knowledge.

4.2.3.4. More advanced WA heuristics

A number of more advanced wavelength assignment heuristics which rely on complete network state information have been proposed [24] [25]. It is assumed in these heuristics that the set of possible future lightpath connections is known in advance. For a given connection, the heuristics attempt to choose a wavelength which minimizes the number of lightpaths in the set of future lightpaths that will be blocked by this connection. It is shown that these heuristics offer better performance than first-fit and random wavelength assignment.

4.2.3.5. Summary

A summary of the different kinds of WA algorithms in the RWA problem are presented in Table 3.

WA heuristics	Characteristics	Advantage	Drawback
First-fit	The wavelengths are indexed, and a lightpath will attempt to select the wavelength with the lowest index before attempting to select a wavelength with a higher index	If global information, outperforms random heuristics.	Possible blocking if simultaneous lightpath connections.
Random	Select one of the wavelengths at random	If local information, outperforms first-fit heuristics. Very simple.	Does not provide optimal wavelength assignment.
Least-used	The wavelength which is the most used in the rest of the network is selected.	Spreads the load evenly across all wavelengths	Global information needed.
Most-used	The wavelength which is the most used in the rest of the network is selected.	Provides maximum wavelength reuse in the network	Global information needed.

Table 3 WA algorithms in the RWA problem

4.2.4. SIGNALLING AND RESOURCE RESERVATION

In order to set up a lightpath, a signalling protocol is required to exchange control information among nodes and to reserve resources along the path. In many cases, the signalling protocol is closely integrated with the RWA algorithms.

Signalling and reservation protocols may be categorized based on whether the resources are reserved on each link in parallel, reserved on a hop-by-hop basis along the forward path, or reserved on a hop-by-hop basis along the reverse path. Algorithms will also differ depending on whether global information is available or not.

4.2.4.1. Parallel Reservation

In [16], the control scheme reserves wavelengths on multiple links in parallel. The scheme, which is based on link-state routing, assumes that each node maintains global information on the network topology and on the current state of the network, including information regarding which wavelengths are being used on each link. Based on this global information, the node can calculate an optimal route to a destination on a given wavelength. The source node then attempts to reserve the desired wavelength on each link in the route by sending a separate control message to each node in the route. Each node that receives a reservation request message will attempt to reserve the specified wavelength, and will send either a positive or negative acknowledgement back to the source. If the source node receives positive acknowledgements from all of the nodes, it can establish the lightpath and begin communicating with the destination.

Nevertheless, [16] does not provide any details of the routing algorithm used. Also, it uses periodical acknowledgements which can put a severe burden on the network. This paper was also first aimed at ATM networks ; so it does not take into account the RWA problem into a broader and deeper framework that GMPLS embraces for IP networks.

The advantage of a parallel reservation scheme is that it shortens the lightpath establishment time by having nodes process reservation requests in parallel. It is also simpler to implement that other reservation schemes such as hop by hop reservation detailed in the next paragraph. The disadvantage is that it requires global knowledge, since both the path and the wavelength must be known ahead of time.

4.2.4.2. Hop-by-Hop Reservation

An alternative to parallel reservation is hop-by-hop reservation in which a control message is sent along the selected route one hop at a time. At each intermediate node, the control message is processed before being forwarded to the next node. When the control message reaches the destination, it is processed and sent back towards the source node. The actual reservation of link resources may be performed either while the control message is travelling in the forward direction towards the destination, or while the control message is travelling in the reverse direction back towards the source.

4.2.4.2.1 Forward Reservation

In forward reservation schemes, wavelength resources are reserved along the forward path to the destination on a hop-by-hop basis. The method of reserving wavelengths depends on whether or not global information is available to the source node.

a) Global knowledge of wavelength usage

If the source node is maintaining complete state information, then it will be aware of which wavelengths are available on each link. Assuming that the state information is current, the source node may then send a connection setup message along the forward path, reserving the same available wavelength on each link in the path.

b) Local knowledge of wavelength usage

For the case in which a node only knows the status of its immediate links, the wavelength selection becomes more complicated, as the source node doesn't know which wavelength will be available along the entire path.

b1). Conservative reservation scheme

The source node may utilize a conservative reservation approach, choosing a single wavelength and sending out a control message to the next node attempting to reserve this wavelength along the entire path; however, there is no guarantee that the selected wavelength will be available along every link in the path. If the wavelength is blocked, the source node may select a different wavelength and reattempt the connection. The limitation of this approach is that it may result in high setup times, since it may take several attempts before a node can establish a lightpath.

b2). Aggressive reservation scheme

An alternate approach to maximizing the likelihood of establishing a lightpath in a forward reservation scheme is to use an aggressive reservation scheme which over-reserves resources [26]. Multiple wavelengths may be reserved on each link in the path, with the expectation that at least one wavelength will be available on all links in the path. In a greedy approach, all feasible wavelengths will be reserved at every link in the path. The source node will first reserve all available wavelengths on the desired outgoing link. A connection request message containing the wavelength reservation information is then sent to the next node along the path.

At each intermediate node, the subset of wavelengths consisting of the intersection of the wavelengths reserved on the previous link and the wavelengths available on the next link will be reserved. For example, if S_1 is the set of wavelengths available on the nth link, then the set of wavelengths reserved on the first link in the path will be S_1, the set of wavelengths reserved on the second link will be $S_1 \cap S_2$, the set of wavelengths reserved on the third link would be $S_1 \cap S_2 \cap S_3$, etc. When the connection request reaches the destination, one wavelength of the remaining set of wavelengths will be chosen, and all of the other wavelengths will be released.

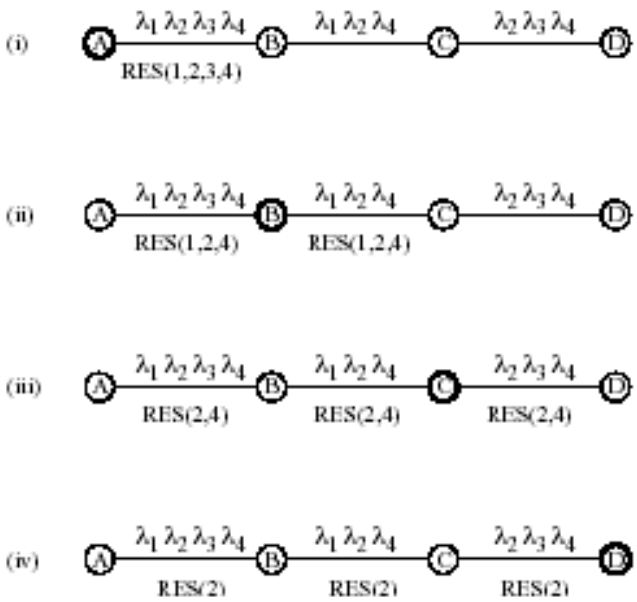


Figure 5: Forward reservation

Figure 5 illustrates the forward reservation of wavelengths when establishing a lightpath from node A to node D. As the control message propagates from A to D, each node reserves the set of wavelengths which have been available on all links traversed by the control message. One disadvantage of over-reserving resources is that, during the time that the resources are reserved, the reserved resources cannot be utilized by other users, even if these resources will never be used by the connection.

In order to reduce the amount of time that an unused wavelength is reserved on a link, the wavelength may be released as soon as it is apparent that the wavelength is not viable for a given connection. For example, if wavelengths λ_1; λ_2, λ_3; and λ_4 are available on the first link, then all four of the wavelengths will be reserved on this link. However, if it is subsequently discovered that only λ_1, λ_2, and λ_4 are available on the second link, then not only will λ_1, λ_2, and λ_4 be reserved on the second link, but λ_3 will immediately be released on the first link.

4.2.4.3. Aggressive with wavelength group reservation scheme

Another approach to limiting the number of wavelengths that are reserved is to divide the wavelengths into groups. When reserving wavelengths on a link, a node will reserve only those wavelengths which belong to a specific group [27]. The choice of the group is made at the source node and is based on the number of available wavelengths in each group. The source node will find the group with the largest number of available wavelengths, and the node will reserve all of the available wavelengths in that group before sending the request on to the next node. The size of the group is a critical parameter. If the group is too large, then too many resources will be reserved, but if the group is too small, then the likelihood of establishing a lightpath will be smaller.

4.2.4.3.1 Backward Reservation

To prevent the over-reservation of resources altogether, reservations may be made after the control message has reached the destination and is headed back to the source. Such reservation schemes are referred to as backward reservation schemes [26].

By reserving wavelengths in the reverse direction, the reserved wavelengths are idle for less time than if the wavelengths are reserved in the forward direction. Another advantage is that the connection request message can gather wavelength usage information along the path in the forward direction. This information can then be used by the destination node to select an appropriate wavelength to reserve.

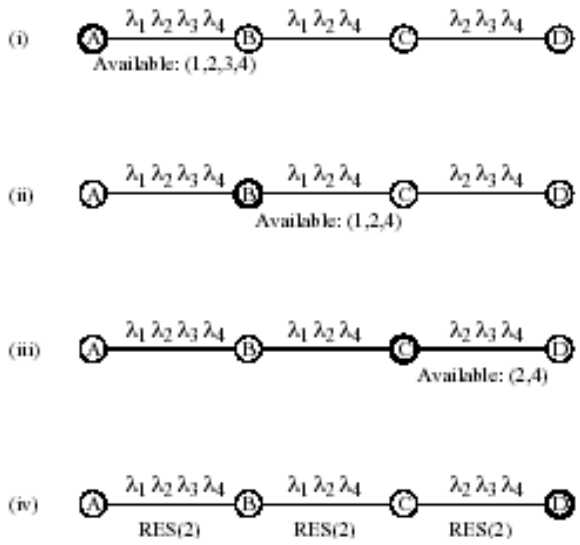


Figure 6: Backward reservation

Figure 6 illustrates the backward reservation scheme. . As the control message propagates from A to D, it records the set of wavelengths that are available. It is shown in [26] that, in general, backward reservation schemes outperform forward reservation schemes for the case in which there is no wavelength conversion.

One possible drawback of a backward reservation scheme is that if multiple connection are being set up simultaneously, it is possible that a wavelength that was available on a link in the forward direction will be taken by another connection request and will no longer be available when the reservation message traverses the link in the reverse direction.

4.2.4.4. Holding Policies

To improve the connection setup probability at the cost of higher setup times, it is possible to hold or buffer connection requests at intermediate nodes if wavelength resources are not immediately available [28] [29]. If an appropriate wavelength becomes available, the connection request will continue towards the destination. If, after waiting for some time, the appropriate resources do not become available, then the connection is blocked.

In [29], it is shown that a holding policy decreases the blocking probability without significantly increasing setup time. However, it is also shown in [28] that a holding policy reduces the network throughput compared to a policy in which calls are blocked immediately if resources are not available.

4.2.4.5. Summary

A summary of the different kinds of reservation algorithms in the RWA problem are presented in Tableau 4.

Reservation scheme				Characteristics	Advantages	Drawbacks
Parallel				Reserves wavelengths on multiple links in parallel. Based on link-state routing, it assumes that each node maintains global information on the network topology and on the current state of the network, including wavelengths usage. Based on this global information, the node can calculate an optimal route to a destination on a given wavelength.	Very fast lightpath establishment Optimal route.	Global information knowledge: network topology and wavelength usage.
Hop-by-hop	Forward	Global knowledge		With global knowledge of wavelength usage, sends a connection setup for an available wavelength.	Very simple.	Global knowledge. Possibly outdated information.
		Local knowledge	Conservative	Chooses randomly a single wavelength and sends out a control message to the next node attempting to reserve this wavelength along the entire path. If the wavelength is blocked, the source node may select a different wavelength and reattempt the connection.	Local knowledge.	High setup times.
			Aggressive	Multiple wavelengths may be reserved on each link in the path, with the expectation that at least one wavelength will be available on all links in the path. The source node will first reserve all available wavelengths on the desired outgoing link. A connection request message containing the wavelength reservation information is then sent to the next node along the path. At each intermediate node, the subset of wavelengths consisting of the intersection of the wavelengths reserved on the previous link and the wavelengths available on the next link will be reserved.	Local knowledge. Decreases blocking.	Over reserve resources.

			Aggressive with group reservation	The source node find the group with the largest number of available wavelengths, and the node will reserve all of the available wavelengths in that group before sending the request on to the next node.		The size of the group is a critical parameter. Less over reservation than for aggressive.
	Backward			Reservations are made after the control message has reached the destination and is headed back to the source. The connection request message can gather wavelength usage information along the path in the forward direction.	Local knowledge. Prevents the over reservation of resources. Outperform forward reservation when no WC.	Higher blocking probability than forward reservation in case of simultaneous reservations
	Holding policy (can be added to either backward or forward reservation schemes).			Buffer connection requests at intermediate nodes if wavelength resources are not immediately available. If an appropriate wavelength becomes available, the connection request will continue towards the destination. If, after waiting for some time, the appropriate resources do not become available, then the connection is blocked.	Improves setup connection probability of backward and forward reservation schemes.	Increases setup time. Reduces the network throughput.

Tableau 4: Reservation mechanisms in the RWA problem

4.3. A FRAMEWORK FOR OPTICAL NETWORKING: GMPLS

4.3.1. THE RWA PROBLEM IN THE GMPLS FRAMEWORK

The RWA problem is now a part of the work undergone under several IETF workgroups, such as Common Control and Measurement Plane (ccamp), Multiprotocol Label Switching (mpls), Open Shortest Path First IGP (ospf) and IS-IS for IP Internets (isis). In the following, an overview of the trends in the industry that make optical networking a reality in all-optical networks is depicted. The motivations and purposes of the GMPLS framework are explained and are linked to the RWA problem.

4.3.1.1. Overview of GMPLS

Generalized MultiProtocol Label Switching, also referred to as MultiProtocol lambda Switching supports not only devices that perform packet switching, but also those that perform switching in the time, wavelength, and space domains. In that sense, it is a “generalized” version of the MPLS technology presented in 4.1.1.

There is presently a great deal of interest in automating lightpath set-ups and teardowns in an optical transport network. An emerging trend in the industry is to utilize an optical-layer control plane, rather than a management plane as being done traditionally, to provision lightpaths. An intelligent optical-layer control plane is expected to offer several benefits including rapid circuit provisioning, service flexibility such as bandwidth on-demand services, enhanced interoperability of network elements from different vendors, and enhanced survivability by providing a dynamic rerouting capability when a failure occurs [30].

A common approach is that the control plane (i.e., routing and signalling) for the optical layer should be based on reusing and leveraging existing control-plane protocols in order to reduce product development cycles and foster rapid deployment of a new class of optical network elements.

It has recently become evident that the industry has gravitated toward GMPLS (also referred before to as MPλS standing for Multi Protocol Lambda Switching), as the control plane solution for next-generation optical networking. GMPLS is an extension to MPLS which enables Generalized Label Switched Paths (G-LSPs) such as lightpaths to be automatically set up and torn down by means of a signalling protocol. This requires the definition of an MPLS label to be generalized so that a label can also be encoded as a time slot, a wavelength, or a spatial identifier. By taking advantage of the new definition of a generalized label, it becomes apparently clear that MPLS can also be extended to control and configure a TDM DXC, a lambda or a fibre OXC.

A good overview of the routing and management enhancements used in the GMPLS framework is [31]. A good collection of links related to GMPLS is www.gmpls.org. Here is an introduction of the particularity of the routing problems in an OTN.

4.3.1.2. Lightpath set-up and restoration

One of the major objective for optical networks is to provide fast end-to-end optical lightpath set-up and restoration. This is done by three different components:

- **Resource discovery**. In resource discovery, state information such as network connectivity, link capacity and special constraints are derived. By and large, this is done by extending an IGP protocol, such as OSPF or IS-IS in order to carry the additional information in the LSAs.

- **Path selection**. Path selection is used to select an appropriate route through the ION for the requested lightpath. This is generally implemented by introducing a Constraint Based Routing (CBR) algorithm, which computes the desired route under physical layer constraints and operational constraints.
- **Path management**. Path management includes path setup and teardown, path maintenance and label distribution. This is done mainly by using extensions of RSVP-TE or CR-LDP.

As the reader may have noticed, the RWA problem concerns mostly the two first points. Nevertheless, as it was said before, the RWA problem can be intricately linked to the path management process, so that it is usually impossible to determinate the results of the RWA algorithm implemented without implementing any kind of path management protocol.

4.3.2. OPTICAL ROUTING ISSUES

Although there are similarities in the routing aspect of IP networks and ION, ION is more complex because it contains added constraints in the routing decision. Such differences are for instance as follows:

- Datagram network vs circuit-switched network. In IP networks, packet forwarding is done on a hop-by-hop basis. On the contrary, in ION, an end-to-end connection, or lightpath must be established according to constraints based on the network topology and resources.
- Separation of control plane and data plane topology. Contrary to IP networks, ION is likely to offer a greater security by managing an out of band control plane; entirely distinct from the data plane topology.

Among the major constraints added at the ION layer are the physical layer constraints that typically deal with the optical signals.

4.3.2.1. Physical layer constraints

A number of physical constraints that influence the lightpath computation results must be taken into account. Power budget at the source node, PMD, chromatic dispersion, ASE, cross talk between channels and other non-linearities are all critical constraints for the lightpath computation. Not much work has been done yet on that subject. A starter is done in [32].

It is possible that other constraints, especially 3R regeneration of signals should be considered by the ingress node undertaking the RWA algorithm when calculating the optimum route to an egress node. For instance, the Hikari GMPLS router is based on a photonic universal platform with the addition of 3R functions and wavelength conversion. If the signal is degraded by fibre loss as well as non-linear effects such as PMD or ASE, the 3R function is activated. In addition, wavelength conversion is also used when signalling is blocked by wavelength overbooking.

4.3.2.2. Wavelength constraints

In all-optical networks, unless there is Wavelength Conversion used, wavelength continuity must be preserved all along the lightpath, which complicates the routing decision. This is known as the Wavelength Continuity Constraint (WCC). This means that the route advertisements must contain information about available wavelength in each fibre link in the ION. Such a solution can pose important scalability problems because the number of wavelengths in the ION tend to be important. Some solutions giving the available wavelength in a fibre link have been discussed. Unfortunately, those methods imply important changes to the routing protocols.

4.3.3. THE GMPLS ARCHITECTURE

The basis of the GMPLS framework is defined in [33]. Extensions to IGPs, such as OSPF and IS-IS, allow nodes to exchange information about optical network topology, resource availability and administrative constraints. The core GMPLS routing specification is available in three parts:

- A routing function description[34],
- IGP extensions such as OSPF-TE extensions [35] and IS-IS extensions.

The GMPLS signalling extends certain base functions of the RSVP-TE and CR-LDP signalling and, in some cases, adds functionality. These changes and additions impact basic LSP properties, how labels are requested and communicated, the unidirectional nature of LSPs, how errors are propagated, and information provided for synchronizing the ingress and egress nodes. The core GMPLS signalling specification is available in three parts:

- A signalling functional description [36],
- RSVP-TE extensions [37],
- CR-LDP extensions.

Dynamic lightpath routing in IP-over-WDM networks is based on GMPLS constraint based routing model. For instance, OSPF is a link state protocol in which the state of each link in the network is periodically broadcast to all nodes in the form of LSAs. This information is used as input to a usual or constraint-based path computation algorithm that computes paths subject to topology, resource, and administrative constraints. To the extent of the author's knowledge, the GMPLS standard does not determine nor impose any kind of routing scheme. This is done for differentiation of implementation between vendors.

Once an appropriate lightpath is selected, a signalling protocol such as CR-LDP or RSVP-TE is then invoked to set up the connection. While the current focus of the IETF is on few specific protocols, GMPLS itself is not restricted to any single routing or signalling protocol. Furthermore, protocols such as OSPF, CR-LDP, and RSVP-TE are flexible and lend themselves to the implementation of various routing and signalling schemes for lightpath establishment.

4.3.3.1. A practical implementation of GMPLS: the Hikari router

An interesting approach to solve the RWA problem is implemented in the Hikari router [38]. In this study, wavelength converters are used only when there is blocking due to the absence of available wavelengths satisfying the wavelength continuity constraint. The Hikari router consists of an IP router, a wavelength router and a GMPLS router-manager as presented in Figure 7.

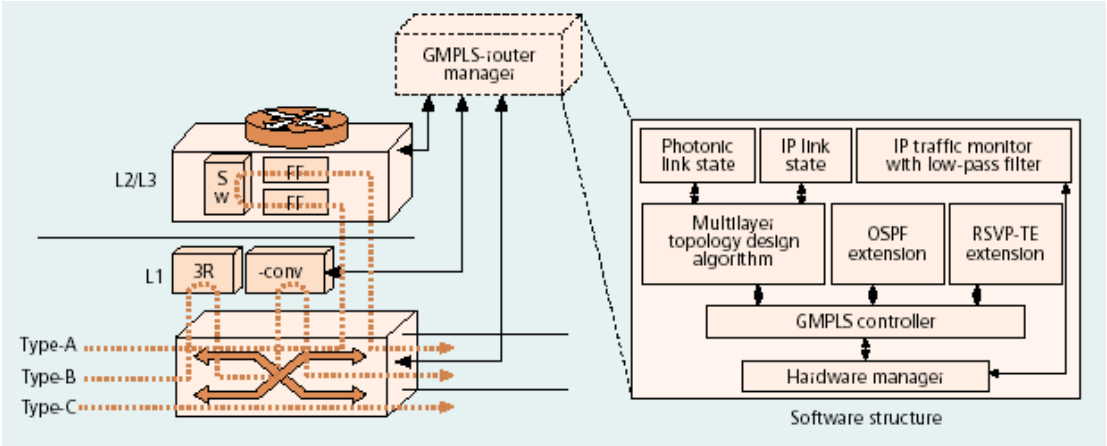


Figure 7: Structure of Hikari router with multi-layer TE based on IP traffic monitoring

To solve the RWA problem, the Hikari router implements an OSPF extension and an RSVP-TE extension to achieve effective route selection as well as to minimize the number of wavelength conversions needed. The OSPF extension allows to build two different link-state databases reflecting the links capability of the network. The usual link-state database contains IP link-state information while the TE database (TED) contains photonic link-state information.

First, for route advertisement, the OSPF extension is used. Each Hikari router advertises its total number of used and unused wavelengths. Edge nodes use this information to discern GMPLS link state and can select the least expensive path The changes include 3R resource information and statistical information such as utilization of each wavelength, and 3R and wavelength conversion resources. This information is typically used for source routing based on a combination of shortest path first and load information.

To enhance the signalling function, an RSVP-TE extension is proposed. The first Hikari router sets the unused wavelength information, using a bit map format, in RSVP signalling. Each transit Hikari router overwrites this information by placing “And” between arriving signalling unused wavelength bitmap and its own unused wavelengths. If there is no unused wavelength, wavelength conversion is used. The router, which offers wavelength conversion, creates a new unused wavelength bitmap and sends it to the next router. This signalling and routing technique minimizes the frequency of wavelength conversion in the network and so can provide very cost effective photonic networks.

The approach used to provide dynamic lightpath establishment described in the Hikari router is particularly interesting. To the author’s knowledge, it is the most state of the art standardized implementation of a DLE. Nevertheless, it doesn’t provide any further information on how the information gathered by the link advertisements are actually processed into the RWA algorithm. Especially, a definition of the links metric is not given and few performance results are presented. Thus, it is possible that poor results concerning the blocking of the connection requests has been concealed by the massive use of wavelength conversion.

To that regard, wavelength conversion is still extremely expensive and not fully commercially available at any reasonable prices. Thus, it seems very likely that there is an actual need for a RWA algorithm that would perform relatively well even under the conditions of the wavelength continuity constraint.

4.3.4. Overview of IETF current drafts and RFCs

In the following, an overview of the format of the route advertisements that are used in OSPF and its following extensions is presented. Such an introduction seemed necessary to the author, because it allows to understand how the GMPLS framework allows practically to establish dynamically lightpath in the ION. In the following, OSPF was chosen as an example, but it could also have been IS-IS. It is possible to use indifferently one of them.

Nevertheless, OSPF is more widely deployed in the Internet that its counterpart IS-IS, the later being mostly known in large provider and carrier networks. Also, OSPF is a free software implemented in C++ [39]. In the following description, only fields that would be relevant in a future implementation of a RWA algorithm based on OSPF are presented.

4.3.4.1. Link state routing protocol extensions of OSPF

4.3.4.1.1 Packets formats

a) Packet header

The packet header has the following structure:

Version #	Type	Packet length
Router ID		

Data

The following fields are relevant in this work:

- **Version #**. The OSPF version number. This will be always 2.
- **Type**. The OSPF packet types that will be used. This includes: Hello (type 1), Database Description (type 2) and Link State Update (type 4). Link State Request (type 2) and Link State Acknowledgment (type 5) will not be used. For simplicity's sake, this simulation considers that the underlying layers provide a reliable service.
- **Packet length**. The length of the OSPF protocol packet in bytes. This length includes the standard OSPF header.
- **Router ID**. The Router ID of the packet's source.

b) Hello Packet

Hello packets are packet type 1. These packets are sent periodically on all interfaces in order to establish and maintain neighbour relationships.

Packet Hdr	
Hello Interval	Router Dead Interval

The following fields can be used:

- **Hello Interval**. The number of seconds between this router's Hello packets.
- **Router Dead Interval**. The number of seconds before declaring a silent router down.

c) Database Description packet

Database Description packets are OSPF packet type 2. These packets are exchanged when an adjacency is being initialised. They describe the contents of the link-state database.

Packet Hdr		
I	M	DD sequence number
LSA		
Etc.		
I	M	DD sequence number
LSA		

The following fields will be used:

- **I-bit**. The Init bit. When set to 1, this packet is the first in the sequence of Database Description Packets.
- **M-bit**. The More bit. When set to 1, it indicates that more Database Description Packets are to follow.
- **DD sequence number**. Used to sequence the collection of Database Description Packets. The initial value (indicated by the Init bit being set) should be unique. The DD sequence number then increments until the complete database description has been sent.
- **LSA**(s).

d) Link State Update packet

Link State Update packets are OSPF packet type 4. These packets implement the flooding of LSAs. Each Link State Update packet carries a collection of LSAs one hop further from their origin. Several LSAs may be included in a single packet.

Packet Hdr
LSAs = N
LSA 1
Etc.
LSA N

The following field will be used:

- **# LSAs**. The number of LSAs included in this update.
- **LSA**(s).

4.3.4.1.2 LSA formats

OSPF includes five distinct types of LSAs. Each LSA begins with a standard LSA header. Each LSA describes a piece of the OSPF routing domain. Every router originates a router-LSA.

a) The LSA header

All LSAs begin with a common header. This header contains enough information to uniquely identify the LSA (LS type, Link State ID, and Advertising Router). Multiple instances of the LSA may exist in the routing domain at the same time. It is then necessary to determine which instance is more recent. This is accomplished by examining the LS age, LS sequence number fields that are also contained in the LSA header.

LS age	LS type
Advertising router	
LS sequence number	

The following fields will be used:

- **LS age**. The time in seconds since the LSA was originated.
- **LS type**. The type of the LSA. Each LSA type has a separate advertisement format. For this work, it is worth mentioning the LS type 1 (Router-LSA) and the type 10 (opaque LSA).
- **Advertising Router**. The Router ID of the router that originated the LSA.
- **LS sequence number**. Detects old or duplicate LSAs. Successive instances of an LSA are given successive LS sequence numbers.

a1). The Router-LSA (LS type 1)

Router-LSAs are the Type 1 LSAs. Each router in an area originates a router-LSA. The LSA describes the state and cost of the router's links (i.e., interfaces) to the area. All of the router's links to the area must be described in a single router-LSA.

The following fields will be used:

- **# links**. The number of router links described in this LSA. This must be the total collection of router links (i.e., interfaces) to the area.
- **Link Type**. A quick description of the router link. In this work, the type 1 (Point-to-point connection to another router) will be only used.

- **Link ID**. Identifies the object that this router link connects to. In this work, as the Link Type is 1, the Link ID identifies the Neighboring router's Router ID.
- **Metric**. The cost of using this router link.

This type of LSA is the one commonly used in OSPF for link state updates in order to calculate the shortest path (and build the link-state database). Instead, in this work, it is legitimate to use the opaque LSA (type 10), as proposed in the OSPF extensions. This LSA is considered in the following.

4.3.4.1.3 LSAs extensions

The extensions of the LSAs are based on the TE extensions of OSPF for MPLS [40] and GPMLS [35]. Those extensions use opaque LSAs [41] [40]. TE extensions, commonly associated with MPLS, are better described by "extended link attributes", as what is proposed is simply to add more attributes to links in OSPF advertisements.

The information made available by these extensions can be used to build an extended link state database just as router LSAs are used to build a regular link state database. The difference is that the extended link state database (referred to as a TED) has additional link attributes. For simplicity's sake, only the relevant LSAs extensions that would be suited to the RWA problem are described.

a1). Opaque LSAs (types 9,10 and 11)

Opaque LSAs provide a generalized mechanism to allow for the future extensibility of OSPF. They consist of a standard LSA header followed by application-specific information. Opaque LSAs are types 9, 10 and 11 link-state advertisements. Standard link-state database flooding mechanisms are used for distribution of Opaque LSAs.

Link-state type 10 represents an area-local scope. Type-10 Opaque LSAs are not flooded beyond the borders of their associated area.

a2). The area scope opaque LSA (type 10)

- **Traffic Engineering extensions to OSPF [40]**

The LSA ID of an Opaque LSA is defined as having eight bits of type and 24 bits of type-specific data. The Traffic Engineering LSA uses type 1.

1	Reserved	Instance

The LSA payload consists of one or more nested Type/Length/Value (TLV) triplets for extensibility. They have the following structure.

Type	Length
Value	

An LSA contains one top-level TLV. There are two top-level TLVs defined: (1) Router Address and (2) Link.

The Link TLV (type 1) describes a single link. It is constructed of a set of sub-TLVs. There are no ordering requirements for the sub-TLVs. The following sub-TLVs are defined:

- 1 - Link type (1 octet)
- 2 - Link ID (4 octets)

- 3 - Local interface IP address (4 octets)
- 4 - Remote interface IP address (4 octets)
- 5 - Traffic engineering metric (4 octets)
- 6 - Maximum bandwidth (4 octets)
- 7 - Maximum reservable bandwidth (4 octets)
- 8 - Unreserved bandwidth (32 octets)
- 9 - Resource class/color (4 octets)

The Link Type and Link ID sub-TLVs are mandatory, i.e., must appear exactly once. All other sub-TLVs defined here may occur at most once. Of particular interest are the following:

- **Link Type**. The Link Type sub-TLV (TLV type 1) defines the type of the link. In this work, the value 1 - Point-to-point – will be used.
- **Link ID**. The Link ID sub-TLV (TLV type 2) identifies the other end of the link. For point-to-point links, this is the Router ID of the neighbour.
- **Traffic Engineering Metric**. The Traffic Engineering Metric sub-TLV (TLV type 5) specifies the link metric for traffic engineering purposes. This metric may be different than the standard OSPF link metric.

- **OSPF extensions in support of GMPLS [35]**

The TE LSA, which is an opaque LSA with area flooding scope, has only one top-level Type/Length/Value (TLV) triplet and has one or more nested sub-TLVs for extensibility. The top-level TLV can take one of two values (1) Router Address or (2) Link. In [35], sub-TLVs for the Link TLV in support of GMPLS are enhanced. Specifically, the following sub-TLVs are added to the Link TLV:

Sub-TLV Type	Length	Name
11	8	Link Local/Remote Identifiers
14	4	Link Protection Type
15	variable	Interface Switching Capability Descriptor
16	variable	Shared Risk Link Group

Two sub-TLVs of particular importance are

- Link Protection Type

The first octet is a bit vector describing the protection capabilities of the link (see Section "Link Protection Type" of [34]). They are: 0x01 Extra Traffic, 0x02 Unprotected, 0x04 Shared, 0x08 Dedicated 1:1, 0x10 Dedicated 1+1, 0x20 Enhanced, 0x40 Reserved, 0x80 Reserved. This is of critical interest if, in a further work, one would try to model the performance of a GMPLS-based network with restoration and self-healing capabilities.

- Interface Switching Capability Descriptor

The Interface Switching Capability Descriptor (ISCD) is the more relevant to direct applications for this work.

As stated in [34], the Interface Switching Capability Descriptor describes switching capability of an interface. Interface Switching Capability Descriptors present a new constraint for LSP path computation.

The general structure of the Interface Switching Capability Descriptor is

Switching Cap	Encoding	Reserved

Max LSP Bandwidth at priority 0
Max LSP Bandwidth at priority 1
Etc.
Max LSP Bandwidth at priority 7
Switching Capability-specific information (variable length)

Following the GMPLS framework [34], the Switching Capability (Switching Cap) field contains one of the following values:

- 1 Packet-Switch Capable-1 (PSC-1)
- 2 Packet-Switch Capable-2 (PSC-2)
- 3 Packet-Switch Capable-3 (PSC-3)
- 4 Packet-Switch Capable-4 (PSC-4)
- 51 Layer-2 Switch Capable (L2SC)
- 100 Time-Division-Multiplex Capable (TDM)
- 150 Lambda-Switch Capable (LSC)
- 200 Fibre-Switch Capable (FSC)

The content of the Switching Capability specific information field depends on the value of the Switching Capability field. When the Switching Capability field is LSC, there is no Switching Capability specific information field present defined at the moment in the current drafts. As it appears, of particular interest in this work is the 150 value for Lambda Switch Capable node.

The Max LSP fields will not be considered in this work.

- Proposed extension of the switching capability specific information

This field will hold the lambda switching capabilities of the node. In particular, it will contain a bitmap format field defined in 5.1.1.5.2.

- Example of a Switching Capability LSA

LS age	LS type = 10
Advertising router	
LS sequence number	
Type = 1 (Link)	Length = 11 bytes
Type = 1 (Link type)	Length = 1 byte
Link Type = 1	
Type = 2 (Link Id)	Length = 4 bytes
Link ID (first neighbour's router ID)	
Type = 15 (ISCD)	Length = 6 bytes
Sw. cap = 150	0x 00 01 01 01 00 : Lambda capability

As one see, this adds traffic overhead, but this is necessary, if further works try to implement any other kinds of OSPF functionalities.

4.4. CONCLUSION

This literature survey has showed that the RWA problem is a very complex one. An important number of studies have provided very different various schemes to solve this problem, but in very different situations. Most of the time, this leads to sophisticated and very evolved heuristics, because the RWA is known as intractable. Unfortunately, even if numerous schemes have been proposed, there are no agreed solution on how to solve the problem and no practical recommendation to implement the problem. There has been also very few work done on how to integrate all different schemes (routing, WA and reservation) together and estimate their performance as a whole.

On the other hand, it has recently become evident that GMPLS is the best control plane solution for next-generation optical networking. In GMPLS, the MPLS label is generalized so that a label can also be encoded as a time slot, a wavelength, or a spatial identifier.

The framework embraced by GMPLS is extremely large and tries to standardize a way to solve the general problem of dynamic lightpath establishment in IONs. The GMPLS is so large that the RWA problem appears as a relatively small part of the GMPLS framework, albeit a very critical part. It seemed important to the author that the solution of the RWA problem must be found within the different recommendations of the IETF for GMPLS. Obvious reasons for that are software reuse for the implementations, easy upgrades of the simulation network (such as link protection, efficient reservation schemes, easily added physical constraints, etc).

Nevertheless, the GMPLS framework is relatively new and very few implementations of it have been done. There is a critical research need in the modelling of lightpath set-up schemes (equivalently RWA algorithms) in a GMPLS-based network in order to estimate the performance of the solutions advocated by the drafts and standards of the IETF.

In particular, it is of foremost importance to estimate the performance of several RWA schemes, especially when build around a link-state routing algorithm in a GMPLS-based ION. There is also a urge to estimate the consequences of special physical constraints implied by the optical domain in the performance of the RWA algorithm.

5. INVESTIGATION: A SIMULATION MODEL OF THE RWA PROBLEM

In the following, the approach in order to solve and simulate the RWA problem in an ION is presented. First, the specifications of the simulation model are enumerated. This includes defining accurately the hypotheses, the objectives and the proposed schemes chosen that will be implemented. Secondly, the guidelines for the implementation of the model are described.

5.1. EXPERIMENTAL METHODOLOGY

5.1.1. HYPOTHESES

5.1.1.1. Definition of the problem

Given a set of dynamic and randomly chosen lightpaths that need to be established in the ION, and given a constraint on the total number of wavelengths into a fibre, the RWA scheme must determine the routes over which these lightpaths should be set up and also determine the wavelengths that should be assigned to these lightpaths so that, as a whole, the maximum number of lightpaths may be established.

While shortest path routes may be most preferable, this choice may have to be sacrificed in order to allow more lightpaths to be set up. Lightpaths that cannot be set up due to constraints on routes and wavelengths are said to be blocked, so the corresponding network optimisation problem is to maximise the probability of set-up for a current connection request while minimizing the blocking probability of future connection requests.

5.1.1.2. Requests' set-up time considerations

Once the routing table is ready, the set-up time of a request is the time taken by the reservation process. If a hop-by-hop reservation scheme is chosen, the set-up time is proportional to the number of nodes and the number of links that the reservation messages will encounter. As it was said before, the routing algorithm will have to sacrifice the choice of shortest path routes and will advertise routes that are not the shortest, but that yield the best wavelength utilization in the network. If the processing in each node is important, this will inevitably increase the set-up time.

Nevertheless, contrary to several other works, the RWA to be implemented will not consider the set-up time as a performance parameter. This is because the lightpath establishment problem is not considered as a whole.

Furthermore, there is no commonly accepted idea concerning the architecture of the control plane for all-optical networks. All types of signalling are viable, that is in-band signalling, out-of-band signalling or common channel signalling. For instance, out-of-band signalling would result of the use of one wavelength for the signalling of the entire network. This possibility is interesting because it allows to carry safely the possible large amount of route advertisements issued by the routing protocol and also the reservation messages.

5.1.1.3. Requests' arrivals

In this work, we consider only the DLE problem. This is different to the SLE problem in which there is a known set of connections that needs to be routed, before the RWA algorithm is performing in the network.

The DLE problem is more complicated than the SLE. Nevertheless, if it is assumed that MPLS is used on the upper layer, it can be considered that the rate of arrivals is somewhat relatively low. By low, the author means that the MPLS label stacking methods allows requests coming from an MPLS-based end-system to be bundled together in a label stack in order to minimize the load entailed by dynamic lightpath requests. For instance, a usual analogy is the one of ATM, that uses a 2-level label stacking (VPI-VCI) in order to simplify the network architecture. Yet, MPLS (and thus GMPLS) is much more powerful than ATM in this regard, because it allows an infinite-level label stacking.

5.1.1.4. Architecture of the ION considered in this work

In this work, the RWA scheme will be tested on a meshed network. This is a usual situation in all-optical networks where several redundant paths are possible between each node of the network.

Each node consists of an OXC controlled by a GMPLS controller using the services of an IP router. Typically, the different tasks of the controller could be the management of all non local management functions, including the management of optical resources, configuration and capacity management, addressing, routing, topology discovery, TE and restoration.

It is assumed that the GMPLS controller functions purely as a controller for the optical layer and carries no IP data traffic. The electronic controllers communicate with each other over a control network, either out-of-band, or in band. It is assumed that it exists a reliable transport protocol within the control network to ensure that messages between controllers are delivered reliably and in sequence.

The model will be tested on a typical network used in many research papers for network performance evaluation. The network topology considered is a meshed network (partially connected graph) such as the Abilene network in the United States.

Abilene is an advanced backbone network that supports the development and deployment of the new applications being developed within the Internet2 community in the United States. Abilene connects regional network aggregation points, called gigaPoPs, to support the work of Internet2 universities as they develop advanced Internet applications. A map of the network is given in Figure 8.

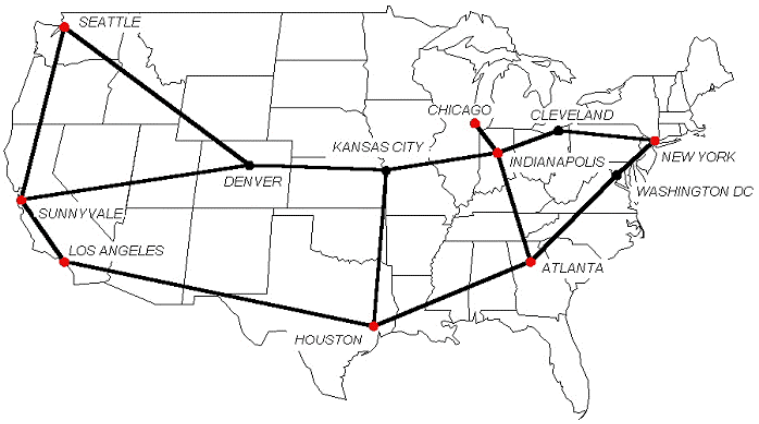


Figure 8 the Abilene network in February 2002

In Abilene, the degree of each node is between 1 and 4, with an average degree of about 3. The total number of nodes |V| is 12.

5.1.1.5. Physical constraints

In this work, parameters that are not wavelength-based constraints are not taken into account. For instance, some other works try to consider other physical constraints and integrate them in the routing decision [38]. This is reserved for further work.

5.1.1.5.1 Wavelength Continuity Constraint

Because the all-optical wavelength-conversion is an expensive and still immature technology, it is assumed that none of the OXCs has wavelength conversion capability. Consequently, in the following, the solution will have to respect the WCC.

The WCC is a unique constraint that is not found in usual telephony circuit-switched networks. Consequently, it is very likely that any algorithm respecting the WCC will suffer higher blocking probability. For instance, in the figure given below, two lightpaths have been established in the network: (1) between node 1 and node 2 on λ_1 and (2) between node 2 and node 3 on λ_2. Now, suppose that a lightpath has to be set up between node 1 and node 3. This is impossible and this will lead to blocking for such a lightpath establishment, because the two last available wavelengths (λ_2 between node 1 and node 2 and λ_1 between node 2 and node 3) are different.

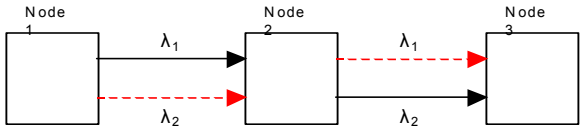


Figure 9 Blocking due to the Wavelength Continuity Constraint

The algorithm must establish dynamically an end-to-end path between any ingress node A and any egress node B in the all-optical network. First, not considering TE aspects, the main goal is to minimize the blocking in the network. Taking this aspect for granted, it means that the optimal path will not be the IGP's optimal path. To obtain such a result, the metrics of the links have to be changed according to this goal. Different metrics (refer to 4.2.2.2.3a1) for dynamic routing algorithms have to be tested.

5.1.1.5.2 Number of wavelengths per link

Let us consider one link between two adjacent nodes. Each link may be composed of several fibres, each fibre being itself composed of several WDM wavelengths. In Figure 10, the link is composed of 3 fibres, each of them containing a certain number of wavelengths. Let $\lambda_{i,j}$ be the wavelength number i in the fibre j.

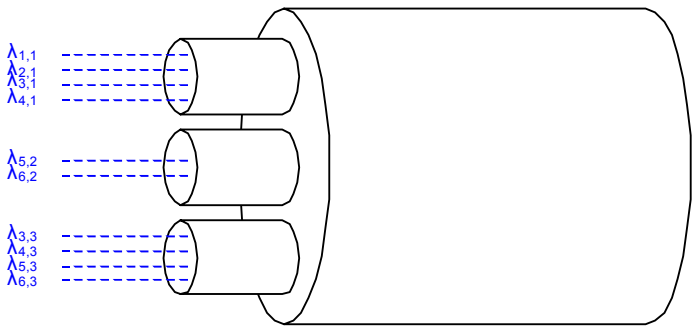


Figure 10: An example of a fibre's trunk

It is assumed that the operating wavelength range is the 1,550-nm band, usually named the Conventional band or C-band by the ITU-T. The wavelengths range in the C-band is 1,530 nm to 1,565 nm. Each wavelength is assigned a specific emission wavelength that is specified by the ITU-T [42].

The ITU-T's frequency plan is followed. Consequently, it is assumed that there is a maximum of 44 wavelengths per fibre. The wavelengths are ordered as follows: $\lambda_{1,i}$ = 1,530.33 nm and $\lambda_{44,i}$ = 1,564.68 nm, with j being the number of fibres used.

In a nutshell, in this work, the following physical-based wavelength constraints are considered:

$$\begin{cases} \lambda_{i,j} = \text{ Wavelength i in fibre j} \\ i \in [1, 44] \\ \lambda_{1,j} = 1{,}530.33 \text{ nm} \; ; \; \lambda_{44,j} = 1{,}564.68 \text{ nm (C-band)} \end{cases}$$

Still, it can be showed that a multi-fibre RWA problem is algorithmically equivalent to a single fibre based optical network. Thus, in the following, an ordered list of wavelengths with only one fibre will be considered.

Also, it is very unlikely that the 44 wavelengths of the C band will be all practically used to transmit. Usually, only a few wavelength are lit ; this actually depends on the optical network designer. Generally, the population of the WDM slots is limited to a multiple of 8 wavelengths, up to 32 or 40.

5.1.2. OBJECTIVES IN THIS WORK

If possible, the RWA algorithm will reuse common algorithms that are well-known and available. It will be as simple to implement as possible. This is to minimize the computation problems that may arise if other physical constraints than wavelength availability were to be added.

The RWA algorithm must be able to identify the maximum of explicit routes in the ION in order to satisfy dynamic requests. The goal is to minimize the blocking of the lightpaths requests in the network while respecting the WCC.

The RWA algorithm will separate the routing, the WA and the reservation mechanisms. In other words, this work will primarily concentrates upon the search of an explicit route in the network that would meet the physical constraints of the ION. Such a separation between routing and reservation protocols eases further work.

In order for this work be continued, it has been thought that the RWA algorithm that will be defined will give as an output an explicit route in the ION. This explicit route will determine each node to be traversed by the lightpath. Besides, a WA algorithm will allow to determine the wavelength satisfying the WCC that will have to be reserved on the lightpath. The explicit route and wavelength will be then treated as the input of a reservation protocol in order to establish the circuit in the network.

Any software written must prone software reuse. This is particularly important, because the RWA problem can grow more and more complex when new physical constraints are added. For instance, in this work, only the wavelength constraints will be considered. Nevertheless, this work must build the foundation for following works, such as taking into account specific ION constraints such as signal regeneration and wavelength conversion.

5.1.3. DESCRIPTION OF THE RWA SCHEME

The present work is to develop a model of a GMPLS-based ION based on an adaptive link-state routing using global network knowledge. This choice is explained in the following.

5.1.3.1. Routing algorithm

Several types of RWA algorithms have been presented. The three main types of routing schemes are fixed, fixed-alternate and dynamic routing. Fixed and fixed-alternate routing principles are very simple. Nevertheless, they have poor capabilities when dealing with self healing capabilities and can yield a very high blocking probability when considering DLE.

Thus, dynamic routing will be considered. The main disadvantage of dynamic routing is the possible high overhead due to route advertisements. Also, dynamic routing algorithms can become very greedy in CPU when considering several constraints. Nevertheless, the choice of an algorithm is always a turnoff between performance and traffic overhead. Considering earlier research studies described in the literature survey, it has been thought that a dynamic routing is one of the best way to solve the RWA problem.

In a first approach, both adaptive routing schemes, that is link-state routing and distance-vector routing have been considered. Contrary to link-state based routing algorithms that flood packets onto the network, distance-vector algorithms send their route advertisements only to their neighbours. In link-state algorithms, the link state update are flooded onto the whole network (or area in OSPF).

It is possible that a distance-vector algorithm can minimize the traffic overhead in the network while still giving relatively low blocking probabilities under high loads. A distance-vector algorithm is also interesting when considering constraint routing. Indeed, It is a property of the Bellman-Ford algorithm that, at its h-th iteration, it identifies the optimal (in our context: maximal number of wavelengths) path between the source and each destination, among paths of at most h hops. This is recommended for QoS routing implementations [19].

Because the Bellman-Ford algorithm progresses by increasing hop count, it essentially provides for free the hop count of a path as a second optimisation criteria. This property is very interesting when applied to all-optical networks. Even if the shortest path will be sacrificed for a longer – in hops – route, the lightpath should not be too long, because this could lead to a poor and / or costly optical communication.

RIP is the most famous implementation of a distance-vector based protocol. It continues to be popular, because it is simple and is well suited to small networks. However, RIP has several flaws that make it particularly unsuitable for ION. Particularly, RIP is unsuitable for large configurations and the convergence of the algorithm can also be lengthy ; it suffers also from the count-to-infinity problem, of which the best remedy is to implement link-state algorithms.

In the context of the ION, in order to compute an explicit route, it is also much easier to use a link-state based routing algorithm. Indeed, the lightpath to be established will be more optimal if each node has a complete knowledge of the network. It is also a property of the Dijkstra algorithm that the complete route from the source to any other node in the network can be easily found by a recursive iteration on the graph. This makes it very easy to use Dijkstra algorithm in order to find an explicit route in the ION while minimizing the requests blocking probability in the ION.

In link-state based routing, information is only sent when changes occur. A node builds up first a description of the topology of the network. Then it may use any routing algorithm to determine the route. Conversely, a distance-vector approach needs to use a distributed algorithm such as the popular Bellman-Ford algorithm.

In our context, the fact that link-state algorithms can use any routing algorithm is particularly important. Any more advanced routing algorithm can be safely added to the RWA algorithm, without having to undergo massive changes. This is particularly important for software reuse. For instance, different RWA algorithms can be implemented, based on different optimisation schemes (partial or total wavelength knowledge).

The OSPF protocol, known mostly for its second version [43], is the most widely known link-state protocol. It has been increasingly popular over RIP, because it is most suitable for large networks. OSPF is an open source algorithm that can be found in different languages, including C++ [39]. This is a particular advantage for software reuse, because the simulation program (OMNeT++) to be used is based on C++.

The previous points explain why a link-state based algorithm has been chosen. Such a choice is also a must, because the GMPLS standard takes it as granted that only link-state routing algorithms will be used. That is why only extensions to OSPF or IS-IS are given in the GMPLS drafts.

5.1.3.2. Description of the routing scheme

5.1.3.2.1 Route advertisements

The route advertisement messages are build according to the OSPF extensions for GMPLS. As the literature survey points it, this work proposes to extend the sub TLV relative to the ISCD field of the GMPLS OSPF extensions.

In order to dynamically monitor the state of the network, each GMPLS router keeps track of all the wavelength capabilities of the whole network in a “links” database. This database is constantly updated each time a ROUTING message is received. The ROUTING message contains the description of the wavelength capabilities of a certain link that have changed very recently.

Those ROUTING messages are actually flooded on the whole network by the node which one of its links’ capability changed. They basically contain the address of the extremities of the link of which the wavelength capability has changed and the state of the changed wavelength.

The links database is composed of records, where each record describes one link of the network. Each record contains the node addresses of the link’s extremities, a wavelength capability field and a metric field. The wavelength capability field describes explicitly the wavelength resources of the considered link. The metric field is the cost to use this link when performing the shortest path calculation. In this work, two different metrics will be implemented. They are both function of the total number of wavelengths and the number of available wavelength(s) on the link (5.1.3.2.2).

5.1.3.2.2 Link metrics

The link metric represents the cost to use a certain link in the network. Intuitively, the link cost is a function of the total number of wavelength and the number of available wavelengths. The more available wavelength, the lower cost the link will be.

The routing scheme is tested under different link metrics: simple TAW or enhanced TAW. Let $\lambda_{i,j}^{a}$ be the number of available (unused) wavelengths on the link (i,j) and $\lambda_{i,j}^{T}$ the total number of possible wavelengths on that link. The simple TAW metric represents the load assigned to a link and is defined by:

$$w_{i,j} = 1 - \frac{\lambda_{i,j}^{a}}{\lambda_{i,j}^{T}} \qquad \forall (i,j) \in E^2 \tag{3}$$

The enhanced TAW metric is to test the metric proposed by [18] that equivalently minimizes the probability of blocking on an explicit route:

$$w_{i,j} = -\log\left[1-\left(1-\frac{\lambda_{i,j}^{a}}{\lambda_{i,j}^{T}}\right)^{\lambda_{i,j}^{a}}\right] \quad \forall(i,j) \in E^2$$

A comparison of the different weights for $\lambda_{i,j}^{a} = 32$ is given in .

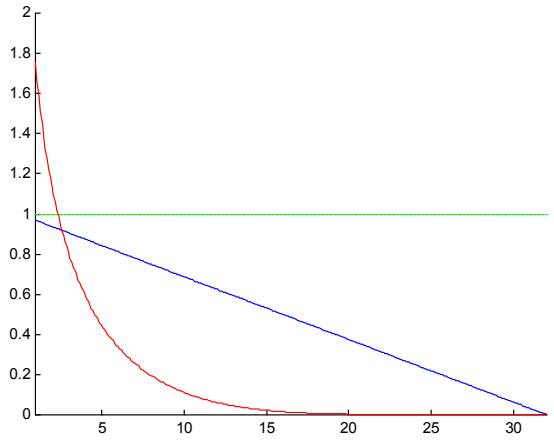


Figure 11 Link metrics for a 32 wavelengths fibre

5.1.3.2.3 Path calculation

The links database that has been freshly updated by the GMPLS router serves as the basis of the path discovery calculation based on the Dijkstra algorithm. This path computation is performed by each node in the network when it receives a ROUTING message, that is an update of a certain link capability in the network.

The routing algorithm actually allows each node to build its own photonic database which is its own representation of the network. The photonic database contains N records, where N is the number of nodes of the network. Each record is based on the following structure:

- A node destination address. This identifies the record ; it is the node to reach.
- A total cost to this destination node. This is the total cost when taking the shortest path route to the destination node.
- An address of the last-but-one node on the shortest path route to the destination node.
- An end-to-end available wavelength capability. This field determinates explicitly the possible wavelengths to be assigned, if any, on the shortest path route.
- An explicit route. This is an ordered list of all the addresses of the nodes on the shortest path route.

Given the links database, the three first fields are actually the direct output of the Dijkstra shortest path algorithm. The two last fields are the result of a sub-routine of the Dijkstra algorithm. Indeed, it is a property of the Dijkstra algorithm that it is possible to find the list of the nodes of each shortest path by a simple recursive call.

5.1.3.2.4 Wavelength assignment

There are numerous possible WA schemes. Possible schemes tested in this work are first-fit and random. First-fit chooses the first wavelength available in an ordered list of wavelength. Random chooses a wavelength randomly between the different available wavelengths.

Those two schemes are used directly at the output of the path calculation algorithm in order to determinate a wavelength to be reserved on the path to each destination in the network from a source node.

5.1.3.2.5 Reservation scheme

For simplicity sake's, a reservation protocol based on parallel reservation has been implemented in order to test the different routing and WA schemes implemented. The reservation is simplified because it does not consider that packets could be lost, allowing not to consider any retransmission mechanisms. Also, it is assumed that the service of a routing protocol (such as OSPF) from hop to hop between the different routers is provided.

The reservation by itself uses the following types of packets: REQUEST, RESERVE, RESPONSE, TAKEDOWN. The process of messages exchanges for reservation is as follows:

- A generator of an end-system of a node emits a REQUEST message for a new connection to another randomly chosen node in the ION. It transmits this request to its associated GMPLS router. Thanks to its photonic database built by the wavelength routing algorithm, the GMPLS router module is able to determine if (1) an explicit route is available to the destination requested ; (2) on this explicit route, choose a wavelength that will maximise the probability of non-blocking.
- If an explicit route and a wavelength are available for the destination, the GMPLS router then reserves in parallel the route and the wavelength assigned for that connection. That is, it sends in parallel a RESERVE message to each node of the explicit route, excepted itself.
- When a node on the explicit route receives the RESERVE message, it checks if its wavelength is available on the link linking the node considered with his predecessor in the explicit route. It then sends a RESPONSE message back to the GMPLS router sub-module that sent the RESERVE message.

A diagram of the reservation process illustrating the REQUEST, RESERVE and RESPONSE messages is given in Figure 12. The end-system attached to the node S sends a REQUEST message to the GMPLS router S (1). Then, S sends in parallel 3 RESERVE messages to nodes 1, 2 and D (2). Finally, each nodes answers independently to S (3).

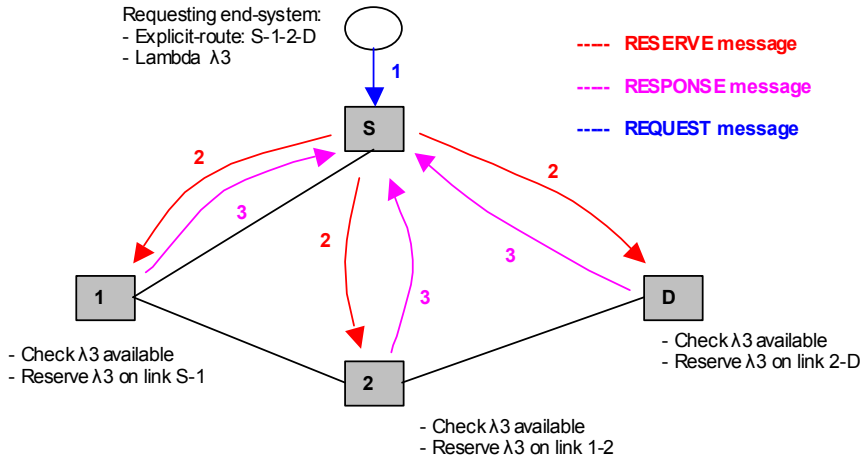


Figure 12 Parallel reservation mechanism – REQUEST, RESERVE, RESPONSE messages

- In case the wavelength to be reserved was not available in one or several node of the explicit route, the requesting GMPLS router module sends a TAKEDOWN message to all the nodes of the explicit route to warn them to reset as available the wavelength that had been reserved.

A diagram of the reservation process illustrating the TAKEDOWN message is given in Figure 13. The node 2 checks its available wavelengths on the link 1-2 ; the wavelength requested by S is not available. It then warns S by sending a RESPONSE message refusing the wavelength reservation (3). Immediately, S sends back to the other nodes of the explicit route a TAKEDOWN message (4) that resets the lambda requested to available.

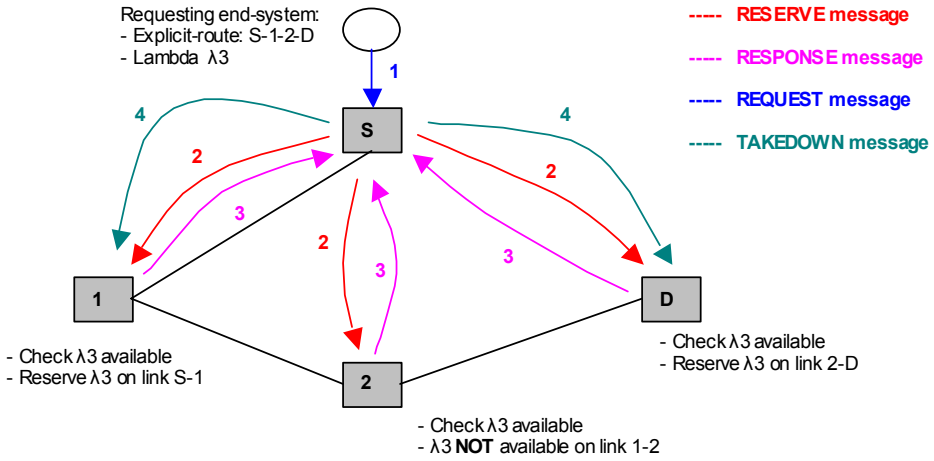


Figure 13 Illustration of the use of TAKEDOWN messages

5.2. MODEL IMPLEMENTATION

The model is implemented using the simulation software OMNeT++. OMNeT++ is becoming increasingly popular among a wide range of scientific communities, and competes well in these communities with established commercial tools such as OPNET. It is being used at a number of universities for research on communication networks [44].

There are several reasons to use OMNeT++. First, this is a free software that is currently available on UNIX at Monash. It is also currently increasingly employed by Monash research students as a simulation tool for different works, including routing protocols. Thus, there is actually an extremely profitable cooperation between research fellows working on OMNeT++ that will definitely boost the research on optical networking simulation topics. OMNeT++ allows also to more efficiently reuse former implementation because it uses C++, an object-oriented programming for simple modules.

5.2.1. IMPLEMENTATION GUIDELINES

5.2.1.1. Network model

The ION model is composed of simple modules – referred to as GMPLS router modules – and compound modules – referred to as EndSystem compound module. The GmplsRouter module contains all the RWA and reservation algorithms, while the EndSystem module is responsible of generating and analysing the response to connection request messages.

The EndSystem sub module is composed of a Generator simple module and a Sink simple module. The generator module task is to generate a certain number of connection requests to randomly chosen destination nodes in the ION. The sink module receives the responses to each connection request generated by its counterpart Generator module. The sink module principle task is to analyse the responses and to build statistics table.

It is worth noting that the Endsystem compound module actually models an end system in an ION. Effectively, the EndSystem module represents one or several end systems such as switches or routers. For instance, it is very likely that such end systems will be ATM switch or IP routers with MPLS-based software. A current area of research is to standardize the physical interface between the clients (end systems) and the transport network (ION). This interface is designated as the UNI interface. An implementation agreement for the UNI has been proposed by the OIF [45].

Each GmplsRouter module is associated to only one Endsystem module. Those modules have the same address. A simplified view of the model is given in Figure 14.

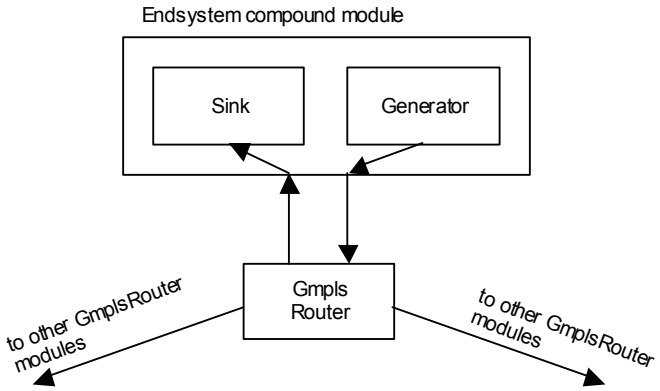


Figure 14: Endsystem, Generator, Sink and GmplsRouter modules

The network model is based on the Abilene network. It is composed of 12 GmplsRouter modules scattered at different GigaPops around the U.S. ; each GmplsRouter module is associated to its own Endsystem module.

5.2.1.2. System parameters

The network traffic is generated in terms of connection requests. A connection request is a request to establish a lightpath from a source node to a randomly chosen destination node. The connection requests arriving at each router is assumed to follow an exponential distribution with mean $\lambda_{poisson}$ per unit of time.

The system parameters that are varied are $\lambda_{i,j}^{T}$, the total number of wavelengths on each link and the connection requests arrivals. $\lambda_{i,j}^{T}$ is varied between 8 and 24 by increment of 8.

The performance parameter considered is the blocking probability P and the link utilization U.

Let P be the probability that a connection request is blocked due to the unavailability of wavelengths for a lightpath. It is not possible to give any analytical definition of P

The average link utilization U is given by the percentage of time that all wavelengths of each link in the network are fully utilised.

$$U = \frac{\sum_{i,j}^{i<j,\, i\neq j}\left(1-\frac{\lambda_{i,j}^{a}}{\lambda_{i,j}^{T}}\right)}{\sum_{i,j}^{i<j,\, i\neq j} 1}$$

5.2.1.3. Object-oriented framework

The UML class diagram (implemented with Rational Rose) of the RWA model is given in Figure 15. This framework describes the relation and contents of each class. The different classes of the model are the following: Generator, Sink, GmplsRouter, Node, LinkInfo, NodeInfo, ExplicitRoute and LambdaCap.

GmplsRouter is the central class of the simulation. It is composed of several databases implemented as vectors of objects.

- Links is a vector of objects called LinkInfo. Each LinkInfo object contains all the necessary information on a certain link (i,j) in the network. Especially, it contains instances of the LambdaCap class.
- Nodes is a vector of objects called Node. Each node object contains information about a node of the network, especially if it needs to be extracted to the graph that will serve as an input to the calcShortestPath method of the GmplsRouter class.
- PhotonicDb is a vector of objects called NodeInfo. Each NodeInfo objects is comprised of one record of the forwarding database of the GmplsRouter module. This database is filled up by the calcShortestPath and calcExplicitRoute methods. The NodeInfo contains one instance of the ExplicitRoute class and one instance from the LambdaCap class.

The class ExplicitRoute is used to represent the explicit route (shortest path) from the GmplsRouter node to a destination node in the network. Several methods have been implemented to facilitate the use of this object.

The class LambdaCap represents the wavelength capability of a link or an explicit route. The reader will note that this class implements the wavelength assignment and the metric schemes.

Figure 15 RWA simulation class diagram

As this framework subjects, it will be very easy for any further work to reuse this model. Thanks to the object-oriented programming in C++, it will just be necessary to overload certain methods of the model to use different and more efficient RWA schemes. For instance, one can simply add another method in GmplsRouter if another routing algorithm has to be tested. Similarly, other wavelength assignment schemes and more evolved metric can be added to the LambdaCap class with very little change to the model framework.

5.2.1.4. OMNeT++ Modules description

As the object oriented framework suggests it, the classes are organised in different OMNeT++ files. The simulation code is shared amongst the following. All those files are given in Annex.

File name	General description
`includes.h`	Defines the constants and the different types of messages
`gen.cc`	Implements the generator of an end system class
`sink.cc`	Implements the sink of an end system class
`gmplsRouter.cc`	Implements the Gmpls Router class. This is the main class of the system. Includes
`gmplsRouter.h`	Implements the Node, NodeInfo, LinkInfo, LambdaCap and ExplicitRoute classes.
`abilene.ned`	Describes the topology of the Abilene transport network
`omnetpp.ini`	Describes the different parameters for a run in OMNeT++

5.2.2. SIMULATION RESULTS PROCESS

In order to have a good estimation of the blocking probability, it is necessary to run several times the algorithm with different requests arrival parameters. Each run is assigned a different random generator, called a random seed.

For selecting good seeds, the seedtool program of OMNeT++ is used. For instance, the command `seedtool g seed0 dist n` generates 'n' seeds that are 'dist' apart, starting at ‘seed0’. The bash script presented in Figure 16 was used to simulate 20 runs with different seeds.

```
#! /bin/bash
for seed in `seedtool g 1 10000000 20`
do
    (
      echo "random-seed = ${seed}"
      echo "output-vector-file = rwa-${seed}.vec"
    ) >parameters.ini
    ./rwa
done
```

Figure 16 Script to perform 20 independant runs

Each run outputs a vector (a cOutVector in OMNeT++). An extract of such a vector is given in the listing of Figure 17.

```
vector 36  "net.endsystem9.sink"  "blockingVsLinkUtil"  2
36    189.780099  0     0.633333333
vector 37  "net.endsystem9.sink"  "blockingVsLinkUtil"  2
37    194.075275  0     0.65
vector 38  "net.endsystem11.sink"  "blockingVsLinkUtil"  2
38    196.36799   0     0.675
vector 39  "net.endsystem11.sink"  "blockingVsLinkUtil"  2
39    197.664505    1     0.675
```

Figure 17: Extract of one simulation run

Each vector is the response to one request made by an end system. There are label lines (beginning with vector) and data lines.

- A `vector` line introduces a new vector. Its columns are: vector ID, module of creation, name of `cOutVector` object, multiplicity of data (single numbers or pairs will be written).
- Lines beginning with numbers are data lines. The columns: vector ID, current simulation time, and one or two double values. Here the first one identifies if the request has been blocked (1) or fulfilled (0). The second value represents the link utilisation when the connection request was made.

Several UNIX utilities such as awk, sed, sort and grep are used to process the vectors and obtain the necessary information, that is the third and fourth fields of the data lines. An extract of the actual C shell script used to process the raw data is given in Annex.

Finally, a short C++ program is used to determine the blocking probality versus the link utilisation data. This program is used to calculate the probability of blocking P at certain link utilisation. This is done by making the average of the blocking on a certain interval ΔU. This program can be found in Annex.

6. RESULTS AND DISCUSSION

One of the goals of this project was to estimate how the developed RWA algorithm performs relatively to the blocking of requests in the network. As it will be seen, because wavelength conversion is not used, the wavelength continuity constraint leads to a high level of blocking when the load increases. First, wavelength assignment schemes are compared. Secondly, the blocking in the network as a function of the average link utilisation is described, analysed and discussed.

6.1. WAVELENGTH ASSIGNMENT SCHEMES COMPARISONS

It was found that the first-fit and the random wavelength assignments perform in very similar ways, either when blocking is plotted versus average link utilisation or versus average node traffic. Actually, the results show that there are no real differences in performance between the two schemes. There was no real explanation found; except that it is maybe a result of the mix of those WA schemes with the routing and reservation schemes that were developed. Nevertheless, it highlights the importance of a model to discover such relations that are not obvious. In the following, the simulation uses a first-fit wavelength assignment scheme.

6.2. BLOCKING AND AVERAGE LINK UTILISATION

The blocking probability P is plotted as a function of the link utilisation U for different values of $\lambda_{i,j}^{T}$, the total number of wavelengths in a fibre. $\lambda_{i,j}^{T}$ is varied between 8 and 24 by increment of 8 and different schemes, including simple TAW metric, enhanced TAW metric, first-fit and random wavelength assignment are tested. The processed data is given in Table 5, Table 6 and Table 7.

- $\lambda_{i,j}^{T}$ = 8

Simple TAW	Enhanced TAW	Aver. Link Utilisation	
Blocking	Blocking	%	Used lambdas
0,00	0,00	3	0,2
0,00	0,00	8	0,6
0,02	0,02	12	0,96
0,00	0,00	18	1,44
0,02	0,00	23	1,84
0,00	0,00	28	2,24
0,00	0,00	33	2,64
0,05	0,00	38	3,04
0,00	0,04	42	3,36
0,17	0,19	47	3,76
0,29	0,31	52	4,16
0,33	0,48	57	4,56
0,57	0,53	62	4,96
0,65	0,54	68	5,44
0,72	0,66	73	5,84
0,79	0,75	78	6,24
0,86	0,82	83	6,64
0,87	0,87	88	7,04
0,90	0,84	93	7,44

Table 5 Processed data for 8 wavelengths per fibre

- $\lambda_{i,j}^{T}$ = 16

Simple TAW	Enhanced TAW	Link Utilisation	
Blocking	Blocking	%	Used Lambdas
0,00	0,00	3	0,4
0,01	0,01	8	1,2
0,01	0,01	12	1,92
0,02	0,02	18	2,88
0,04	0,00	23	3,68
0,03	0,00	28	4,48
0,00	0,00	33	5,28
0,02	0,01	38	6,08
0,00	0,01	42	6,72
0,02	0,00	47	7,52
0,14	0,13	52	8,32
0,43	0,39	57	9,12
0,60	0,48	62	9,92
0,66	0,61	68	10,88
0,76	0,70	73	11,68
0,79	0,77	78	12,48
0,84	0,83	83	13,28
0,85	0,84	88	14,08
0,82	0,86	93	14,88

Table 6 Processed data for 16 wavelengths per fibre

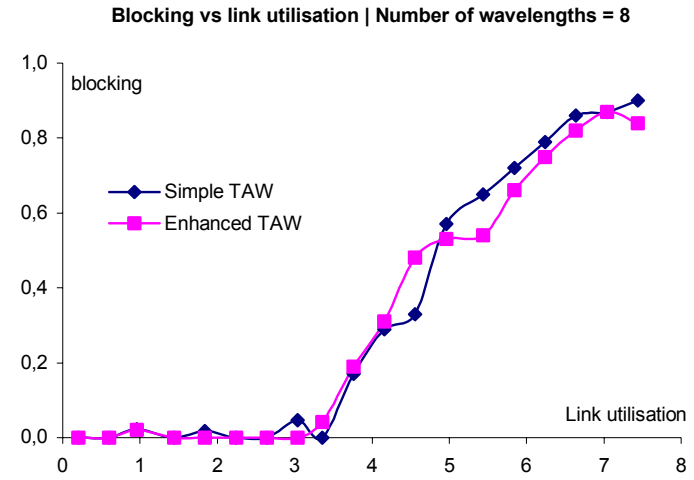
- $\lambda_{i,j}^{T}$ = 24

TAW1 - FF	TAW2 - FF	Link Utilisation	
Blocking	Blocking	%	Used Lambdas
0,01	0,01	3	0,6
0,01	0,01	8	1,8
0,01	0,02	12	2,88
0,02	0,03	18	4,32
0,01	0,01	23	5,52
0,01	0,01	28	6,72
0,03	0,01	33	7,92
0,05	0,02	38	9,12
0,05	0,02	42	10,08
0,07	0,00	47	11,28
0,14	0,12	52	12,48
0,42	0,35	57	13,68
0,55	0,53	62	14,88
0,65	0,62	68	16,32
0,77	0,70	73	17,52
0,84	0,78	78	18,72
0,85	0,86	83	19,92
0,88	0,88	88	21,12

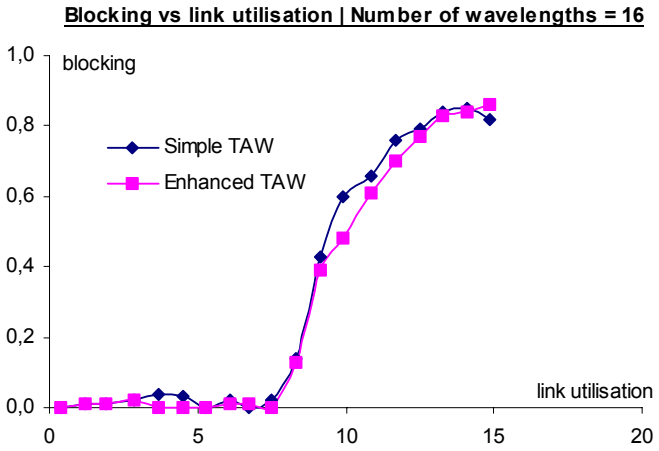
Table 7 Processed data for 24 wavelengths per fibre

The plots of the blocking versus the average link utilisation for different values of $\lambda_{i,j}^{T}$ are given in Figure 18. Graphs a), b) and c) represent the blocking versus the average link utilisation for $\lambda_{i,j}^{T}$ = 8, $\lambda_{i,j}^{T}$ = 16, $\lambda_{i,j}^{T}$ = 24 respectively. In each graph, the blocking versus the average link utilisation is plotted for the two metrics that were expected to be compared, that is simple TAW and enhanced TAW.

a)



b)



c)

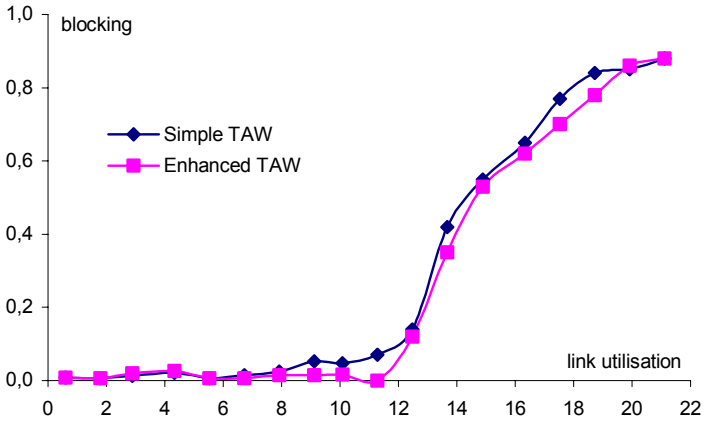


Figure 18. Blocking versus link utilisation for different $\lambda_{i,j}^{T}$

a) Number of wavelengths: 8.

b) Number of wavelengths: 16

c) Number of wavelengths: 24

6.2.1. GENERAL OBSERVED BEHAVIOURS FOR THE BLOCKING IN AN ION

The different graphs show a somewhat similar behaviour regarding to the blocking in the ION. At low link utilisation, that is when less than about half of the wavelengths are used, the blocking probability is extremely low (almost zero). When more than half of the wavelengths are used on average on the links in the network, the blocking increases suddenly and steadily and tends to 100 % of blocking when the average link utilisation goes to 100 %.

A low blocking level is defined for a blocking inferior to 5 %. Higher blocking is for blocking superior or equal to 5 %. If this threshold is used, the following is observed from the graphs:

- The low blocking threshold is reached by the simple TAW at 33 % link utilisation while the enhanced TAW reaches it at 42 % for $\lambda_{i,j}^{T}$ = 8.
- The low blocking threshold is reached by the simple TAW at 47 % link utilisation while the enhanced TAW reaches it at 47 % for $\lambda_{i,j}^{T}$ = 16.
- The low blocking threshold is reached by the simple TAW at 33 % link utilisation while the enhanced TAW reaches it at 47 % for $\lambda_{i,j}^{T}$ = 24.

As a rule of thumb, whatever the metric is used, it seems that high blocking appears at higher link utilisation when more wavelengths are used. Nevertheless, a real behaviour is difficult to interpret here because only 3 different numbers of wavelengths have been used.

However, even with 24 wavelengths, somewhat poor performance is observed. Indeed, relatively high blocking appears fastly at higher link utilisation than about 50 %. This poor performance is mostly due to the hypotheses of this work. Especially, no wavelength conversion has been taken into account ; the wavelength continuity constraint then leads to much higher blocking at relatively medium link utilisation.

6.2.2. COMPARISON BETWEEN SIMPLE AND ENHANCED TAW METRICS

In this work, two metrics are compared, simple TAW and enhanced TAW. The performance of the two metrics are compared, that is their performance relatively to the blocking at low utilisation and high utilisation are estimated.

U_t is defined as the threshold link utilisation where blocking is equal to 5 %. The data comparing the two metrics are given in Table 8, Table 9 and Table 10.

- $\lambda_{i,j}^{T}$ = 8

	Blocking difference	Blocking diff at low blocking	Blocking diff at high blocking
	0%	0%	
	0%	0%	
	0%	0%	
	0%	0%	
	2%	2%	
	0%	0%	
	0%	0%	
	5%		5%
	-4%		-4%
	-2%		-2%
	-2%		-2%
	-15%		-15%
	4%		4%
	11%		11%
	6%		6%
	4%		4%
	4%		4%
	0%		0%
	6%		6%
Mean	**5%**	**1%**	**7%**

Table 8 Comparison data between simple and enhanced TAW metrics for 8 wavelengths per fibre

- $\lambda_{i,j}^{T}$ = 16

	Blocking difference	Blocking diff at low blocking	Blocking diff at hith blocking
	0%	0%	
	0%	0%	
	0%	0%	
	0%	0%	
	4%	4%	
	3%	3%	
	0%	0%	
	1%	1%	
	-1%	-1%	
	2%	2%	
	1%		1%
	4%		4%
	12%		12%
	5%		5%
	6%		6%
	2%		2%
	1%		1%
	1%		1%
	-4%		-4%
Mean	**3%**	**2%**	**4%**

Table 9 Comparison data between simple and enhanced TAW metrics for 16 wavelengths per fibre

- $\lambda_{i,j}^{T}$ = 24

Blocking diff	At low util	At high util
0%	0%	
0%	0%	
-1%	-1%	
-1%	-1%	
0%	0%	
1%	1%	
1%	1%	
4%		4%
3%		3%
7%		7%
2%		2%
7%		7%
2%		2%
3%		3%
7%		7%
6%		6%
-1%		-1%
0%		0%
Mean	**3%**	**1%**	**3%**

Table 10 Comparison data between simple and enhance TAW metrics for 24 wavelengths per fibre

The major observations from these are given in the following table.

	Simple TAW	Enhanced TAW	Performance comparison (mean blocking difference)	
	U_t		$U < U_t$	$U > U_t$
$\lambda_{i,j}^{T}$ **= 8**	33 %	42 %	1 %	7 %
$\lambda_{i,j}^{T}$ **= 16**	47 %	47 %	2 %	4 %
$\lambda_{i,j}^{T}$ **= 24**	33 %	47 %	1 %	3 %

Tableau 11 Recapitulation table for metrics comparison

It seems difficult to explain why in the case of $\lambda_{i,j}^{T}$ = 16, blocking superior to 5 % appears at relatively high link utilisation for the simple TAW. One answer would be that only 20 runs have been used, which is not really sufficient to have very exact values of the blocking probability at one given link utilisation.

The main conclusions are as follows:

- At low link utilisation, simple TAW and enhanced TAW perform the same way. The blocking in this case stays very low (strictly inferior to 5 %).
- At higher link utilisation, the enhanced TAW metric performs much better than the simple TAW metric. This is particularly true when very few wavelengths are used.
- When the number of wavelengths increase, blocking superior to 5 % appears at higher link utilisation.

In general, it has been observed that the enhanced TAW metric performs better than the simple TAW metric at higher utilisation. One reason is that the enhanced metric tries to minimizes the weight of the explicit route while still trying to minimize the probability of blocking for further requests. Conversely, the simple TAW metric does not take into account any probabilities. It just tries to minimize the total link utilisation on a possible explicit route from the ingress node to the egress node.

7. CONCLUSIONS

Built on different researches in the field, a RWA scheme including a routing, a wavelength assignment and a reservation scheme were proposed. First, the proposed routing scheme was based on the implementation of OSPF and different associated drafts that extend OSPF capabilities for routing in all-optical networks. A proposal for an extension of the ISCD field in type 10 opaque LSA was given, allowing to consider two different metrics for the shortest path calculation based on Dijkstra algorithm. Secondly, two different wavelength assignment schemes were tested, that is first-fit and random. Finally, a simplified parallel reservation scheme was implemented in order to estimate the performance of the routing and wavelength assignment schemes previously mentioned.

An object-oriented framework of the RWA scheme was designed in order to ease software reuse for further work. This framework was used to develop a simulation model of the proposed schemes in OMNeT++.

The model was tested under different parameters, including total number of wavelengths per fibre and different schemes: simple TAW, enhanced TAW for the routing scheme, first-fit and random for the wavelength assignment. The ION tested was the Abilene network, the research network linking U.S. universities.

The main conclusions of the simulation model are:

- The probability of a request to be blocked P is very low and stable (< 5 %) for link utilisation varying between 0 % and up to a threshold U_t varying between 33 % (worst schemes and parameters combination) and 47 % (better schemes and parameters combination). At higher link utilisations than U_t, ($U > U_t$), blocking increases very fastly when link utilisation increases.
- The enhanced TAW metric implemented performs better at link utilisations higher than the simple TAW metric particularly when the number of wavelengths per fibre is low. Conversely, at low link utilisation ($U < U_t$), both metrics have very similar results whatever the number of wavelengths are used.
- The wavelengths assignment schemes implemented, that is first-fit and random, yield extremely similar performance when associated with any routing and reservation schemes implemented in this work.

8. RECOMMENDATIONS FOR FURTHER WORK

There are currently numerous research studies that can be build onto the RWA algorithm proposed in this work. The purpose of any of this work may be to build a more accurate model of an all-optical GMPLS-based ION. Further work may involve building, developing and simulating a more efficient RWA algorithm or reservation protocol. Finally, other possible work include building and simulating a restoration scheme in an ION.

8.1. IMPROVEMENTS OF THE RWA ALGORITHM

The RWA algorithm proposed in this work should be improved so that the blocking probability is lower than the one demonstrated, particularly at link utilisation superior than 50 %. A way to diminish the blocking probability of requests is to use wavelength conversion. The RWA should be also able to consider other physical constraints that only the number of available and total wavelengths as it was done in this work.

8.1.1. WAVELENGTH CONVERSION CAPABILITY

It is possible to improve the blocking probability of the RWA algorithm by using wavelength converters [14, 38, 46-48]. It is necessary to bear in mind that wavelength conversion is still a very costly solution, because most of the wavelength converters are currently laboratory devices. This is one of the reasons why it was not considered in this work.

Thus, in practical implementations using wavelength conversion, not every node in the ION will have a wavelength conversion capability. The nodes should be able to know at which node there is a possible wavelength conversion. Another good idea is that the use of wavelength conversion should be performed only if blocking in the system attains a certain threshold, that, if passed, is undesirable [38].

8.1.2. PHYSICAL CONSTRAINTS

It is possible to extend the RWA algorithm to take into account other physical constraints than wavelength availability. For instance, regeneration of the signal could impose to go through a node after a certain distance. A study should be followed on other physical constraints to be taken into account when performing the RWA algorithm, such as the use of wavelength converters, the use of signal regeneration to combat ASE and PMD [38] [32].

8.2. STUDY OF A RESERVATION PROTOCOL IN A GMPLS-BASED NETWORK

In this work, a simple parallel reservation has been implemented. Nevertheless, more complex reservation protocols can significantly decrease the blocking probability of the requests in the ION [26, 49, 50]. Two reservation protocols are proposed in the GMPLS architecture: CR-LDP and RSVP-TE [33]. It seems that RSVP-TE has the favour in the industry, since it is implemented by Cisco and Juniper. The advantage of using RSVP is that it is possible to reuse RSVP implementations that are mainly free of bugs, because RSVP is already an “old” protocol. Conversely, CR-LDP is based on LDP. This protocol has been mainly designed and supported by Nortel for the LDP and signalling protocol in MPLS. This is a new software, that may not be as mature as RSVP.

There are a lot of studies and drafts on lightpath set-up in GMPLS-based networks implemented with RSVP-TE [30] [20] [31] [51] [52] [37] [36] .

8.3. RESTORATION CAPABILITIES

Restoration capabilities should be added to this work’s model, so that the breakdown of a node or a link be bypassed. Several studies currently deal with this problem and are implemented according to several GMPLS drafts [31] [53].

9. REFERENCES

1. Stallings, W., ed. *High Speed Networks and Internets, Performance and Quality of Service*. 2nd ed. 2002, Prentice Hall.
2. Dijkstra, E., ed. *A Note on Two Problems in Connection with graphs*. 1959, Numerical Mathematics.
3. Laffra, C., *Dijkstra's Shortest Path Algortithm Animation in Java*. 2002.
4. Stallings, W., *Multi Protocol Label Switching*. Sept. 2001, Cisco.
5. Black, U., *Optical networks, Third Generation Transport systems*. 2002: Prentice Hall PTR.
6. Chlamtac, I.G., A.; Karmi, G., *Lightpath communications: an approach to high bandwidth optical WAN's.* Communications, IEEE Transactions on, July 1992. **40**(7): p. 1171 -1182.
7. Banerjee, D. and B. Mukherjee, *A Practical Approach for Routing and Wavelength Assignment in Large Wavelength-Routed Optical Networks.* IEEE Journal on Selected Areas in Communications, 1996. **14**(5): p. 903-908.
8. Baroni, S. and P. Bayvel, *Wavelength Requirements in Arbitrary Connected Wavelength-Routed Optical Networks.* IEEE/OSA Journal of Lightwave Technology, 1997. **15**(2): p. 242-251.
9. Alanyali, M. and E. Ayanoglu, *Provisioning algorithms for WDM optical networks.* IEEE/ACM Transactions on Networking, Oct. 1999. **7**(5): p. 767 -778.
10. Ohta, S.G. and A. Greca. *Comparison of routing and wavelength assignment algorithms for optical networks*. in *High Performance Switching and Routing, 2001 IEEE Workshop on , 2001*. 2001.
11. Bala, K. and T.E. Stern. *Algorithms for routing in a linear lightwave network*. in *INFOCOM '91. Proceedings. Tenth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking in the 90s.* 1991: IEEE.
12. Birman, A. and A. Kershenbaum. *Routing and Wavelength Assignment Methods in Single-Hop All-Optical Networks with Blocking*. in *Proceedings, IEEE Infocom '95*. Apr. 1995. Boston, MA.
13. Harai, H.M., M.; Miyahara, H. *Performance of alternate routing methods in all-optical switching networks*. in *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*. 1997.
14. Ramamurthy, S.M., B. *Fixed-alternate routing and wavelength conversion in wavelength-routed optical networks*. in *Global Telecommunications Conference, 1998. GLOBECOM 1998. The Bridge to Global Integration. IEEE*. 1998.
15. Mokhtar, A.A., M., *Adaptive wavelength routing in all-optical networks.* Networking, IEEE/ACM Transactions on, April 1998. **6**(2): p. 197 -206.
16. Ramaswami, R.S., A. *Distributed network control for wavelength routed optical networks*. in *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation., Proceedings IEEE*. 1996.
17. Zang, H., et al. *Connection management for wavelength-routed WDM networks*. in *Global Telecommunications Conference, 1999. GLOBECOM '99*. 1999.
18. Fabry-Asztalos, T., N.M. Bhide, and K.M. Sivalingam. *Adaptive Weight Wunctions for Shortest Path Routing Algorithms for Multi-Wavelength Optical WDM Networks*. in *IEEE Intl. Conference on Communications*. 2000. New Orleans, LA.
19. Apostolopoulos, G., et al., *QoS Routing Mechanisms and OSPF Extensions - RFC 2676*. 1999.
20. Zang, H., et al., *Dynamic lightpath establishment in wavelength routed WDM networks.* IEEE Communications Magazine, Sept. 2001. **39**(9): p. 100 -108.
21. Li, L. and A.K. Somani, *Dynamic wavelength routing using congestion and neighborhood information.* Networking, IEEE/ACM Transactions on, Oct. 1999. **7**(5): p. 779 -786.
22. Jue, J.P. and G. Xiao. *An adaptive routing algorithm for wavelength-routed optical networks with a distributed control scheme*. in *Computer Communications and Networks, 2000. Proceedings. Ninth International Conference on*. 2000.
23. Zang, H., J.P. Jue, and B. Mukherjee, *Review of Routing and Wavelength Assignment Approaches for Wavelength-Routed Optical WDM Networks*, in *Optical Networks Magazine*. 2000. p. 47-60.

24. Subramaniam, S. and R.A. Barry. *Wavelength assignment in fixed routing WDM networks*. in *Communications, 1997. ICC '97 Montreal, Towards the Knowledge Millennium. 1997 IEEE International Conference on*. 1997.
25. Zhang, X. and C. Qiao. *Wavelength Assignment for Dynamic Traffic in Multi-fiber WDM Networks,"*. in *Proceedings, 7th International Conference on Computer Communications and Networks*. Oct. 1998. Lafayette, LA.
26. Yuan, X., et al., *Distributed Control Protocols for Wavelength Reservation and Their Performance Evaluation.* Photonic Network Communications, 1999. **1**(3): p. 207-218.
27. Sotica, A.G. and A. Sengupta. *On a Dynamic Wavelength Assignment Algorithm for Wavelength-Routed All-Optical Networks*. in *Proceedings, SPIE/IEEE/ACM OptiComm 2000*. Oct. 2000. Dallas, TX.
28. Mei, Y. and C. Qiao. *Efficient Distributed Control Protocols for WDM All-Optical Networks*. in *Proceedings, International Conference on Computer Communication and Networks*. Sept. 1997. Las Vegas, NV.
29. Sengupta, A., et al., *Algorithms for Dynamic Routing in All-Optical Networks.* Photonic Network Communications, 2000. **2**(2): p. 163-184.
30. Widjaja, I. and A.I. Elwalid. *Study of GMPLS lightpath setup over lambda-router networks*. in *Communications, 2002. ICC 2002. IEEE International Conference on*. 2002.
31. Banerjee, A., et al., *Generalized multiprotocol label switching: an overview of signaling enhancements and recovery techniques*, in *IEEE Communications Magazine*. 2001. p. 144 - 151.
32. Strand, J., A.L. Chiu, and R. Tkach, *Issues for routing in the optical layer*, in *IEEE Communications Magazine*. Feb. 2001.
33. Ashwood-Smith, P., et al., *Generalized Multi-Protocol Label Switching (GMPLS) Architecture - draft-ietf-ccamp-gmpls-architecture-02.txt*. 2002.
34. Kompella, K., et al., *Routing Extensions in Support of Generalized MPLS - draft-ietf-ccamp-gmpls-routing-04.txt*. 2002.
35. Kompella, K., et al., *OSPF Extensions in Support of Generalized MPLS - draft-ietf-ccamp-ospf-gmpls-extensions-07.txt*.
36. Berger, L., et al., *Generalized MPLS - Signaling Functional Description - draft-ietf-mpls-generalized-signaling-08.txt*. 2002.
37. Berger, L., et al., *Generalized MPLS Signaling - RSVP-TE Extensions - draft-ietf-mpls-generalized-rsvp-te-07.txt*. 2002.
38. Ken-ichi Sato, N.Y., Yoshihiro Takigawa, Masafumi Koga, Satoru Okamoto, and E.O. Kohei Shiomoto, and Wataru Imajuku, NTT Corporation, *GMPLS-Based Photonic Multilayer Router (Hikari Router) Architecture: An Overview of Traffic Engineering and Signaling Technology*, in *IEEE Communications Magazine*. March 2002. p. 96 -101.
39. *OSPFD Release 2.0*.
40. Katz, D., D. Yeung, and K. Kompella, *Traffic Engineering Extensions to OSPF - draft-katz-yeung-ospf-traffic-06.txt*. 2002, IETF.
41. Coltun, R., *The OSPF Opaque LSA Option, RFC 2370*. 1998.
42. Hecht, J., *Understanding Fiber Optics*. 4th ed. 2002: Prentice Hall.
43. Moy, J., *OSPF Version 2 - RFC 2328*. 1998.
44. Varga, A., *OMNeT++*. 2002.
45. Working Group: Architecture, O.P., PLL, & Signaling Working Groups, *User Network Interface (UNI) 1.0 Signaling Specification*. 2001, Optical Internetworking Forum (OIF).
46. Swaminathan, M.D.S., K.N. *Practical routing and wavelength assignment algorithms for all optical networks with limited wavelength conversion*. in *Communications, 2002. ICC 2002. IEEE International Conference on*.
47. Xiao, G., Y.-W. Leung, and K.-W. Hung, *Two-stage cut saturation algorithm for designing all-optical networks.* Communications, IEEE Transactions on, June 2001. **49**(6): p. 1102 - 1115.
48. Zhang, Y., et al. *An efficient heuristic for routing and wavelength assignment in optical WDM networks*. in *Communications, 2002. ICC 2002. IEEE International Conference on*. 2002.
49. Shami, A., et al., *Performance Evaluation of Two GMPLS-Based Distributed Control and Management Protocols for Dynamic Lightpath Provisioning in Future IP Networks.* IEEE, 2002.

50. Zheng, J. and H.T. Mouftah. *Routing and wavelength assignment for advance reservation in wavelength-routed WDM optical networks*. in *Communications, 2002. ICC 2002. IEEE International Conference on*. 2002.
51. Semaria, C., *RSVP signalling extensions for MPLS TE*. 2002, Juniper.
52. Braden, R., et al., *RFC2205 Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification*. 1997.
53. Sengupta, S. and R. Ramamurthy, *From network design to dynamic provisioning and restoration in optical cross-connect mesh networks: an architectural and algorithmic overview.* IEEE Network, 2001. **15**(4): p. 46 -54.

10. APPENDIX

10.1. SIMULATION PROGRAM

10.1.1. INCLUDES.H

```
// Defines the different types of messages
#include <map>

#ifndef __INCLUDES_H
#define __INCLUDES_H

typedef std::map< int, int, std::less<int> > mid;

const int UNDEF  = 60000;          // undefined node
const double INF = 60000;          // infinity metric

const int hops = 1;
const int TAW1 = 2;
const int TAW2 = 3;

const int waFF = 1;
const int waR = 2;

const long ROUTING = 1;
const long REQUEST = 2;
const long RESPONSE = 3;
const long RESERVE = 4;
const long RESERVE_ACK = 5;
const long TAKEDOWN = 6;

const long PT_HELLO = 1;           // Hello packet
const long PT_DD = 2;              // Database Description
const long PT_LSREQ = 3;           // Link State Request
const long PT_UPD = 4;             // Link State Update
const long PT_LSACK = 5;           // Link State Acknowledgment

/* Definitions for the "lsType" field */
const long LST_RTR = 1;            // Router-LSAs
const long LST_NET = 2;            // Network-LSAs
const long LST_SUMM = 3;           // Summary-link LSAs (inter-area routes)
const long LST_ASBR = 4;           // ASBR-summaries (inter-area)
const long LST_ASL = 5;            // AS-external_LSAs
const long LST_GM = 6;             // Group-membership-LSA (MOSPF)
const long LST_NSSA = 7;           // NSSA externals
const long LST_EXATTR = 8;         // External-attributes LSA
const long LST_LINK_OPQ = 9;       // Link-scoped Opaque-LSA
const long LST_AREA_OPQ = 10;      // Area-scoped Opaque-LSA - the one that we use
const long LST_AS_OPQ = 11;        // AS-scoped Opaque-LSA
const long MAX_LST = 11;           // Maximum number of supported LSA types

/* Assigned TLV Types*/
const long TLV_T_ROUTERADDR = 1;   // Router Address top-level TLV
const long TLV_T_LINK = 2;         // Link top-level TLV

/* Assigned sub TLV Types*/
const long SUB_TLV_T_LINKTYPE = 1; // sub TLV link type
                                   // identifies the type of the link
                                   // in this work, 1 is used (point to point)
const long SUB_TLV_T_LINKID = 2;   // sub TLV link id
                                   // identifies the other end of the link
const long SUB_TLV_T_ISCD = 15;    // sub TLV Interface Sw. Cap. Descriptor (ISCD)
                                   // Defines the switching capabilities

/* Assigned switching capability descriptors for the ISCD sub tlv */
const long SW_CAP_T_LSC = 150;     // Lambda Switching Capable link
#endif
```

10.1.2. GEN.CC

```
#include <omnetpp.h>
#include "includes.h"

class Generator : public cSimpleModule {
  Module_Class_Members(Generator,cSimpleModule,16384)
  int myAddress;
  int total_sentmsg;
  int numberPacketsToSend;
  virtual void activity();
};

Define_Module( Generator );

void Generator::activity() {

  cPar rnd("rnd");
  int num_nodes = par("num_endnodes");
  myAddress = par("address");
  int degree = par("degree");
  int totalLambdas = par("totalNumberLambdas");
  double delay = par("delta");
  numberPacketsToSend = (degree * totalLambdas);
  simtime_t d;
  total_sentmsg = 0;

  // Idle time between 0 and 1 min
  simtime_t idleTime = uniform(0, 60);

  // Wait for a random time before sending packets
  cout << "Time to wait in " << myAddress << " before sending any packet = "
       << simtimeToStr(idleTime) << endl;
  wait(idleTime);
  cMessage *toSend = new cMessage;
  toSend->addPar("sendRequest") = (bool) true;
  scheduleAt(simTime() + 0.01, toSend);

  for (;;) {
    cout << "we are here\n";
    cMessage *mes = receive();
    cout << "we are here\n";

    if (mes->hasPar("sendRequest")) {
      if (total_sentmsg <= numberPacketsToSend) {
      cMessage *msg = new cMessage();

      // add the type of packet
      msg->addPar("messageType") = REQUEST;

      //select a destination randomly
      int dest = myAddress;
      while (dest == myAddress) {
        dest = (int) intuniform(1,num_nodes);
      }

      d = simTime();

      // set a name for packet
      cout << endl << "CONNECTION REQUEST from " << myAddress << " to "
           << dest << " at time " << simtimeToStr(d) << endl;

      //add dest to the message
      msg->addPar("destAddr") = dest;

      //send the packet
      send(msg,"out");
      total_sentmsg++;

      if (total_sentmsg < numberPacketsToSend) {

        // Next packet to send scheduled in delay time
        cMessage *nextMes = new cMessage;
        nextMes->addPar("sendRequest") = (bool) true;

        simtime_t timeForNextPacket = simTime() + delay;
        scheduleAt(timeForNextPacket, nextMes);

        // Wait 3 minutes to tear down the route
```

```
      }
      }
    }

    if (mes->hasPar("sendTearDown")) {

    }
  }
}
```

10.1.3. SINK.CC

```
#include <omnetpp.h>
#include "includes.h"

class Sink : public cSimpleModule {
  Module_Class_Members(Sink,cSimpleModule,16384)

  virtual void activity();
};

Define_Module( Sink );

void Sink::activity() {
  int numberPacketsToReceive = 0;
  int numberPacketsReceived = 0;
  int numberRequestAccepted = 0;
  int numberRequestBlocked = 0;
  int myAddress = par("address");
  int degree = par("degree");
  int totalLambdas = par("totalNumberLambdas");
  numberPacketsToReceive = degree * totalLambdas;

  for(;;) {
    //cOutVector blockingVsNodeTraffic("blockingVsNodeTraffic",2);
    cOutVector blockingVsLinkUtil("blockingVsLinkUtil",2);
    cOutVector blockingVsRequest("blockingVsRequest",2);

    cMessage *msg = receive();
    numberPacketsReceived++;
    bool requestAccepted = msg->par("accepted");
    double linkUtil = msg->par("linkUtil");
    double nodeTraffic = msg->par("nodeTraffic");

    if (requestAccepted) {
      numberRequestAccepted++;
      blockingVsRequest.record(0, numberPacketsReceived);
      blockingVsLinkUtil.record(0, linkUtil);
    }
    else {
      blockingVsRequest.record(1, numberPacketsReceived);
      blockingVsLinkUtil.record(1, linkUtil);
    }

    simtime_t t = simTime()-msg->timestamp();

    numberRequestBlocked = numberPacketsReceived - numberRequestAccepted;

    cout << endl << "SINK : response for a request"  << " at "
       << simtimeToStr(t) << endl
       << "In node " << myAddress << endl
       << "Total Request accepted = " << numberRequestAccepted << endl
       << "Total Request blocked  = " << numberRequestBlocked << endl
       << "the average link utilisation is " << linkUtil << endl
       << "the node traffic for " << myAddress << " is " << nodeTraffic << endl;

    delete msg;
  }
}
```

10.1.4. GMPLSROUTER.CC

```
#include <omnetpp.h>
#include <iostream>
#include <vector>
#include <map>
#include <cassert>
```

```
#include <algorithm>
#include <cmath>
#include "includes.h"
#include "gmplsRouter.h"

class GmplsRouter : public cSimpleModule {
  Module_Class_Members(GmplsRouter, cSimpleModule, 56384)

    int myAddress;
  double degree;
  double totalLambdas;

  // For stats purposes
  int numberNodes;
  int numberLinks;
  int respToBeReceived;
  int respReceived;
  int numberLightpathEstablished;

  // Types of schemes used by this node
  int metricType;  // type of the metric: hops, TAW1, TAW2, etc
  int waType;      // type of wa: first fit, random, etc

  cTopology graphe;
  std::vector<NodeInfo> photonicDb;
  std::vector<NodeInfo> temp;

  std::vector<LinkInfo> links;
  std::vector<Node> nodes;

  // map
  mid pairs;

  virtual void activity();
  virtual void finish();
  virtual void printLinks();
  virtual void printPhotonicDb();
  virtual void printTemp();
  virtual void printNodes();
  virtual void printMap();
  virtual void initGraph();
  virtual void sendUpdate(cModule *destMod, int neighbourAddress, int lambda);
  virtual void sendReserve(int lambda, cModule *destMod, int beforeNode);
  virtual void sendReserveResponse(bool accepted, int src);
  virtual void sendResponseToSink (bool accepted);
  virtual void sendTakedown(cModule *destMod, int lambda, int neighbour);
  virtual void receiveUpdate (cMessage *mes);
  virtual void receiveResponse(cMessage *mes);
  virtual void receiveRequest(cMessage *mes);
  virtual void receiveReserve(cMessage *mes);
  virtual void receiveTakedown(cMessage *mes);
  virtual void takedownRoute(int dest, int lambda);
  virtual void initTemp();
  virtual void calcExplicitRoute();
  virtual void calcShortestPath();
  virtual double linkUtil();
  virtual double nodeTraffic();

};

// Module registration
Define_Module(GmplsRouter);

// *****************  ACTIVITY  ******************************
void GmplsRouter::activity() {
  myAddress = par("address");
  totalLambdas = par("totalNumberLambdas");
  degree = par("degree");

  // parameters for testing different schemes
  metricType = par("metricType");
  waType = par("waType");

  //cout << "Enter activity - We are in module: " << myAddress << endl;

  // extract all the nodes in the topology
  graphe.extractByModuleType("GmplsRouter", NULL);
  numberNodes = graphe.nodes();
  // cout << "The graphe extracted has " << numberNodes << " nodes" << endl;

  // fills up the links and nodes databases
```

```
//cout << "Init the links and nodes databases " << endl;
numberLinks = 0;
for (int i=0 ; i<numberNodes ; i++) {
  sTopoNode *node = graphe.node(i);
  cModule *nodeMod = node->module();
  int nodeAddress = nodeMod->par("address");
  Node n(nodeAddress, true);
  nodes.push_back(n);

  for (int j=0; j<node->outLinks(); j++) {
    sTopoNode *neighbour = node->out(j)->remoteNode();
    cModule *neighbourMod = neighbour->module();
    int neighbourAddress = neighbourMod->par("address");
    bool linkIncluded = false;

    // avoids to put twice the same link in the links db
    for (unsigned int k=0 ; k<links.size() ; k++) {
    if (links[k].srcAddress() == neighbourAddress &&
        links[k].destAddress() == nodeAddress) {
      linkIncluded = true;
    }
    }

   if (linkIncluded == false) {
    // sets by default the metric to 0
    LinkInfo link(nodeAddress, neighbourAddress);
    links.push_back(link);
    numberLinks++;
    }
  }
}

// init the graph for sp calculation
// cout << "Init the graph for sp calculation " << endl;
initGraph();

//  cout << "Calculate shortest path - fills up the database links " << endl;
calcShortestPath();

// Endless loop
for(;;) {
  // cout << endl << "Enters loop of node " << myAddress << endl;
  cMessage *msg = receive();

  if (msg->hasPar("messageType")) {
    long mes = msg->par("messageType");
    switch (mes) {
    case ROUTING:
    // ROUTING packet
    cout << endl << "ROUTING packet received in node " << myAddress << endl;
    receiveUpdate(msg);
    break;

    case REQUEST:
    // REQUEST packet
    cout << endl << "REQUEST packet received in node " << myAddress << endl;
    receiveRequest(msg);
    break;

    case RESERVE:
    // RESERVE packet
    cout << endl << "RESERVE packet received in node " << myAddress << endl;
    receiveReserve(msg);
    break;

    case RESPONSE:
    // RESPONSE packet
    cout << endl << "RESPONSE packet received in node " << myAddress << endl;
    receiveResponse(msg);
    break;

    case TAKEDOWN:
    // Takedown message
    cout << endl << "TAKEDOWN message received in node " << myAddress << endl;
    receiveTakedown(msg);
    break;

    default:
    cout << "Reception of an unknown type of message !!!";
    }
  }
```

```
    else
      cout << "Reception of an unknown type of message !!!";
  }
}


// ******************  FINISH  ******************************
void GmplsRouter::finish() {
  cout << "*** Module: " << fullPath() << "***" << endl;
  cout << "Stack allocated:      " << stackSize() << " bytes";
  cout << " (includes " << ev.extraStackForEnvir() << " bytes for environment)" << endl;
  cout << "Stack actually used: " << stackUsage() << " bytes" << endl;

}


// Prints of the different containers
void GmplsRouter::printLinks() {
  cout << "Contents of the links database for node " << myAddress << endl;
  for (unsigned int i=0 ; i<links.size() ; i++) {
    links[i].print();
    cout << endl;
  }
  cout << endl;
}

void GmplsRouter::printPhotonicDb() {
  cout << "Contents of the Photonic Db for node " << myAddress << endl;
  for (unsigned int i=0 ; i<photonicDb.size() ; i++) {
    photonicDb[i].print();
    cout << endl;
  }
  cout << endl;
}

void GmplsRouter::printTemp() {
  cout << "Contents of the temp vector" << endl;
  for (unsigned int i=0 ; i<temp.size() ; i++) {
    temp[i].print();
    cout << endl;
  }
  cout << endl;
}

void GmplsRouter::printNodes() {
  cout << "The nodes db" << endl;
  for (unsigned int i=0 ; i<nodes.size() ; i++) {
    nodes[i].print();
  }
  cout << endl;
}

void GmplsRouter::printMap() {
  cout << "The map are" << endl;
  mid::const_iterator iter;
  for (iter = pairs.begin() ; iter != pairs.end() ; iter++) {
    int dest = iter->first;
    int lambda = iter->second;
    cout << "Lambda: " << lambda << " - Dest: " << dest << endl;
  }
}

void GmplsRouter::initGraph() {
  // disables all the nodes that must not be considered for the sp calculation
  /*  for (int i=0 ; i<graphe.nodes() ; i++) {
    if (nodes[i].toBeExtracted() == false && nodes[i].address != myAddress)
      graphe.node(i)->disable();
      }*/

  //cout << "Enters initGraph\n";

  // Sets the metrics of the graph
  for (int i=0 ; i<graphe.nodes() ; i++) {
    sTopoNode *node = graphe.node(i);
    cModule *nodeMod = node->module();
    int nodeAddress = nodeMod->par("address");

    //cout << "In node " << nodeAddress << endl;
    for (int j=0 ; j<node->outLinks() ; j++) {
      sTopoLinkOut *link = node->out(j);
      sTopoNode *neighbour = node->out(j)->remoteNode();
```

```
      cModule *neighbourMod = neighbour->module();
      int neighbourAddress = neighbourMod->par("address");

      //cout << "the size of links is " << links.size();
      //printLinks();
      for (unsigned int k=0 ; k<links.size() ; k++) {
      if ((links[k].srcAddress() == nodeAddress &&
           links[k].destAddress() == neighbourAddress) ||
          (links[k].srcAddress() == neighbourAddress &&
           links[k].destAddress() == nodeAddress)) {

        link->setWeight(links[k].metric());
        //cout << "in the graph, we set link from " << nodeAddress
        //   << "to " << neighbourAddress
        //   << "at the value " << links[k].metric() << endl;
      }
      }
    }
  }
  //cout << "Out of initgraph\n";
}


void GmplsRouter::sendUpdate(cModule *destMod, int neighbourAddress, int lambda) {
  cout << "Enters sendUpdate in node " << myAddress << endl;
  cMessage *mes = new cMessage;

  // message type
  mes->addPar("messageType")= ROUTING;

  // Packet header
  mes->addPar("packetType") = PT_UPD;
  mes->addPar("source") = myAddress;

  // Update packet header
  mes->addPar("numberLsa") = 1;

  // LSA header
  mes->addPar("lsType") = LST_AREA_OPQ;

  // top level TLV Type
  mes->addPar("topLevelTlvType") = TLV_T_LINK;

  // sub TLV link
  mes->addPar("subTlvLinkType") = SUB_TLV_T_LINKTYPE;
  mes->addPar("subTlvLinkTypeValue") = 1;

  // sub TLV link ID
  mes->addPar("subTlvLinkIdType") = SUB_TLV_T_LINKID;
  mes->addPar("subTlvLinkIdTypeValue") = (long) neighbourAddress;

  // sub TLV switching capability
  mes->addPar("subTlvIscdType") = SUB_TLV_T_ISCD;
  mes->addPar("swCapType") = SW_CAP_T_LSC;
  mes->addPar("lambda") = (long) lambda;

  //double delay = uniform(0.001,0.005);
  sendDirect(mes, 0, destMod, "in", 0);
  int dest = destMod->par("address");
  cout << "Send update to " << dest << " - The link (" << myAddress << ","
       << neighbourAddress << ") has the lambda " << lambda << " unavailable\n";
}

void GmplsRouter::sendReserve(int lambda, cModule *destMod, int beforeNodeAddress) {
  cout << "Enters sendReserve in node " << myAddress << endl;
  cMessage *res = new cMessage;
  res->addPar("messageType") = RESERVE;
  res->addPar("lambda") = lambda;
  res->addPar("srcAddress") = myAddress;
  res->addPar("beforeNodeAddress") = beforeNodeAddress;

  int destAddress = destMod->par("address");

  //double delay = uniform(0.001,0.005);
  sendDirect(res, 0, destMod, "in", 0);
  cout << "Reserve message sent to " << destAddress << endl
       << "that has beforeNode: " << beforeNodeAddress
       << " in the explicit route" << endl;

}
```

```
void GmplsRouter::sendReserveResponse(bool accepted, int src) {
  cout << "Enters sendReserveResponse in node " << myAddress << endl;
  cMessage *resp = new cMessage;
  resp->addPar("messageType") = RESPONSE;
  cPar& lambdaAvailable = resp->addPar("lambdaAvailable");
  lambdaAvailable.setBoolValue(accepted);

  for (int i=0 ; i<graphe.nodes() ; i++) {
    sTopoNode *node = graphe.node(i);
    cModule *mod = node->module();
    int address = mod->par("address");
    if (address == src) {
      //double delay = uniform(0.001,0.005);
      sendDirect(resp, 0, mod, "in", 0);
      cout << "Reserve response sent to " << address << endl
         << "The lambda requested "
         << (accepted == true ? " is " : " is NOT ") << "available" << endl;
    }
  }
}


void GmplsRouter::sendResponseToSink (bool accepted) {
  cout << "Enters sendResponseToSink in node " << myAddress << endl;
  cMessage *mes = new cMessage();
  mes->addPar("accepted") = (bool) accepted;
  mes->addPar("linkUtil") = (double) linkUtil();
  mes->addPar("nodeTraffic") = (double) nodeTraffic();
  cPar rnd("rnd");
  send(mes, "to_sink", 0);
}

void GmplsRouter::sendTakedown(cModule *destMod, int lambda, int neighbour) {
  cout << "Enters sendTakedown in node " << myAddress << endl;
  cMessage *mes = new cMessage;
  mes->addPar("messageType") = TAKEDOWN;
  mes->addPar("lambda") = lambda;
  mes->addPar("srcAddress") = myAddress;
  mes->addPar("neighbourAddress") = neighbour;

  cout << "Lambda to be taken down is " << lambda << endl;

  //double delay = uniform(0.001,0.005);
  sendDirect(mes, 0, destMod, "in", 0);
}

void GmplsRouter::receiveUpdate (cMessage *mes) {
  cout << "Enters receiveUpdate in node " << myAddress << endl;
  int src = mes->par("source");
  int neighbour = mes->par("subTlvLinkIdTypeValue");
  int lambda = mes->par("lambda");

  cout << "Updating the links database for link (" << src << ","
       << neighbour << ") with lambda " << lambda << " non available" << endl;

  if (mes->par("packetType").longValue() == PT_UPD &&
      mes->par("numberLsa").longValue() == 1 &&
      mes->par("lsType").longValue() == LST_AREA_OPQ &&
      mes->par("topLevelTlvType").longValue() == TLV_T_LINK &&
      mes->par("subTlvLinkType").longValue() == SUB_TLV_T_LINKTYPE &&
      mes->par("subTlvLinkTypeValue").longValue() == 1 &&
      mes->par("subTlvLinkIdType").longValue() == SUB_TLV_T_LINKID &&
      mes->par("subTlvIscdType").longValue() == SUB_TLV_T_ISCD &&
      mes->par("swCapType").longValue() == SW_CAP_T_LSC) {

    // Update the links db
    for (unsigned int i=0 ; i<links.size() ; i++) {
      if ((links[i].srcAddress() == src &&
        links[i].destAddress() == neighbour) ||
        (links[i].srcAddress() == neighbour &&
         links[i].destAddress() == src)) {
      links[i].lC().setLambdaNonAvailable(lambda);
      links[i].setMetric(links[i].lC().linkWeight(metricType));
      }
    }

    //cout << endl << "Links database after reception of an update" << endl;
    // printLinks();

    // run the shortest path algo on the new architecture
    // cout << "Updating the photonic db" << endl;
```

```
    calcShortestPath();

    //cout << endl << "Photonic database after reception of an update" << endl;
    // printPhotonicDb();
  }

  delete mes;
}

void GmplsRouter::receiveResponse(cMessage *mes) {
  cout << "Enters receiveResponse in node " << myAddress << endl;
  bool mesRes = mes->par("lambdaAvailable");
  delete mes;

  if (mesRes == true) {
    respReceived++;
    if (respReceived == respToBeReceived) {
      cout << "Send message to sink - this request can be fulfilled\n";
      numberLightpathEstablished++;
      sendResponseToSink(true);
    }
  }

  else {
    // This lambda is not available => response is blocked
    sendResponseToSink(false);
    cout << "Send message to sink - this request canNOT be fulfilled\n";

    // we need to unreserve the lambdas on the explicit rte
    mid::const_iterator iter;
    for (iter = pairs.begin() ; iter != pairs.end() ; iter++) {
      int dest = iter->first;
      int lambda = iter->second;
      //cout << "We added dest " << dest << " and lambda " << "to map\n";

      takedownRoute(dest, lambda);
    }
  }
}

void GmplsRouter::receiveRequest(cMessage *mes) {
  cout << endl << "Enters receiveRequest in node " << myAddress << endl;
  int dest = mes->par("destAddr").longValue();
  cout << "A request has been received - Destination node: "
       << dest << endl;

  for (unsigned int i=0 ; i<photonicDb.size() ; i++) {
    if (photonicDb[i].node() == dest) {
      //  ExplicitRoute &er = photonicDb[i].explicitRoute();
      cout << "The explicit route to reach " << dest << " is ";
      photonicDb[i].explicitRoute().print();
      cout << endl;

      if (photonicDb[i].explicitRoute().numberNodes() >= 2 &&
        photonicDb[i].e2eAvailableLambdas().numberUnusedLambdas() != 0) {
      respToBeReceived = photonicDb[i].explicitRoute().numberNodes()-1;
      respReceived = 0;

      int lambda;
      for (int j=0 ; j<photonicDb[i].explicitRoute().numberNodes()-1 ; j++) {
        int nodeAddress = photonicDb[i].explicitRoute(j);
        int beforeNode = photonicDb[i].explicitRoute(j+1);

        // Wavelength assignment (only done once)
        if (j == 0) {
          //cout << "nodeinfo\n";
          //photonicDb[i].print();
          lambda = photonicDb[i].e2eAvailableLambdas().wa(waType);
          cout << "The lambda chosen is " << lambda << endl;

          // Creates a map of the lambda to be set and the dest of the exp rte
          pairs.insert(mid::value_type(dest, lambda));
        }

        for (int k=0 ; k<graphe.nodes() ; k++) {
          sTopoNode *node = graphe.node(k);
          cModule *mod = node->module();
          int ad = mod->par("address");
          if (ad == nodeAddress) {
            sendReserve(lambda, mod, beforeNode);
          }
```

```
        }
      }
      }
      // The request is blocked
      else {
      sendResponseToSink(false);
      }
    }
  }
  delete mes;
}

void GmplsRouter::receiveReserve(cMessage *mes) {
  cout << "Enters receiveReserve in node " << myAddress << endl;
  int lambda = mes->par("lambda");
  int srcAddress = mes->par("srcAddress");
  int beforeNodeAddress = mes->par("beforeNodeAddress");
  delete mes;

  cout << "receiveReserve is asked to reserve " << lambda << endl;
  // Check the resources
  // Reserve the resources and relay the RESERVE message
  for (unsigned int i=0 ; i<links.size() ; i++) {
    if ((links[i].srcAddress() == beforeNodeAddress &&
      links[i].destAddress() == myAddress) ||
      (links[i].srcAddress() == myAddress &&
      links[i].destAddress() == beforeNodeAddress)) {
      LambdaCap &l = links[i].lC();

      if (l.lambdaAvailable(lambda)) {
      // reserve the lambda on that link
      l.setLambdaNonAvailable(lambda);
      // recalculate the metric on that link
      links[i].setMetric(l.linkWeight(metricType));

      cout << endl << "Reserve the resource in our own links database" << endl;
      printLinks();

      // flood an update of our link change
      for (int i=0 ; i<graphe.nodes() ; i++) {
        sTopoNode *node = graphe.node(i);
        cModule *mod = node->module();
        if (mod->id() != this->id())
          sendUpdate(mod, beforeNodeAddress, lambda);
      }

      // Then send a response to the src
      sendReserveResponse(true, srcAddress);
      }

      // this lambda is not available
      else {
      sendReserveResponse(false, srcAddress);
      }
    }
  }
}

void GmplsRouter::receiveTakedown(cMessage *mes) {
  int lambda = mes->par("lambda");
  int neighbour = mes->par("neighbourAddress");
  delete mes;

  for (unsigned int i=0 ; i<links.size() ; i++) {
    if ((links[i].srcAddress() == neighbour &&
       links[i].destAddress() == myAddress) ||
      (links[i].srcAddress() == myAddress &&
       links[i].destAddress() == neighbour)) {
      LambdaCap &l = links[i].lC();

      l.setLambdaAvailable(lambda);
      // recalculate the metric on that link
      links[i].setMetric(l.linkWeight(metricType));
    }
  }
}

// Sends a lambda tear down message to all the routers of an explicit route
// to a destination dest node
void GmplsRouter::takedownRoute(int dest, int lambda) {
  for (unsigned int i=0 ; i<photonicDb.size() ; i++) {
```

```
    if (photonicDb[i].node() == dest) {
      for (int j=0 ; j<photonicDb[i].explicitRoute().numberNodes()-1 ; j++) {
      int nodeAddress = photonicDb[i].explicitRoute(j);
      int beforeNode = photonicDb[i].explicitRoute(j+1);

      for (int k=0 ; k<graphe.nodes() ; k++) {
        sTopoNode *node = graphe.node(k);
        cModule *mod = node->module();
        int ad = mod->par("address");
        if (ad == nodeAddress) {
          sendTakedown(mod, lambda, beforeNode);
        }
      }
      }
    }
  }
}

void GmplsRouter::initTemp() {
  for (int i=0 ; i<numberNodes ; i++) {
    if (nodes[i].address() != myAddress) {
      NodeInfo node(nodes[i].address(), INF, UNDEF);
      temp.push_back(node);
    }
    else {
      NodeInfo node(myAddress, 0, UNDEF);
      temp.push_back(node);
    }
  }
}

void GmplsRouter::calcExplicitRoute() {
  for (unsigned int i=0 ; i<photonicDb.size(); i++) {
    if (photonicDb[i].node() != myAddress) {
      bool srcReached = false;
      int location = 0;
      int currentNode = photonicDb[i].node();
      int beforeNode = photonicDb[i].beforeNode();
      double cost = photonicDb[i].cost();

      if (cost != INF && beforeNode != UNDEF) {

      photonicDb[i].explicitRoute().addNode(currentNode);
      location++;

      //cout << "Calculating explicit route from " << myAddress
      //     << "To " << currentNode << endl;

      while (!srcReached) {
        // determinates the available lambdas on the explicit route
        for (unsigned int j=0 ; j<links.size() ; j++) {
          if ((links[j].srcAddress() == currentNode &&
             links[j].destAddress() == beforeNode)  ||
            (links[j].srcAddress() == beforeNode &&
             links[j].destAddress() == currentNode)) {
            LambdaCap &l = links[j].lC();
            //cout << "lambdas capability to be added taken from links with nodes  "
            //   << currentNode << " and " << beforeNode << endl;;
            // l.print();
            photonicDb[i].e2eAvailableLambdas() += l;
            //cout << "\nE2E availalble lambdas after sum of two lambda:  " << endl;
            // photonicDb[i].e2eAvailableLambdas().print();
            //cout << endl;
          }
        }

        // determinates the explicit route
        //cout << "Det the explicit route\n";
        if (beforeNode == myAddress) {
          srcReached = true;
          photonicDb[i].explicitRoute().addNode(myAddress);
        }
        else {
          photonicDb[i].explicitRoute().addNode(beforeNode);
          location++;
          for (unsigned int j=0 ; j<photonicDb.size() ; j++) {
            if (photonicDb[j].node() == beforeNode) {
            currentNode = photonicDb[j].node();
            beforeNode = photonicDb[j].beforeNode();
            //cout << "next link is "
            // << currentNode << " " << beforeNode << endl;
```

```
            }
          }
        }
      }
      }
    }
  }
  //cout << "Exits calcExplicitRoute";
}

void GmplsRouter::calcShortestPath() {
  //cout << "Enters calcShortestPath in node: " << myAddress << endl;
  if (graphe.nodes() == 0)
    cout << "the graphe is empty\n";
  else {
    // Erase the contents of the temp structure
    temp.clear();

    // init the temp structure used in the dijkstra algo
    initTemp();

    // init the graph with the correct weights determined in the links db
    initGraph();

    // Erase the photonicDb
    photonicDb.clear();

    //cout << "Enters algorithm\n";
    while(!temp.empty()) {
      // Sort temp based on the cost
      sort(temp.begin(),temp.end());

      // Put the lowest NodeInfo cost into the photonic db
      NodeInfo node(temp[0]);
      temp.erase(temp.begin());
      photonicDb.push_back(node);

      for (int k=0 ; k<graphe.nodes() ; k++) {
      sTopoNode *node = graphe.node(k);
      cModule *nodeMod = node->module();
      int nodeAddress = nodeMod->par("address");
      int sizeDb = photonicDb.size();

      if (photonicDb[sizeDb-1].node() == nodeAddress) {
        for (int i=0 ; i<node->outLinks() ; i++) {
          sTopoLinkOut *link = node->out(i);
          sTopoNode *neighbour = link->remoteNode();
          cModule *mod = neighbour->module();
          int neighbourAdd = mod->par("address");

          for (unsigned int j=0 ; j<temp.size() ; j++) {
            if (neighbourAdd == temp[j].node() &&
              (photonicDb[sizeDb-1].cost() + link->weight()) <
              temp[j].cost()) {
            temp[j].setCost(photonicDb[sizeDb-1].cost() + link->weight());
            temp[j].setBeforeNode(photonicDb[sizeDb-1].node());
            }
          }
        }
      }
      }
    }


  }

  //cout << "Before entering calcExplicitRoute\n";
  //printPhotonicDb();

  // set the explicit route and the wavelength availibilities
  //cout << "Enters calcExplicitRoute\n";
  calcExplicitRoute();
}

double GmplsRouter::linkUtil() {
  double temp = 0;
  for (unsigned int j=0 ; j<links.size() ; j++) {
    temp += links[j].lC().linkLoad();
  }
  return temp/numberLinks;
}
```

```
double GmplsRouter::nodeTraffic() {
  double temp = 0;
  for (unsigned int k=0 ; k<links.size() ; k++) {
    if (links[k].srcAddress() == myAddress ||
      links[k].destAddress() == myAddress) {
      temp += (totalLambdas - links[k].lC().numberUnusedLambdas());
    }
  }
  return temp/(totalLambdas * degree);
}
```

10.1.5. GMPLSROUTER.H

```
#include <omnetpp.h>
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <cassert>
#include "includes.h"

#define totalNumberLambdas 24
#define maxNodes 20

// lambda capabilities
class LambdaCap {
 private:
  int lCap[totalNumberLambdas];
 public:
  // default constructor : sets all the bits to one
  LambdaCap() {
    for (int i=0 ; i<totalNumberLambdas ; i++)
      lCap[i] = 1;
  }
  // copy constructor
  LambdaCap(const LambdaCap &l) {
    for (int i=0 ; i<totalNumberLambdas ; i++) {
      lCap[i] = l.lCap[i];
    }
  }
  // overloaded equality operator
  const LambdaCap &operator=(const LambdaCap &right) {
    if (&right != this) {
      for (int i=0 ; i<totalNumberLambdas ; i++) {
      lCap[i] = right.lCap[i];
      }
    }
    return *this;
  }
  // overloaded compare equal
  bool operator==(LambdaCap &lc) {
    for (int i=0 ; i<totalNumberLambdas ; i++) {
      if (lCap[i] != lc.lCap[i])
      return false;
    }
    return true;
  }
  // overloaded addition operator
  LambdaCap &operator+=(LambdaCap &right) {
    for (int i=0 ; i<totalNumberLambdas ; i++) {
      if (right.lCap[i] == 0 || lCap[i] == 0)
      lCap[i] = 0;
    }
    return *this;
  }
  // overloaded subscript operator
  int &operator[](int i) {
    return lCap[i];
  }
  int numberUnusedLambdas() {
    int count = 0;
    for (int i=0 ; i<totalNumberLambdas ; i++) {
      if (lCap[i] == 1) {
      count++;
      }
    }
    return count;
```

```
}
double ratio() {
  double totalLambdas = totalNumberLambdas;
  double unusedLambdas = numberUnusedLambdas();
  return unusedLambdas/totalLambdas;

}
double linkLoad() {
    return (1.0000 - ratio());
}
bool lambdaAvailable(int number) {
  if (lCap[number] == 1)
    return true;
  return false;
}
void setLambdaNonAvailable(int number) {
  lCap[number] = 0;
}
void setLambdaAvailable(int number) {
  lCap[number] = 1;
}
void print() {
  cout << "'";
  for (int i=0 ; i<totalNumberLambdas ; i++)
    cout << lCap[i] << " ";
  cout << "'";
}

// wavelength assignment - different schemes can be implemented here
// First fit wavelength assignment
int wa(const int waType) {
  switch (waType) {
  case waR:
    cout << "Random wa is chosen\n";
    return random();
    break;
  case waFF:
    cout << "First Fit wa is chosen\n";
    return firstFit();
    break;
  default:
    cout << "Unknown type of metric !!!\n";
  }
  return -1;
}

int firstFit() {
  for (int i=0 ; i<totalNumberLambdas ; i++) {
    if (lCap[i] == 1)
    return i;
  }
  cout << "There is no available wavelength !!!" << endl;
  return UNDEF;
}

// Random wavelength assignment
int random() {
  int unusedLambdas = numberUnusedLambdas();
  if (unusedLambdas != 0) {
    std::vector<int> v;
    for (int i=0 ; i<totalNumberLambdas ; i++) {
    if (lCap[i] == 1) {
      v.push_back(i);
    }
    }
    return v[0];
  }
  else {
    cout << "There is no available wavelength !!!" << endl;
    return UNDEF;
  }
}

// Adaptive weight functions - different schemes can be implemented here
  // Very simple (default) based on available lambdas and total number of lambdas
// hops based (no lambda information needed)
double linkWeight(const int metricType) {
  switch (metricType) {
  case hops:
    cout << "Metric chosen : hop count\n";
    return metricHops();
```

```
      break;
    case TAW1:
      cout << "Metric chosen : simple TAW\n";
      return metricTaw1();
      break;
    case TAW2:
      cout << "Metric chosen : enhanced TAW\n";
      return metricTaw2();
      break;
    default:
      cout << "Unknown type of metric !!!\n";
    }
    return -1;
  }

  double metricHops() {
    return 1.0;
  }

  double metricTaw1() {
    if (this->numberUnusedLambdas() != 0) {
      double totalLambdas = totalNumberLambdas;
      return (1 - this->numberUnusedLambdas()/totalLambdas);
    }
    else
      return INF;
  }

  double metricTaw2() {
    double weight = INF;
    double nuw = this->numberUnusedLambdas();
    cout << "nuw = " << nuw << endl;
    if (nuw != 0) {
      weight = 0.0001 - log(1 - pow((1 - nuw/totalNumberLambdas), nuw));
      return weight;
    }
    else
      return INF;
  }

};

class LinkInfo {
 private:
  int src;
  int dest;
  LambdaCap lCap;
  double met;
 public:
  // default constructor
  LinkInfo() : lCap() {
    src = UNDEF;
    dest = UNDEF;
    met = INF;
  }
  // constructor
  LinkInfo(int a, int b) : lCap() {
    src = a;
    dest = b;
    met = 0.0001;
  }
  // copy constructor
  LinkInfo(const LinkInfo &right) {
    src = right.src;
    dest = right.dest;
    lCap = right.lCap;
    met = right.met;
  }
  // copy assignment operator
  LinkInfo &operator=(LinkInfo &right) {
    if (&right != this) {
      src = right.srcAddress();
      dest = right.destAddress();
      lCap = right.lC();
      met = right.metric();
    }
    return *this;
  }
  int srcAddress() { return src; }
  int destAddress() { return dest; }
  LambdaCap &lC() { return lCap; }
```

```
  double metric() { return met; }
  void setMetric(double value) { met = value; }
  void setSrcAddress(int value) { src = value; }
  void setDestAddress(int value) { dest = value; }
  void print() {
    cout << "     " << "Link source address = " << src << endl
       << "     " << "Link dest address   = " << dest << endl
       << "     " << "Lambda capability   = ";
    lCap.print();
    cout << endl
       <<"     " << "Metric              = ";
    if (met != INF)
      cout << met << endl;
    else
      cout << "Infinity " << endl;
  }
};

class Node {
 private:
  int add;
  bool toExt;
 public:
  Node(int a, bool b) {
    add = a;
    toExt = b;
  }
  bool toBeExtracted() { return toExt; }
  int address() { return add; }
  void print() {
    cout << "     " << "Node Address: "
       << add << (toExt == false ? " is NOT" : " is")
       << " to be extracted" << endl;
  }
};

class ExplicitRoute {
 private:
  int expRte[maxNodes];
  int nodes;
 public:
  // defaut constructor
  ExplicitRoute() {
    nodes = 0;
    for (int i=0 ; i<maxNodes ; i++) {
      expRte[i] = UNDEF;
    }
  }
  // copy constructor
  ExplicitRoute(ExplicitRoute &init) {
    nodes = init.numberNodes();
    for (int i=0 ; i<init.numberNodes() ; i++) {
      expRte[i] = init.expRte[i];
    }
  }
  // copy assignment operator
  const ExplicitRoute &operator=(const ExplicitRoute &right) {
    nodes = right.nodes;
    for (int i=0 ; i<right.nodes ; i++) {
      expRte[i] = right.expRte[i];
    }
    return *this;
  }
  // compare equal operator
  bool operator==(ExplicitRoute &er) {
    for (int i=0 ; i<er.numberNodes() ; i++) {
      if (expRte[i] != er.expRte[i])
      return false;
    }
    return true;
  }
  // overloaded subscript operator
  int &operator[](int index) {
    return expRte[index];
  }
  ~ExplicitRoute() {}
  // add a node at the end of the explicit route
  void addNode(int value) {
    expRte[nodes] = value;
    nodes++;
  }
```

```
  int numberNodes() { return nodes; }
  // Print function
  void print() {
    cout << "'";
    for (int i=0 ; i<nodes ; i++)
      cout << expRte[i] << " ";
    cout << "'";
  }
};

class NodeInfo {
 private:
  int nod;      // a destination node in the graph
  double cos;     // the lowest cost to the destination
  int beforeNod;      // the last node before reaching the destination node
  LambdaCap lambdas;    // Available lambdas (explicit route)
  ExplicitRoute rte;      // Explicit route
 public:
  // default constructor
  NodeInfo() : lambdas(), rte() {
    nod = UNDEF;
    cos = INF;
    beforeNod = UNDEF;
  }
  NodeInfo(int n, double c, int bn) : lambdas(), rte() {
    nod = n;
    cos = c;
    beforeNod = bn;
  }
  // copy constructor
  NodeInfo(const NodeInfo &init) {
    nod = init.nod;
    cos = init.cos;
    beforeNod = init.beforeNod;
    lambdas = init.lambdas;
    rte = init.rte;
  }
  // copy assignment operator
  NodeInfo &operator=(NodeInfo &right) {
    if (&right != this) {
      nod = right.node();
      cos = right.cost();
      beforeNod = right.beforeNode();
      lambdas = right.e2eAvailableLambdas();
      rte = right.explicitRoute();
    }
    return *this;
  }
  // overloaded less than operator
  bool operator<(const NodeInfo &right) const {
    if (cos <= right.cos) {
      if (cos == right.cos) {
      if (nod < right.nod) {
        return true;
      }
      else {
        return false;
      }
      }
      else {
      return true;
      }
    }
    else {
      return false;
    }
  }
  // getters
  int node() { return nod; }
  double cost() { return cos; }
  int beforeNode() { return beforeNod; }
  LambdaCap &e2eAvailableLambdas() { return lambdas; }
  ExplicitRoute &explicitRoute() { return rte; }
  int explicitRoute(int i) {
    return rte[i];
  }
  // setters
  void setNode(int value) { nod = value; }
  void setCost(double value) { cos = value; }
  void setBeforeNode(int value) { beforeNod = value; }
```

```
  // print
  void print() {
    cout << "     " << "Node address dest   = ";
    if (nod == UNDEF)
      cout << "Not defined" << endl;
    else
      cout << nod << endl;
    cout << "     " << "Lowest cost to dest = ";
    if (cos == INF)
      cout << "Infinity" << endl;
    else
      cout << cos << endl;
    cout << "     " << "Node before dest    = ";
    if (beforeNod == UNDEF)
      cout << "Not defined" << endl;
    else
      cout << beforeNod << endl;
    cout << "     " << "Lambda capacity     = ";
    lambdas.print();
    cout << endl
       << "     " << "Explicit route      = ";
    rte.print();
    cout << endl;
  }
};
```

10.1.6. ABILENE.NED

```
// ABILENE NETWORK (US Universities backbone network)
//    - 12 nodes
//    - Degree between 1 and 4

channel wan_link

endchannel

//-------------------------------------------------------------
// generator --
// generates lightpaths requests to a random node
//-------------------------------------------------------------

simple Generator
    parameters:
        num_endnodes : numeric,
        totalNumberLambdas : numeric,
        address : numeric,
        degree : numeric,
        delta : numeric;
    gates:
        out: out;
endsimple

//-------------------------------------------------------------
// Sink --
// Creates statistics
//-------------------------------------------------------------
simple Sink
    parameters:
        num_endnodes : numeric,
        totalNumberLambdas : numeric,
        address : numeric,
        degree : numeric;
    gates:
        in: in;
endsimple

//-------------------------------------------------------------
// Router
//
//-------------------------------------------------------------

simple GmplsRouter
    parameters:
      totalNumberLambdas : numeric,
      degree: numeric,
      metricType : numeric,
      waType : numeric,
        address : numeric;
    gates:
```

```
        in: from_gen[];
        in: in[];
        out: to_sink[];
        out: out[];
endsimple


//-------------------------------------------------------------
//  endsystem = generator + sink
//-------------------------------------------------------------

module EndSystem
    gates:
        out: out;
        in: in;
    submodules:
        gen: Generator;
            parameters:
                num_endnodes = ref ancestor num_endnodes,
                totalNumberLambdas = ref ancestor totalNumberLambdas,
                address = input,
                degree = input,
                delta = ref ancestor delta;
            display: "b=40,24;p=139,35";
        sink: Sink;
            parameters:
                num_endnodes = ref ancestor num_endnodes,
                totalNumberLambdas = ref ancestor totalNumberLambdas,
                address = input,
                degree = input;
            display: "b=40,24;p=183,35";
    connections:
        sink.in <-- in;
        gen.out --> out;
endmodule

//-------------------------------------------------------------------
// Network =
// endsystems + routers
// ------------------------------------------------------------------

module Net
    parameters:
      totalNumberLambdas: const,
      delta : const,
      num_endnodes : const,
      metricType : const,
      waType : const;
    submodules:
        // End systems
        endsystem1: EndSystem;

            display: "b=32,32;p=45,115";
        endsystem2: EndSystem;

            display: "b=32,32;p=45,251";
        endsystem3: EndSystem;

            display: "b=32,32;p=65,39";
        endsystem4: EndSystem;

            display: "b=32,32;p=216,103";
        endsystem5: EndSystem;

            display: "b=32,32;p=301,131";
        endsystem6: EndSystem;

            display: "b=32,32;p=256,335";
        endsystem7: EndSystem;

            display: "b=32,32;p=384,55";
        endsystem8: EndSystem;

            display: "b=32,32;p=456,111";
        endsystem9: EndSystem;

            display: "b=32,32;p=432,343";
        endsystem10: EndSystem;

            display: "b=32,32;p=528,63";
        endsystem11: EndSystem;
```

```
    display: "b=32,32;p=632,215";
endsystem12: EndSystem;

    display: "b=32,32;p=616,87";
// routers
Sunnyvale: GmplsRouter; //
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[3],
        out[3],
        from_gen[1],
        to_sink[1];
    display: "p=81,179;b=32,32";
LosAngeles: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[2],
        out[2],
        from_gen[1],
        to_sink[1];
    display: "p=121,259;b=32,32";
Seattle: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[2],
        out[2],
        from_gen[1],
        to_sink[1];
    display: "p=121,47;b=32,32";
Denver: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[3],
        out[3],
        from_gen[1],
        to_sink[1];
    display: "p=201,171;b=32,32";
KansasCity: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[3],
        out[3],
        from_gen[1],
        to_sink[1];
    display: "p=294,193;b=34,34";
Houston: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[3],
        out[3],
```

```
        from_gen[1],
        to_sink[1];
    display: "p=317,314;b=32,32";
Chicago: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[1],
        out[1],
        from_gen[1],
        to_sink[1];
    display: "p=389,106;b=32,32";
Indianapolis: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[4],
        out[4],
        from_gen[1],
        to_sink[1];
    display: "p=453,186;b=32,32";
Atlanta: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[3],
        out[3],
        from_gen[1],
        to_sink[1];
    display: "p=485,298;b=32,32";
Cleveland: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[2],
        out[2],
        from_gen[1],
        to_sink[1];
    display: "p=533,122;b=32,32";
WashingtonDC: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[2],
        out[2],
        from_gen[1],
        to_sink[1];
    display: "p=573,234;b=32,32";
NewYork: GmplsRouter;
    parameters:
    totalNumberLambdas = totalNumberLambdas,
    degree = input,
    metricType = metricType,
    waType = waType,
    address = input;
    gatesizes:
        in[2],
        out[2],
        from_gen[1],
        to_sink[1];
    display: "p=605,146;b=32,32";
```

```
    connections:
        endsystem1.out --> Sunnyvale.from_gen[0];
        endsystem1.in <-- Sunnyvale.to_sink[0];
        endsystem2.out --> LosAngeles.from_gen[0];
        endsystem2.in <-- LosAngeles.to_sink[0];
        endsystem3.out --> Seattle.from_gen[0];
        endsystem3.in <-- Seattle.to_sink[0];
        endsystem4.out --> Denver.from_gen[0];
        endsystem4.in <-- Denver.to_sink[0];
        endsystem5.out --> KansasCity.from_gen[0];
        endsystem5.in <-- KansasCity.to_sink[0];
        endsystem6.out --> Houston.from_gen[0];
        endsystem6.in <-- Houston.to_sink[0];
        endsystem7.out --> Chicago.from_gen[0];
        endsystem7.in <-- Chicago.to_sink[0];
        endsystem8.out --> Indianapolis.from_gen[0];
        endsystem8.in <-- Indianapolis.to_sink[0];
        endsystem9.out --> Atlanta.from_gen[0];
        endsystem9.in <-- Atlanta.to_sink[0];
        endsystem10.out --> Cleveland.from_gen[0];
        endsystem10.in <-- Cleveland.to_sink[0];
        endsystem11.out --> WashingtonDC.from_gen[0];
        endsystem11.in <-- WashingtonDC.to_sink[0];
        endsystem12.out --> NewYork.from_gen[0];
        endsystem12.in <-- NewYork.to_sink[0] display "m=,100,0,100,0";

        Sunnyvale.in[0] <-- wan_link <-- LosAngeles.out[0];
        Sunnyvale.out[0] --> wan_link --> LosAngeles.in[0];
        Sunnyvale.in[1] <-- wan_link <-- Seattle.out[0];
        Sunnyvale.out[1] --> wan_link --> Seattle.in[0];
        Sunnyvale.in[2] <-- wan_link <-- Denver.out[0];
        Sunnyvale.out[2] --> wan_link --> Denver.in[0];

        LosAngeles.in[1] <-- wan_link <-- Houston.out[0];
        LosAngeles.out[1] --> wan_link --> Houston.in[0];

        Seattle.in[1] <-- wan_link <-- Denver.out[1];
        Seattle.out[1] --> wan_link --> Denver.in[1];

        Denver.in[2] <-- wan_link <-- KansasCity.out[0];
        Denver.out[2] --> wan_link --> KansasCity.in[0];

        KansasCity.in[1] <-- wan_link <-- Indianapolis.out[0];
        KansasCity.out[1] --> wan_link --> Indianapolis.in[0];
        KansasCity.in[2] <-- wan_link <-- Houston.out[1];
        KansasCity.out[2] --> wan_link --> Houston.in[1];

        Houston.in[2] <-- wan_link <-- Atlanta.out[0];
        Houston.out[2] --> wan_link --> Atlanta.in[0];

        Chicago.in[0] <-- wan_link <-- Indianapolis.out[1];
        Chicago.out[0] --> wan_link --> Indianapolis.in[1];

        Indianapolis.in[2] <-- wan_link <-- Cleveland.out[0];
        Indianapolis.out[2] --> wan_link --> Cleveland.in[0];
        Indianapolis.in[3] <-- wan_link <-- Atlanta.out[1];
        Indianapolis.out[3] --> wan_link --> Atlanta.in[1] display "m=,100,0";

        Atlanta.in[2] <-- wan_link <-- WashingtonDC.out[0];
        Atlanta.out[2] --> wan_link --> WashingtonDC.in[0];

        Cleveland.in[1] <-- wan_link <-- NewYork.out[1];
        Cleveland.out[1] --> wan_link --> NewYork.in[1];

        WashingtonDC.in[1] <-- wan_link <-- NewYork.out[0];
        WashingtonDC.out[1] --> wan_link --> NewYork.in[0];
    display: "p=10,18;b=713,475";
endmodule

//   *********   Network parameters    **********************
//   ....   Metric type TAW1        2     OK
//                      TAW2        3     OK
//
//   .... WA type       FirstFit          1     OK
//                      Random            2     OK
//   ****************************************************/

network net : Net
    parameters:
      totalNumberLambdas = 24,
      delta = 60s,
```

```
      num_endnodes = 12,
      metricType = 3,
      waType = 2;
endnetwork
```

10.1.7. OMNETPP.INI

```
[General]
network = net
include parameters.ini
ini-warnings = no
warnings= no

[Cmdenv]
module-messages = no
verbose-simulation = no
extra-stack = 72768

[Tkenv]
default-run=1
use-mainwindow = no
print-banners = yes

[Parameters]

net.endsystem1.gen.degree = 3
net.endsystem1.gen.address = 1
net.endsystem1.sink.degree = 3
net.endsystem1.sink.address = 1

net.endsystem2.gen.degree = 2
net.endsystem2.gen.address = 2
net.endsystem2.sink.degree = 2
net.endsystem2.sink.address = 2

net.endsystem3.gen.degree = 2
net.endsystem3.gen.address = 3
net.endsystem3.sink.degree = 2
net.endsystem3.sink.address = 3

net.endsystem4.gen.degree = 3
net.endsystem4.gen.address = 4
net.endsystem4.sink.degree = 3
net.endsystem4.sink.address = 4

net.endsystem5.gen.degree = 3
net.endsystem5.gen.address = 5
net.endsystem5.sink.degree = 3
net.endsystem5.sink.address = 5

net.endsystem6.gen.degree = 3
net.endsystem6.gen.address = 6
net.endsystem6.sink.degree = 3
net.endsystem6.sink.address = 6

net.endsystem7.gen.degree = 1
net.endsystem7.gen.address = 7
net.endsystem7.sink.degree = 1
net.endsystem7.sink.address = 7

net.endsystem8.gen.degree = 4
net.endsystem8.gen.address = 8
net.endsystem8.sink.degree = 4
net.endsystem8.sink.address = 8

net.endsystem9.gen.degree = 3
net.endsystem9.gen.address = 9
net.endsystem9.sink.degree = 3
net.endsystem9.sink.address = 9

net.endsystem10.gen.degree = 2
net.endsystem10.gen.address = 10
net.endsystem10.sink.degree = 2
net.endsystem10.sink.address = 10

net.endsystem11.gen.degree = 2
net.endsystem11.gen.address = 11
net.endsystem11.sink.degree = 2
```

```
net.endsystem11.sink.address = 11

net.endsystem12.gen.degree = 2
net.endsystem12.gen.address = 12
net.endsystem12.sink.degree = 2
net.endsystem12.sink.address = 12


net.Sunnyvale.address = 1;
net.Sunnyvale.degree = 3;

net.LosAngeles.address = 2;
net.LosAngeles.degree = 2;

net.Seattle.address = 3;
net.Seattle.degree = 2;

net.Denver.address = 4;
net.Denver.degree = 3;

net.KansasCity.address = 5;
net.KansasCity.degree = 3;

net.Houston.address = 6;
net.Houston.degree = 3;

net.Chicago.address = 7;
net.Chicago.degree = 1;

net.Indianapolis.address = 8;
net.Indianapolis.degree = 4;

net.Atlanta.address = 9;
net.Atlanta.degree = 3;

net.Cleveland.address = 10;
net.Cleveland.degree = 2;

net.WashingtonDC.address = 11;
net.WashingtonDC.degree = 2;

net.NewYork.address = 12;
net.NewYork.degree = 2;
```

10.2. RESULTS PROCESSING

The following C shell script is run in order to process raw data accumulated in `cOutVector .vec` files. This script must be run in the directory where all the output vectors are located. The script calls a.out, the executable of a short C++ program. The a.out executable must be also present in the same directory.

```
#!/bin/csh
foreach vector(*.vec)
      cat $vector >> data
end
awk 'NF == 4 {print $4, $3}' data > result
./a.out
```

10.2.1. PROCESSING C++ PROGRAM

```
#include <iostream>
using std::cout;
using std::cin;
using std::ios;
using std::cerr;
using std::endl;
#include <fstream>
using std::ifstream;
#include <iomanip>
using std::setiosflags;
using std::resetiosflags;
using std::setw;
using std::setprecision;
```

```
#include <cstdlib>
#include <vector>
#include <numeric>
#include <algorithm>

const double delta = 0.05;

int main(int argc, char *argv[]) {
  char nameInput[30];
  char nameOutput[30];
  int blocked;
  double util;
  double prob;
  double x=0;
  double average;
  int sum;
  int lines;

  cout << "Enter file name for data to be processed" << endl;
  cin >> nameInput;
  ifstream dataIn(nameInput, ios::in);

  if (!dataIn) {
    cerr << "File could not be opened\n";
    exit(1);
  }

  //cout << "Enter records interval" << endl;
  //cin >> delta;
  //delta = (double) argv[2];

  cout << "Enter file name for processed data" << endl;
  cin >> nameOutput;
  ofstream dataOut(nameOutput, ios::out);

  if (!dataOut) {
    cerr << "File could not be opened\n";
    exit(1);
  }

  while (x+delta <= 1.00) {
    dataIn.seekg(0);
    dataIn.clear();
    sum = 0;
    lines = 0;
    average = 0.00;
    while (dataIn >> util >> blocked) {
      if (util >= x && util < x+delta) {
      lines++;
      if (blocked == 1) {
        sum++;
      }
      }
    }
    if (lines != 0) {
      average = (double) sum/lines;
      //cout << "Made the aver of sum " << sum << " / lines " << lines << endl;
      dataOut << setprecision(2) << average << " " << x+delta/2 << "\n";
    }
    x += delta;
  }
  return 0;
}
```