2010

Reconfigurations of Logical Topologies for WDM Mesh Networks

Aktaruzzaman AKM
University of Windsor

Reconfigurations of Logical Topologies for WDM Mesh Networks.

By

AKM, Aktaruzzaman

A Thesis

Submitted to the Faculty of Graduate Studies and Research through the School of Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Science at the University of Windsor

Windsor, Ontario, Canada

2010

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract:

In static lightpath allocation, the logical topology of a WDM mesh network is determined, based on the long-term traffic demands. These traffic demands change with time. When a logical topology is incapable of supporting the current traffic demands, the logical topology has to be changed. The change is made by adding a minimum number of edges to the logical topology.

The objective of this research is to find an optimal new *Logical Topology* which can support the current traffic demands with as little change to the existing topology as possible. We have proposed a Hill-Climbing algorithm to solve the reconfiguration problem of logical topologies in WDM networks. Our problem can be divided into two sub-problems. The first is to find an optimal logical topology and the second is to route the traffic optimally on the logical topology.

Keywords: Optical Networks, Mesh Networks, WDM Networks, Optimization, Logical Topology, Reconfiguration, Heuristic.

Acknowledgement

It is my great pleasure to thank those who made this thesis possible. I am thankful to my supervisor, Dr. Subir Bandyopadhyay, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. These studies could not be achieved without his wise supervision and continuous encouragements. I would like to thanks to Mr. Quazi Rahman, PhD student, who helped in all aspects throughout my thesis work. Also I would like to thank our thesis committee members, Dr Dan Wu and Dr. Jagadish Pathak for their valuable comments and suggestions.

Finally I would like to my wife Lipika Yasmin, my two daughters (Nosheen and Noreen), my parents, my brothers-sisters, and my friends. Their love and care always were with me during these years. Without their support it would be impossible to make this thesis. I am deeply owed to all the people directly or indirectly has supported.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.

AKM, Aktaruzzaman

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Preamble

Wavelength division multiplexing (WDM) technology enables *optical networks* to properly utilize the huge *bandwidth* capacity of *optical fibers* for carrying traffic [2]. A major advantage of an optical network is that it is able to reconfigure its logical topology to adapt to changing traffic patterns dynamically [3]. Another key feature of second generation of optical networks is the use of tunable *transmitters* and/or *receivers*, which allows the logical connectivity to be optimized to adapt to changing traffic conditions. In this thesis we consider re-arrangeable *multihop lightwave* WDM networks, where each node is equipped with a pre-determined number of transmitters and receiver.

1.2 Problem Description

In static *lightpath* allocation, the *logical topology* of a *WDM optical* network is determined, based on long-term traffic demands. These traffic demands however change with time. When a *logical topology* is incapable of supporting the current traffic demands, the logical topology has to be modified or *reconfigured*. The objective is to find an optimal new logical topology which can support the current traffic demands with as little change to the existing topology as possible. This is known to be a difficult problem [14] and many researchers have studied this recently. In this research, we have addressed the *reconfiguration* problem of the logical topology of a WDM optical network.

Our main objective in this research is to observe how much the *congestion* can be reduced by reconfiguring the logical topology when the traffic demand increases beyond the capacity of the network. In order to compute the congestion value, the policy of routing the traffic over the logical topology must be determined. Therefore, the problem can be split into two sub-problems as follows:

i. Create a new logical topology by reconfiguring the existing logical topology.

ii. Route the traffic demand over the new logical topology in an optimal manner to determine the congestion.

We have used a hill climbing procedure [11] to create a new logical topology by reconfiguring the existing logical topology as little as possible, such that the new logical topology can handle the changing traffic patterns. In each iteration, we have tried to find the best logical topology with relatively little change to the existing logical topology, thus minimizing the disruption to the network and the reconfiguration time. We have used the CPLEX optimizer to route the traffic over the logical topology.

Many researchers have worked on this issue and we have summarized the work briefly in Chapter 2. The problem and the motivation have been discussed in Section 1.3, the fundamentals of WDM optical networks are discussed in Section 1.4, and the thesis outline is shown in Section 1.5.

1.3 The Problem Investigation and motivation

A lightpath in a logical topology carries certain amount of traffic (typically 10 Mbits/sec at the moment). When the traffic increases above this capacity, a new logical topology needs to be determined, requisite lightpaths set up and traffic grooming strategies determined to accommodate the increases in the traffic. The

second generation WDM network technology can dynamically change its logical topology corresponding to the changing traffic conditions [4]. The problems for the researcher are as follows:

- How frequently should reconfigurations be carried out?
- How to keep the network performance optimized?
- How much should be the cost due to reconfiguration in the transition period, when the network is switched from one logical topology to another one?
- How to handle faults in the logical topology at the time of reconfiguration?

Many researchers have worked on different aspects of optimization in optical networks, including optimizing network performances, improving the delay and the throughput metrics, minimizing the hardware costs, minimizing the disruptions, proposing heuristics to design logical topology and decreasing the *congestion*. The term congestion is discussed later in this section.

In this thesis, we have investigated the reconfiguration of the logical topologies so that the congestion of the networks can be minimized and we have tried to answer the following questions:

i. How to find a “good” reconfiguration fairly quickly for logical topology.

ii. How to minimize the disruptions of the network as little as possible.

iii. How to ensure that the new topology is capable of carrying the new traffic.

iv. How to route the traffic demand very effectively and find the congestion.

Many methods for designing the logical topology and routing the traffic optimally have been studied. Some of the studies are reviewed in Chapter 2. Mixed integer linear programming (MILP) has been used in previous studies [2], for designing a new logical topology. The MILP may take an exponential amount of time to obtain an exact solution, even for small networks. As an alternative to using a MILP, a heuristic algorithm may be used, although it is known that the quality of a heuristic solution is unknown. The tabu search algorithm has been used to design a logical topology [17] and properly designed tabu search [11] is known to be very effective on overcoming the well-known problem of getting trapped in local optimum.

It is known that the CPLEX optimizer provides the power to solve linear programs (LP) with many constraints and continuous variables, within a reasonable amount of time. We have used *LP* to route the traffic over the logical topology. The LP equations are used by the CPLEX optimizer tool as an input to find the congestion of the network. The CPLEX optimizer tool expedites our process of routing quickly and efficiently.

1.4 Fundamentals of WDM Optical Networks

An *optical network* is a network where computers or *end nodes* are connected using *optical fibers*. A typical optical network is shown in Fig. 1.4.1 with 4 end nodes E1-E4 and 4 routers R_1-R_4. An optical fiber is a thin glass cylinder or a filament which carries signals in the form of light (optical signals). It is the replacement of earlier computer networks, where copper wires were used as the communication medium. The technology of using multiple optical signals on the same fiber is called *wavelength division multiplexing* (WDM). Using this technique, it is possible to utilize the bandwidth of optical network (50 tera-bits) in an efficient way.

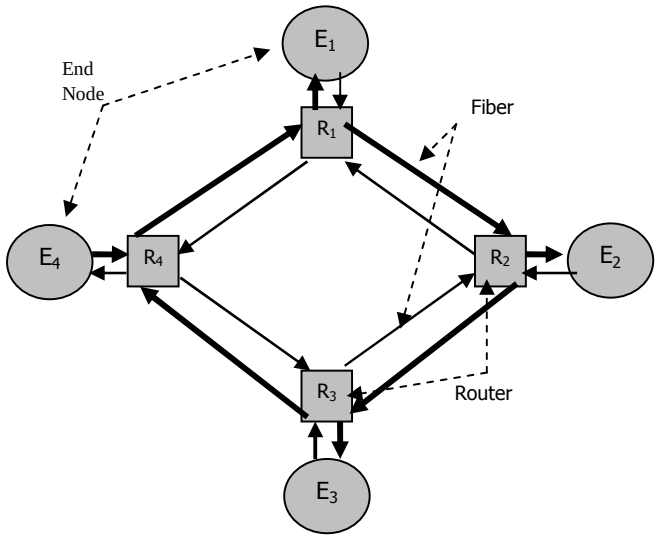

Figure 1.4.1: The physical topology of a typical WDM network with four end-nodes E1-E4 and four routers R1-R4.

In *optical networks*, the source or the destination of a data transmission is called an *end-node*. It could be a computer, a router or any other device that stores and sends data. An optical router is an important component of optical networks to route the incoming data to an appropriate outgoing destination. Each optical router has a number of incoming fibers and a number of outgoing fibers to carry one or more incoming or outgoing optical signals. It is convenient to describe the *Physical Topology* of an optical network as a graph *G*. Such a graph is shown in Figure 1.4.1 above, where an end-node or a router is a node of graph *G*, and the fiber from one node to another can be defined as an edge of graph *G*.

A *lightpath* is an optical connection from one end node to another, used to carry data in the form of encoded optical signals. It is also convenient to view the lightpaths as edges of a directed graph G_L shown in figure 1.4.2, where the nodes of G_L are the end nodes of the physical topology. Such a graph is called the *logical topology* of an optical network and the edges of such a graph are called *logical edges* [21].

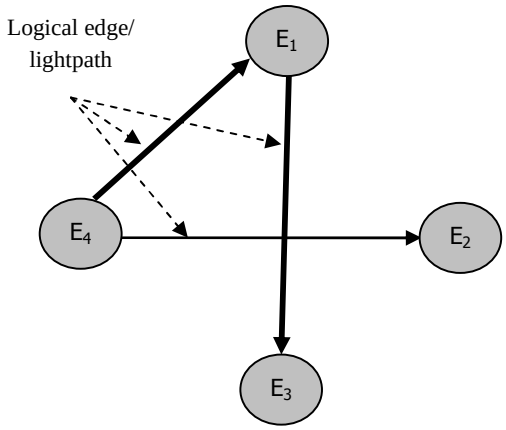

Figure 1.4.2: Logical Topology

The routing over a logical topology determines, for each source destination pair*(S, D)*, which logical paths are to be used to communicate data from *S* to *D* and how much data has to be carried out by each logical path from *S* to *D*.

In a network with *N* end nodes, an *N x N* traffic matrix may be used to define the traffic requirement for all source-destination pairs. A traffic matrix shows how much data is to be sent form one source end-node to another destination end-node. A typical 4 x 4 traffic matrix is shown in Table 1.4.1.

Node	1	2	3	4
1	0.00	0.30	0.5	0.30
2	0.20	0.00	0.30	0.20
3	**0.55**	0.10	0.00	0.30
4	0.00	0.20	0.10	0.00

Congestion

Table 1.4.1: A Typical [4 x 4] Traffic Matrix

Another very important research topic is *congestion* optimization in a logical topology. The maximum total traffic on a logical edge defines the *congestion* of that network for that traffic matrix. In the example shown in Table 1.4.1, the logical edge from end node 3 to end node 1 is carrying 0.55 unit of traffic, which is the current congestion for this particular network for this specific example. Here the unit of traffic is the capacity of a lightpath.

1.5 Thesis Outline

In Chapter 2, we have discussed the background study and have reviewed the literature. We have specified the problem in detail in Chapter 3. In Chapter 4 we have discussed the implementation details. In Chapter 5, we have described the experimental results and observation. Chapter 6, we have given our conclusion and have discussed future works.

Chapter 2: Background study/ Review of literature.

2.1 Preamble

During the early stages of computer networks, copper wire was used as the medium of communication. Due to the limitations of copper wire (such as high attenuation, susceptibility to malicious attacks, and electromagnetic interference), for the last twenty years, better ways of communication between computers have become one of the most important research topics in the computer revolution. The tremendous growth of the Internet and the ever increasing data transfer rates have made very high-bandwidth optical networks a very important technology in network infrastructure. This bandwidth property of optical fibers makes optical technology very attractive for *backbone* networks. Multiple optical signals can be transported on the same optical fiber [20] (explained in Section 2.2). Using this technique, the bandwidth of optical network is 50 tera-bits per second and is useful for handling the increasing demands for communication.

The nodes of an optical network are computers or any other devices (often called end nodes) which can generate or store data in an electronic form. Selected pairs of nodes are connected using *optical fibers*. Optical *fibers* are basically very thin glass cylinders or filaments which carry signals in the form of light (optical signals). Optical networks also include *transmitters to* generate optical signals for communication, and *receivers* to detect the optical signals and to convert the signals to electronic form. An optical *Router* is a device which routes signals from the incoming fibers to the appropriate outgoing fibers of the router. A *lightpath* is an optical connection from one end node to another, and is used to carry data in the form of encoded optical signals. A directed graph to represent the lightpaths connecting pairs of end nodes is called a *logical topology.* A path through a logical topology

from a source to a destination is known as a *logical path*. In the remaining sections in this chapter, we have discussed some of the optical hardware components and key terminology in detail.

2.2 Component and Key Terminology of WDM Networks

There are many devices used in second generation of WDM networks, such as couplers, optical transmitters, optical receivers and filters, optical amplifiers, optical routers, and switches are the most common. A few key devices are described below.

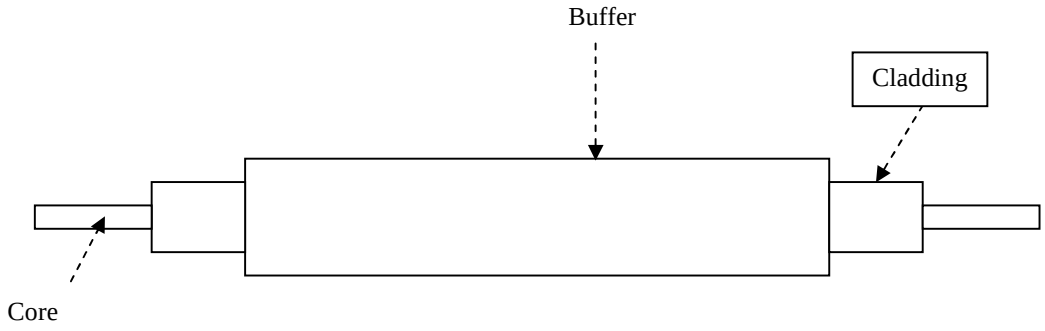

Figure 2.2.1.1: Optical Fiber

An optical fiber consists of a very fine cylinder of glass (core) of silica with refractive index, say μ_1, through which optical signals propagates. The core is surrounded by a concentric layer of glass (cladding) of silica with a lower refractive index, say μ_2, which is protected by a thin plastic jacket. The *buffer* shown in Fig. 2.2.1.1 surrounding the cladding encapsulates the fiber for mechanical isolation and for protection from physical damage [21]. A cross-section of a fiber is shown in Fig. 2.2.1.2.

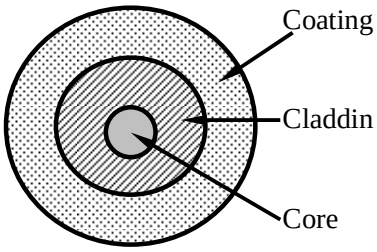

Figure 2.2.1.2: The cross-section of a fiber [21]

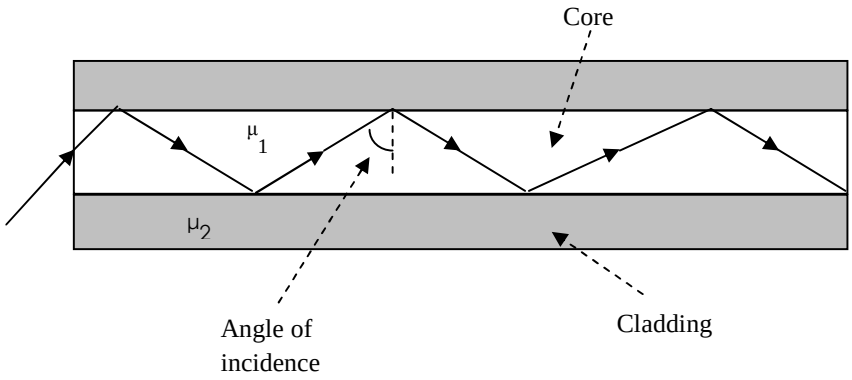

Figure 2.2.1.3: Optical Signal Propagation through a Fiber

If the angle of incidence (Fig. 2.2.1.3) is greater then the critical angle ($\sin^{-1} \mu_2/ \mu_1$) then total internal reflection takes place and all the light is reflected back into the medium. Total internal reflection forms the basis of optical transmission through fibers.

2.2.2 Wavelength Division Multiplexing (WDM)

The technology of using multiple optical signals using different carrier wavelengths on the same fiber is called Wavelength Division Multiplexing (WDM) [21].

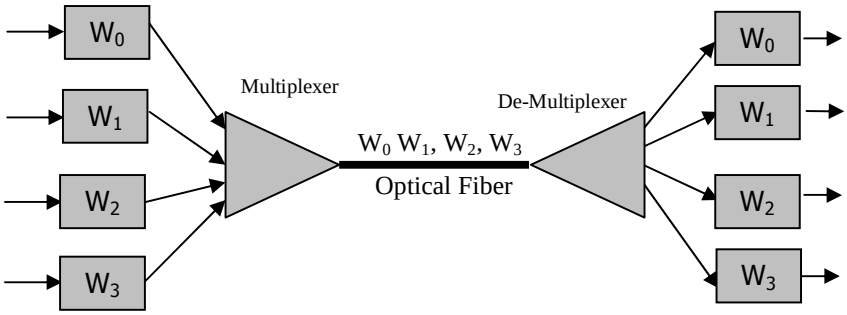

Figure 2.2.2a: WDM

Fig. 2.2.2a shows a *Multiplexer* that combines 4 distinct signals (W_0 – W_3) and sends them on an *Optical Fiber*. The De-multiplexer on the other side splits those 4 signals from the fiber and generates 4 outputs. In WDM networks, the available bandwidth of the fiber can be visualized as a set of channels. *Channel spacing* (Fig. 2.2.2b) is the separation of one channel from the next channel. Channel spacing is used to avoid interference between different optical signals and must exceed a certain minimum bandwidth.

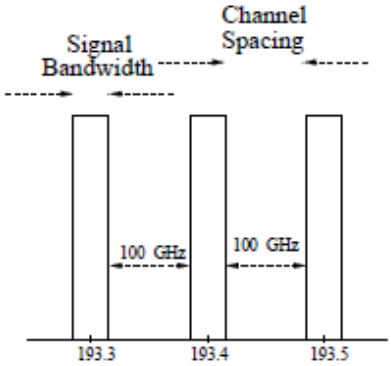

Figure 2.2.2b: Signal bandwidth and channel spacing [21]

The important advantages of WDM networks are as follows:

- *Low signal attenuation*- As a signal propagates through fibers; the signal strength goes down at a low rate (0.2 dB/km). This means that the number of optical amplifiers needed is relatively small.
- *Low signal distortion* - As a signal is sent along a fiber optic network, it degrades with respect to shape and phase. Signal regenerators are needed to restore the shape and timing. Low signal distortion means that signal regeneration is needed infrequently.
- *Low power requirement.*
- *Low material usage.*
- *Small space requirements.*
- *Low cost.*

WDM networks can be classified into two types - *wavelength-routed networks* and *Broadcast-and-select networks*. Since our study is based on wavelength-routed WDM networks, a brief description of wavelength-routed WDM networks is given

below:

2.2.2.1. Wavelength-Routed Networks

In a wavelength-routed network, a lightpath starts from an end-node, passes through 0 or more routers and terminates at another end-node. This type of networks may contain a large number of end nodes.

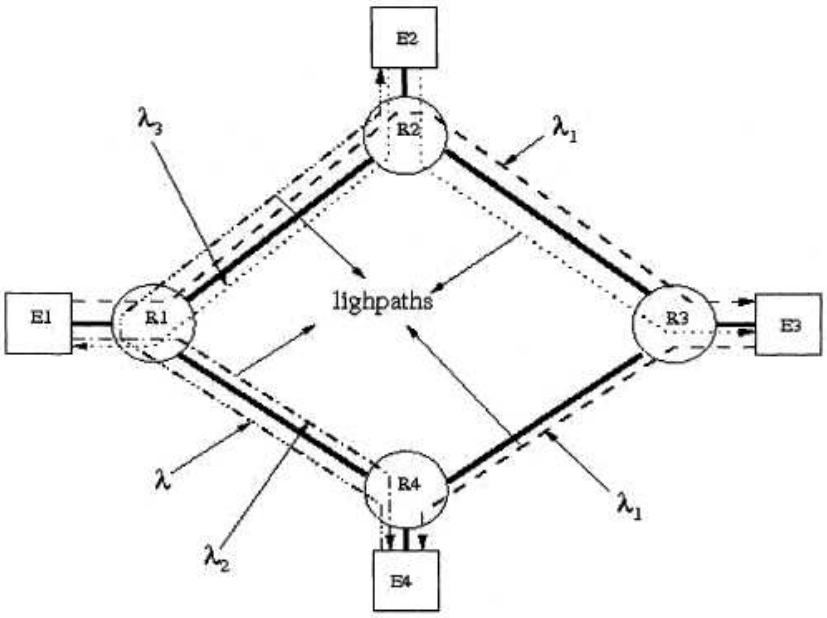

Figure 2.2.2.1: A Wavelength-Routed Network [22]

The network shown in Fig. 2.2.2.1 is a wavelength-routed network, since the end nodes communicate using lightpaths where the routing of a lightpath to its destination is based on its carrier wavelength. There are 4 end nodes E_1 – E_4, connected with 4 optical routers respectively in a physical topology and dashed lines show lightpaths which are sent over the physical topology. When an end node E_1 tries to communicate to end node E_4, the signal passes through routers R_1, R_2, R_3, *and* R_4 using the path $R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_4$. The router R_1 sends the signal at wavelength λ_1 to R_2 and so on. When the router R_4 receives this signal, it is passed to end node E_4.

Since end node E_4 is tuned at wavelength λ_1, it receives the signal.

2.2.2.2 Single-hop and Multi-hop WDM Networks

In single-hop network, data remains in the form of an optical signal from the time it leaves the source end node until it reaches its destination end node. All data communication involves a path length of one logical edge. In other words, exactly one lightpath is involved in each communication. Since the signal remains in the optical domain all the way, such a network is also called an *all-optical network* [6]. In a network with N_E nodes, the number of end node pairs is N_E $(N_E$ *-1)*, so that the number of lightpaths becomes impossibly large, even for moderate values of N_E. Since the number of available channels, the number of transmitters and receivers are all limited, single-hop networks are not feasible even for moderate values of N_E.

In a *multi-hop* network, the signal transmitted from a source end node to the destination end node via one or more intermediate end nodes. Signals are converted from the optical form to the electrical form at each intermediate end-node. If there is no direct optical link available between a source end node and a destination end node, multi-hop communication is used. In Figure 2.2.6.1 shows a typical multi-hop optical network.

2.2.3 Multiplexer and De-multiplexer

The use of multiplexers makes it possible to have multiple data streams on the same fiber. A multiplexer combines different distinct signals on different input fibers into one output which can be communicated using a single fiber. A typical multiplexer is shown in Figure 2.2.3.1, which combines 4 input signals each using a distinct channel from $c_1 - c_4$, and the combined signal can be transmitted through a single fiber to the corresponding destinations.

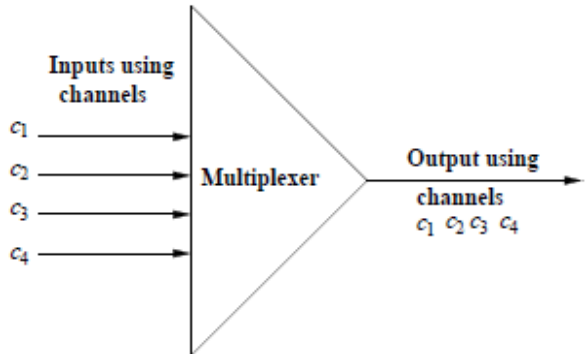

Figure 2.2.3.1: 4 – Input Multiplexer [21]

A de-multiplexer splits the signals carried by an incoming fiber into different outputs, each with a signal using a distinct channel. A typical de-multiplexer is shown in Figure 2.2.3.2, in which the signals on the incoming fiber are separated by a de-multiplexer into 4 outputs, each carrying a distinct signals and using one of the channels $c_{1,}$ $c_{2,}$ c_3 and c_4.

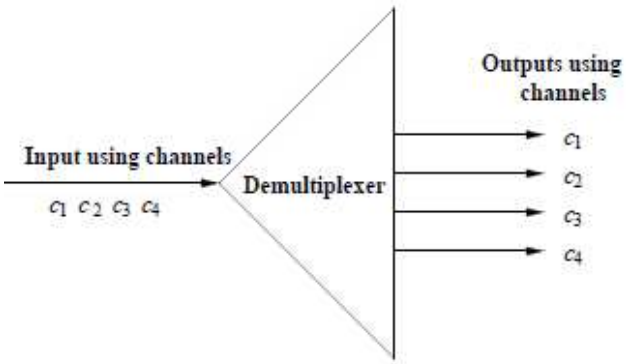

Figure 2.2.3.2: A Typical 4- Output De-Multiplexer [21]

2.2.4 Add-drop multiplexer (ADM)

The ADM is an useful component of optical networks as it has the capability to add one or more new wavelength channels to an existing multi-wavelength WDM signal, and/or to drop (remove) one or more channels, passing those signals to another network. A typical Add-drop multiplexer consists of a multiplexer and a de-multiplexer as shown in Figure 2.2.4.1. This figure shows that signals using two channels from the output of de-multiplexer c_2 and c_3 are dropped (in other words, are not sent to the input of multiplexer). These two signals may be converted into electrical signals and used in the end node attached to this ADM. The end node also creates two signals using channels c_2 and c_3 and sends them to the inputs of the multiplexer. The multiplexer generates the output, which is fed to an outgoing fiber.

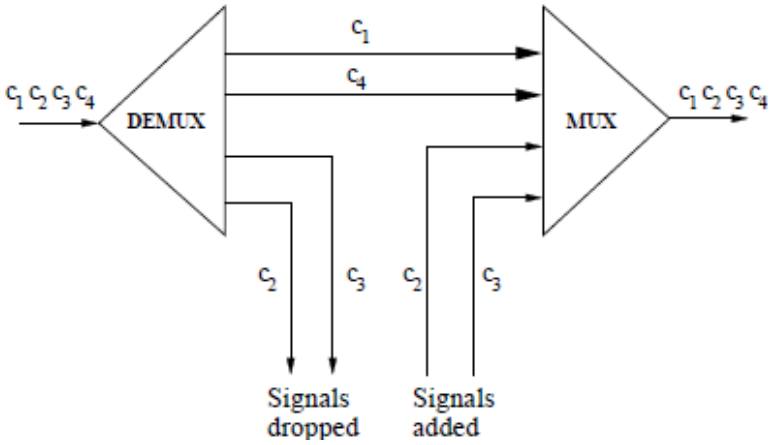

Figure 2.2.4.1: An Optical Add-Drop Multiplexer [21].

2.2.5. Wavelength Router

The Wavelength Router is a device used to route an optical signal to its output destination according to its wavelength. An optical router has the same number of

input ports and output ports, each carrying many optical signals. The optical router determines which incoming signal has to be routed to which outgoing fiber.

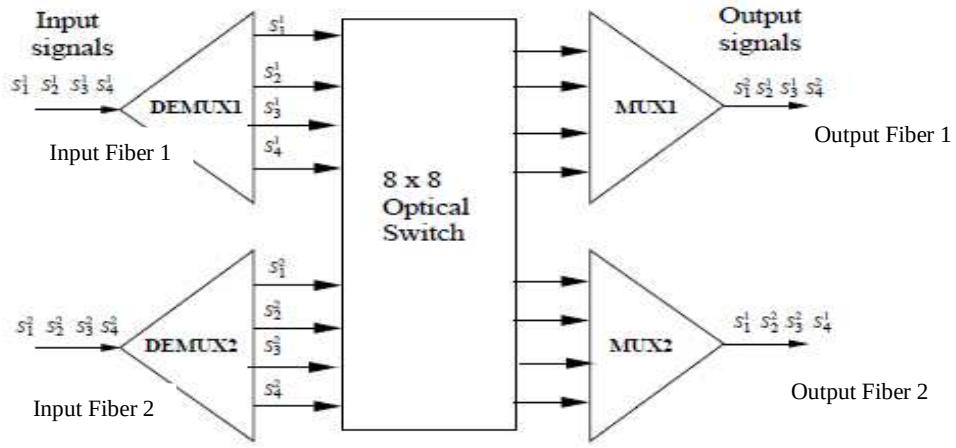

Figure 2.2.5.1: Router Node [21]

Fig. 2.2.5.1 shows a basic diagram of an optical router node with *multiplexers and de-multiplexers* and having two input fibers and two output fibers. The figure shows that input signal s_1^1 is coming through input fiber 1, and passes through demux1, routed by the 8 x 8 optical switches to the appropriate input of MUX2 and ultimately is routed to output fiber 2.

2.2.6 Physical Topology

The physical topology of a WDM network consists of many optical devices. Figure 2.2.6.1 shown below is a simplified diagram of a typical physical topology of an optical network. In the diagram an oval represents an *end node* which is the source/destination of data generated by the user. A square represents a router which directs the data signal to the proper destination. A directed line represents a

fiber. A fiber connection allows unidirectional communication and the arrow on the line gives the direction in which optical signals can flow. It is convenient to represent the simplified topology as a graph $G_p = (V, E)$ in which each end node or router in the network is a vertex in set V, each fiber optic link between two nodes is an arc in set E.

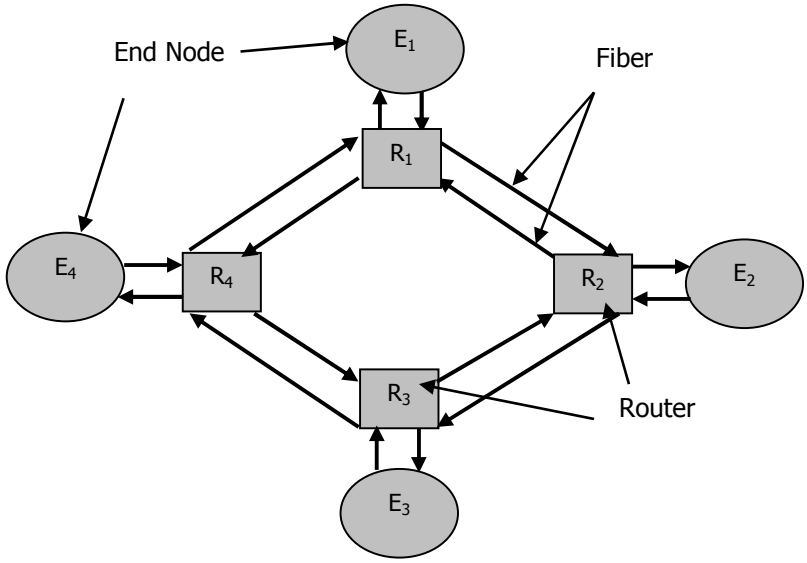

Figure 2.2.6.1: The physical topology of a typical WDM network with four end-nodes E1-E4 and four routers R1-R4

2.2.7 Lightpath

A *lightpath* is an optical connection from one end node to another, used to carry data in the form of encoded optical signals. Such a lightpath always starts from an end node, traverses a number of fibers and router nodes, and ends in another end node [21]. Figure 2.2.7.1 shows a number of lightpaths using the physical topology. For example lightpath L_1 started from E_1, terminated at node E_3 and uses path $E_1 \rightarrow R_1 \rightarrow R_4 \rightarrow R_4 \rightarrow E_3$.

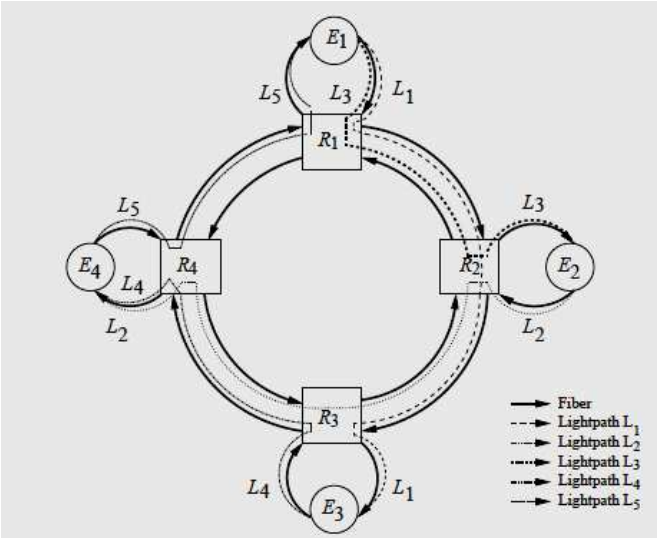

Figure 2.2.7.1: Lightpaths On the physical topology [21]

The characteristics of a lightpath are as follows:

- A lightpath is an all-optical connection from one end-node to another that is used to carry data in the form of encoded optical signals.
- The lightpath uses a path consisting of a sequence of physical links from the source to destination of the lightpath.
- It is possible for a lightpath to have several wavelengths on different fibers in its path. Such a lightpath must use an *all-optical* wavelength converter.
- Current optical networks do not have wavelength converters and a lightpath must use the same wavelength on all fibers in its path.

An all-optical network refers to the class of networks where the information remains in the optical domain in the entire path from the source node to the destination node; in such networks, there is no conversion between optical signals to

electrical signals along the path used for data transmission.

2.2.8 Logical Topology

Since the lightpaths determine which end nodes can directly communicate with other end nodes, once the lightpaths are settled up as shown in Fig. 2.2.8.1, the physical topology is irrelevant for determining a strategy for communication. It is convenient to view the lightpaths as edges of a directed graph G_L where the nodes of G_L are the end nodes of the physical topology. Such a graph is called the *logical topology* of an optical network and the edges of such a graph are called *logical edges* [21]. The directed graph shown in the figure 2.2.8.2 is a logical topology. The lightpath forms a basic data communication link from a given source end node to destination end node in an optical network. In general, an optical network has many lightpaths, defining optical connections between selected pairs of end nodes. In Figure 2.2.8.1, a logical edge starts from end node E_4 and ends at end node E_1. Another lightpath is from end node E_1 to end node E_3. When a lightpath is established for data communication between one source end nodes to another destination end node, it can pass through intermediate end nodes.

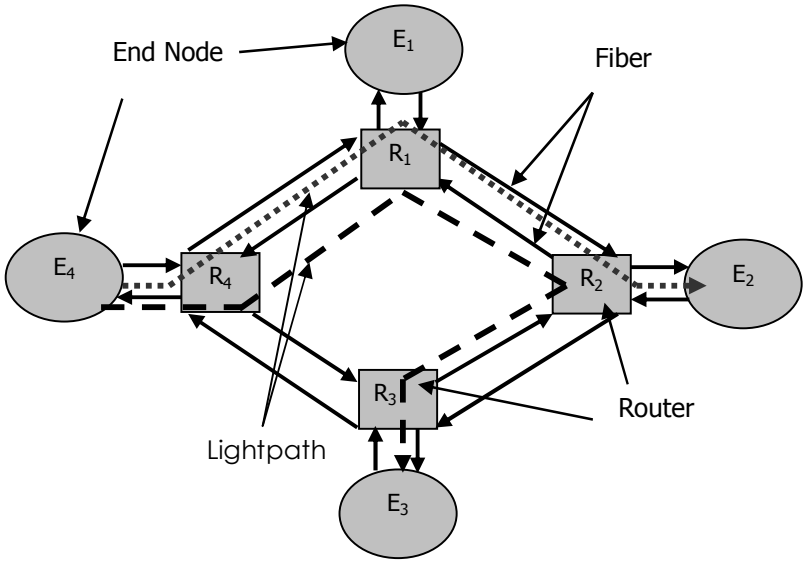

Figure 2.2.8.1: Lightpath shown by dashed-line on Physical Topology

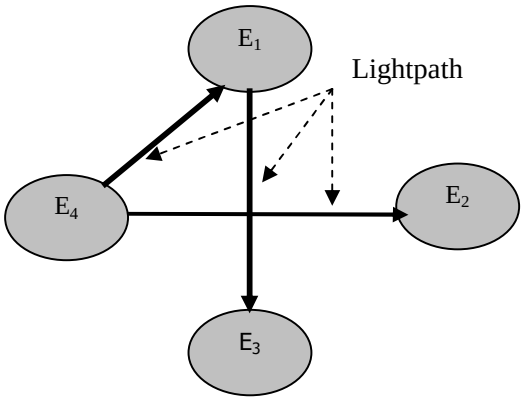

Figure 2.2.8.2: A logical topology drawn from physical topology has shown Fig. 2.2.8.1.

2.3 Route and Wavelength Assignment (RWA)

As we have mentioned in Section 2.2.2, *WDM* allows the same fiber to carry many signals independently, as long as each uses a different carrier wavelength and maintains sufficient channel spacing. The channel assigned to the lightpaths should be such that two lightpaths sharing a fiber are never assigned the same channel. Determining the path of a lightpath and the channel number is known as the *routing and wavelength Assignment* (RWA) problem.

Two versions of the RWA problem have been considered by researchers [21]. If the set of lightpaths to be set up is known in advance, the problem is called the *off-line RWA* problem. In this formulation the traffic demand is known in advance and is not expected to change in the near future. This is also called *static lightpath allocation* since the lightpaths, once established, are not modified until the traffic pattern changes sufficiently to warrant a different set of lightpaths.

The other version of the problem is to set up the lightpaths on demand and is called *online RWA* (also called *dynamic lightpath allocation*) problem [25]. In this problem, requests for data communication are considered as and when they occur. In dynamic allocation, lightpaths are set up when needed and are taken down when the communication is over.

The following approaches primarily used for *RWA* problem:

- RWA as graph coloring problem
- Integer linear programming
- RWA using a heuristic.

The solutions for static lightpath allocation using mathematical programming are computationally intractable even for medium-sized networks. Heuristics are useful for solving the problem, within a reasonable amount of time [22].

2.4 Logical Topology Design Problem

The problem of designing logical topologies is to find a set of *lightpaths* defining which pairs of end nodes are to be connected by logical edges, in order to handle the expected traffic in an economic manner. In a *WDM* network, the optical fibers can support a number of channels for data communication. The actual number of channels supported by a fiber depends on the technology used and the type of fiber used. To design an optimal logical topology, we need to consider the following:

- The set of *lightpaths* to be created,
- For each lightpath, the route through the *physical topology* and the channel to be used on each fiber in its route,
- The strategy for routing the traffic over the logical topology.

All the points mentioned above are not independent. Some early researchers solved these three problems in one formulation, using some *MILP* (Mixed Integer Linear Programming) for logical topology design and for routing[1]. These formulations are intractable for practical sized networks, since they involve large numbers of integer variables. This fact has motivated the development of efficient heuristic solutions that are "reasonably" good and solve the logical topology design problem in a reasonable time frame [7]. The logical topology design problem has been sub-divided into three sub-problems as follows:

i) Find the logical topology,

ii) Carry out *RWA* for each lightpath,

iii) Find the optimal traffic routing strategy.

2.5 Traffic Matrix

It is convenient to represent the traffic requirements in the form of a matrix $T = [t(i, j)]$, often called a *traffic matrix*. The entry $t(i, j)$ in row i and column j of traffic matrix T denotes the amount of traffic from end node E_i to E_j, where $i \neq j$.

The signal rate is normally expressed, using the *Optical Carrier level* notation (OC-*n*), where the base rate (OC-1) is 51.84 Mbps and OC-*n* means ($n \times 51.84$) Mbps, depends on the technology used. A typical traffic matrix is shown in Table 2.5.1 below.

[1] MILP appears in Section 2.7.

T =

Node	1	2	3	4
1	0.00	0.30	0.5	0.30
2	0.20	0.00	0.30	0.20
3	0.35	0.10	0.00	0.30
4	0.00	0.20	0.10	0.00

Table 2.5.1: A Traffic Matrix

In our thesis, for simplicity, we have defined the traffic load as a fraction of the capacity of a lightpath and the maximum is 1.0. As shown in table 2.5.1, the entry t (1, 4) is 0.20, meaning that the amount of data to be communicated from end node E_1 to end node E_4 is 0.20 units. If a single lightpath can carry data at the rate of 10.0 Gbps (i.e., OC-192), the expected data communication rate from end node E_1 to end node E_4 is 10 * 0.20 = 2 Gbps.

It is convenient to consider the traffic corresponding to each pair of end nodes as a commodity and attach a distinct number k with such a commodity. We will designate commodity K^k as the traffic corresponding to the pair of end nodes ($E_{sk,}$ E_{dk}) having $t(s_k, d_k) > 0$. In the above Figure 2.5.1, there are eleven (11) nonzero entries, so that there are eleven commodities. For example, K^1 is for pair ($E_{1,}$ E_2), corresponding to row 1, column 2 and has traffic 0.30. Similarly commodity K^{11} is for pair ($E_{4,}E_3$), corresponding to row 4, column 3 and has traffic 0.10.

2.6 Congestion Minimization

A request from a specific source to a specific destination constitutes a commodity. Each entry of the traffic matrix $t(s, d) > 0$, is a distinct commodity as described in

Section 2.5. The congestion is the value of the highest traffic that is flowing on any particular edge of a logical link. The total traffic flowing on an edge is the sum of all the flows for all the commodities on that edge. Figure 2.6.1, which is drawn from Table 2.5.1, shows that there is a total of 0.60 unit of traffic flowing on the edge $E_2 \rightarrow E_3$ for two different commodities (C_1 and C_2) for a specific logical topology and routing scheme. Since, the traffic flowing on this edge is the maximum traffic; this edge is the congested edge.

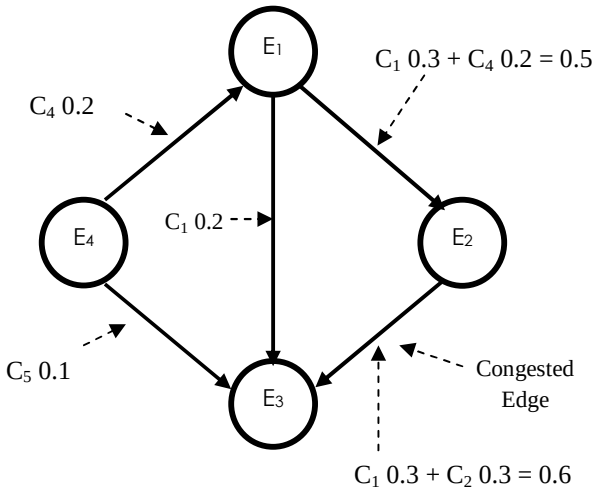

Figure 2.6.1: Congestion of a Network

Congestion *minimization* helps to decrease the chances of a *bottleneck* in the context of communication. Minimizing the value of congestion is one way to reduce the cost of a network since each lightpath means additional cost due to the optical (electrical) hardware at the source and the destination of the lightpath. A lower value of congestion allows greater possibility of scaling up the traffic in the network without changing the routing strategy - more traffic can be arranged to flow without additional resources to the network and is therefore more economical.

2.7 MILP- based solution of the logical topology design and the routing problem

In a linear programming formulation, if some of the variables are constrained to have integer values and others are continuous variables, the formulation is called a *mixed integer linear program* (*MILP*). The problem of logical topology design is to find which pairs of end nodes are to be connected by a lightpath. A lightpath may be represented by using a binary variable having a value of 1 (0). This binary variable may be used to denote whether a lightpath exists (does not exist). Therefore, these variables are needed in each pair of end nodes. When the logical topology is known, the problem of routing strategy is to determine how the traffic may be handled optimally over the logical topology. The details of the routing problem are discussed in Section 2.9.

The formulation of the *MILP* has been provided below [21]. The term *L-congestion* used here to describe the maximum traffic Λ_{max} on a logical link[2]. To determine the value of *L-congestion*, we first have to determine an optimal logical topology and then find an optimal routing over the logical topology.

Objective function:

$$\text{Minimize } \Lambda_{max} \qquad (7.1)$$

Subject to

1. Ensure that Λ_{max} is the *L-congestion*.

[2] An explanation for the notation used in this thesis appears in appendix 1.

$$\sum_{s=1}^{N_E}\sum_{d=1}^{N_E} x_{ij}^{sd} \leq \Lambda_{\max},\ \forall i, j,\ 1 \leq i, j \leq N_E,\ i \neq j \qquad (7.2)$$

2. Part of the traffic *t(s, d)* can flow on the logical edge $E_i \Rightarrow E_j$ only if the logical edge exists. Also x_{ij}^{sd} , the part of the traffic *t(s, d)* flowing on the logical edge $E_i \Rightarrow E_j$ cannot exceed the traffic *t(s, d).*

$$x_{ij}^{sd} \leq b_{ij} \cdot t(s, d), \qquad \forall i, j, s, d,\ 1 \leq i, j, s, d \leq N_E, i \neq j, s \neq d \qquad (7.3)$$

3. Apply the flow conservation rules.

$$\sum_{i=1}^{N_E} x_{ij}^{sd} - \sum_{j=1}^{N_E} x_{ji}^{sd} = \begin{cases} t(s,d) & if\ s=i \\ -t(s,d) & if\ d=i, \\ 0 & otherwise \end{cases} \quad \forall i,s,d, 1 \le s,d,i \le N_E, s \ne d \qquad 7.4)$$

4. The number of lightpaths ending at (starting from) a given end node cannot exceed Δ_{in} (Δ_{out}).

$$\sum_{i=1}^{N_E} b_{ij} \le \Delta_{in}, \quad \forall j, 1 \le j \le N_E$$

(7.5)

$$\sum_{j=1}^{N_E} b_{ij} \le \Delta_{out,} \quad \forall j, 1 \le j \le N_E$$

(7.6)

Equation (7.2) ensures that $\Lambda_{\max}$ is greater or equal to the traffic on any edge. If a lightpath from end node E_i to end node E_j exists (in other words, if b_{ij} is 1) then equation (7.3) ensures that part of the traffic *t (s, d)* may be routed over logical edge $E_i \rightarrow E_i$. Equation (7.4) ensures that the total traffic flowing out from E_i is $\sum_{i=1}^{N_E} x_{ij}^{sd}$ and the total traffic flowing into E_i is $\sum_{j=1}^{N_E} x_{ji}^{sd}$. If E_i is the source, then there is no traffic flowing into E_s. If E_s is the destination, then there no traffic flowing out of E_s, otherwise for all intermediate node, the incoming and outgoing traffic difference should be 0. Equations (7.5) and (7.6) ensure that the number of

lightpaths do not exceed the number of transmitters and/or receivers.

2.8 A Heuristic for Designing a Logical Topology

Heuristic is defined as any technique used to obtain an approximate (i.e., close to the optimum value) solution of a given problem. As we have mentioned in our problem definition in Chapter 1, we will be using a heuristic for designing the initial logical topology. The pseudo code for a heuristic to solve the logical topology design problem [21] is given below.

The steps of the heuristic are as follows:

Step 1: Find the entry t ($i_{max,,}\ j_{max}$) in T having a maximum value among all the entries in T. If there is no nonzero entry in T, stop.

Step 2: Using any RWA technique (Discussed in 2.3), check if it is possible to establish a lightpath from E_{imax} to E_{jmax}. If RWA is not possible, set t ($i_{max,}\ j_{max}$) to 0 and go to Step 1.

Step 3: Create a logical edge from end node E_{imax} to end node E_{jmax}. Set t($i_{max,}\ j_{max}$) to

t($i_{max,}\ j_{max}$) - $c_{lightpath}$ or 0, whichever is greater. Go to Step 1.

Note: We did not consider any restrictions on Δ_{in} (receiver) or Δ_{out} (transmitter) when designing our initial logical topology using this heuristic. We also assumed that it is possible to carry out a RWA to set up a lightpath from any source to any destination.

2.9 Routing over a Logical Topology

As we have mentioned in Section 2.5 that the traffic requirement of a N_E node network can be represented as a form of matrix T where, each individual entry of the matrix can be presented as t_{ij} ($0 \leq i,j \leq N_E$). This gives us the amount of traffic to be

routed from source end node E_i to destination end node E_j.

Now, if the logical topology is already known, then the other part of our problem is to determine the routing optimally for the each traffic entry t_{ij}. This type of problem may be viewed as *MCNF* (Multi-Commodity Network Flow) problem [7]. For each pair of end nodes, the distinct number of commodity has to be assigning for transportation from one source node E_s to destination node E_d. Therefore, we can say that there is k commodity has single source E_{sk} and a single destination E_{dk}. In this situation, a directed graph $G = (V, E)$ may be viewed as the transportation network where V is the set of vertices in G and each V_i ($0 \leq i \leq N$) represents possible source (destination). Each $i \rightarrow j$ represents the logical edge (link) from source i to destination j. If there are n_{sd}^{light} lightpaths from source node E_s to destination node E_d and $C_{lightpath}$ is the capacity of a link, then (n_{sd}^{light} * $C_{lightpath}$) amount of data may be handled from E_s to E_d. In a transportation network when there is single commodity to be considered, the problem is called a *single commodity* flow problem; otherwise, it is called a *multi commodity* network flow problem.

Networks consisting of less than 30 end nodes are considered small to medium-sized networks. Since, the first part of our problem is known, the logical topology is fixed, and routing can be done by the formulation described below, since, the values b_{ij} ($1 \leq i, j \leq N_E$) are known, it is a LP program.

Routing this type of flow problem can be formulated as follows: [21]

Objective function:

$$\text{Minimize } \Lambda_{max} \qquad (2.9.1)$$

Subject to

1. Ensure that $\Lambda_{\max}$ is the *L-congestion*.

$$\sum_{k=1} x_{ij}^{k} \leq \Lambda_{\max}, \forall (i, j) \in L$$

(2.9.2)

2. Apply the flow conservation rules.

$$\sum_{j:(i,j)\in L} x_{ij}^{k} - \sum_{j:(j,i)\in L} x_{ji}^{k} = \begin{cases} t^{k} \text{ if } src_{k} = i, \\ -t^{k} \text{ if } dest_{k} = i, \\ 0 \quad \text{otherwise} \end{cases}$$

(2.9.3)

(2.9.3) has to be applied to node *i*, for all *i*, $1 \leq i \leq N_E$, and has to be repeated for each commodity *k* , $1 \leq k \leq q$.

The notation used in this formulation has been given in Appendix1.

2.10 Reconfiguration of Logical Topology

The motivation of logical topology design is to optimize the network performance, improving the congestion, the delays and the throughput metrics. The *WDM* networks have the property of dynamically changing its logical topology corresponding to the changing traffic conditions [8]. This ability to dynamically optimize the network for changing traffic patterns is one of the key features of multi-

wavelength optical network. As we have mentioned in Chapter 1, traffic does not remain the same all the time. Therefore, when the given logical topology is incapable of supporting changing traffic demands, the logical topology needs to be changed. This can be viewed as a *reconfiguration* of the logical topology. The general approach to the logical topology reconfiguration problem has been a two-phase operation: the first phase being a logical topology design for the new traffic conditions and the second phase being a routing scheme to handle the increased traffic demand over the logical topology, in order to meet the objective function, such as minimizing the congestion of the network.

In this discussion, we have an arbitrary logical topology based on some physical topology, which was capable of handling the traffic that existed when the logical topology was designed, but after a certain period of time, the traffic load will change and the current logical topology cannot handle the new traffic. In this situation we need to change the logical topology or reconfigure the logical topologies that can handle the increased traffic in an optimal manner.

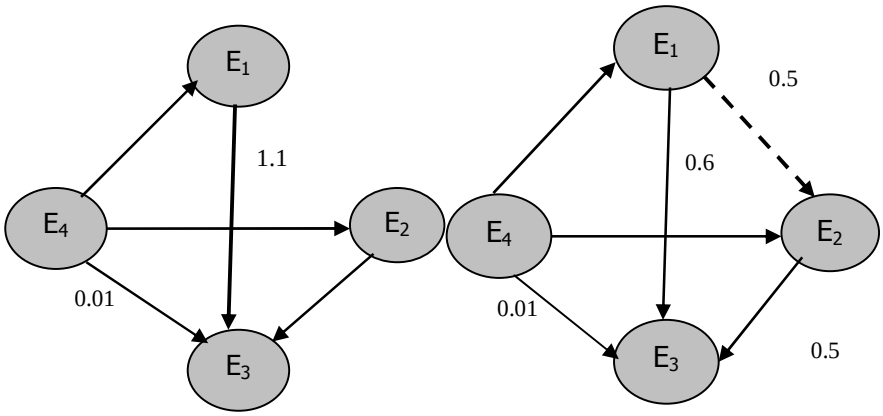

Fig. 2.10.1a: Old Topology

Fig. 2.10.1b: New Topology

Figure 2.10.1: Reconfiguration of Logical Topology

In Fig. 2.10.1a, end node E_1 sending traffic of 1.1 (which is greater than the maximum allowed value of 1.0) to end node E_3 and end node *E4* sending a relatively very small traffic (0.01) to end node E_3. The network shown in Fig. 2.10.1a is an overloaded network, since edge $E_1 \rightarrow E_3$ is carrying more traffic than its capacity. In this situation, two strategies could be considered to change (reconfigure) the logical topology. One is to find the best logical edge (s) to be added to the topology, another one is to remove one logical edge and add one logical edge. If very little traffic is flowing on an edge, the edge is a candidate to be deleted. As a result of the deletion, there will be no significant effect on the traffic carried by the network. In our thesis, we have implemented the first option only.

In Fig. 2.10.1b shows that we have added one edge from end node E_1 to end node E_2 and then routed the traffic accordingly. The traffic flowing on every edge is below the capacity of logical edge. Therefore, the reconfiguration has been done on the existing l*ogical topology*.

In summary, when a *lightpath* is created from source to destination, a source node communicates directly with the destination node. A *lightpath* can carry a certain amount of traffic. When traffic increases more than the capacity of a logical edge, the *logical topology* needs to be changed in order to get an optimal logical topology capable of handling the new traffic. An optimal *logical topology* is always desirable, because it improve the performance of the network.

Our main objective is to reconfiguration of *logical topology*, focusing on minimizing the *congestion* of the network. There are a few key terminologies, related to reconfiguration is described below.

2.10.1. Optimization

In general, the *optimization* means, to find the best value of some objective function. In optical networks, the optimization problem for a given logical topology could be to find the optimum strategy to handle all the traffic for data communication in the network. Some of the optimization strategy could be as follows:

- The maximum number of hops from the source to any destination.
- The use of resources, such as the number of channels used.
- Some linear combination of the number of receivers and transmitters used.
- The set of lightpaths to be created.
- Minimizing the route through the physical topology.
- Minimizing the number of channels on each fiber in its route.
- Minimizing the total traffic load on a path from source to destination

2.10.2. Optimization approach

In general, *logical topology* design problems can be formulated as *optimization* problems aimed at *maximizing* network throughput by *minimizing* some objective function or other performance measure of interest. Routing the traffic optimally on the *lightpaths* is also usually seen to be a part of the *logical topology* design problem. For this purpose, the problem can be decomposed into the following sub problems.

- Static traffic control
- Dynamic traffic control

2.11. CPLEX Optimizer

ILOG CPLEX (often informally referred to simply as CPLEX) is an optimization software package. It is named for the simplex method and the C programming language, although today it contains interfaces for the C++, C#, and Java languages. It was originally developed by Robert E. Bixby and sold via CPLEX Optimization Inc., which was acquired by ILOG in 1997; ILOG was subsequently acquired by IBM in January 2009. The CPLEX tools solve integer linear programming (*ILP*) problems in medium sized network very efficiently.

2.12 Hill Climbing Heuristic

The Hill climbing is a class of methods, which start with an initial solution to be improved and generates a sequence of new solutions by perturbing the current solution, and then accepting the new solution permanently, temporarily, or rejecting it completely. A simple illustration of a hill climbing algorithm is given in fig. 2.12.1.

1. *current* ← initial state or solution
2. **loop do**
    a. *next* ← a highest-valued successor of *current*
    b. if VALUE[*next*] < VALUE[*current*] **then return** *current*
    c. *current* ← *next*
3. **end**

Figure 2.12.1: Hill Climbing Algorithm

It is simply a loop that continually moves in the direction of increasing value. The algorithm does not maintain a search tree, so the state data structure need only record the state and its evaluation , which we can denoted by VALUE. In this solution there is no guarantee that there are no local minima of the objective function. A solution is called a local minimum if a better solution can be obtained only after a sequence of perturbations, some of which are not improvements. A hill climbing algorithm [11] also makes use of short-term memory of searching algorithm, if there is no restrictions are applied. In our problem solution is a classic hill climbing is used to search the neighborhood and select better solution in the each iteration.

2.13 Overview from Previous Research Paper

An overview of reconfiguration issues in virtual topology design is given in [7]. Two approaches have been discussed, i) Cost approach: The concern is to minimize the cost of the reconfiguration, in terms of the number of Wavelength Routers that need to have their optical switching reprogrammed, or the total number of optical switching that need to be changed to implement the new lightpath and eliminate old ones. ii) Optimization Approach: In this approach, the virtual topology

is given, when the traffic changed, the reconfiguration necessary to optimize some objective function is carried out.

Two sub-problems have been mentioned in this paper [9]. One is to find connectivity and another is to route the traffic. Algorithms have been proposed in this paper to iteratively reconfigure the logical topology to minimize the congestion in response to changes in the traffic pattern using a local search technique. For routing they have used a minimum hop routing algorithm. They have also worked on minimizing the disturbance to the network at the time of reconfiguration by a "branch-exchange" technique. The author claimed that since each change to the logical topology is small, the disturbance to the network is small.

The reconfiguration issues arising in single-hop lightwave networks have been studied in [13]. They claimed that this is the first in-depth study of the tradeoffs involved in carrying out the reconfiguration process. They developed and compared reconfiguration policies to determine *when* to reconfigure the network, and presented an approach to carry out the network transition by describing a class of strategies that determine *how* to retune the optical transceivers. The reconfiguration problem is formulated in this paper as a *Markovian decision process* [28].

The complete logical topology design problem including traffic routing, lightpath selection and *RWA* has been considered in [14]. The problem is formulated as an *MILP*, where the objective is to minimize the congestion. For large networks, a solution based on LP-relaxation of the integer variables is presented.

An *MILP* for optimal logical topology design and an evaluation of a number of existing heuristics for logical topology design, in terms of both performance and complexity, has been given in [18].

A tabu-search-based meta-heuristic for optimal logical topology design, objective is to minimize the congestion of the multi-hop network, has been presented in [5]. The paper assumes that the physical topology and a stochastic description of

the traffic pattern is given, and does not consider constraints on the number of wavelengths per fiber. The move selection procedure has been given in the following Fig.2.12.1 below:

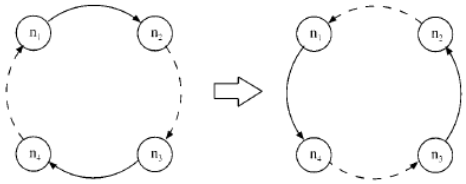

Figure 2.13.1: Move Selection Procedure [5]

A tabu search heuristic for solving the routing and wavelength assignment (*RWA*) problem in optical *WDM* networks has been proposed in this paper [15], considering the wavelength continuity constraint and a given set of connections to satisfy. For a number of available wavelengths on each link, this algorithm attempts to maximize the number of routed connections. Using the *tabu search* algorithm, they solved the problem of *RWA* in two steps. During Step 1, a set of paths is selected in the graph. Then Step 2 uses paths selected during Step 1 to build a solution of the problem.

The heuristic approach presented in this paper [16], adapts the tabu search strategy proposed in [4] for throughput maximization, to the multi-criteria problem of simultaneously minimizing congestion and total flow. Tabu search is implemented as a two phase strategy dealing with diversification as well as intensification of search. A local search based on branch-exchange and tabu strategy is used to explore different virtual topologies.

A *Tabu Search* Algorithm have been proposed in this paper [17], to design a logical topology for packet switched traffic over *WDM* mesh networks. The algorithm evaluate the cost-performance trade-off between,

i) Designing logical topology with small congestion and large number of lightpath.

ii) Designing a less expensive topology with higher congestion.

They have solved the routing problem with *ILP* formulation and Tabu Search technique used for move selection procedure. The process used depicted below in Fig. 2.10.2.

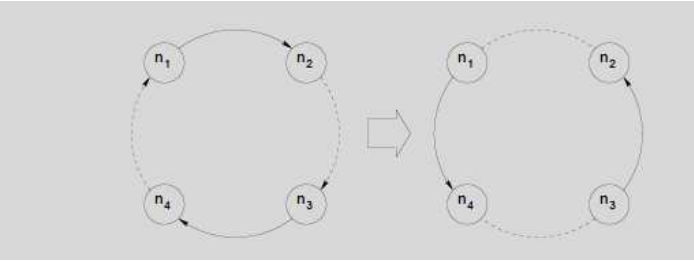

Figure 2.13.2: Move Selection Procedure in Tabu Search [17]

Chapter 3: Problem Specification

3.1 Problem definition

In Section 2.4, we have discussed the problem of designing an initial logical topology. We now define the problem of reconfiguring the logical topology in a WDM optical network. Given an initial logical topology, when the traffic demand increases and the current logical topology is incapable of handling the increased traffic demand, our problem is to find a new logical topology, capable of handling the current demand, with minimal changes to the original logical topology.

It is well known that the problem of determining an optimal logical topology is computationally intractable [21]. Reconfiguring the logical topology is a variation of the same problem and is intractable for the same reason. Our objective is to reconfigure the topology so that the congestion is below the capacity of a lightpath. We need to find an optimal logical topology with an acceptable congestion of the network. This problem has two sub problems as follows:

1. **Find Topology** (***FT***): Find the new logical topology by reconfiguring the initial logical topology.

2. **Route Traffic (*TR*):** Route the current traffic optimally over the logical topology.

Fig. 3.1.1 shows a schematic description of the problem including the input parameters and the output.

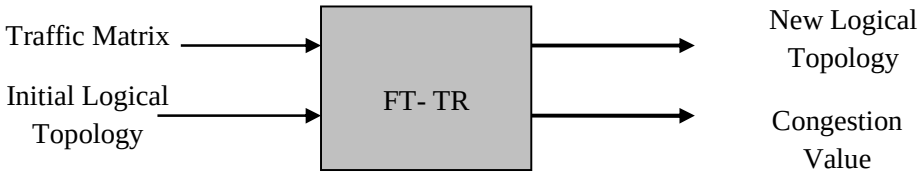

Figure 3.1.1: Schematic Description of the Problem

Since, we cannot route the traffic without determining a logical topology, these two problems are interrelated and it should be solved simultaneously. The problem of finding a new optimal logical topology can be viewed as a reconfiguration of the logical topology since we start with the original topology and add a minimum number of logical edges. In this search we can view the current logical topology as the current state and a new logical topology as the new state - the result of a move. The term "move" here means the application of a perturbation to the current logical topology to create a new logical topology. The search space for a given logical topology in a WDM network is, in general, vast. We have used a hill climbing technique using four strategies to limit the time needed to find a candidate optimal logical topology. Once a new optimal logical topology is determined, the CPLEX optimizer program has been used to route the traffic over the logical topology to determine the congestion. The CPLEX program can be used only on small to medium sized networks for determining an optimal solution.

3.2 Identifying the move

We have mentioned in Chapter 2 that the congestion of a logical topology for a given traffic matrix and routing strategy is the value of the traffic on the edge carrying the maximum load [21]. Our objective is to reduce the congestion to a value below the capacity of a light path. A move in our problem identifies an edge to be

added to the current logical topology to reduce the congestion. Let the logical edge carrying the maximum load be the edge from source node i_{max} to destination node j_{max} (Figure 3.3.1). Traffic flowing on the edge i_{max} to j_{max} could be due to the following:

1. The source for a communication could be i_{max} itself, where part of the traffic resulting from the communication is being routed, using the edge $i_{max} \rightarrow j_{max}$. The arrow '→' is used in between two nodes to describe a logical edge in our thesis.

2. The source for a communication could be some other node where some or all of the traffic for the communication is being routed, using the edge $i_{max} \rightarrow j_{max.}$

3. Destination of the traffic flow could be j_{max} itself.

To keep the problem tractable, we have limited the search space we investigated to four strategies involving the edges that either start from or end at $i_{\max}$ or $j_{\max}$ as follows:

3.3 Strategy 1.

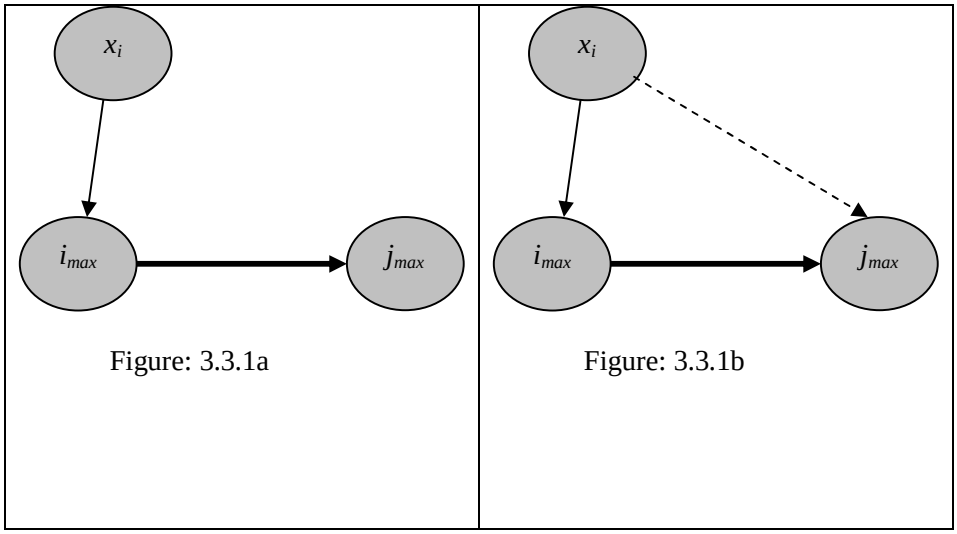

Figure: 3.3.1a

Figure: 3.3.1b

Figure 3.3.1: Adding edge from Strategy1

Condition for applying the strategy: This strategy is applicable if there is an edge from node x_i to i_{max} but there is no edge from node x_i to j_{max}, shown in Fig. 3.3.1a.

Details of the strategy: Strategy1 adds an additional edge from node x_i to j_{max} as shown, using a dashed line in Fig 3.3.1b.

The rationale for the strategy: Part of the traffic flowing on edge $x_i \rightarrow i_{max}$ is likely being routed through the edge $i_{max} \rightarrow j_{max}$. In this case, adding an edge from end node x_i to j_{max} could be promising. Part of the traffic flowing to end node j_{max} using edge $x_i \rightarrow i_{max} \rightarrow j_{max}$, could flow directly from node x_i to node j_{max} as an alternative path (Figure 3.3.1b) and may reduce the congestion of the network.

Potential benefit for the strategy: In this strategy, we calculate the potential benefit for adding the edge $x_i \rightarrow i_{max}$ as follows.

In general, there are a number of commodities flowing on the edge $x_i \rightarrow i_{max}$ and i_{max}

$\rightarrow j_{max}$. The following process calculates the benefit for commodity k:

i. Let the traffic request t_1 be flowing on the edge $x_i \rightarrow i_{max}$ for commodity k_i

ii. Let traffic t_2 be flowing on edge $i_{max} \rightarrow j_{max}$ for commodity k_i.

iii. Let t_{min} be the minimum of t_1 and $t_{2..}$

We can calculate the total benefit for all commodities, simply by adding up all the t_{min} found from step iii for all the commodities. After we find out the total benefit for the edge $x_i \rightarrow i_{max}$ using the Steps 1 to 3, we calculate the total benefit for all other edges incoming to i_{max} by using the same process. Finally we select the edge which has the highest benefit among all the edges and save the information of that edge as the best potential move from Strategy 1.

The rationale for considering the minimum traffic flow for a particular commodity k_i is illustrated by an example below. There are two cases to be considered to achieve the best benefit.

Case-1: The edge $i_{max} \rightarrow j_{max}$ is carrying a lesser amount of commodity k_i compared to the edge $x_i \rightarrow i_{max}$.

Case-2: The edge $i_{max} \rightarrow j_{max}$ is carrying a greater amount of commodity k_i compared to the edge $x_i \rightarrow i_{max}$.

<u>An example for Case-1:</u>

Let 0.45 units of commodity k_i flow on the edge $i_{max} \rightarrow j_{max}$ and 0.55 units of commodity k_i flow on the edge $x_i \rightarrow i_{max}$ (Fig. 3.3.2). Now if we add an edge from the end node x_i to j_{max}, as shown using a bold dashed line, a maximum of 0.45 units of commodity k_i can be diverted to the alternative path ($x_i \rightarrow j_{max}$). After 0.55 units of commodity k_i reached the end node i_{max}, 0.45 units flows on the edge $i_{max} \rightarrow j_{max}$ and 0.10 units of commodity k_i might be flowing to some other end node x_n which is not a concern, because our concern is to reduce the traffic load on the edge $i_{max} \rightarrow j_{max}$.

Therefore, the maximum benefit can be achieved by diverting 0.45 units of commodity k_i, which is the minimum of 0.55 and 0.45.

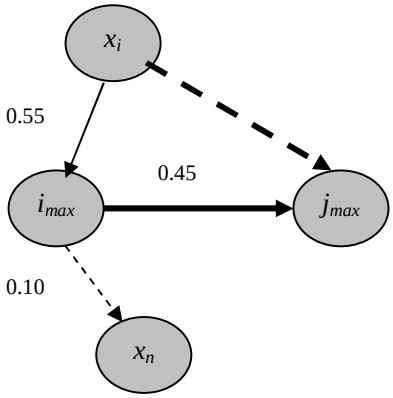

Figure 3.3.2: Benefit calculation for case-1.

An example for Case-2: For commodity k_i, let 0.55 (0.45) units of commodity k_i flow on edge $i_{max} \rightarrow j_{max}$ ($x_i \rightarrow i_{max}$) (Fig. 3.3.3). Now if we add an edge from end node x_i to j_{max}, as shown using a bold dashed line, a maximum of 0.45 units of commodity k_i can be diverted to this alternative path ($x_i \rightarrow j_{max}$). Because 0.45 units of commodity k_i is flowing from x_i to j_{max}, using the path $x_i \rightarrow i_{max} \rightarrow j_{max}$, which can be diverted using the edge $x_i \rightarrow j_{max}$. The 0.10 units of commodity k_i is added to the edge $i_{max} \rightarrow j_{max}$ from some other end node (x_n), again which is not a concern. Therefore, we may divert 0.45 units of commodity k_i which is the minimum value of 0.55 and 0.45.

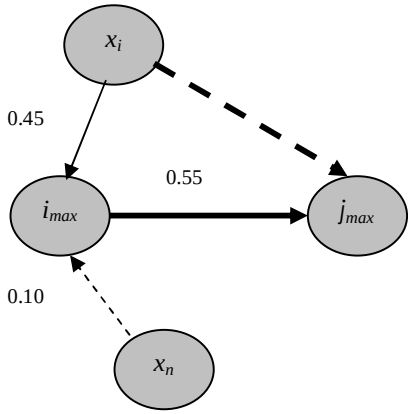

Figure 3.3.3: Benefit calculation for case-2.

Therefore, in both cases, it is beneficial to consider the minimum value of a particular commodity that is flowing through the path $x_i \rightarrow i_{max} \rightarrow j_{max}$.

3.4 Strategy 2

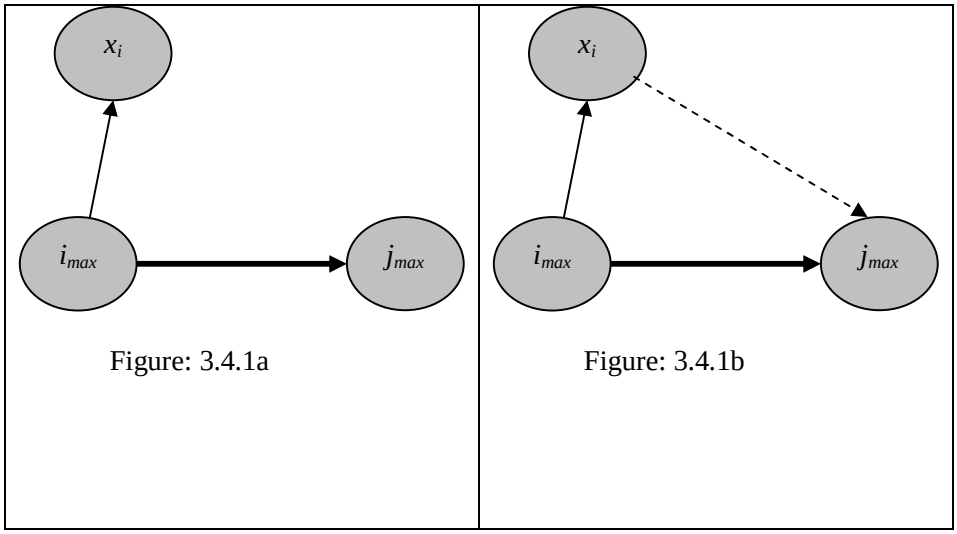

Figure: 3.4.1a

Figure: 3.4.1b

Figure 3.4.1 Adding edge from Strategy 2.

Condition for applying the strategy: This strategy is applicable if there is an edge from node i_{max} *to* x_i but there is no edge from node x_i to j_{max} as shown in Fig. 3.4.1a.

Details of the strategy: Strategy 2 adds an additional edge from node x_i to j_{max} as shown in Fig 3.4.1b.

The rationale for the strategy: Part of the traffic flowing on the edge $i_{max} \rightarrow j_{max}$, could be diverted using the edge $x_i \rightarrow j_{max}$ as shown in Fig. 3.4.1b. In this case, adding an edge from node x_i to j_{max} could be promising, because part of the traffic could be diverted through the path $i_{max} \rightarrow x_i \rightarrow j_{max}$ as shown in Fig. 3.4.1b. As a result, it may reduce the congestion of the network.

Potential benefit for the strategy: The process of calculating the benefit for this strategy is as follows:

i. Let L_{max} be the current congestion value,

ii. Let the total traffic flowing on the edge $i_{max} \rightarrow x_i$ be the minimum traffic (T_{min}),

iii. Therefore, the maximum benefit that can be achieved from Strategy 2 is the average of the difference between L_{max} and $T_{min..}$

The rationale for considering the edge which is carrying the minimum traffic load is explained with an example below.

Let the traffic be flowing on the network shown in (Fig. 3.4.2).

Edge	Data
$i_{max} \rightarrow x_1$	0.45
$i_{max} \rightarrow x_2$	0.35
$i_{max} \rightarrow j_{max}$	0.95

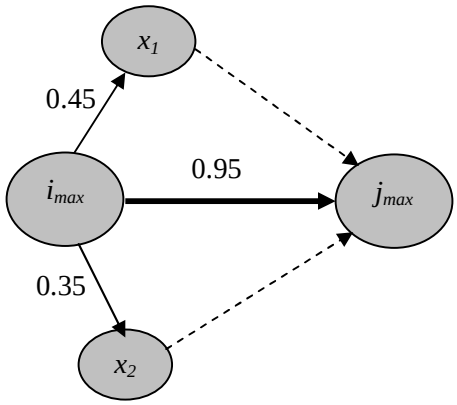

Figure 3.4.2: Benefit Calculation for Strategy 2.

In Fig 3.4.2, there are 3 outgoing edges from i_{max} as follows:

- An edge from i_{max} to j_{max} carrying a flow of 0.95
- An edge from i_{max} to x_1 carrying a flow of 0.45
- An edge from i_{max} to x_2 carrying a flow of 0.35

We note that, considering all the edges from i_{max}, the edge from i_{max} to x_2 is carrying the least amount of traffic and the edge from i_{max} to x_1 is carrying most amount of traffic. In order to reduce the total flow from i_{max} to j_{max}, under Strategy 2, there are two choices:

i) add an edge from x_1 to j_{max}

ii) add an edge from x_2 to j_{max}

Choice 1- adds an edge from x_1 to j_{max}:

In this case, the traffic has to be routed through the path $i_{max} \rightarrow x_1 \rightarrow j_{max}$. Note that the edge $i_{max} \rightarrow x_1$ is carrying the most traffic. As per the process of calculating the benefit from this strategy, the benefit can be achieved by the average of the difference of traffic flow between the edges $i_{max} \rightarrow x_1$ and $i_{max} \rightarrow j_{max}$, which is 0.25 [(0.95 – 0.45) / 2].

Choice 2- add an edge from x_2 to j_{max}:

In this case, the traffic has to be routed through $i_{max} \rightarrow x_2 \rightarrow j_{max}$. As per the process of calculating the benefit from this strategy, the benefit can be achieved by the average of the difference of traffic flowing between the edges $i_{max} \rightarrow x_2$ and $i_{max} \rightarrow j_{max}$, which is 0.30 [(0.95 – 0.35) / 2].

We observed that the maximum benefit can be achieved from choice 2 by adding an edge from $x_2 \rightarrow i_{max}$ and the benefit is the average of the difference of the traffic flow between the edges $i_{max} \rightarrow x_2$ and $i_{max} \rightarrow j_{max}$ as mentioned above. Note that the edge $i_{max} \rightarrow x_2$ is carrying the least amount of traffic.

To justify the cases, let us reduce 0.30 units of data from the edge $i_{max} \rightarrow j_{max}$, which is the average of the difference between L_{max} and T_{min} mentioned in step (iii), and divert this traffic through the path $i_{max} \rightarrow x_2 \rightarrow j_{max}$. After routing the traffic, the total load on the edge $i_{max} \rightarrow x_2$ is 0.65 (0.35 + 0.30), and total load on the edge $i_{max} \rightarrow j_{max}$ is 0.65 (0.95 – 0.30) as shown in Fig. 3.4.3. It shows that, the corresponding traffic on the edges $i_{max} \rightarrow j_{max}$ and $i_{max} \rightarrow x_2$ are uniformly distributed.

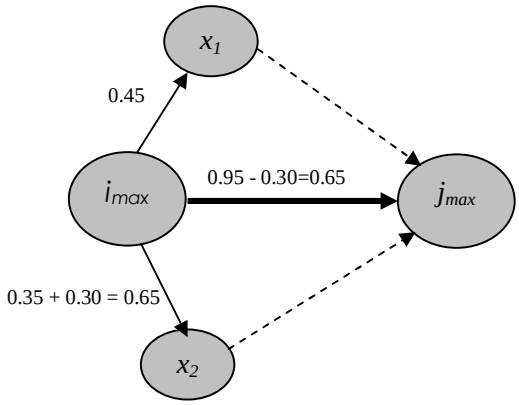

Figure 3.4.3a: Choice 2 - Traffic routing after removing 0.30 units.

In contrast, if we reduce 0.35 units of data from the edge $i_{max} \rightarrow j_{max}$, which is 0.05 units more then the average of the difference between L_{max} and T_{min} mentioned in step (iii), and route that traffic through the path $i_{max} \rightarrow x_2 \rightarrow j_{max}$, the total load on the edge $i_{max} \rightarrow x_2$ is 0.70 (0.35 + 0.35) and total load on the edge $i_{max} \rightarrow j_{max}$ is 0.60 (0.95 – 0.35). In this way, the corresponding traffic on the edge $i_{max} \rightarrow x_2$ and $i_{max} \rightarrow j_{max}$ is not uniformly distributed. As a result, the edge $i_{max} \rightarrow x_2$ becomes overloaded than $i_{max} \rightarrow j_{max}$ as shown in Fig. 3.4.3b.

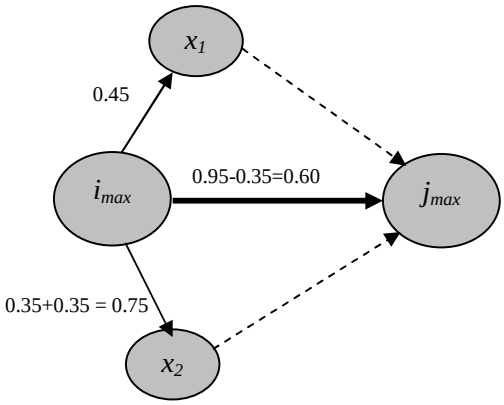

Figure 3.4.3b: Choice 2 - Traffic routing after removing 0.35 units

Therefore, adding an edge from x_2 to j_{max} is more beneficial then adding an edge from x_1 to $j_{max.}$

3.5 Strategy 3

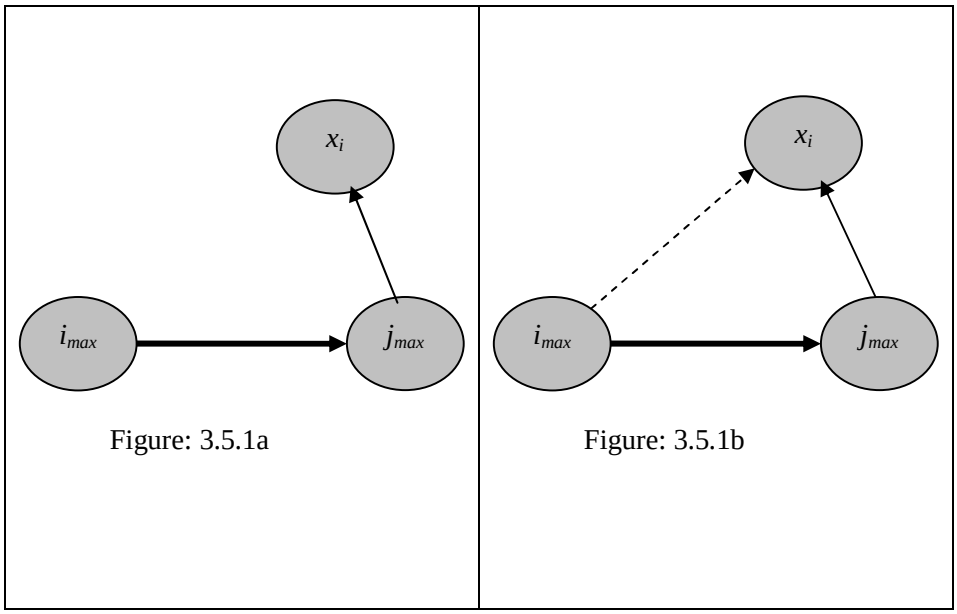

Figure: 3.5.1a

Figure: 3.5.1b

Figure 3.5.1 Adding edge from Strategy 3.

Condition for applying the strategy: This strategy is applicable if there is an edge from node j_{max} *to* x_i but there is no edge from node i_{max} to x_i as shown in Fig. 3.5.1a.

Details of the strategy: Strategy 3 adds an additional edge from node i_{max} to x_i as shown in Fig 3.5.1b.

The rationale for the strategy: Part of the traffic flowing to node x_i, is also likely being routed through the edge $i_{max} \rightarrow j_{max}$. In this case, adding an edge from node i_{max} to x_i will be promising, since the traffic routing through the edge from node i_{max} to j_{max}

could be diverted through the edge $i_{max} \rightarrow x_i$.

Potential benefit for the strategy: The addition of the edge $i_{max} \rightarrow x_i$ could reduce the traffic of the edge $i_{max} \rightarrow j_{max,}$ and as a result it could reduce the congestion of the network. The process of calculating the benefit for this strategy is similar to Strategy 1, discussed above.

3.6 Strategy 4

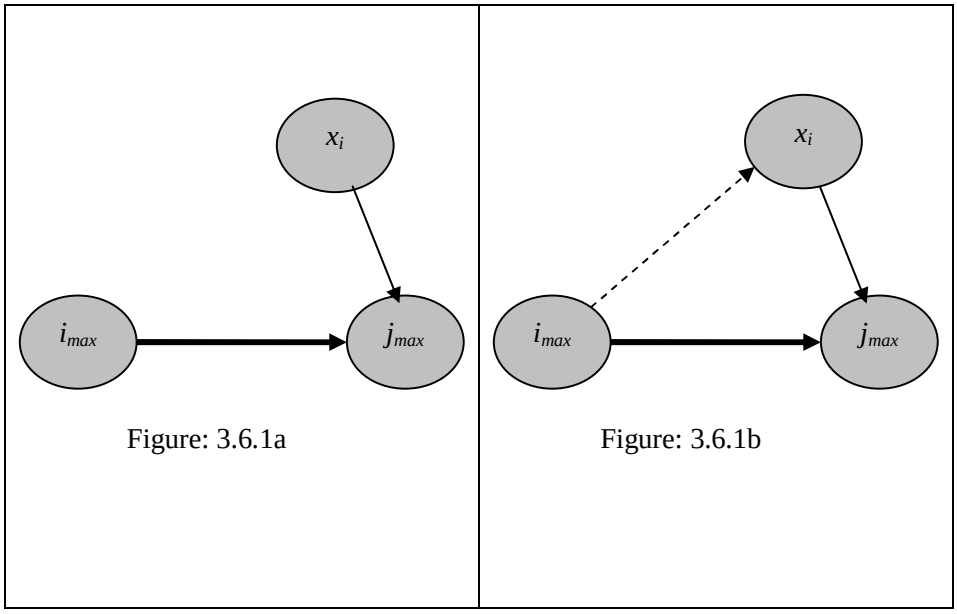

Figure: 3.6.1a

Figure: 3.6.1b

Figure 3.6.1: Adding edge from Strategy 4.

Condition for applying the strategy: This strategy is applicable if there is an edge from node x_i to j_{max} but there is no edge from node i_{max} to x_i as shown in Fig. 3.6.1a.

Details of the strategy: Strategy 4 adds an additional edge from i_{max} to x_i as shown in Fig. 3.6.1b.

The rationale for the strategy: Part of the traffic flowing on the edge $i_{max} \rightarrow j_{max}$

could be diverted using the edge $i_{max} \rightarrow x_i$.

Potential benefit for the strategy: In this case, adding an edge from node i_{max} to x_i will be promising because there will be an alternative path to divert the traffic flowing through the edge $i_{max} \rightarrow j_{max}$ as shown in Fig. 3.6.1b. As a result, it may reduce the congestion of the network. The process of calculating the benefit from this strategy is similar to the Strategy 2, discussed above.

3.7 Overall Scheme with Block Diagram

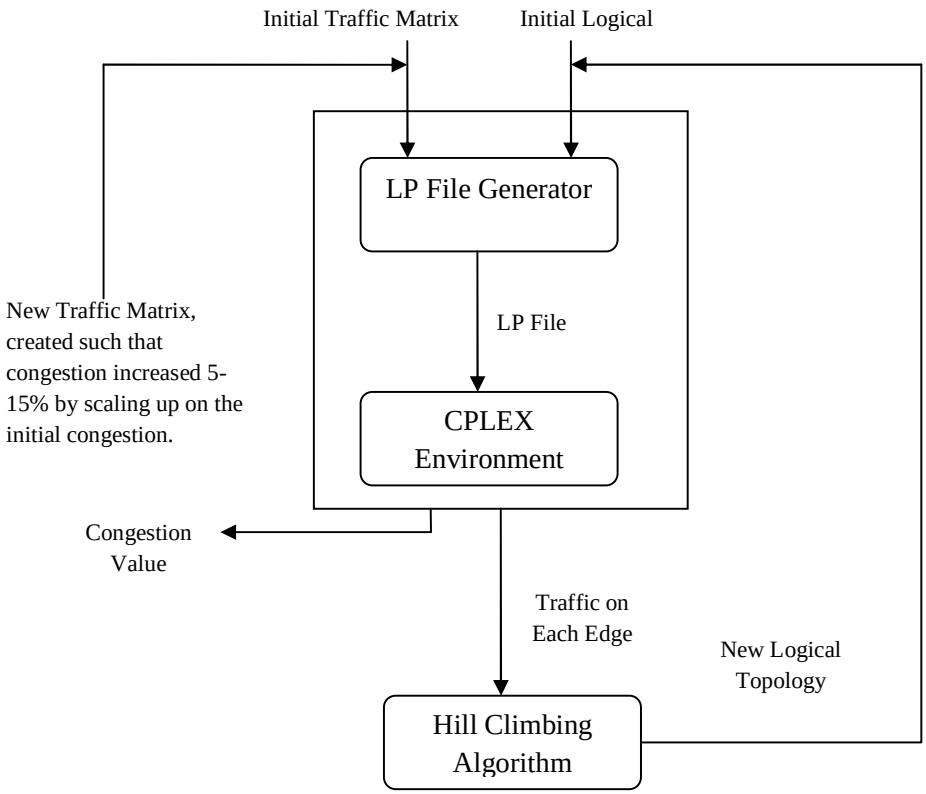

Figure 3.7: Overall Scheme with Block Diagram.

The overall scheme of our work is shown in Figure 3.7, where the input for *LP File Generator* is an initial logical topology which is generated by the heuristic mentioned in section 2.8, and an initial traffic matrix which is randomly generated discussed in Section 2.5. The LP File Generator is a C program which generates a Linear Programming (*LP*) specification file, discussed in section 2.9. The CPLEX[3] environment is called with the *LP* file[4] as an input to route the traffic over the logical topology optimally.

In the first iteration, the CPLEX Optimizer calculates the initial congestion value of the network. Then we create three new traffic matrices such that congestion value increases to 1.05, 1.10 and 1.15 on the initial congestion by applying an appropriate scaling factor to each entry in the traffic matrix. The *lp* file is then converted to a file which shows how much traffic is flowing on an edge for a commodity[5]. This file is an input to the *Hill Climbing* algorithm to generate a new logical topology. The Hill Climbing algorithm is constructed using 4 strategies designed in Section 3.6.

In the subsequent iterations, we run the CPLEX optimizer with new logical topology and the new traffic matrix and observe the *congestion* of the network. This task will be carried out with a number of iteration until the stopping-criterion[6] is met.

---

[3] The CPLEX in an optimization tools, discussed in Chapter 2.

[4] A sample of lp file is discussed in Chapter 4 and also appeared in Appendix 1.

[5] Multiple commodities are flowing on a logical edge; discussed in section 2.5.

[6] The stopping criterion is to reduce the congestion value below 1 in our thesis.

Chapter 4: Implementation Details

The implementation details of our problem of reconfiguration of logical topology have been discussed in this chapter. Various aspects of implementation of our thesis have been pointed out below:

1. We have designed an "initial logical topology" using a heuristic algorithm.
2. Created a fractional traffic matrix and sorted traffic matrix (will be discussed later in this chapter in detail).
3. Designed a CPLEX driver program (Linear Program Specification) for routing the traffic optimally by calling the CPLEX environment.
4. Defined a reconfiguration algorithm based on the Hill-Climbing Search methodology to create new logical topology.

4.1 Design initial logical topology

We have designed an initial logical topology based on a heuristic algorithm as mentioned in Chapter 2. We could have used any heuristic for this purpose and we chose to use the HDL heuristic [19]. We kept the design as simple as possible; we have assumed that all resources such as transmitters or receivers have been used at each end node properly at the time of designing the initial logical topology. The input to this heuristic algorithm is a randomly generated sorted traffic matrix (discussed later in this chapter); sorted in the sense that the largest traffic has to be handled first. The straight forward approach of generating an initial logical topology has been given below:

1. Read sorted traffic request matrix
2. For each request, get the source-destination pair.
3. Using Breadth-first Search algorithm, search for a path in the current logical topology, which has sufficient residual capacity to send the request.
4. If a path is found, update the residual capacity of the path from the source to the destination for that request.
5. If path cannot be found, create a new logical path, consisting of a single logical edge from the source to the destination of the request and update the residual capacity of the path.

The flow diagram shown in Fig. 4.1.1, describes the process of generating initial logical topology.

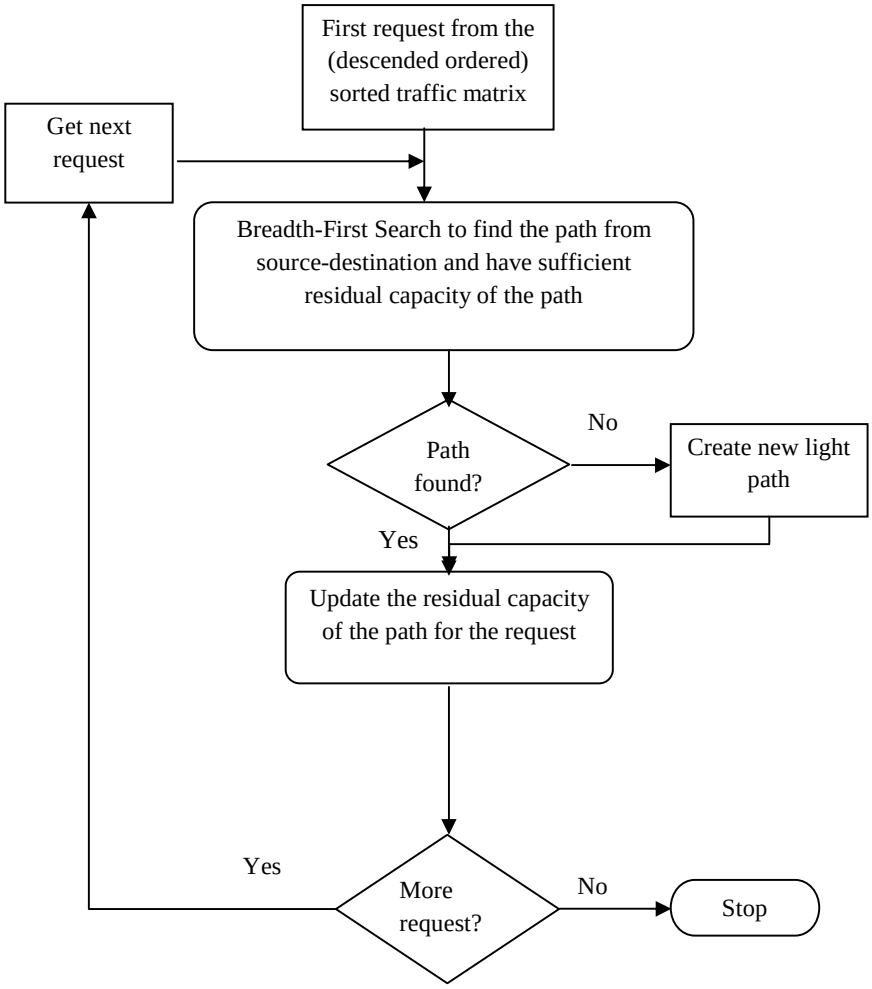

Figure 4.1.1: Generate Initial Logical Topology.

As mentioned, we have used the breadth-first search for finding a path from the source to the destination of a traffic request. A brief algorithm for the Breadth-First search is given below:

1. Enqueue the source node.
2. Search all neighboring nodes from source node
3. Dequeue a node from neighboring node and examine it.
    - If the destination node is found, quit the search and return the path from the source to the destination.
    - Otherwise enqueue any successors nodes that have not yet been visited.
4. If the queue is empty, every node on the networks has been examined – quit the search and return "not found".
5. Repeat from Step 3 otherwise.

4.1.2 An example of creating a 4-node logical topology

We used an N x N matrix to represent the logical topology of a network with N end nodes. In this representation, the element in row i and column j of the matrix, is either a 0 or a 1. If the element in row i and column j *is* 1(0), it means that there is a (no) logical edge from end node i to end node j. When the heuristic for creating a logical topology starts, there is no logical edge and so the entries in the matrix are all 0's. For instance, a network with 4 end-nodes may be represented by a matrix of size 4×4 with suitable elements. An initial 4×4 matrix M representing a logical topology and a traffic matrix T is shown in Table 4.1.2.1 and Table 4.1.2.2 is respectively. M is initialized with all zeros since there is no logical edge initially. The

traffic request matrix *T* is randomly generated and then sorted in descending order, according to the amount of traffic to be communicated.

In the following example, the very first request is to send 0.5 units of traffic from source node 0 to destination node 2; second request is 22 traffic units from source node 1 to destination node 0 and so on.

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Table 4.1.2.1: 4 Nodes logical topology (Initialize with zeros)

Source	Destination	Traffic
0	2	0.5
1	0	0.4583
1	3	0.4583
2	0	0.4375
2	3	0.3125
3	2	0.3125
3	0	0.2292
0	1	0.2083
1	2	0.2083
2	1	0.2083
3	1	0.1042
0	3	0.0833

Table 4.1.2.2: Sorted Traffic Request Matrix

Step1: According to the algorithm, we take the first request, which is to send 0.5 units of data. The breadth-first search looks for a path from 0 to 2, which is sufficient to carry 0.5 units. In this case, there is no path. So we add a new logical edge from 0 to 2. Now, we have a path from 0 to 2 with a capacity of 1 units[7]. We draw a lightpath from the source node 0 to the destination 2 (0 → 2) shown in Fig. 4.1.2.1. In Table 4.1.2.3 shows that 1 has been entered in row 0, column 2, meaning there is a logical edge from source node 0 to destination node 2. Table 4.1.2.4, shows the all the updates.

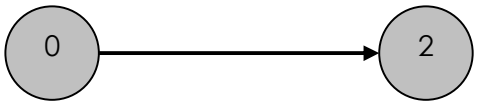

Figure 4.1.2.1: Graph diagram of Logical Topology with one Logical Edge

0	0	**1**	0
0	0	0	0
0	0	0	0
0	0	0	0

Table 4.1.2.3: Logical Topology Matrix

Source	Destination	Lightpath	Current Load	Residual Capacity
0	2	0 → 2	0.5	1.0 – 0.5 = 0.5

Table 4.1.2.4: Updates of Residual Lightpath Capacity, Step1

[7] We assume, the capacity of a logical edge is OC-48.

Step2: Take the next request, which is 0.4583, and source-destination pair is (1, 0), there is no path from node 1 to 0; we draw a new lightpath as show in Fig. 4.1.2.2, update of traffic matrix and the residual lightpath capacity for the edge 1 to 0 is 0.5417 (1.0 – 0.4583) as shown in Table 4.1.2.5 and Table 4.1.2.6 respectively.

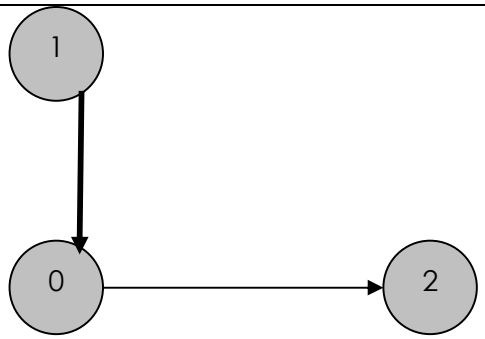

Figure 4.1.2.2: Graph diagram of Logical Topology with two Edges

0	0	1	0
1	0	0	0
0	0	0	0
0	0	0	0

Table 4.1.2.5: Logical Topology Matrix

Source	Destination	Lightpath Flow	Current Load	Residual Capacity
0	2	0 → 2	0.5	1.0 – 0.5 = 0.5
1	0	1 → 0	0.4583	1.0 –0.4583 = 0.5417

Table 4.1.2.6: Updates of Residual Lightpath Capacity, Step2

Similarly, if no logical path exists for a source-destination pair of a request; a new path will be created following the procedure mentioned in Step 1 and Step 2. The next four such requests and their source-destination pairs are shown in the table 4.1.2.7 in our example is taken from table 4.1.2.1. The request 0.4583 is routing from node 1 to node 3, request 0.4375 routing from node 2 to node 0 and so on.

1	3	0.4583
2	0	0.4375
2	3	0.3125
3	2	0.3125

Table 4.1.2.7: Source-Destination pair of requests

After handling the four requests, the graph diagram and the traffic matrix update are shown in Fig. 4.1.2.3 and Table 4.1.2.8 respectively. The updates of residual lightpath capacity are shown in Table 4.1.2.9.

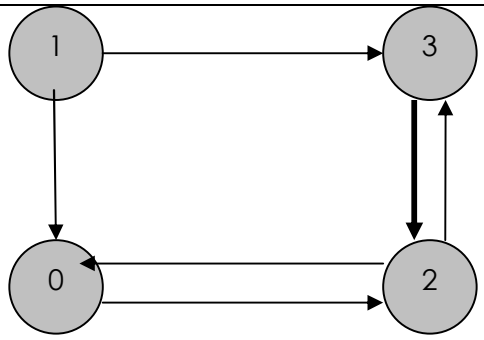

Figure 4.1.2.3: Graph diagram of Logical Topology with 6 Edges.

0	0	1	0
1	0	0	1
1	0	0	1
0	0	**1**	0

Table 4.1.2.8: Updated Logical Topology

Source	Destination	Lightpath Flow	Current Load	Residual Capacity
0	2	0 → 2	0.5	1.0 – 0.5 = 0.5
1	0	1 → 0	0.4583	1.0 –0.4583 = 0.5417
1	3	1 → 3	0.4583	1.0 –0.4583 = 0.5417
2	0	2 → 0	0.4375	1.0 – 0.4375 = 0.5625
2	3	2 → 3	0.3125	1.0 – 0.3125 = 0.6875
3	2	3 → 2	0.3125	1.0 – 0.3125 = 0.6875

Table 4.1.2.9: Updates of Residual Lightpath Capacity

Step 3: In this step, I have demonstrated the procedure where a logical path exists for a source-destination pair of a request. In this case, if the residual capacity of the existing path is sufficient to handle the request, it will use that path instead of creating a new link (path). For example, from the source node 3 to the destination node 0, the request 0.2292 units of data to communicate as shown in table 4.1.2.2. In this case since there is a logical lightpath (3 → 2 →0) as can be seen from Fig. 4.1.2.3, it does not need a new lightpath from the source node 3 to the destination node 0. Since, the residual capacity of lightpath 3 → 2 is 0.6875, and for the lightpath 2 → 0 is 0.5625, it is sufficient to send the request (0.2292) using the path 3 → 2 →0 as shown in the Table 4.1.2.10 in bold-italic text.

Source	Destination	Lightpath Flow	Current Load	Remaining Capacity
0	2	0 → 2	0.5	1.0 – 0.5 = 0.5
1	0	1 → 0	0.4583	1.0 – 0.4583 = 0.5417
1	3	1 → 3	0.4583	1.0 – 0.4583 = 0.5417
2	0	***2 → 0***	0.4375 + 0.2292	1.0 – 0.6667 = 0.3333
2	3	2 → 3	0.3125	1.0 – 0.3125 = 0.6875
3	2	***3 → 2***	0.3125 + 0.2292	1.0 – 0.5417 = 0.4583
3	0	3 → 2 → 0	Xxx	Xxx

Table 4.1.2.10: Update of Residual Lightpath Capacity

Similarly, after handling the entire set of requests, using this heuristic, the following logical topology and the corresponding logical topology matrix is generated as shown in Fig. 4.1.2.4 and Table 4.1.2.11 respectively.

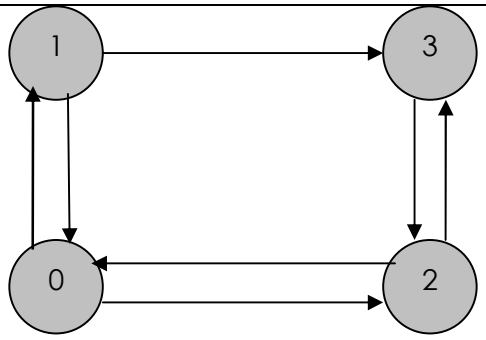

Figure 4.1.2.4 Graph diagram of Logical Topology with 7 Edges.

0	1	1	0
1	0	0	1
1	0	0	1
0	0	1	0

Table 4.1.2.11: Logical Topology Matrix.

4.2 Generating a traffic matrix

The traffic matrix is an *N x N* matrix, where *N* is the number of end nodes in the networks. The matrix entries are randomly generated using the native C compiler. Table 4.2.1 is an example which is generated for the network with 4 end nodes using the random number generator. The diagonal entries are filled with 0's, since there cannot be any traffic from a node to itself. For example, the entry for source-destination pair (0, 3) is 0.083333, which means that the source node 0 is sending request (0.083333) units of data to the destination end node 3. Here the unit is the capacity of a lightpath.

0.000000	0.208333	0.500000	0.083333
0.458333	0.000000	0.208333	0.458333
0.437500	0.208333	0.000000	0.312500
0.229167	0.104167	0.312500	0.000000

Table 4.2.1: Randomly Generated Traffic Matrix

The rationale for generating a fractional traffic matrix is discussed in Chapter 2. Since the lightpath capacity has been fixed to OC-48 in our experiments, as mentioned in Section 4.1, we have generated traffic in between OC-1 to OC-48 randomly and then divided each value by 48, which gives us entry with fraction number in between 1/48 to 48/48. Below are the straight forward steps for generating traffic matrix.

1. Initialize each cell of matrix *M* with zeros.

2. For each entry of *N x N* matrix *M*, generate a random number n_i in between 1 to 48.

3. Divide n_1 by edge-capacity (Edge-capacity = 48 OC).

4. Fill each entry with the value which is got from Step2, except the diagonal entries.

4.2.1 A sorted traffic matrix

We have also generated the sorted traffic matrix as shown in Table 4.1.2.2, using the native C compiler. The sorted traffic matrix has been used to generate the initial logical topology as mentioned in section 4.1. As we have mentioned in our problem definition in Chapter 2, the initial logical topology is needed to find the initial congestion for a given traffic. The rationale for generating the sorted traffic matrix in descending order is that we wish to handle the largest traffic first. For the smaller requests, we may not need to create a new lightpath if the request can be send by some existing logical path.

4.3 Determining the Congestion

We have used the CPLEX optimizer program to route the traffic optimally. The input to the CPLEX optimizer is a file specifying the Linear Program (*LP*). The LP file is generated using a logical topology matrix file and a traffic matrix file. The CPLEX optimizer program routes the traffic through the logical topology and calculates the routing strategy within a short period of time for a small or medium size of networks. The routine *CPXNETgetobj* function is used to access the objective function values in the network. The routine returns a zero when it is successful and a nonzero value if an error occurs. The main predefined function of CPLEX used in our

problem is described briefly below:

1. The routine *CPXopenCPLEX* function initializes a CPLEX environment. The routine returns a pointer to a CPLEX environment.
2. Next the *CPXcreateprob* function is called to create the problem. The routine CPXcreateprob() creates a CPLEX problem object in the CPLEX environment. The arguments to CPXcreateprob() define an LP problem name. The problem that is created is an LP minimization problem with zero constraints, zero variables, and an empty constraint matrix. The CPLEX problem object exists until the routine CPXfreeprob () is called. If successful, CPXcreateprob () returns a pointer that can be passed to other CPLEX routines to identify the problem object that is created. If not successful, a NULL pointer is returned.
3. The routine CPXreadcopyprob () reads an LP file into an existing LP problem object. The problem can then be optimized using any one of the optimization routines. The routine returns a zero on success, and a nonzero if an error occurs.
4. The routine CPXlpopt () is called after a linear program has been created via a call to CPXcreateprob (), to find a solution to that problem using one of CPLEX's linear optimizers.
5. The routine CPXgetstat () is used to access the solution status of the problem after an LP or mixed integer optimization. If no solution exists, CPXgetstat () returns the value 0.
6. The routine *CPXNETgetobj* function is used to access the objective function values in the network stored in a network problem object. The routine returns a zero on success, and a nonzero if an error occurs.

4.4 Creating a new Logical Topology using the Hill Climbing Search Technique

The overall process of creating a new logical topology using the Hill-Climbing search technique is given in Figure 4.4a. First, we have created the move[8] list from four different strategies. Each strategy generates one best move by searching all potential moves in the neighbourhood of the current move and calculates how much benefit may be achieved from it. As a result, there are 4 moves generated from four strategies. Finally, we calculated one best move from the 4 moves, based on the move which has the highest benefit. We implemented the best move to the current logical topology to create a new logical topology. The flow diagram is shown in Fig. 4.4a. The details of each process are discussed later in this chapter.

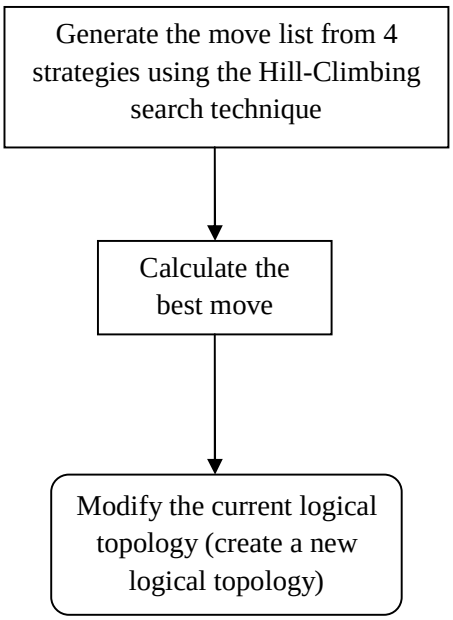

Figure 4.4a: Generating new Logical Topology.

[8] The term "move" has been discussed in Chapter 3.

4.4.1. Implementation details and calculating the best move from Strategy 1

The details of designing Strategy 1 have been discussed in Chapter 3, Section 3.3. In this section, the implementation of this strategy and calculating the best move is discussed in following subsections:

- Implementation of Strategy 1.
- Implementation of determining the best move in Strategy 1.
- Implementation of calculating benefit for a move (A logical edge).

4.4.1.1. Implementation of Strategy 1

To implement Strategy 1, we have executed the steps shown in Fig. 4.4.1.1a.

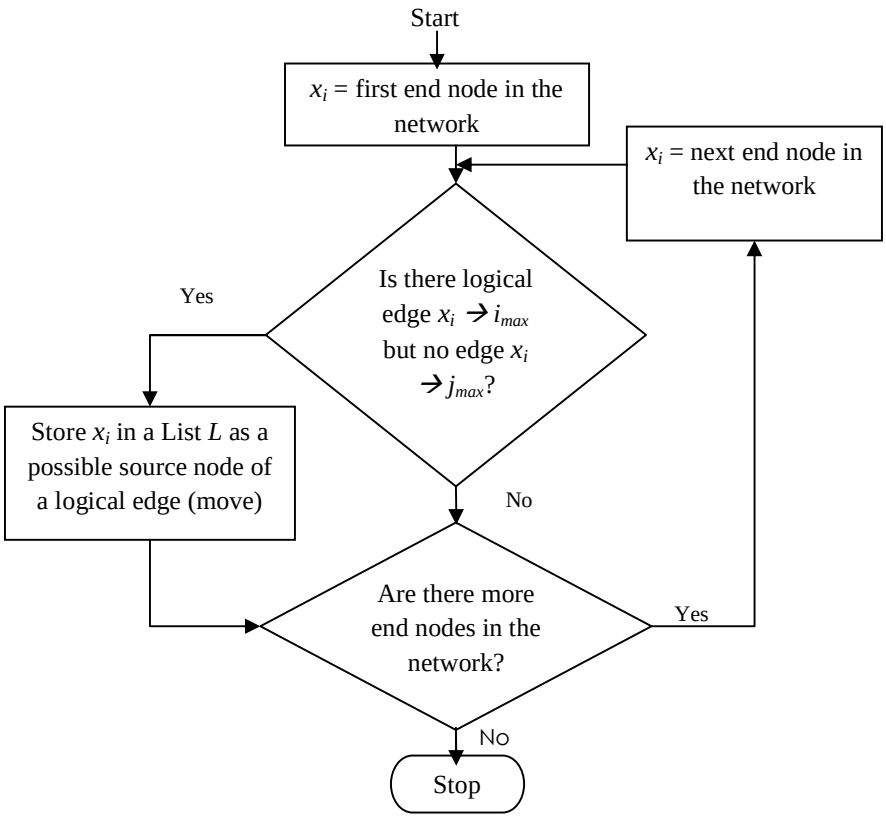

Figure 4.4.1.1a: Implementation details of Strategy 1.

In this process, our objective is to create the list of incoming end nodes, such that if there is an edge from the end node x_i to i_{max} but there no edge from the x_i node to j_{max} as shown in Fig.3.3.1 in Chapter 3. We assume, the highest traffic flow is taking place on edge $i_{max} \rightarrow j_{max}$ in this particular network.

4.4.1.2 Implementation details of determining the best move in Strategy 1

For determining the best move from all potential moves from Strategy 1, we implement the process shown in Fig. 4.4.1.2a. Due to the lack of space, we use T(R) to denote transmitter (receiver) in this diagram.

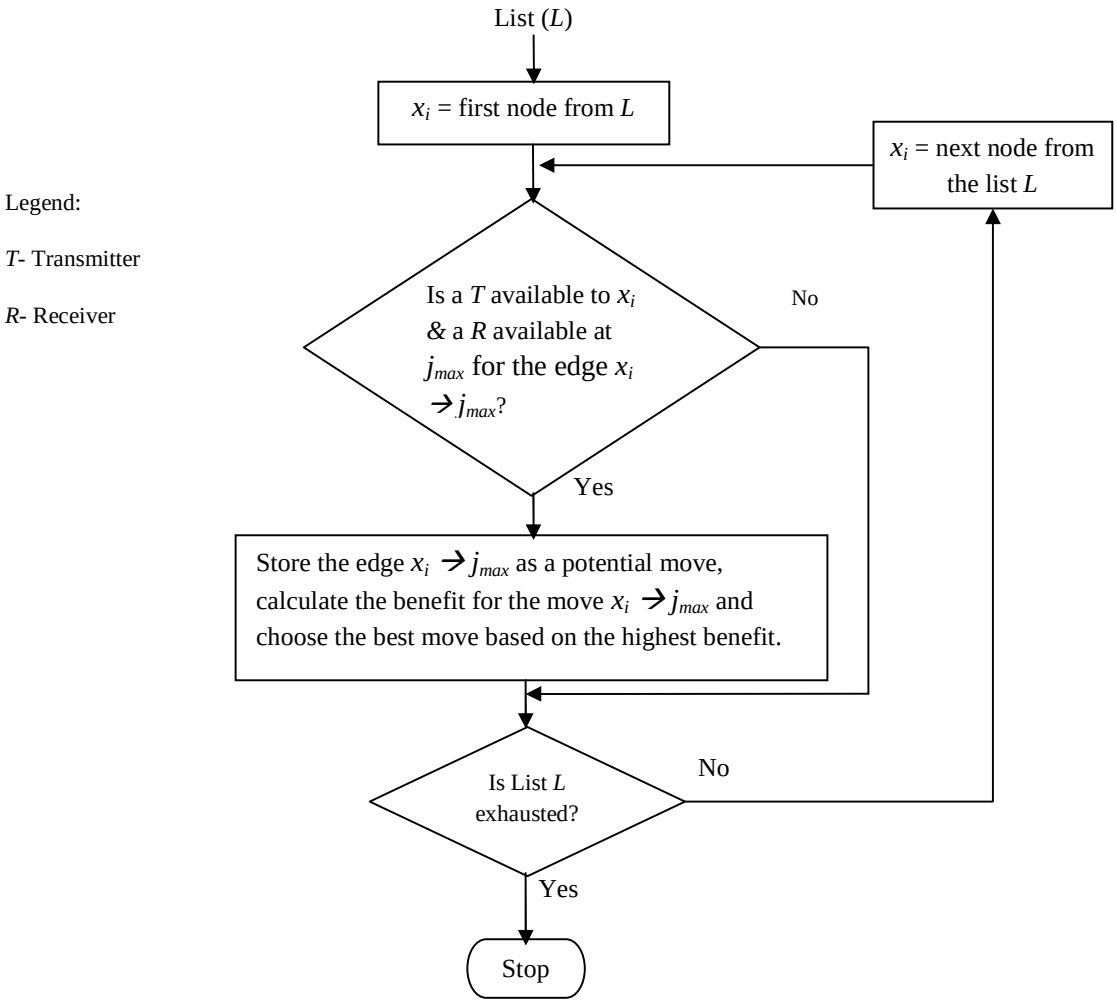

Figure 4.4.1.2a: Implementation of determining best move in Strategy1.

We checked each node x_i from the list L, such that there is a spare transmitter at the node x_i and a spare receiver at the node j_{max} available. The list L is created from the previous section 4.4.1.1. We stored the edge $x_i \rightarrow j_{max}$ considering a potential move. Then we calculated the benefit[9] for each such potential move (edge). Finally, we considered a move which has the highest benefit.

[9] The benefit calculation is shown later in the Section 4.4.1.3.

4.4.1.3. Implementation for calculating the benefit of a move in Strategy 1

The process of implementation of the scheme for calculating the benefit of the move $x_2 \rightarrow i_{max}$ is shown in Fig. 4.4.1.3a.

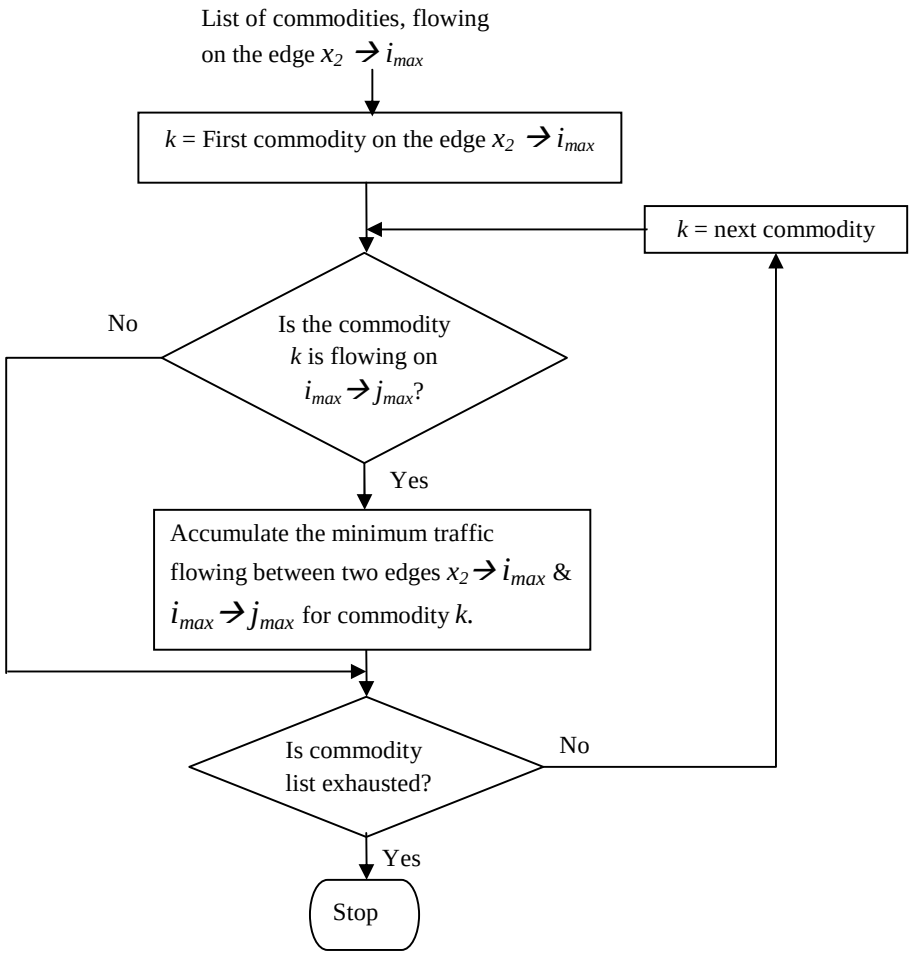

Figure 4.4.1.3a: Implementation for calculating the benefit of a move in Strategy 1.

For our discussions below, we have taken an example when discussing the strategies we used. As we have mentioned in Section 2.9, routing the traffic over a

logical topology is a multi-commodity flow problem. An edge could have multiple commodities flowing for a particular routing scheme. There are two edges $x_1 \rightarrow i_{max}$ and $x_2 \rightarrow i_{max}$ to the node i_{max} as shown in Fig.4.4.1.3b. We calculate the benefit for all such incoming edges to end node i_{max} and finally we consider the edge which has the highest benefit. For example, we consider the edge $x_2 \rightarrow j_{max}$ as a move for this particular network shown in bold dashed line. Since we consider the edge $x_2 \rightarrow j_{max}$ as a potential move for this particular network, a part of the commodities flowing on the edge $x_2 \rightarrow i_{max}$ are routed through the path $x_2 \rightarrow i_{max} \rightarrow j_{max}$. Therefore, we show the calculation of the benefit for the edge $x_2 \rightarrow i_{max}$.

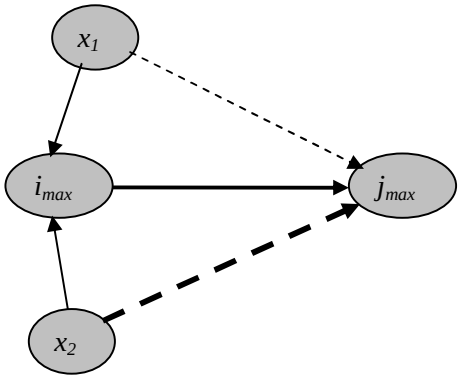

Figure 4.4.1.3b: Benefit calculation for the edge $x_2 \rightarrow i_{max}$ in Strategy 1

In this implementation process, we assume that a commodity *k* is flowing on the edge $x_2 \rightarrow i_{max}$. If the same commodity *k also* flows on the edge $i_{max} \rightarrow j_{max}$, then we calculate the minimum traffic value between the two edges $x_2 \rightarrow i_{max}$ and $i_{max} \rightarrow j_{max}$ for the commodity *k*. We calculated the minimum traffic value for all the other commodities that is flowing between the two edges and accumulates by adding them up. As a result, the accumulated value is the maximum possible benefit for the edge $x_2 \rightarrow i_{max}$. An example for calculating the benefit with a data flow and rationale for choosing the minimum traffic flow for a commodity has been shown in Chapter 3.

4.4.2 Implementation details and calculating the best move from Strategy 2

The details of designing Strategy 2 have been discussed in Section 3.4. In this section, the implementation of this strategy and calculating the best move is discussed in following subsections.

- Implementation of Strategy 2.
- Implementation of determining the potential moves.
- Implementation of selecting the best move and calculate the benefit.

4.4.2.1: Implementation of Strategy 2

To implement Strategy 2, we have executed the steps shown in Fig. 4.4.2.1a.

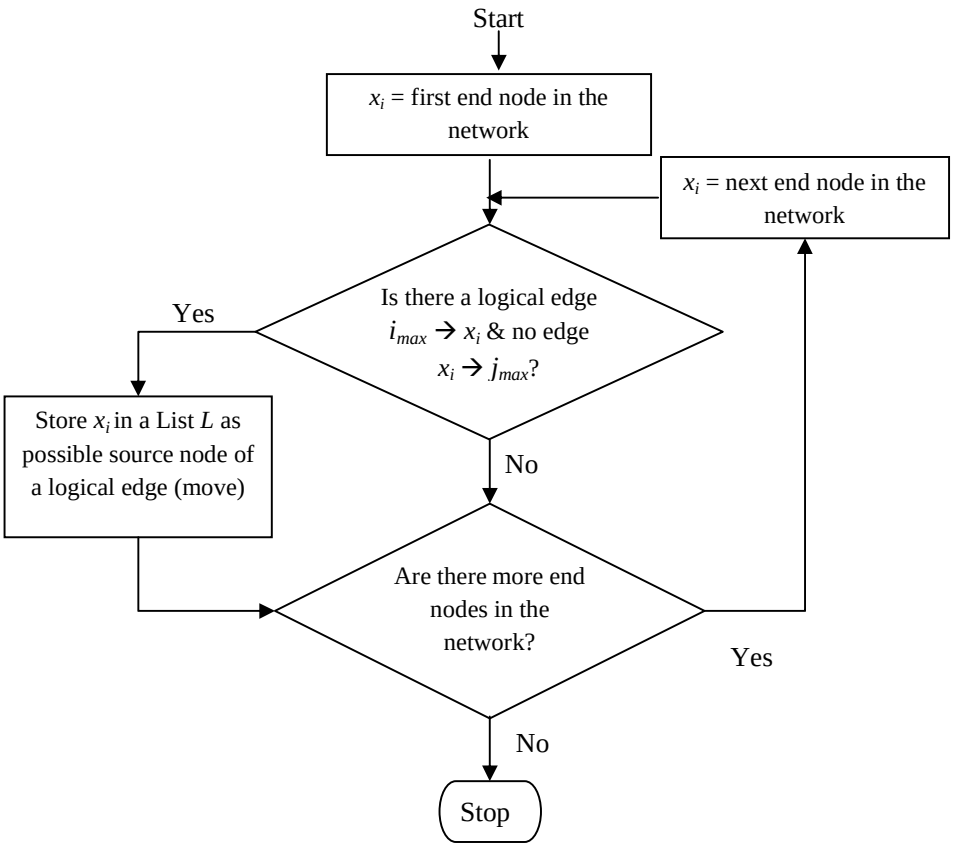

Figure 4.4.2.1a: Implementation details of Strategy 2

In this process, our objective is to create the list of end nodes x_i, such that there is an edge from node i_{max} to end node x_i but no edge from node x_i to j_{max} as shown in Fig. 3.4.1 in Chapter 3. We assume that the highest traffic flow is taking place on the edge $i_{max} \rightarrow j_{max}$ in the network.

4.4.2.2. Implementation detail of determining the potential moves from Strategy 2

For determining the potential moves using Strategy 2, we implemented the process shown in Fig. 4.4.2.2a. Due to the lack of space, we use T (R) to denote Transmitter (Receiver) in this diagram.

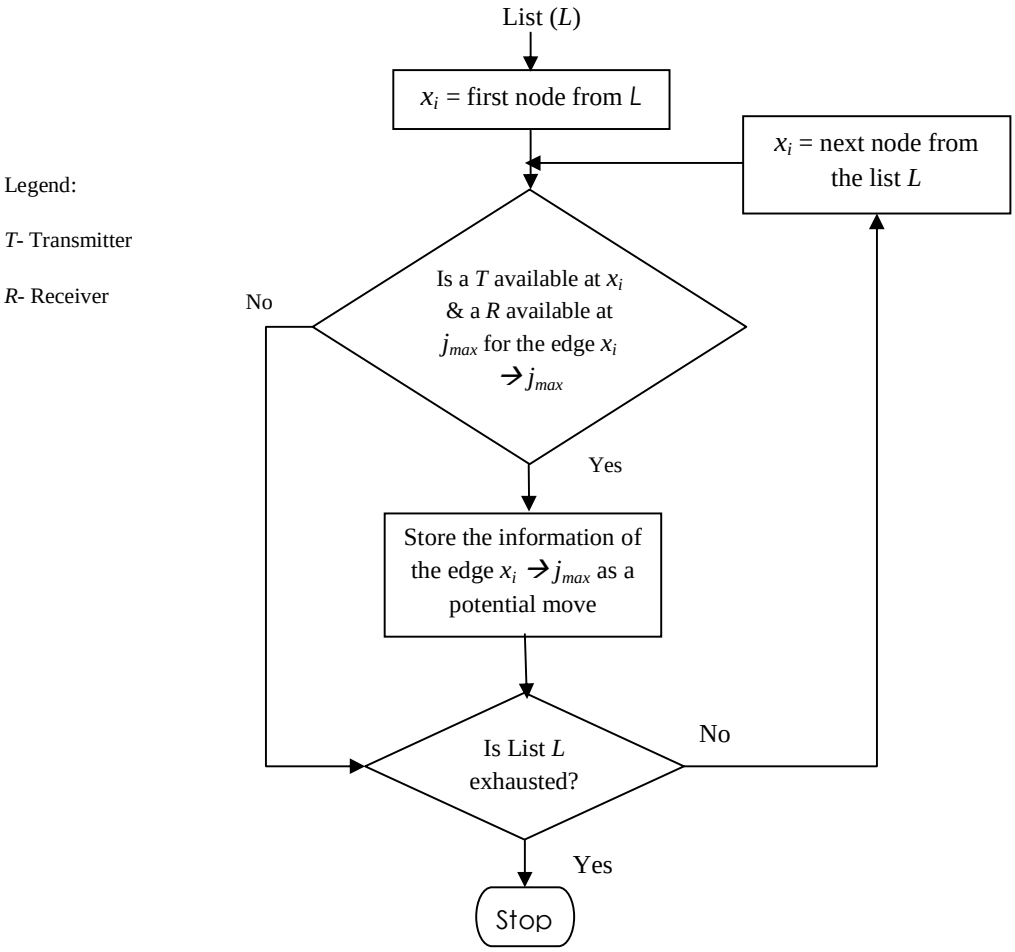

Figure 4.4.2.2a: Implementation of determining best move in Strategy 2.

We stored the information of the edge $x_i \rightarrow j_{max}$ considering a potential move if there is a spare transmitter at node x_i and a spare receiver at node j_{max} available of the edge $x_i \rightarrow j_{max}$. Node x_i is a member of list L which is created from procedure

outlined in Section 4.4.2.1. In this process, we checked all such moves and calculated the benefit for each move. In the next section, we show how we calculate the benefit for selecting the best potential move.

4.4.2.3. Implementation of selecting the best move in Strategy 2

The details of how we selected the best move by calculating the benefit is shown in Fig. 4.4.2.3a.

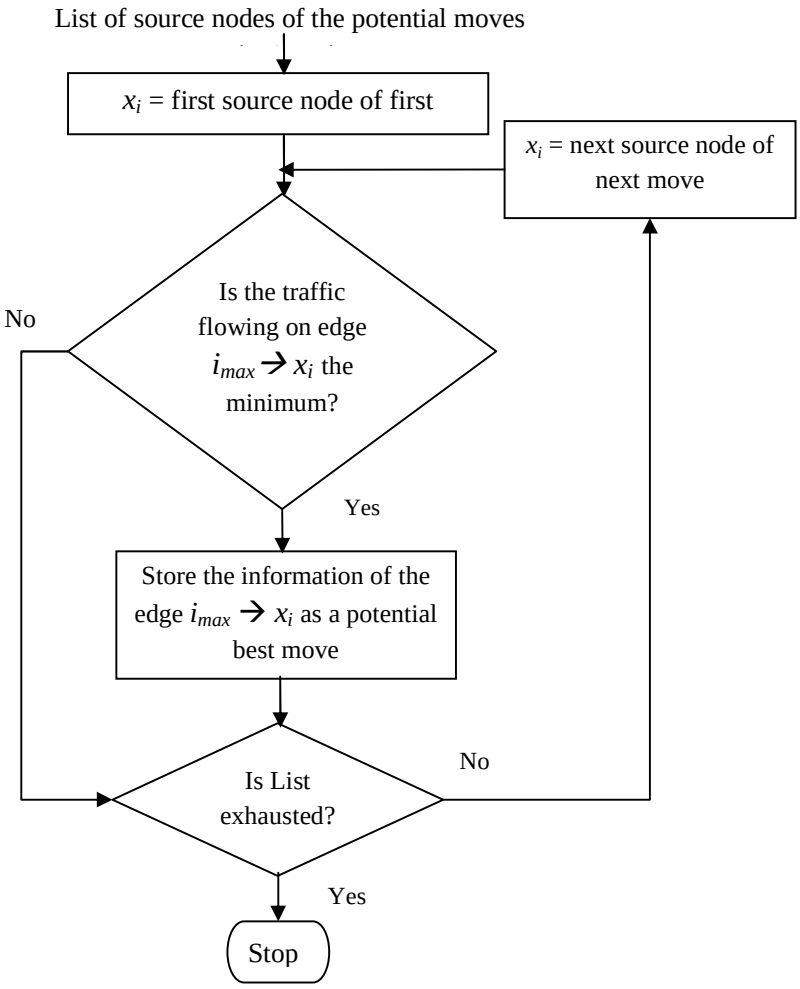

Figure 4.4.2.3a: Implementation of selecting the best move in Strategy 2.

An example of selecting the best move is shown in Fig. 4.4.2.3b. There are two potential moves ($x_1 \rightarrow j_{max}$ and $x_2 \rightarrow j_{max}$) in this particular example. The node x_1 and x_2 are two source nodes of the two moves.

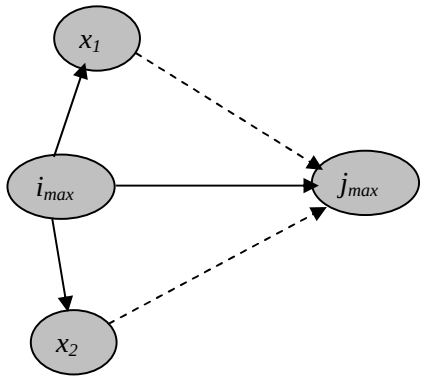

Figure 4.4.2.3b: Selecting the best move in Strategy 2.

In order to choose the best move, we checked edge $i_{max} \rightarrow x_1$ and $i_{max} \rightarrow x_2$. Each edge is carrying a certain amount of traffic for different commodities. We selected the edge which is carrying the least amount of traffic as the best move. For example, if edge $i_{max} \rightarrow x_1$ is carrying a smaller amount of traffic than edge $i_{max} \rightarrow x_2$, then we select the edge $x_1 \rightarrow j_{max}$ as the best move in this particular network. The rationale for choosing the edge which carrying least amount of traffic is discussed with an example in Chapter 3, Section 3.4.

4.4.3. Implementation details and calculating the best move from Strategy 3

The details of designing Strategy 3 have been discussed in Section 3.5. In this section, we describe the following issues for implementing of this strategy and calculating the best move:

- Implementation of Strategy 3
- Implementation of determining the best move in Strategy 3
- Implementation of calculating benefit for a move (A logical edge)

4.4.3.1. Implementation of Strategy 3

To implement Strategy 3, we have executed the steps shown in Fig. 4.4.3.1a.

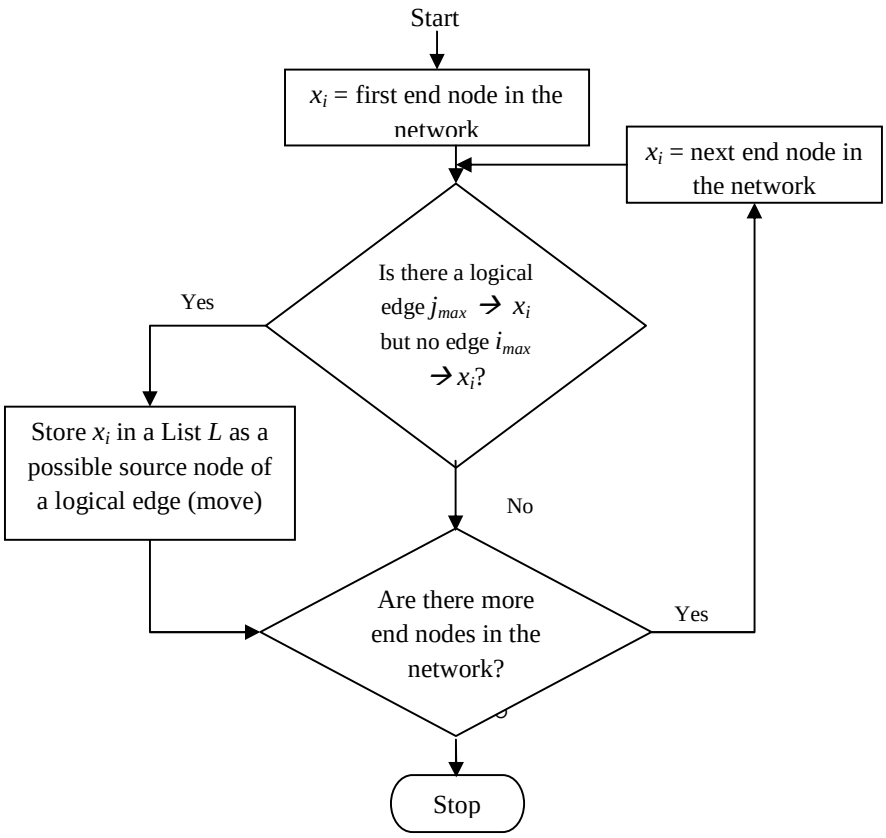

Figure 4.4.3.1a: Implementation details of Strategy 3.

In this process, our objective is to create the list of outgoing end nodes from j_{max}, such that if there is an edge from the end node x_i to i_{max} but there no edge from the x_i node to j_{max} as shown in Fig.3.5.1 in Chapter 3. We assume, the highest traffic flow is taking place on edge $i_{max} \rightarrow j_{max}$ in this particular network.

4.4.3.2 Implementation details of determining the potential moves in Strategy 3

For determining the best move from all potential moves from Strategy 3, we implemented the process shown in Fig. 4.4.3.2a. Due to lack of space, we used T(R) to denote a transmitter (receiver) in this diagram.

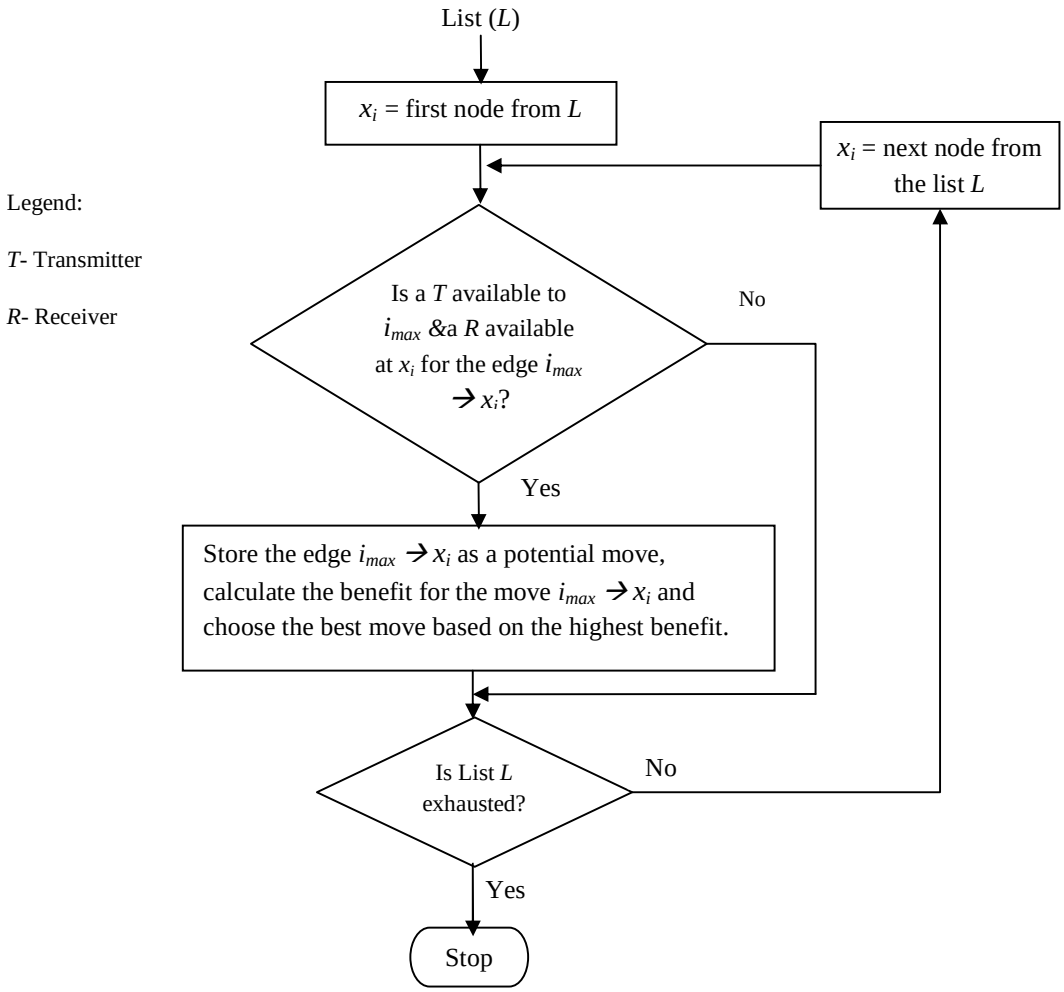

Figure 4.4.3.2a: Implementation of determining best move in Strategy 3.

We considered adding the edge $i_{max} \rightarrow x_i$ as a potential move, if a spare transmitter at node i_{max} and a spare receiver at node x_i available. A list L is created

with node x_i in Section 4.4.3.1. We calculated the benefit for each potential move (edge). Finally, we considered the move from which the highest benefit that can be achieved.

4.4.3.3. Implementation of the scheme for calculating the benefit of a move in Strategy 3

The process of calculating the benefit of the move $x_1 \rightarrow i_{max}$ (Fig. 4.4.3.3b) is shown using the flow diagram shown in Fig. 4.4.3.3a.

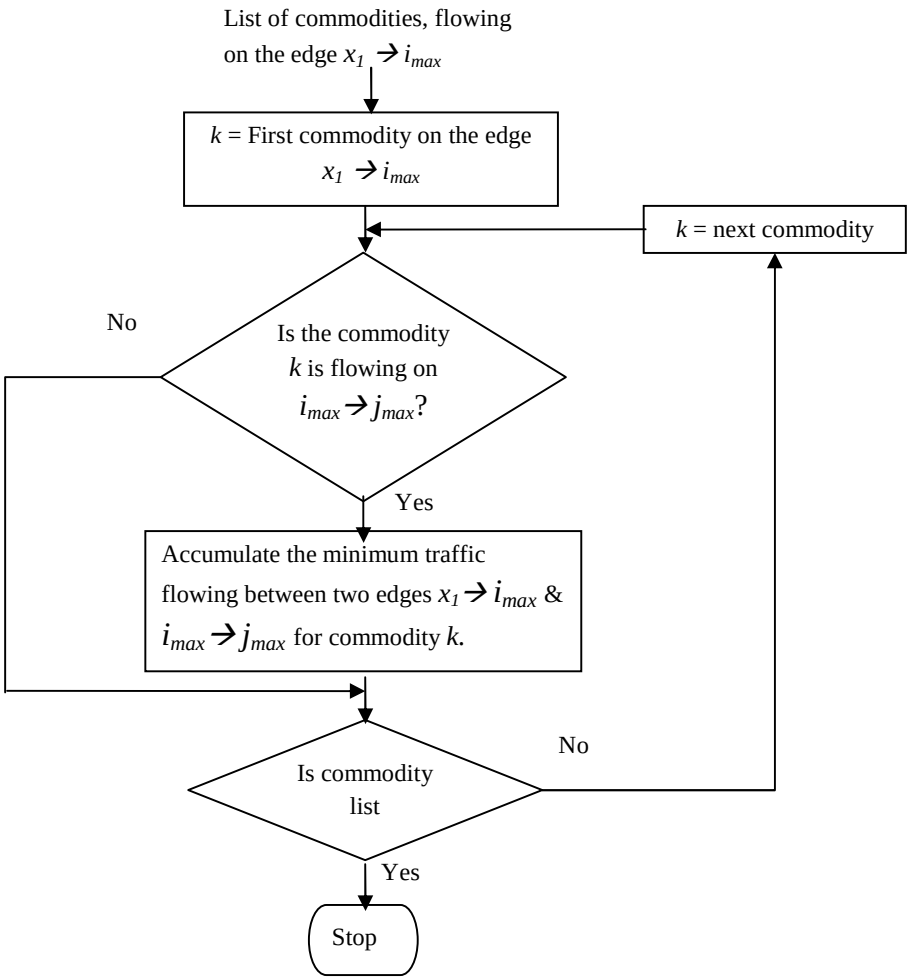

Figure 4.4.3.3a: Calculating the benefit of a move using Strategy 3.

As we have mentioned when describing Strategy 1, routing the traffic over a logical topology is a multi-commodity flow problem and an edge could have multiple commodities flowing for a particular routing scheme. There are two edges $j_{max} \rightarrow x_1$ and $j_{max} \rightarrow x_2$ from the node j_{max} as shown in Fig.4.4.3.3b. We calculate the benefits for all such edges that are outgoing from end node j_{max} and finally we considered the edge which has the highest benefit. For example, we considered the edge $i_{max} \rightarrow x_1$ as a move for this particular network shown using a bold dashed line. Since, we considered the end node x_1 is a potential destination of commodities for this particular network; the commodities are possibly being routed through the path $i_{max} \rightarrow j_{max} \rightarrow x_1$.

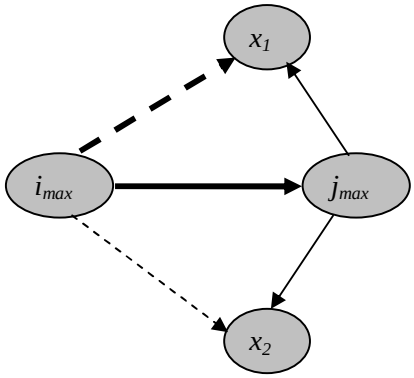

Figure 4.4.3.3b: Benefit calculation for the edge $j_{max} \rightarrow x_1$ in Strategy 3.

The process for calculating the benefit for the edge $j_{max} \rightarrow x_1$ follows an approach similar to Strategy 1 discussed above. An example for calculating the benefit with a data flow has been shown in Chapter 3.

4.4.4 Implementation details and calculating the best move from Strategy 4

The details of designing Strategy 4 have been discussed in Section 3.6. In this section, the implementation of this strategy and calculating the best move is discussed in the following subsections:

- Implementation of Strategy 4.
- Implementation of determining the potential moves.
- Implementation of selecting the best move and calculate the benefit.

4.4.4.1: Implementation of Strategy 4

To implement Strategy 4, we have executed the steps shown in Fig. 4.4.4.1a.

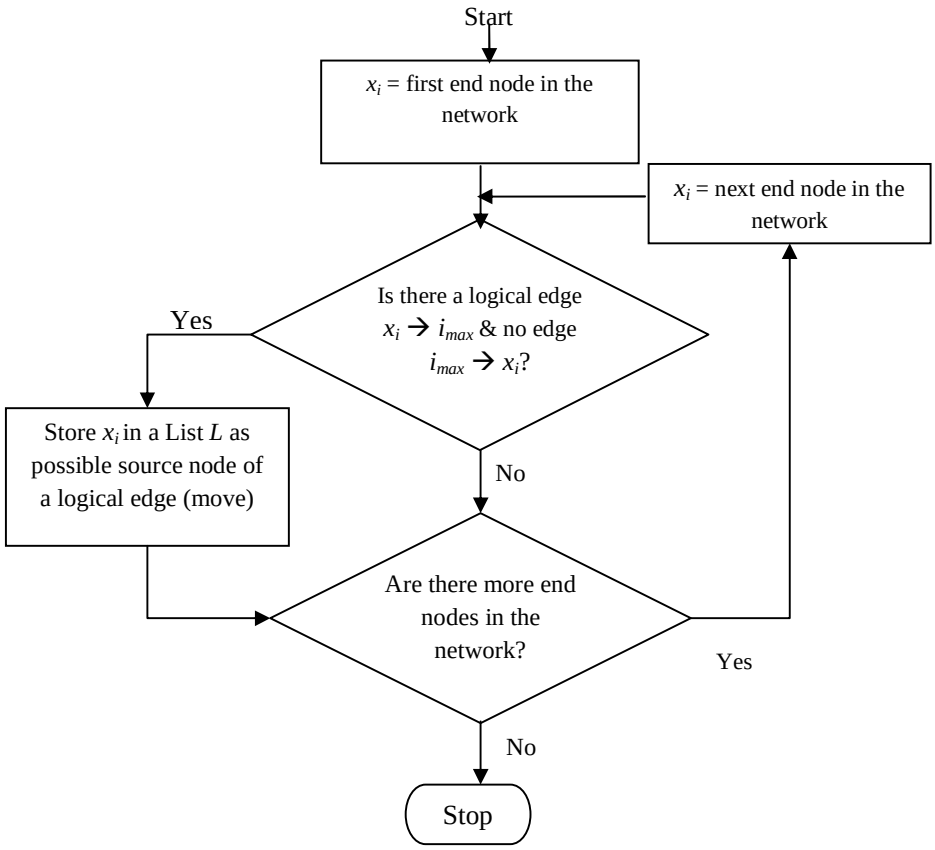

Figure 4.4.4.1a: Implementation details of Strategy 4.

In this process, our objective was to create the list of end nodes, such that there is an edge from node x_i to end node j_{max} but there is no edge from node i_{max} to x_i (Fig. 3.6.1 in Chapter 3). We assumed that the highest traffic flow is on the edge $i_{max} \rightarrow j_{max}$ in the network.

4.4.4.2. Determining the potential moves from Strategy 4

To determine the potential moves from Strategy 4, we implemented the process shown in Fig. 4.4.4.2a. Due to lack of space, we have used T (R) to denote a Transmitter (Receiver) in this diagram.

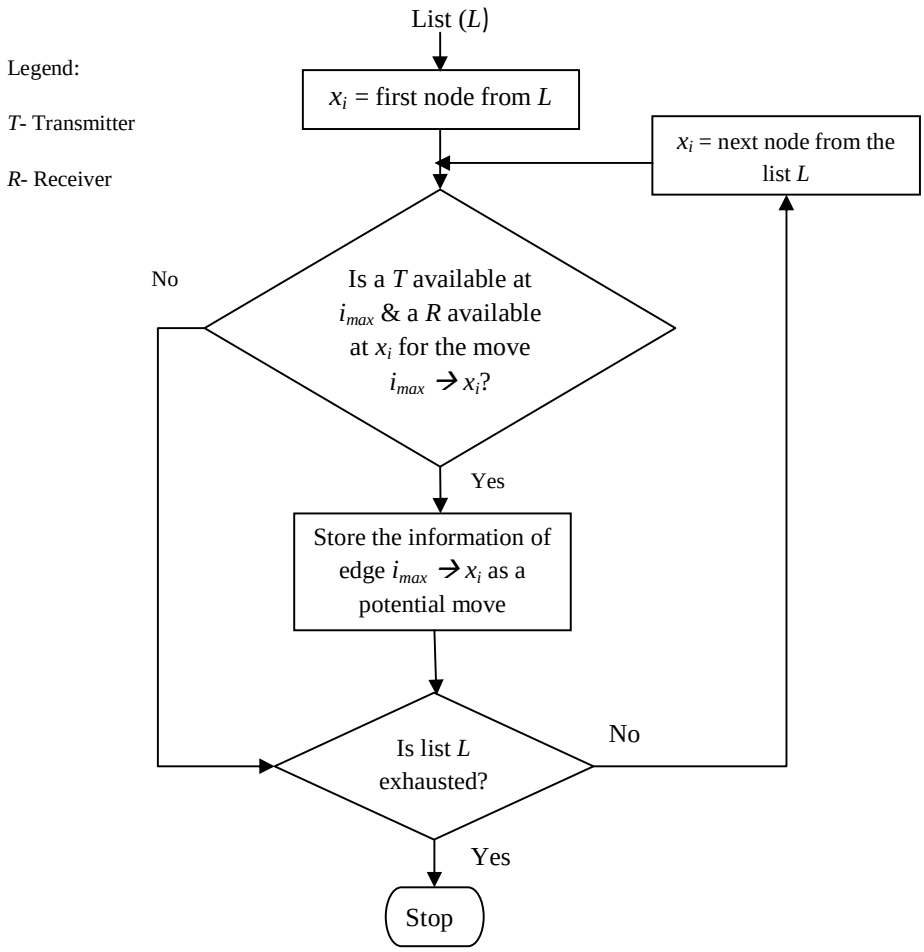

Figure 4.4.4.2a: Implementation of determining potential moves in Strategy 4.

We stored the information regarding the edge $i_{max} \rightarrow x_i$ after considering a potential move, if there is a spare transmitter at the node i_{max} and a spare receiver

available at the node x_i of the edge $i_{max} \rightarrow x_i$. The node x_i is a member of list L which is created from Section 4.4.4.1. In this process, we check all such moves and calculate the benefit for each move. In the next section, we show the benefit calculation for selecting the best potential move.

4.4.4.3. Implementation of the scheme for selecting the best move in Strategy 4

The details of the process of selecting the best move by calculating the benefit is shown in Fig. 4.4.2.3a.

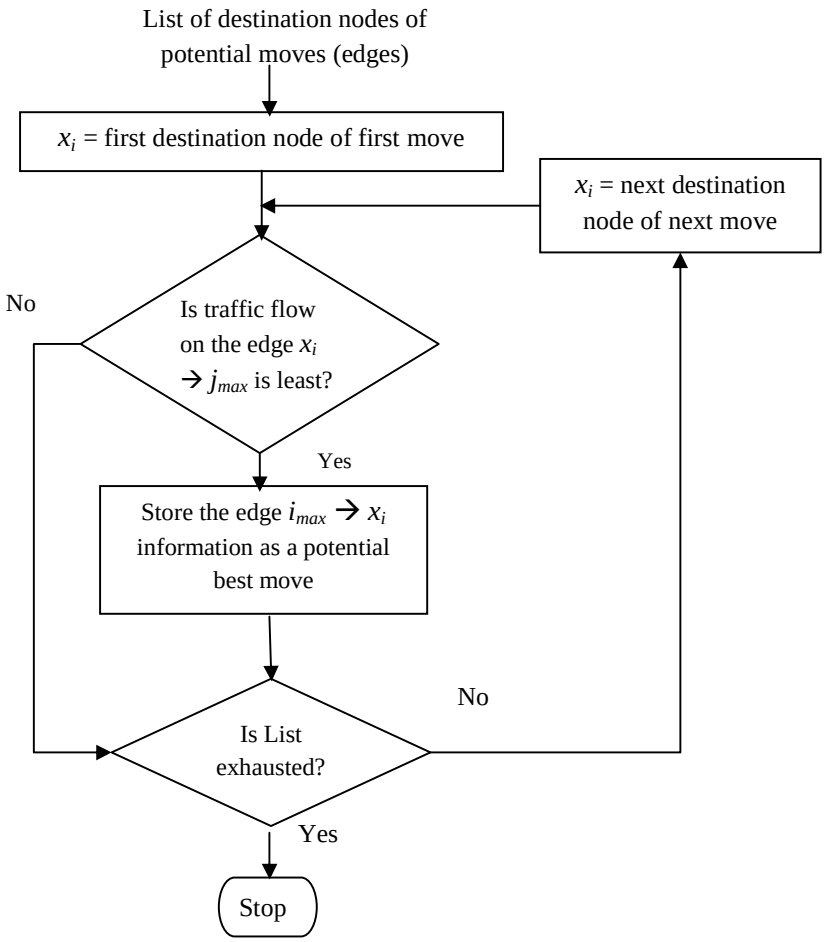

Figure 4.4.4.3a: Implementation of selecting the best move in Strategy 4.

An example of selecting the best move is shown in Fig. 4.4.4.3b. There are two potential moves $i_{max} \rightarrow x_1$ and $i_{max} \rightarrow x_2$ in this particular example. The node x_1 and x_2 are two destination nodes of the two moves.

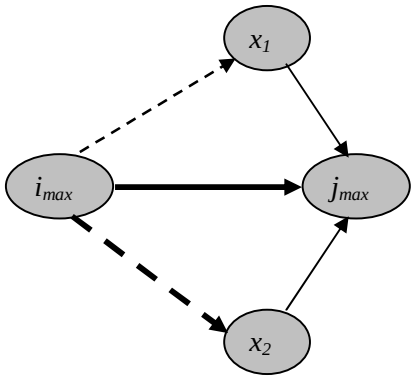

Figure 4.4.4.3b: Benefit calculation for the edge $x_2 \rightarrow j_{max}$ in Strategy 4.

In order to choose the best move, we checked edge $x_1 \rightarrow j_{max}$ and $x_2 \rightarrow j_{max}$. Each edge is carrying a certain amount of traffic for the different commodities. We selected the edge which is carrying the least amount of traffic as the best move. For example, if the edge $x_2 \rightarrow j_{max}$ is carrying a lesser amount of traffic than the edge $x_1 \rightarrow j_{max}$, then we selected the edge $x_2 \rightarrow j_{max}$ as the best move in this particular network.

The process for calculating the benefit for the edge $x_2 \rightarrow j_{max}$ follows an approach similar to Strategy 2 discussed above. An example for calculating the benefit with a data flow is shown in Chapter 3.

4.5 An Example of a 4 Node Network

An example of a 4 node network is given below to show the overall basic steps to solve a specified problem. Initially, we routed the traffic using the CPLEX optimization tool for an initial logical topology to observe the initial congestion of the network. As we have discussed in Section 4.3, the CPLEX is an optimization tool which is used to route the traffic optimally over a logical topology.

Table 4.5.1 is an example of a randomly generated traffic matrix[10] and Figure 4.5.1 is an initial logical topology[11] shown as a graph. The traffic matrix generated such that the congestion must be below 1.

0.000000	0.208333	0.500000	0.083333
0.458333	0.000000	0.208333	0.458333
0.437500	0.208333	0.000000	0.312500
0.229167	0.104167	0.312500	0.000000

Table 4.5.1: Initial Traffic Matrix

[10] The details of traffic matrix is discussed in Section 4.2

[11] The details of initial logical topology is discussed in Section 4.1

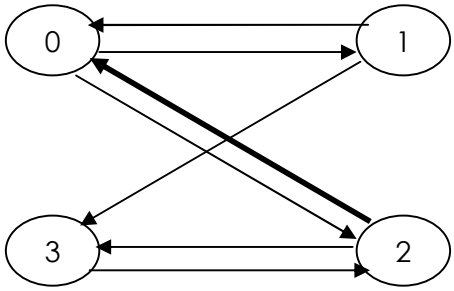

Figure 4.5.1: Initial Logical Topology.

Let the initial congestion of the network be 0.98 for initial logical topology with randomly generated initial traffic matrix. We increased the congestion to 1.05 by multiplying each entry of initial traffic matrix by an appropriate factor as shown in Table 4.5.2. In the subsequent iterations we created a new logical topology using our hill-climbing search algorithm and called the CPLEX optimizer with the new logical topology and the new traffic matrix (Table 4.3.2) to observe the new congestion value. We created a possible move list with 4 best moves from 4 strategies. Finally, we selected one best move, based on the highest benefit. We have shown only iteration to demonstrate the process.

0.000000	0.223214	0.535714	0.089285
0.491071	0.000000	0.223214	0.491071
0.468750	0.223214	0.000000	0.334821
0.245536	0.111607	0.334821	0.000000

Table 4.5.2: New Traffic Matrix

Iteration 1:

Let the edge carrying the maximum traffic be 2 → 0 ($i_{max} \rightarrow j_{max}$) as shown in Fig. 4.5.1 by a bold line at the beginning of the iteration.

Step 1- Find the possible move list: The Table 4.5.3 shows the possible move list from 4 different strategies and the benefit of each move.

From	To	Benefit	Strategy
3	0	0.330000	1
3	0	0.295000	2
2	1	0.310000	3
2	**1**	**0.670000**	**4**

Table 4.5.3: The possible move list

Step 2- Select the best move: The highest benefit can be achieved by adding the edge from the source node 2 to the destination node 1 as shown in the Table 4.5.3 from the strategy 4.

Step 3- Create the new logical topology: As a result, the new logical topology is created by adding the edge from node 2 to 1 and shown in dashed line in Figure 4.5.2.

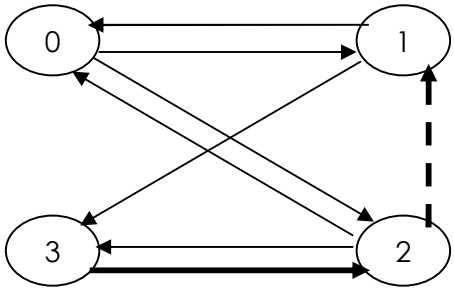

Figure 4.5.2: New Logical Topology.

Step 4 – Calculate the congestion: We call the CPLEX environment with new logical topology (Fig. 4.5.2) and new traffic matrix (Table 4.5.3). The new congestion value is 0.9725.

The conclusion is that, the initial congestion was 0.98 for the initial logical topology and the congested edge was 2 → 0 shown by bold line in Fig. 4.5.1. A new traffic matrix created such that the congestion scaled up to 1.05 from 0.98, which is 5% more then the capacity of the edge. After the reconfiguration of the logical topology, by adding an edge from node 2 to node 1, the new logical topology is shown in Fig. 4.5.2. Then by calling the CPLEX optimizer, we observe the congestion reduced to 0.9725 from 1.05 and new congested edge become 3 → 2 as shown in bold line in Fig. 4.5.2. By adding an edge, we are able to reduce the congestion below 1.

4.6 Implementation of LP formulation.

The LP formulation has been implemented using a C program discussed in Chapter 2, Section 2.9. The LP program generates the *lp* file as an input to the CPLEX optimizer tool. A *traffic matrix*[12] is shown in Table 4.6.1, where each non-zero entry corresponds to a commodity and gives the amount of traffic for a source-destination pair. In this table, we represented the commodity from *i* to *j*

[12] Traffic matrix is discussed in Section 4.2

by k_i^j , where the value of k_i^j is the entry of i^{th} row and j^{th} column of the traffic matrix. The first commodity k_0^1 is in row 0, column 1 and has a value 0.21 and denotes the data flowing from end node *0* to end node *1*. Similarly, the second commodity k_0^2 is the entry in row 0, column 2 and has a value 0.50. This denotes the traffic flowing from end node *0* to end node *2* and so on.

0.00	0.21	0.50	0.08
0.46	0.00	0.21	0.46
0.44	**0.21**	0.00	0.31
0.23	0.10	0.31	0.00

Table 4.6.1: Traffic Matrix

The implementation of the LP formulation is informally discussed with an example. Let us consider commodity k_2^1 , where 0.21 units of traffic need to be routed optimally from source node 2 to destination node 1 as shown in Table 4.6.1.

Some lines of an *.lp* file format are shown below. The line *C3* corresponds to the constraint of the form (2.9.2) and line *C36, C37* and *C38* corresponds to the constraint of the form (2.9.3) in Section 2.9 of chapter 2.

C3: + X_0_1_0 + X_1_1_0 + X_2_1_0 + X_3_1_0 + X_4_1_0 + X_5_1_0 + X_6_1_0 +

X_7_1_0 + X_8_1_0 + X_9_1_0 + X_10_1_0 + X_11_1_0 - Lmax <= 0

C36: +X_7_2_0 + X_7_2_3 - X_7_0_2 - X_7_3_2 = 0.21

C37: +X_7_1_0 + X_7_1_3 - X_7_0_1 = -0.21

C38: + X_7_0_1 + X_7_0_2 - X_7_1_0 - X_7_2_0 = 0

Explanations of the symbol:

i. *C1…Cn* (where *n* is an integer value) in the beginning of the equation represents the constraint number of the LP file.

ii. *X_k_i_j*: a continuous variable used to define the amount of the traffic flowing for commodity number *k* on the logical edge *i* → *j* (where *k*, *i* and *j* are integer values)

iii. $L_{max:}$ The congestion value of the network

The details of the form of LP formulation is discussed below with an example.

An example of constrain C3:

This equation corresponds to the *LP* form $\sum_{k=1} x_{ij}^{k} \leq \Lambda_{\max}, \forall (i, j) \in L$ (2.9.2). This LP constraint ensures that the congestion of the network is greater than or equal to the total traffic on any edge. In other words, the summation of the amount of the traffic is flowing through an edge for all the commodities is less than or equal to the congestion value of the network. The line *C3* of the *.lp* file (shown above) is an example for such an equation. According to the traffic matrix (table 4.6.1), the equation also shows that there are 11 commodities flowing on a particular 4-node logical topology shown in Fig. 4.6.1. The variable "*X_0_1_0*" denotes that commodity 0 flows on the edge 1 → 0, the variable "X_1_1_0" denotes that

commodity 1 flows on the edge 1 → 0 and so on.

Let the congestion value of a particular 4-node network be 0.98. As mentioned, we would like to use commodity k_2^1 for this example, this particular commodity is flowing from source node 2 to destination node 1 using the path 2 → 0 → 1 with the value 0.21. Since there is no direct path from node 2 to 1 (Fig 4.6.1), the total traffic flowing on the edge 2 → 0 and 0 → 1 is 0.98 (0.44 + 0.21 + 0.23 + 0.10) and 0.52 (0.21 +0.21 + 0.10) respectively as per as data shown in table 4.6.2.

According to the formulation, the total traffic for all commodities flowing on any edge must be less than or equal to the congestion value. From the above example, the edge 2 → 0 carrying 0.98 units of data which is equal to the congestion value and the edge 0 → 1 is carrying 0.52 units of data, which is less than the congestion value. Therefore, the equation for this constraint in line *C3* satisfies the condition.

An example of Constrain C36:

Constraint C36 corresponds to the first condition of the form 2.9.3 discussed in Chapter 2, if an end node is a *source* of a commodity flowing from a source node *i* to a destination node *j*.

This constraint ensures that if an end node is a *source* of a commodity, then there is no traffic is flowing into that node and the traffic flowing out from this node is equal to the entry of the traffic matrix for that source *i* to destination node *j*. The line *C36* is an example of such equation of the *.lp* file (shown above). This equation represents the fact that the difference between the data flowing in and flowing out must be equal to the entry of the traffic matrix for the given pair of source node *i* to the destination node *j*.

For example, 0.21 units of data are flowing from the source node 2 to destination node 1 for commodity k_2^1, using the logical path 2 → 0 →1, as shown in Fig. 4.6.1 and in Table 4.6.1. There are two incoming edges 0 → 2 and 3 →2 to the

source node 2 and two outgoing edges 2 → 0 and 2 → 3 from the source node 2 and on traffic flowing on the edge 2 → 3, 0 → 2 and 3 →2 as per the data shown in Table 4.6.2. Therefore, if we plug-in the value of the four variables in *C36*, the result is 0.21 + 0.0 - 0.0 - 0.0 = 0.21, which satisfies the condition of the equation.

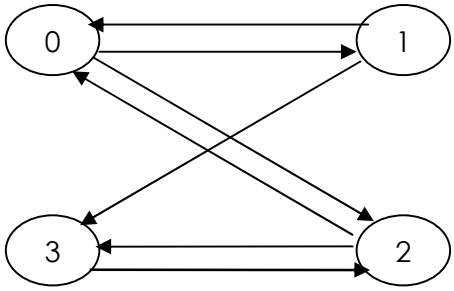

Figure 4.6.1: Logical Topology

Commodity	From	To	Traffic
0	0	1	0.21
7	0	1	0.21
10	0	1	0.1
1	0	2	0.5
2	0	2	0.08
4	0	2	0.21
3	1	0	0.46
4	1	0	0.21
5	1	3	0.46
6	2	0	0.44
7	2	0	0.21
9	2	0	0.23
10	2	0	0.1
2	2	3	0.08
8	2	3	0.31
9	3	2	0.23
10	3	2	0.1
11	3	2	0.31

Table 4.6.2: Amount of traffic flowing on edges for the commodity

An example of Constrain C37:

The constraint C37 corresponds to the second condition of the form 2.9.3 discussed in Chapter 2, if an end node is a *destination* of a commodity flowing from a source node *i* to a destination node *j*.

This constraint ensures that if an end node is a destination for a commodity, then there is no traffic flowing out from that node and the traffic flowing into the destination node is the entry of the traffic matrix of the source *i* to the destination *j* pair. The line *C37* is an example of such an equation of the *.lp* file shown above.

For example, the destination of the commodity k_2^1 is node 1. There are two outgoing edges (1 → 3 and 1 → 0) and one incoming (0 →1) edge for the destination

node 1 (Fig.4.6.1) and no data flows on the edge 1 → 3 and 1 → 0 as per the data shown in Table 4.6.2. Therefore, if we plug-in the value of 3 variables to the equation *C37*, the results is 0.0 + 0.0 – 0.21 = -0.21. Thus the equation *C37* satisfies the condition.

An example of Constraint C38:

The constraint C38 corresponds to the third condition of the form 2.9.3 discussed in Chapter 2, when an end node is an *intermediate* node of a commodity flowing from a source node *i* to destination node *j*.

This constraint ensures that for all intermediate nodes where the data is flowing from the source node to the destination node, incoming flow must be matched by the outgoing flow, so that the difference must be 0 (zero). The line *C38* is an example of such an equation of the *.lp* file shown above.

For example, commodity k_2^1 is flowing from source node 2 to destination node 1 using the path 2 → 0 → 1. Since there is no direct link from source node 2 to destination node 1 (Fig. 4.6.1), the traffic is flowing through intermediate node *0*. The amount of data flowing into the node *0* is 0.21 from node *2* using the edge 2 → 0 and the amount of data flowing out from node *0* to node *1* is also 0.21 using the edge 0 → 1 as per the data shown in the Table 4.6.2. Therefore, if we plug in the values of 4 variables into the equation *C38* as per the data shown in table 4.6.2, such as 0.21 + 0.0 + 0.0 + 0.21 = 0. Thus it is satisfying the condition.

Chapter 5: Experimental Results

We carried out our experiments using networks having three sizes - 6 nodes, 10 nodes and 14 nodes. For networks of a given size, we created 5 initial logical topologies. We tested each logical topology with 5 different traffic matrices, created using a random number generator generating, for each pair of end-nodes, values between 0 and 1. In this way we tested 25 cases for each size of network to observe how many logical edge(s) is (are) required for reconfiguration. Finally we computed the averages of the 25 tests cases. We generated the traffic matrices randomly as discussed in Section 4.2.

As we have mentioned in our problem definition, the set of requests for data communication that the network has to handle does not remain the same and changes with time. Therefore, when the traffic increases beyond the capacity of a logical edge, the logical topology has to be augmented so that the new topology can handle the new traffic. Our objective is to observe how many new logical edge(s) is (are) required to reduce the congestion to the expected level that can be carried by a lightpath. We fixed the expected congestion level after reconfiguration to 0.90, meaning that no lightpath will carry a traffic more than 90% of the capacity of a lightpath.

We carried out our experiments using the following steps:

1. Determine the initial congestion of the network, using the supplied logical topology and traffic matrix.

2. Create a traffic matrix such that the edge carrying the maximum traffic was x% above the capacity of a logical edge (we used three values of x – 5, 10 and 15).

3. Create a new logical topology by augmenting the initial logical topology by

adding a new logical edge using our Hill-Climbing algorithm as mentioned in Section 4.4.

4. Determine the value of the congestion by using the CLEX optimization tool with the traffic matrix created in Step 2 and the new logical topology generated in Step 3.

5. If the congestion in the new topology is greater than the target value (0.9 in our experiments), go to Step 3. Otherwise stop.

In Step 2, we accomplished this by multiplying each entry of the traffic matrix with an appropriate factor, based on the initial congestion as mentioned in Chapter 4. When the above process terminates, we noted how many edge(s) is (are) required to reduce the congestion up the expected level.

As mentioned before, for each size of the network, we created five logical topologies and five traffic matrices. For our discussion below, given a size of the network, we will refer to the five logical topologies (traffic matrices) as logical topology (traffic matrix) 1, 2…5. The first column represents the test case scenarios. For example, in Table 5.1, 6-1-1 stand for the situation where the size of the network is 6, logical topology number 1 and tested with traffic matrix number 2. Columns 2, 3 and 4 represents, how many edges are required when the traffic has increased, so that the congestion is 5%, 10% and 15% over the capacity of a logical edge.

Table 5.1 represents the test results for networks with 6 nodes. We tested a total of 75 test cases. We observed that approximately 2 (respectively 3 and 4) logical edges are needed to reduce the congestion level below 0.90, when the starting congestion was 5%, (respectively 10% and 15%). Similarly, Table 5.2 and 5.3 represents the test results for 10-node and 14-node networks. We have included the detailed experimental data in Appendix 3.

A bar graph for 3 sizes of network are given in this section. The x-axis represents

the test case number and y-axis represents total edges required in 3 different situations of data communications.

5.1 Experimental Results with 6-Nodes

Test_Case Node# -#LT-#TM	New Edge Required-Congestion increased 5% above edge capacity	New Edge Required-Congestion increased 10% above edge capacity	New Edge Required-Congestion increased 15% above edge capacity
6-1-1	1	1	3
6-1-2	1	2	2
6-1-3	2	2	2
6-1-4	2	2	2
6-1-4	4	5	5
6-2-1	1	1	2
6-2-2	2	2	3
6-2-3	3	3	3
6-2-4	1	2	3
6-2-5	2	4	5
6-3-1	3	3	4
6-3-2	2	3	3
6-3-3	1	1	1
6-3-4	3	3	4
6-3-5	3	3	5
6-4-1	1	1	1
6-4-2	1	2	2
6-4-3	1	2	2
6-4-4	1	2	2
6-4-5	1	2	7
6-5-1	3	5	6
6-5-2	2	2	3
6-5-3	2	2	3
6-5-4	1	2	5
6-5-5	2	2	7
Average edge required	1.84	2.36	3.4

Table 5.1: Experimental Results with 6-Nodes

5.2 Experimental Results with 10-Nodes

Test_Case Node# -#LT-#TM	New Edge Required-Congestion increased 5% above edge capacity	New Edge Required-Congestion increased 10% above edge capacity	New Edge Required-Congestion increased 15% above edge capacity
10-1-1	3	3	7
10-1-2	4	6	6
10-1-3	3	6	8
10-1-4	4	7	11
10-1-5	2	2	3
10-2-1	4	9	10
10-2-2	4	12	27
10-2-3	10	11	12
10-2-4	9	10	19
10-2-5	3	5	7
10-3-1	10	12	14
10-3-2	2	4	7
10-3-3	5	11	24
10-3-4	7	8	9
10-3-5	7	7	11
10-4-1	8	20	21
10-4-2	8	13	18
10-4-3	11	16	18
10-4-4	7	7	12
10-4-5	7	9	16
10-5-1	14	16	19
10-5-2	4	18	20
10-5-3	4	6	11
10-5-4	7	9	14
10-5-5	3	4	6
Average edge required	6	9.24	13.2

Table 5.2: Experimental Results with 10-Nodes

5.3 Experimental Results with 14-Nodes

Test_Case Node# -#LT-#TM	New Edge Required- Congestion increased 5% above edge capacity	New Edge Required- Congestion increased 10% above edge capacity	New Edge Required- Congestion increased 15% above edge capacity
14-1-1	7	9	14
14-1-2	7	10	11
14-1-3	11	12	14
14-1-4	9	10	12
14-1-5	8	9	10
14-2-1	9	14	19
14-2-2	9	10	45
14-2-3	8	16	22
14-2-4	10	15	27
14-2-5	8	24	25
14-3-1	1	4	8
14-3-2	12	15	39
14-3-3	8	11	17
14-3-4	9	13	16
14-3-5	17	20	31
14-4-1	6	9	11
14-4-2	7	10	16
14-4-3	10	21	16
14-4-4	9	11	17
14-4-5	9	10	15
14-5-1	9	10	16
14-5-2	8	16	20
14-5-3	7	15	34
14-5-4	8	14	23
14-5-5	11	52	50
Average edge required	8.68	14.4	21.12

Table 5.3: Experimental Results with 14-Nodes

5.4 Results with line chart graph

5.4.1: Chart bar Graph for 6-Node Network

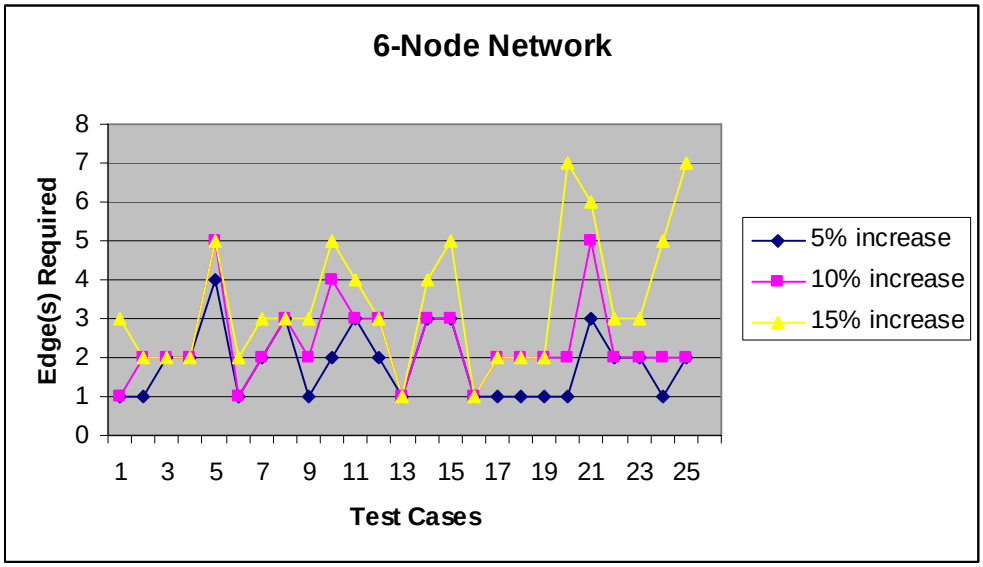

Figure 5.4.1a: Edge(s) Required for 6-Node Network

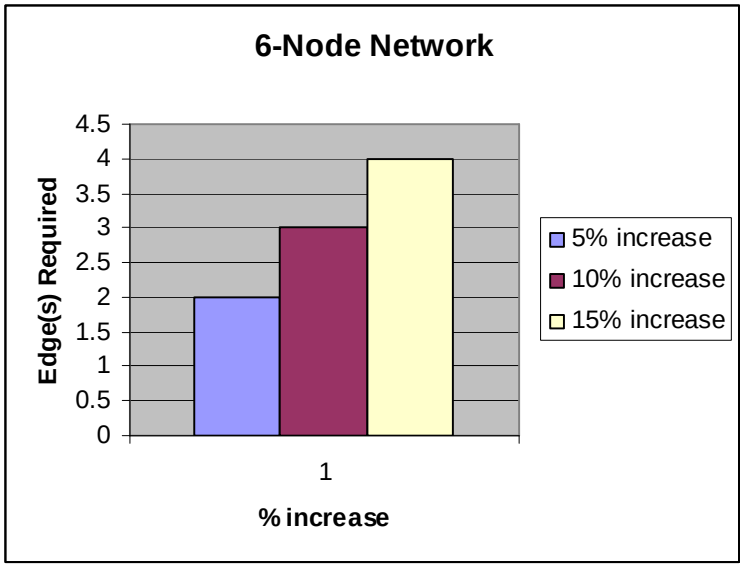

Figure 5.4.1b: Average Edge(s) Required for 6-Node Network

5.4.2: Chart bar Graph for 10-Node Network

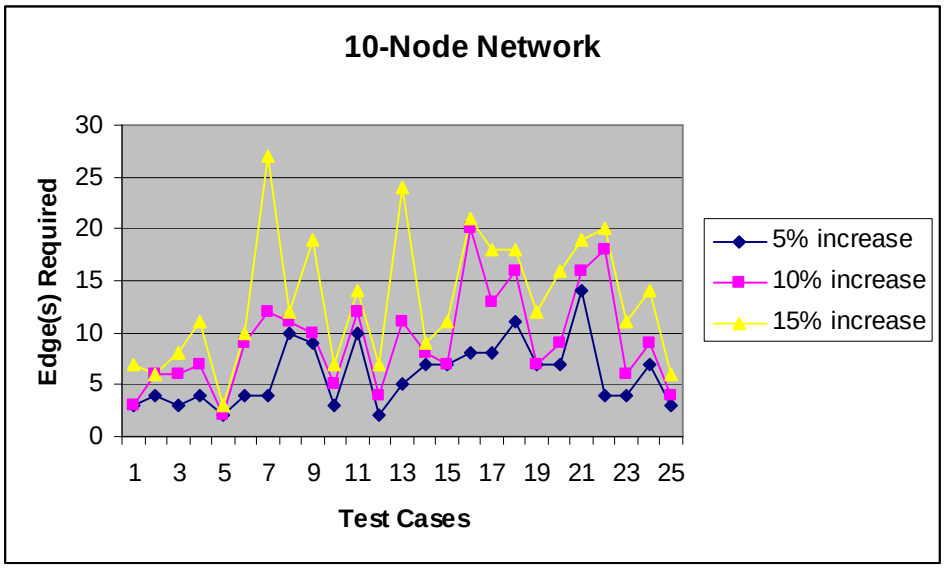

Figure 5.4.2a: Edge(s) Required for 10-Node Network

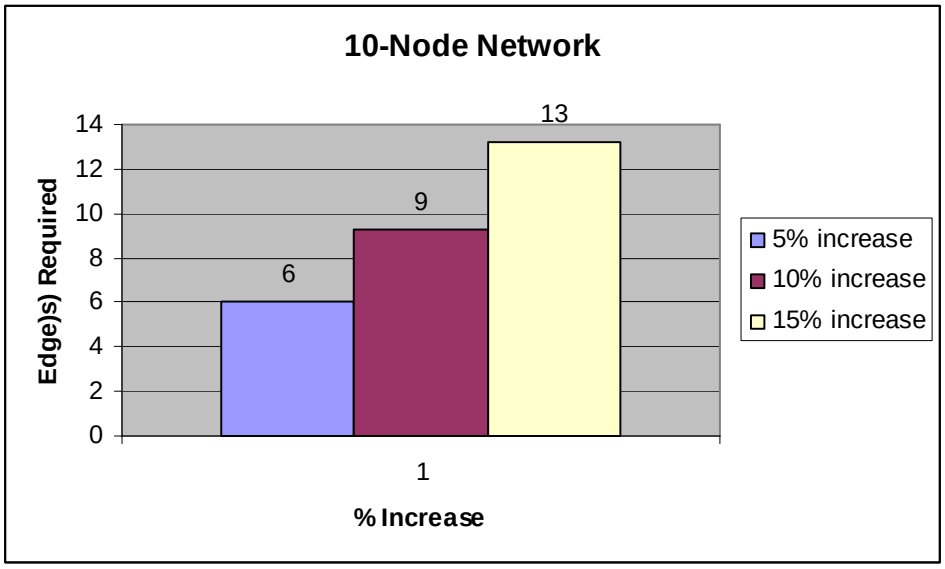

Figure 5.4.2b: Average Edge(s) Required for 10-Node Network

5.4.3: Chart bar Graph for 14-Node Network

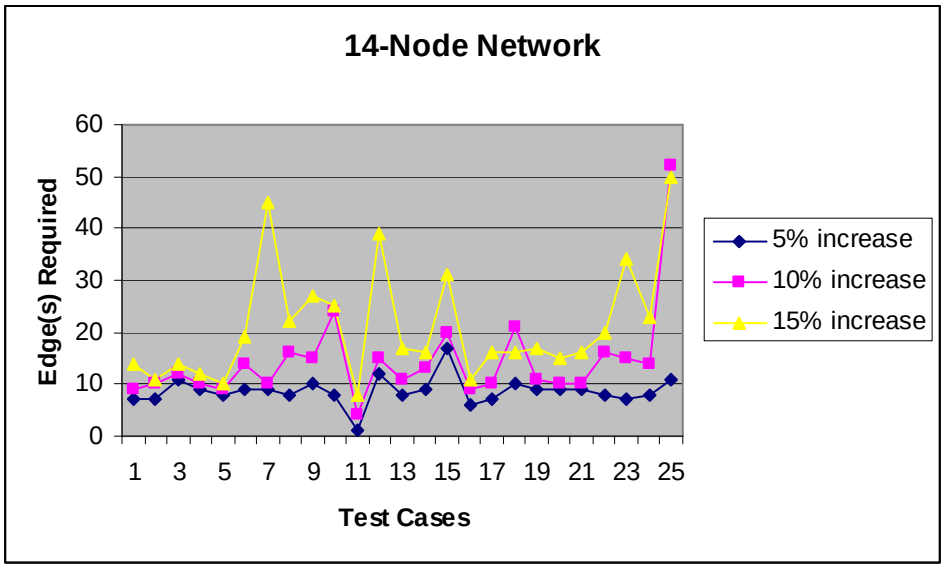

Figure 5.4.3a: Edge(s) Required for 14-Node Network

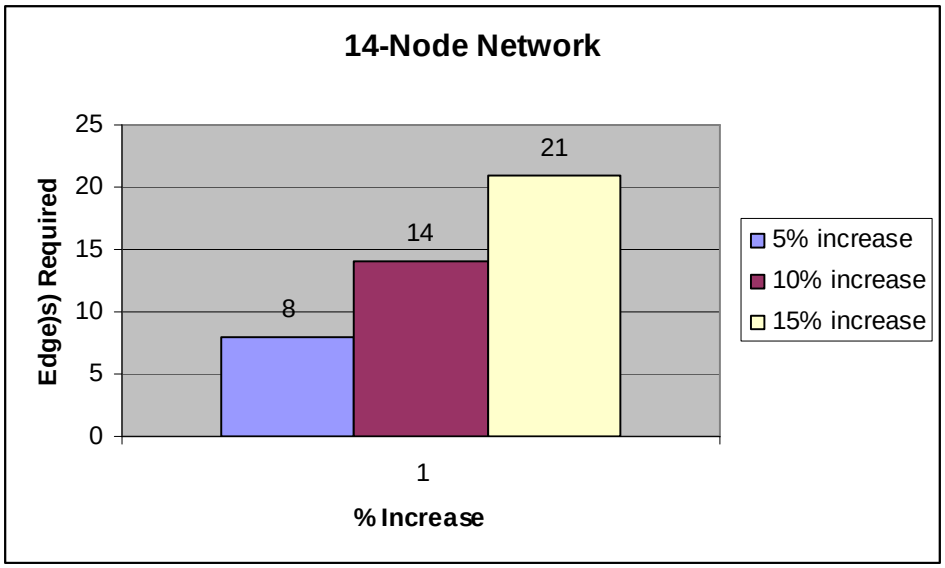

Figure 5.4.3b: Average Edge(s) Required for 14-Node Network

5.5 Observation

There are some significant observations from our experiments:

1. The congestion reduced as new edges are added to an old logical topology in the each iteration and the required number of edges increased when the traffic increased gradually. As an example, shown in Table 5.2, case number 10-3-2, the new edges required are 2 for 5%, 4 for 10% and 7 for 15% increase of traffic.

2. When the traffic increases, the average number edges required also increases for each type of network as shown in Table 5.3, the number of edges required almost doubles from 5% to 15% increase of traffic in 14-node network. Similarly, for 6-node and 10-node networks, the results are shown in Table 5.1 and 5.2.

3. In our problem specification, we mentioned that there are basically three steps for reconfiguration of logical topology.

    - Design an initial logical topology.
    - Reconfigure the logical topology as necessary when traffic demand changes with the time.
    - Route the traffic optimally and determine the congestion of the network.

Since, our initial step is to design a logical topology and we have used any heuristic algorithm to design the initial logical topology, we can not claim that it is optimal. The designing of initial logical topology has significant importance in our problem solution, because we could not guarantee that initial congestion is minimal at the time of designing initial logical topology; it could be either very high or very low.

Chapter 6: Conclusions and Future work

6.1 Conclusion

As we mentioned in Chapter 1, in static lightpath allocation, the logical topology of a WDM optical network is determined, based on long-term traffic demands. These traffic demands however do not remain the same at subsequent points in time. When a logical topology is incapable of supporting the current traffic demands, the logical topology has to be modified or reconfigured. Our objective is to find an optimal new logical topology which can support the current traffic demands with as little change to the original topology as possible.

In this Thesis, we have implemented a Hill-Climbing algorithm to solve the problem of reconfiguring logical topologies. We have tried to determine how many new edges are required when the congestion of a network exceeds beyond the capacity of a logical edge. We have shown in Chapter 5, how many edges are required when congestion increases above the capacity of a logical edge by a factor that ranges from 5% to 15%.

Utilizing the Hill-Climbing algorithm we have described an iterative process where we identify, in the each iteration, the “most promising” edge to be added to the initial logical topology. We monitored the congestion value after routing the traffic after adding the promising edge. Although the Hill-Climbing algorithm has the problem of getting into the trap of local optima, we have found that our solution gives fairly good results when traffic increases beyond the capacity of a lightpath. We have compared 3 situations, and found out how many total new edges are required in each situation.

6.2 Future work

On the basis of our experimental observation discussed in Section 5.4, there are at least two approaches that can be taken:

1. Design the initial logical topology optimally
2. Apply alternative strategic approaches in Hill-Climbing algorithm

It is well known that designing an optimal logical topology is considered to be intractable using a mathematical programming approach. Since, it is very important to have an optimal logical topology initially; we could use meta-heuristics instead of any heuristics, such as the *Tabu Search* algorithm.

For the logical topology reconfiguration, the Hill-Climbing algorithm has been investigated. There are many alternatives strategic approaches can be taken to improve the search criteria, in order to determine the best logical edge to be added to change (reconfigure) the logical topology.

Appendix 1

1.1 The notation used in LP Formulation

N_E : Number of end nodes in the network.

T: An N_E x N_E matrix, called the traffic matrix, giving the traffic requirements for the network.

T(I, j): entry in row I and column j of matrix T. it represents the traffic from source end node E_i to destination end node E_j.

Λ_{max} : The congestion of the network.

B_{ij} : A binary variable such that

$$b_{ij} = \begin{cases} 1 \ \ if \ a \ lightpath \ exist \ \ from \ end \ node \ Ni \ to \ end \ node \ Nj \\ \\ 0 \ \ otherwise \end{cases}$$

χ_{ij}^{sd} : A continuous variable to denote the portion of traffic t(s, d) : t(s,d) > 0, that is routed through the logical edge $E_i \Rightarrow E_j$.

Δ_{in} (Δ_{out}) : A constant denoting the number of transmitter or receiver at every end node.

1.2 C-Programming Code for LP Formulation

```
/******  objective function ( 1a) *******/
fprintf(lpFile,"Minimize\n obj: Lmax\n");


fprintf(lpFile,"\nSubject to\n");
        //count=0;
/********* constraint (1)  **************/
commodity = 0;
for(i = 0; i < endNodes; i++)
  for(j = 0; j < endNodes; j++)
        if( Tmatrix[i][j] > 0)
                commodity++;

count=0;
for (m = 0; m < endNodes; m++)
  for (n = 0; n < endNodes; n++)
         if(Lmatrix[m][n] == 1)
         {
           count++;
         fprintf(lpFile, "c%d: ", count);

                for(i = 0; i < commodity; i++)
       fprintf(lpFile, "+ X_%d_%d_%d ",i, m, n);

                fprintf(lpFile, "- Lmax <= 0\n");
         }
```

```
commodity = 0;
        for(i = 0; i < endNodes; i++)
          for(j = 0; j < endNodes; j++)
                 if( Tmatrix[i][j] > 0)
                 {
              for ( m = 0; m < endNodes; m++)
              {
                count++;
                fprintf(lpFile, "c%d: ", count);

                for ( n = 0; n < endNodes; n++)
                  if(Lmatrix[m][n] == 1)
                    fprintf(lpFile, "+ X_%d_%d_%d ", commodity, m, n);

                                for ( n = 0; n < endNodes; n++)
                                if(Lmatrix[n][m] == 1)
                                  fprintf(lpFile, "- X_%d_%d_%d ", commodity

                                if(m == _src(commodity))
                                        fprintf(lpFile, "= %.2f\n", Tmatrix[i][j]);
                                else  if( m == _des(commodity))
                                        fprintf(lpFile, " = -%.2f\n", Tmatrix[i][j]
                                else
                                        fprintf(lpFile, " = 0\n");
                        }//end of each row

                        commodity++;
                 }


                fprintf(lpFile,"\nEnd\n");


        fclose(lpFile);//close the file pointer

                fclose(ltFile);//close the file pointer

                fclose(trFile);//close the file pointer
```

1.3. Notation used for the heuristic formulation in Section 2.4.2

The inputs to the heuristic are the following:

- Δ^{i}_{in} denoting the number of receivers at end node $E_{i,}$ $1 \leq I \leq N_E$.
- Δ^{i}_{out} denoting the number of transmitters at end node $E_{i,,}$ $1 \leq I \leq N_E$.
- The traffic matrix $T = [t(I, j)]$ giving the required volume of data so that $t(I, j)$ is the traffic from end node Ei to end node Ej in the traffic matrix T, for all I, j, $1 \leq I, j \leq N_{E.}$
- The physical topology of the network.
- The number of channels n_{ch} per fiber.
- $c_{lightpath}$ is the capacity of a lightpath.

1.4 Routing over logical topology described in Section 2.7

N_E: Number of end nodes

q: Number of commodities src_k and $dest_k$: source s_k and destination d_k for commodity k,

$1 \leq k \leq q$)

t^k : traffic $t(s_k, d_k)$ for commodity k, $(1 \leq k \leq q)$

x^{k}_{ij} : amount of traffic flowing on logical edge $I \Rightarrow j$ for commodity k,

Λ_{max} : Congestion on the network

L: set of all pair of end node, lightpath exists on end node E_i to end node E_j.

Appendix-2

2.1 Reference

[1]. S. Sinha, N. Rammohan and C. S. Murthy. Dynamic Virtual Topology Reconfiguration Algorithms for Groomed WDM Networks, Volume 9, No. 2, March, 2005.

[2]. D. Banerjee and B. Mukherjee. Wavelength-Routed Optical Networks: Linear Formulation, Resource Budgeting Tradeoffs, and a Reconfiguration Study, IEEE/ACM Transactions on Networking, Volume 8, No. 5, October 2000.

[3]. B. Mukherjee, Optical Communication Networks. New York: Mc-Graw Hill, 1997.

[4]. J-F. Labourdette, A. Acampora, Logically Rearrangeable Multihop Lightwave

Networks, IEEE Transactions on Communication 39(1991), P: 1223-1230.

[5]. A. Grosso, E. Leonardi, M. Mellia, and A. Nucci. Logical topologies design over WDM Wavelength routed networks robust to traffic uncertainties. IEEE Communications Letters, Volume 5, No. 4, P: 172-174, April 2001.

[6]. R. Ramaswami and K. N. Sivarajan. Routing and wavelength assignment in All-Optical networks. IEEE/ACM Transactions on Networking, Volume 3, No. 5, P: 489-500,October 1995.

[7]. R. Dutta and G. N. Rouskas. A survey of virtual topology design algorithms for wavelength routed optical networks. SPIE Optical Networks Magazine, Volume 1, No.1, P: 73-89, January 2000.

[8]. E. Medova. Network flow algorithms for routing in networks with wavelength division multiplexing. IEE Proceedings Communications, Volume142, No. 4, P: 238-242, August 1995.

[9]. A. N. Tam and E. Modiano. Dynamic load balancing in WDM based packet networks with and without wavelength constraints. IEEE Journal on Selected Areas in Communications, Volume 18, No.10, P: 1972-1979, October 2000.

[10]. A. Lokketangen and F. Flober. Solving zero-one mixed integer programming problems using tabu search", European Journal of Operations Research, P: 624-658, 1998.

[11]. Fred Glover. Tabu Search: A Tutorial, P: 74-94, in: Interface 20, 4 July-August 1990.

[12]. Fred Glover. Tabu Search and Adaptive Memory programming- Advance, Applications and Challenges, in: Interfaces in Computer Science and Operation, 1996.

[13]. I. Baldine and G. N. Rouskas. Traffic adaptive WDM networks: a study of re-configuration issues. IEEE/OSA Journal of Lightwave Technology, Volume19, No. 4, P: 433-455, April 2001.

[14]. R. M. Krishnaswamy and K. N. Sivarajan. Design of logical topologies: A linear formulation for wavelength-routed optical networks with no wavelength changers. IEEE/ACM Transactions on Networking, Volume 9, No. 2, P: 186-198, April 2001.

[15]. C. Dzongang, P.Galinier, and S. Pierre. A Tabu Search Heuristic for the Routing and Wavelength Assignment Problem in optical Networks, IEEE Communications Letters, Volume 9, No. 5, MAY 2005.

[16]. V.Boljuncic, D. Skorin-Kapov, J. Skorin-Kapov, A Tabu Search Approach Towards Congestion and Total Flow Minimization in Optical Networks, Journal of Systems Science and Systems Engineering, Volume 13, No. 2, April, 2004.

[17]. J. Kuri, N. Puech and M. Gagnaire, A Tabu Search Algorithm to Solve a Logical Topology Design Problem in WDM Networks considering Implementation Cost, In SPIE Asian Pacific Optical Conference, Shangai, 2002.

[18]. E. Leonardi, M. Mellia, and M. A. Marsan. Algorithms for the logical topology design in WDM all-optical networks. Optical Network Magazine, Volume 1, No. 1, P: 35-46, January, 2000.

[19]. R. Ramaswami and K. N. Sivarajan, Design of Logical Topologies for Wavelength-Routed Optical Networks, IEEE Journal on Selected Areas in Communications, Volume 14, P: 840-851, 1996.

[20]. I. Chlamtac, A. Farago, and T. Zang. Lightpath (wavelength) routing in large WDM networks. IEEE Journal on Selected Areas in Communications, Volume 14, No. 5, P: 909-913, June, 1996.

[21]. Dr. S. Bandyopadhyay. Dissemination of Information in Optical Networks: From Technology to Algorithms, Springer 2007.

[22]. A. Sood, Logical Topology Design for WDM Networks using Tabu Search, Thesis work, University of Windsor.

[23]. R. Ramaswami and K. N. Sivarajan. Optical Networks: A Practical Perspective. Morgan Kaufmann Publishers, 2002.

[24]. J. Kurki et al, Wavelength Router as a Transport Platform for IP, NOC 2000, June 6-9, 2000.

[25]. O. Gerstel and S. Kutten. Dynamic wavelength allocation in all-optical ring networks. In IEEE International Conference on Communications (ICC), Volume 1, P: 432-436, June 1997.

[26]. P. Garg, Genetic Algorithms, Tabu Search and simulated Annealing: A Comparison between three approaches for the cryptanalysis of transposition cipher.

[27] F. Glover, "Tabu Search and Adaptive Memory Programming- Advances, Applications and Challenges", in: Interfaces in Computer Science and Operations Research, 1996

[28] M. L. Puterman. *Markov Decision Processes*. Wiley, 1994

[29]. P.N.Tran, U. Killat, Distributed algorithm for dynamic logical topology reconfiguration in IP over WDM networks, Computers and Communications, 2009. ISCC 2009. IEEE Symposium on, P748 – 756

[30] P.N. Tran, U. Killat, Dynamic reconfiguration of logical topology for WDM networks under traffic changes, Network Operations and Management Symposium, 2008. NOMS 2008. IEEE, P279 – 286, 2008.

[31]. R.J. Duran, R.M. Lorenzo, N. Merayo, I. Miguel, P. Fernandez, J.C. Aguado, E.J Abril, Efficient reconfiguration of logical topologies: Multiobjective design algorithm and adaptation policy, Broadband Communications, Networks and Systems, BROADNETS 2008. 5th International Conference on 2008, P544 – 551.

Appendix 3

3.1 Details raw data of test results for 6- node network

Test_Case		Congestion increased 5% above edge capacity		Congestion increased 10% above edge capacity		Congestion increased 15% above edge capacity	
Node# -#LT-#TM	Initial Congestion	New Edge Required	Final Congestion	New Edge Required	Final Congestion	New Edge Required	Final Congestion
6-1-1	0.701389	1	0.845977	1	0.886262	3	0.871039
6-1-2	0.743056	1	0.736332	2	0.836703	2	0.862499
6-1-3	0.526042	2	0.873267	2	0.838614	2	0.850166
6-1-3	0.654948	2	0.809941	2	0.848510	2	0.654948
6-1-5	0.546875	4	0.762858	5	0.843333	5	0.744763
6-2-1	0.973959	1	0.752407	1	0.590588	2	0.824065
6-2-2	0.812500	2	0.8502135	2	0.813248	3	0.776282
6-2-3	0.677083	3	0.608462	3	0.637937	3	0.666412
6-2-4	0.901041	1	0.886127	2	0.727890	3	0.696243
6-2-5	0.770833	2	0.879731	4	0.772974	5	0.808109
6-3-1	0.838542	3	0.721428	3	0.658695	4	0.601241
6-3-2	0.753472	2	0.764170	3	0.696774	3	0.858527
6-3-3	0.927084	1	0.690168	1	0.723032	1	0.755897
6-3-4	0.692709	3	0.726315	3	0.898746	4	0.795487
6-3-5	0.756945	3	0.799542	3	0.696330	5	0.727981
6-4-1	0.989582	1	0.806844	1	0.845264	1	0.883686
6-4-2	0.947917	1	0.796154	2	0.834067	2	0.891978
6-4-3	0.979167	1	0.871276	2	0.867447	2	0.715692
6-4-4	0.927084	1	0.884830	2	0.840448	2	0.878650
6-4-5	0.875001	1	0.824999	2	0.864285	7	0.451786
6-5-1	0.585937	3	0.883556	5	0.829689	6	0.602311
6-5-2	0.729167	2	0.754286	2	0.788571	3	0.674999
6-5-3	0.656249	2	0.643334	2	0.673969	3	0.730160
6-5-4	0.609376	1	0.897433	2	0.699486	5	0.766666
6-5-5	0.807292	2	0.839998	2	0.879999	7	0.636645

Ave. edge required		1.84		2.36		3.4	

3.2 Details raw data of test results for 10-node network

Test_Case		Congestion increased 5% above edge capacity		Congestion increased 10% above edge capacity		Congestion increased 15% above edge capacity	
Node# -#LT-#TM	Initial Congestion	New Edge Required	Final Congestion	New Edge Required	Final Congestion	New Edge Required	Final Congestion
10-1-1	0.957532	3	0.849555	3	0.819707	7	0.875732
10-1-2	0.915720	4	0.794287	6	0.861307	6	0.840499
10-1-3	0.940972	3	0.688700	6	0.775857	8	0.838630
10-1-3	0.902778	4	0.742673	7	0.765253	11	0.880192
10-1-5	0.937500	2	0.633899	2	0.858611	3	0.858611
10-2-1	0.883333	4	0.848172	9	0.888562	10	0.619301
10-2-2	0.975695	4	0.867971	12	0.678790	27	0.675267
10-2-3	0.802828	10	0.608526	11	0.637503	12	0.882217
10-2-4	0.769097	9	0.839052	10	0.685327	19	0.778782
10-2-5	0.897569	3	0.864255	5	0.889556	7	0.880850
10-3-1	0.847373	10	0.822855	12	0.898225	14	0.863444
10-3-2	0.971875	2	0.863880	4	0.894070	7	0.872053
10-3-3	0.789062	5	0.884357	11	0.658306	24	0.834985
10-3-4	0.751157	7	0.733339	8	0.873305	9	0.893069
10-3-5	0.893995	7	0.862526	7	0.875828	11	0.744634
10-4-1	0.807870	8	0.618266	20	0.852242	21	0.756232
10-4-2	0.859373	8	0.871817	13	0.733333	18	0.766666
10-4-3	0.785416	11	0.6220116	16	0.699354	18	0.722505
10-4-4	0.718149	7	0.774059	7	0.791788	12	0.895002
10-4-5	0.756740	7	0.859980	9	0.787386	16	0.756740
10-5-1	0.828125	14	0.700000	16	0.733334	19	0.766667
10-5-2	0.776042	4	0.848701	18	0.720138	20	0.790765
10-5-3	0.655449	4	0.853694	6	0.878453	11	0.867203
10-5-4	0.631325	7	0.883559	9	0.870277	14	0.687203
10-5-5	0.737939	3	0.873699	4	0.899732	6	0.841539
Ave. edge required		6		9.24		13.2	

3.3 Details raw data of test results for 14-node network

Test_Case		Congestion increased 5% above edge capacity		Congestion increased 10% above edge capacity		Congestion increased 15% above edge capacity	
Node# -LT-TM	Initial Congestion	New Edge Required	Final Congestion	New Edge Required	Final Congestion	New Edge Required	Final Congestion
14-1-1	0.725356	7	0.898144	9	0.899189	14	0.888424
14-1-2	0.673129	7	0.877986	10	0.898554	11	0.896345
14-1-3	0.637555	11	0.896655	12	0.888146	14	0.884330
14-1-4	0.683989	9	0.870337	10	0.885141	12	0.894790
14-1-5	0.706597	8	0.878088	9	0.899663	10	0.896920
14-2-1	0.683323	9	0.891054	14	0.811109	19	0.897986
14-2-2	0.664394	9	0.864604	10	0.874732	45	0.779807
14-2-3	0.664931	8	0.859576	16	0.796997	22	0.833224
14-2-4	0.691406	10	0.806205	15	0.799624	27	0.814312
14-2-5	0.646388	8	0.891809	24	0.886336	25	0.776679
14-3-1	0.836805	1	0.899194	4	0.888590	8	0.894710
14-3-2	0.695234	12	0.845602	15	0.893997	39	0.745216
14-3-3	0.694762	8	0.887799	11	0.872763	17	0.894513
14-3-4	0.688836	9	0.897081	13	0.897798	16	0.894746
14-3-5	0.708334	17	0.862940	20	0.864779	31	0.884243
14-4-1	0.730841	6	0.899869	9	0.887521	11	0.898190
14-4-2	0.682589	7	0.893269	10	0.879248	16	0.895029
14-4-3	0.656250	10	0.888272	21	0.993198	16	0.876190
14-4-4	0.670139	9	0.897668	11	0.885884	17	0.896336
14-4-5	0.674306	9	0.855431	10	0.887110	15	0.860676
14-5-1	0.836805	9	0.862656	10	0.896889	16	0.878009
14-5-2	0.800000	8	0.875000	16	0.866339	20	0.862500
14-5-3	0.763889	7	0.822273	15	0.825000	34	0.730501
14-5-4	0.736458	8	0.888119	14	0.847950	23	0.842575
14-5-5	0.746817	11	0.870792	52	0.782488	50	0.818005
Ave. edge required		8.68		14.4		21.12	

Vita Auctoris

NAME:	AKM Aktaruzzaman
PLACE OF BIRTH:	Natore, Bangladesh
YEAR OF BIRTH:	1966
EDUCATION:	Jigori High School, Natore, Bangladesh. (1977-1982)
	Electrical Engineering, Polytechnic Institute, Pabna, Bangladesh, (1982- 1985)
	Department of Computer Science, University of Windsor, Windsor, Ontario, Canada, B.Sc. (Honourse), in Computer Science. (2002-2006)
	Department of Computer Science, University of Windsor, Windsor, Ontario, Canada, M.Sc. in Computer Science. (2007-2010)