1999

Design of fault-tolerant optical network.

Sanchita. Mal-Sarkar
University of Windsor

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Design of Fault-tolerant Optical Network

by
Sanchita Mal-Sarkar

A thesis
Submitted to the College 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
1999

Abstract

With the recent growth of telecommunication systems, fault management has received much importance. As networks grow in size and complexity, the amount of disruption caused by a network-related outage becomes more and more significant. Several techniques exist to ensure that networks can continue to provide reliable service even in the presence of faults. The basic principle of fault tolerance is redundancy beyond the minimum requirement for normal operation. These spare resources help avoid faults. Since any redundancy introduced in the system to make it fault-tolerant increases the system cost, it is very important to determine the right type and extent of spare resources to maximize the reliability while maintaining a moderate increase in cost. Intensive theoretical research work has been done on the notion of survival route graphs based on graph theory. However, the present work is the first attempt to realize the applicability of survival route graphs in fault-tolerant optical networks.

In this thesis, we have proposed and simulated two fault-tolerant schemes using the De Bruijn graph as a network physical topology where we treat some special nodes as spare resources. In our first scheme, we have attempted to design an all-optical single-hop fault-tolerant network. However, when we tested this scheme by randomly generating faults, it could not provide alternate paths in the presence of faults in most cases. Our second scheme, a hybrid system of single-hop and multihop system can successfully manage multiple faults almost in all cases. However, the disadvantage of our second scheme is that our network is no longer an all-optical network, and its throughput is lower because of electrical buffering at some intermediate points.

Another disadvantage of this scheme is that the system can result in intolerable restoration delay for some communications due to the high demand on certain links since there is no upper limit on the number of communications through any edge. We have tried to distribute the communications through different links as uniformly as possible to improve this delay. The advantage of this approach is that we need to distribute the communications only once when any edge happens to be faulty. The rate of success of our second scheme is statistically significant.

Dedicated to the memory of my father

Acknowledgments

I express my sincere gratitude to my supervisor Dr. S. Bandyopadhyay for introducing me to this field and being my supervisor over the years. His patience, guidance and support throughout this thesis helped me a lot. I would like to acknowledge my other committee members: Dr. Christie Ezeife and Dr. Hon K. Kwan for their critical reviews and fruitful discussions that improved the quality of this thesis.

I would like to thank Mr. Walid Mnaymneh who introduced me to the concept of networks and has provided all technical help whenever I needed. I would like to thank all the graduate students for their help, encouragement and support. I extend special appreciation to Mary Mardegan, Margaret Garabon, and Gloria Mensah for their generous help.

I am extremely grateful to the Villanova University where I did most of my thesis work. They provided me all computing facilities and, without their help, it would be hard to complete the thesis.

I would also like to acknowledge the help and support I got from Mrs. Bharati Bandyopadhyay and Mr. Dan Boles during this period. I sincerely appreciate the encouragement and inspiration from Prof. Tapas K. Modak who first instilled the joy of learning in me.

Finally, I would like to thank my family members, my parents, brothers, sisters, for their constant encouragement and support. Special thanks goes to my husband, Tarun for enduring me and for statistical help.

My daughter, Tatini and her sparkling smile is always my source of inspiration. I couldn't have finished the work without her.

Table of contents

List of tables

List of figures

Chapter 1

Introduction

1.1 Background

With the increasing dependence on telecommunications, our requirement for high speed communication has increased dramatically. Today's Internet and asynchronous transfer mode (ATM) networks can not meet our bandwidth demands. Fiber optic technology has a potential to satisfy our demands because of its huge bandwidth (nearly 50 terabit per second), low signal attenuation, low signal distortion, low power requirement, low material usage, small space requirement, and low cost [33]. Optical fiber cable carries messages in the form of a fluctuating beam of light - electromagnetic radiation in the visible and near-visible spectrum using glass fiber as the medium.

Since the maximum rate at which an end-user can access the network is limited by the electronic speed (few Gbps), to exploit the huge bandwidth of fiber optical communication network architectures and protocols should be designed to introduce concurrency among multiple user transmissions. This concurrency may be achieved according to their wavelength or frequency (WDM), time slots (TDM), space (SDM), or code (CDM) [24]. At present, WDM is preferred over other multiplexing technologies since all of the end-user equipment need to operate only at the peak electronic processing speed where as in TDM and CDM, some part of an end user's

network interface must operate at a rate higher than electronic speed.

Wavelength division multiplexing (WDM) provides the way to partition the optical bandwidth into a large number of channels operating at different carrier frequencies on a single optical fiber[17]. Multiple users, spread over a geographical area, can use the optical fiber simultaneously using different wavelengths and leading to a dramatic increase in the total capacity of the network. It is anticipated that the next generation of the Internet will employ WDM-based optical backbones [33].

An optical WDM network requires optical switches. An optical switch can be either an opto-electronic switch or an all-optic switch. In an opto-electrical network, there is an opto-electronic conversion resulting in a low bit rate. In all optical networks, messages are transmitted and processed entirely in the optical domain, ensuring a high bit transmission rate. The advantages of all optical networks include transparency, reduced processing, and reduced management [15]. The next generation of optical network will use all-optic switches. However, an all-optic switch has an important limitation of signal attenuation. Therefore, the number of switches should be as small as possible.

There are several approaches to implement optical WDM networks: single-hop, multihop, and wavelength routing [17]. In single-hop networks, messages are routed from the source to the destination in one hop without being processed at intermediate points, that is, there is no electro-optical conversion [30][33]. An all-optical network requires a significant amount of dynamic

coordination between the source and the destination nodes [31][39]. The tuning ranges of current optical transceivers are limited, and their tuning times are also significantly long compared to the duration of packet transmission. For these reasons, the total number of wavelengths used and the maximum number of users are limited in single-hop networks.

In multi-hop networks, messages are routed from the source to the destination in several hops and are processed at the intermediate points, that is, there are several electro-optical conversions resulting in a low bit rate and therefore, these are not all-optical networks[31][33] [48]. The average hop distance of this network should be kept as small as possible since the longer the hop distance, the longer the time a message stays in the network. Thus more network resources are consumed and the resulting throughput becomes smaller. Since high-speed networks do not allow long processing time, the routing scheme should be very simple and packet buffering should be minimized. In multi-hop networks, the wavelengths to which a node's transmitters or receivers are to be tuned are fixed and the transceiver tuning times do not play an important role to determine the performance. Another advantage of a multihop network is the independence of physical topology and logical topology.

A different approach to implement all-optical WDM networks is wavelength routing [17] [40][41][42]. It is defined as the selective routing of optical signals according to their wavelengths as they move from the source to the destination. Another advantage of this approach is that we can reuse each wavelength on different paths as long as these paths do not coexist on the same fiber to avoid interference. This is the fundamental requirement of the wavelength

routed optical network [33].

An optical network can be modeled as a directed graph where each node of the graph represents a network node [2] [9] [12][19][20]. There is an edge from node X to node Y if station X can directly transmit to station Y. A message may have to hop through several intermediate nodes before reaching its destination. A node is equipped with a number of transmitters and receivers. The transmitters and receivers can be tuned to operate at different wavelengths. If the transmitter of a node is tuned to many wavelengths, the node can transmit using those wavelengths. Similarly, if the receiver of a node is tuned to those wavelengths, the second node can receive information from the first node. Since we can use a limited number of transmitters or receivers, the number of nodes that a node can transmit to is also limited.

1.2 What is a fault-tolerant optical network?

Fault-tolerance in an optical network can be studied in terms of two system attributes: reliability and survivability. Network reliability is the ability of a system to perform its function, the lower the probability that a system will fail to perform its function, the more reliable it is. Network survivability is the ability of a network to maintain an acceptable level of performance in the presence of network failure by employing various restoration techniques [43].

The basic principle behind a fault-tolerant system is to provide the system with redundant

resources beyond the minimum requirements for normal operation. These spare resources help avoid faults. Since any redundancy introduced in the system to make it fault-tolerant increases the system cost, it is very important to determine the right type and extent of spare resources to maximize the reliability while maintaining a moderate cost. Spare resources for every possible scenario is not a feasible solution since it would be prohibitively expensive. Therefore, it is a challenge for network planners and engineers to maintain a satisfactory level of survivability with minimum cost.

Different users need different level of survivability depending on an application's cost, service, and performance requirements, optimizing some attributes at the expense of others. A single network scheme may not be a satisfactory solution to all users. Therefore, the aim of a network designer is to combine several restoration techniques to meet different demand requirements and to realize a simple, efficient, fast, cost-effective fault-tolerant optical network.

Network restoration technique can be protection switching, rerouting, or self-healing. In protection switching, the new route is known in advance depending on the equipment residing in either the connecting or terminating points of the path and it is not controlled by the Operating System / Network Management System (OS/NMS) [50]. Example includes APS and self-healing ring [53].

In rerouting, the new route is generally not known in advance and it is established depending on the network resources available at the point of failure. This is controlled by the Operating System

/ Network Management System (OS/NMS) [50]. An example includes the centralized control DCS network restoration [53].

In self-healing, the new route is not known in advance. The new route is established depending on the network elements and resources available at that point of failure and is not controlled by the Operating System / Network Management System (OS/NMS) [50]. Self-healing allows reconfiguration of a network around failures quickly without disturbing the calls that are not affected by the network failures [25]. It requires redundant facilities and intelligence in the network. Some reconfiguration techniques do not require any spare resources and eliminate the faulty component and run the system on the remaining resources with degraded performances. They are useful where repair of failed components are not possible (e.g. unmanned spacecraft) [37].

1.3 Motivation - Why do we need a fault-tolerant optical network?

With the increase of the network's speed and flexibility, the possibility of faults in optical network increases. Communication may fail due to cable cuts, hardware malfunctions, software errors, natural disasters (e.g., floods, fires, etc.), and human error (e.g., improper maintenance). Whenever there is a fault in the network, a number of routes will be unusable resulting in a loss of service to users. This can be frustrating to researchers and other users, and can cause a tremendous loss of revenue to operating companies. Since many of the causes of communication

failures are beyond the control of the network providers, it is very important to design an optical network that are inherently fault-tolerant against events such as cable cuts, central office failures, and hardware failures. The goal is to continue communication even in the presence of faults. A fault-tolerant network provides alternate routes between a pair of nodes in case of fault. The objective of setting a level of survivability is to ensure that network performance will not degrade below a predetermined level.

1.4 Work done in the thesis

The main purpose of this thesis is to design a call rerouting mechanism which provides restoration of network services in the case of link failures. Our focus is on the design of optimized networks in which the rerouting scheme to avoid fault is as simple as possible. The routing scheme is based on the notion of survival route graphs, derived from graph theory [5]. If any node or edge in a normal route between a source and a destination fails, the normal route will be unusable. To ensure reliable communication between the source and the destination, we have to find an alternate route that does not have any faulty node or edge.

The basic idea of this fault-tolerant scheme is to provide the network with some special nodes : *concentrator*, *predecessor*, and *successor nodes*, as spare resources. These spare resources will be used only when any node or edge fails. The idea of concentrator node was introduced in [35]. A set of nodes that satisfy the following conditions are called concentrator nodes [45].

- There is no edge from one concentrator node to another concentrator node.
- No two concentrator nodes can have the same predecessor node.
- No two concentrator nodes can have the same successor node.
- The predecessor of one concentrator node can not be the successor of another.

A set of nodes that have edges to a particular concentrator node are called the predecessor nodes of that concentrator node and if a concentrator node has edges to a set of nodes, that set of nodes are called successor nodes of that concentrator node. If a normal route between a source and destination fails, we have to look for

- a path from the given source to a predecessor node that does not have any faulty edge,
- a path from the predecessor node to a successor node through the concentrator node that does not involve any faulty edge, and
- a path from the successor node to the given destination that does not have any faulty edge.

If we are successful in finding the above three paths, we can replace the original light path from the source to the destination by a composite lightpath consisting of lightpath segments from source to predecessor node, predecessor to successor node through concentrator node, and successor node to destination.

In our first scheme, the number of communication through the physical edge from a source (or successor) to any predecessor (or destination) is limited to one. Once that edge has been already

used to take care of some faulty lightpath, it can not be used to handle another faulty path. Therefore, our first scheme will fail to provide alternate path in most cases in the presence of failures.

In our second scheme, more than one communication can use any link from a source node to any predecessor node, any predecessor node of a given concentrator node to the successor node of that concentrator node, and from any successor node to any destination node. Therefore, our scheme is fault tolerant against any number of faults. However, the high demand on certain links can cause an unbalanced network in which some edges and nodes are heavily loaded while others remain under-loaded resulting in an intolerable restoration delay for some communications. To minimize this queuing delay, we have tried to distribute the number of communications (load) throughout the network as uniformly as possible by employing an adaptive rerouting algorithm that introduces additional computational requirements. The advantage of our approach is that we need to uniformly distribute the communications only once when any edge happens to be faulty. We have coded all optimization algorithms in the 'C' programming language. Our scheme will be very useful for networks and services that require higher level of survivability but can endure delay due to rerouting.

1.5 Structure of the thesis

This paper is structured as follows.

In chapter 2, we have reviewed related literature for this thesis. We have described some network topologies, different approaches to implement WDM networks, types of faults, several fault-tolerant architectures, and survival route graphs.

In chapter 3, we have described our design of fault-tolerant WDM networks. We have discussed the design topology, introduced the notion of *deflector nodes* (spare resources), the fault-tolerant routing schemes proposed and simulated in this thesis and explained why our first scheme failed in most cases. We have also given examples of alternate paths through the special nodes of the network in the presence of failures using our second scheme.

In chapter 4, we have given simulation results with critical comments. We have provided the maximum number of communications through different edges for different runs. Using SYSTAT, we have plotted histograms for the distributions of maximum number of communications through different edges and calculated the mean, median, variance, and standard deviation for each case.

We have concluded with a critical summary and proposed possible future directions for the further development in section 5.

In Appendix-A we have provided the C source code for our second fault-tolerant optical network, in Appendix-B we have presented some of the results, in Appendix-C we have given glossary of terms and abbreviations. It is followed by the bibliography.

Chapter 2

Review of related literature

2.1 Topology - physical and logical

The topology of a network defines how the nodes of a network are connected. A network is defined by a physical topology and a logical connectivity graph among the nodes [40][42]. Physical topology (e.g. Broadcast star or bus) provides the physical connections between every pairs of nodes in the network. Logical topology provides dedicated connections between certain selected source-destination pairs using the underlying physical topology.

In a network, the stations (end-nodes) are interconnected by optical fibers via routers. Each fiber can carry a number of signals modulated at different wavelengths. Each usable wavelength on a given fiber is called a channel. This network supports lightpaths which are end-to-end circuit switched communications traversing one or more fibers using one WDM channel per fiber [33][43]. The logical topology of a network is a graph where the node of the graph corresponds to the end-node of the network and two nodes A and B are connected by a directed edge from A to B if there is a lightpath from the end-node A to the end-node B.

The major factors to design the topology of a network are the reliability, fault-tolerance, the

message routing delay, the routing schemes. To design a good topology, fault tolerance should be maximized while minimizing the message routing delay. However, fault-tolerance is always at least one less than the minimum number of connections per node in case of bidirectional links and by one less than the minimum of number of incoming and outgoing connections per node in case of unidirectional links. A network will be optimally fault-tolerant when the fault tolerance reaches this upper limit. A topology is said to be a regular topology if the number of connections for each node is the same throughout the network. In regular topology, it is easy to implement routing scheme because of its regular structure but it is not optimally fault-tolerant [31].

An interconnection network can be modeled as a directed or undirected graph G in which the nodes represent processors and the edges represent communication links. G is a directed graph if the communication link is unidirectional and it is undirected if the communication link is bidirectional. A communication link is bidirectional if for any ordered pair (x, y), the route from x to y and the route from y to x are assigned to the same path. Directed graph as a network model has received attention as optical fiber is unidirectional.

2.2 Different approaches to implement WDM network

2.2.1 Single-hop Systems

In a single-hop system, a message is transmitted from a source node to a destination node in one

hop without being routed through different end-nodes of the network and is communicated in the optical medium all along the way. Since it is infeasible to have separate transmitters and receivers for different wavelengths at all nodes, it generally requires tunable transmitters and/or tunable receivers to provide connection on demand [22]. There are two major problems to design a good single-hop network: tuning time of transceiver and lack of efficient mechanism to establish dynamic coordination between a pair of nodes that are wishing to communicate so that at least one of the transmitters of the source node and one of the receivers of the destination node are tuned to the same wavelength to confirm transmission [1] [31]. Many multiple access schemes are proposed [30], however either they cannot satisfy the efficiency requirements, or they solely depend on rapidly tunable transceivers [10]. A more realistic approach is described in [22][23].

In a broadcast-select network, the inputs from all transmitting nodes are combined in a star coupler and broadcast to all end-nodes. It can be implemented in different ways depending on the tunability of lasers and the receivers: tunable laser and fixed receiver, fixed laser and tunable receiver, fixed laser and fixed receiver, and tunable laser and tunable receiver [30]. If the input lasers are made tuned to fixed wavelengths and the output receivers are tuned to fixed wavelengths, the number of users will be limited but a control channel is not required.

If the input transmitters are made tunable but the output receivers are tuned to fixed wavelengths, the broadcast-select architecture will support only point-to-point connection and it would not support multi-cast (point-to-multipoint) connections. If the input transmitters are tuned to fixed

wavelength but the output receivers are made tunable, the broadcast-select architecture will support multi cast connections [7].

If both the transmitters and receivers are made tunable, the number of wavelengths required may be reduced but there may not be enough wavelengths available to support simultaneous N x N connections resulting in network or switch blocking in the wavelength domain [7]. It is observed that network with fixed lasers and tunable receivers are more useful than others [10].

However, the scalability of this network is limited to the number of supported wavelengths as it does not support reuse of wavelengths. Due to physical constraints (e.g. crosstalk, filter resolution, requirement of high-speed tunable filter), the number of resolvable wavelengths are also limited [18], [8]. Another disadvantage is that the power from each transmitter is broadcast to all receivers. Mostly it is wasted as many receivers are not using it. For these reasons, the use of such network is limited to high-speed local and metropolitan area networks.

To compensate for the power losses due to splitting, amplifiers are required in this network. However, the maximum power that it can supply is also limited. To minimize the number of optical amplifiers, most of the studies have addressed the issue by adding the constraint that all wavelengths, present at a particular point in a fiber, will be at the same power level. Ramamurthy et al. in [38] have proposed a method that allows the different wavelengths on the same fiber to be at different power levels. Though the method will increase the complexity but many networks, specially smaller networks will be benefitted by requiring fewer amplifiers. Various experimental

demonstrations and prototypes of single-hop WDM networks are LAMDANET, RAINBOW, FOX, HYPASS, STAR-TRACK.

2.2.2 Multihop network

In a multi-hop system, the message from a source to a destination may have to hop through zero or more intermediate nodes. In this system, the wavelength to which a node's receiver and transmitter will be tuned generally does not change and the tuning time is not as important as in the case of a single-hop system [31][24]. An important property of the multihop scheme is the relative independence between the logical topology and the physical topology [24]. In order to have an efficient system, the logical topology should be chosen such that either the average hop distance or the average packet delay or the maximum flow on any link must be minimal. Another important issue is the simplicity of routing[34] .

An example of multi-hop system is shown where the physical topology is a star (Fig 1a) and the logical topology is a 2 x 2 torus (Fig 1b). Topology of a network defines how the nodes of a network are connected. Physical topology provides the physical connectivity patterns, whereas the logical topology defines the ability to communicate from one end-node to another end-node by a single lightpath. In Fig 1b, node 1 can send information to node 2 and 3 directly using frequencies $\omega 1$ and $\omega 2$. For this reason, node 1 is logically connected to node 2 and node 3, although they are not directly physically connected. However, if node 1 wants to send

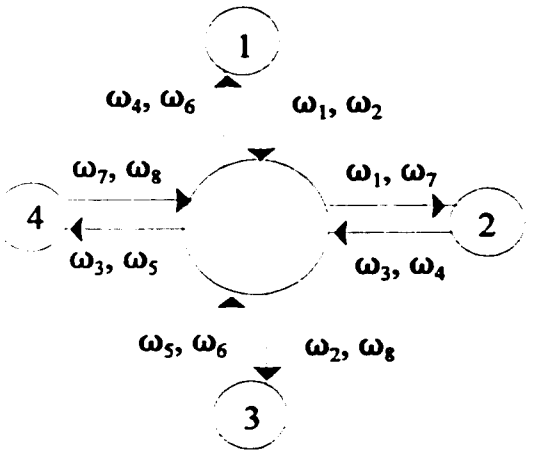


Fig.1a: Physical topology of a 2 x 2 multi-hop network.

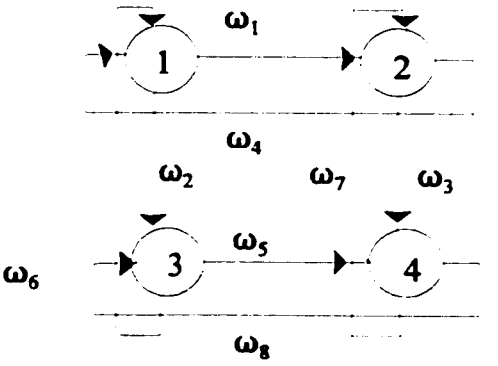


Fig.1b: Logical topology of a 2 x 2 multi-hop network

information to node 4, it has to multi-hop through either node 2 or node 3 since node 1 is not directly logically connected to node 4. If node 1 communicates with node 4 via node 2, there will be a wavelength conversion from ω1 to ω3 at node 2 and which is done electro-optically i.e the signal is converted from optical domain to electrical domain and reconverted to optical domain after wavelength conversion resulting in lower bit rate. This is the main disadvantage of multi hop network.

Another disadvantage is that power is split to irrelevant destinations which does not support long distance transmission. But using this network, we can reduce the number of required wavelengths for communication. To overcome the power splitting problem, wavelength selective devices can be used [21]. Another concern is that load imbalance might be serious problem with the increase of network size. However, in [14], the authors have shown that imbalance load on

multihop network reduces throughput per user to a factor which is not very sensitive to the network size.

Multi-hop systems can be of two types : irregular and regular [31]. In an irregular multi-hop system, it is easy to address the optimal problem and it can be optimized for arbitrary loads, but the problem is its routing complexity as it does not have regular structured node connectivity pattern [31]. On the other hand, in regular multi-hop system, routing strategy is very simple due to its regular structured connectivity but the problem is to achieve optimal condition and generally it can be optimized for uniform loads due to its regular structure. Another disadvantage of complete regular structure includes the number of nodes that it can support will be a discrete set of integers, instead of an arbitrary integers. Regular multi-hop system has received more attention because of its simple routing.

A number of different regular multi-hop architectures, having different operational features and different performance characteristics, are possible : ShuffleNet, Manhattan Street Network (Torus), Hypercube, and De Bruijn graph. We have employed the De Bruijn graph as a physical topology for developing fault-tolerant optical network because of its small diameter and constant distant paths which reduces the signal dynamics at the receiver and simplifies the receiver implementation [29].

De Bruijn graph

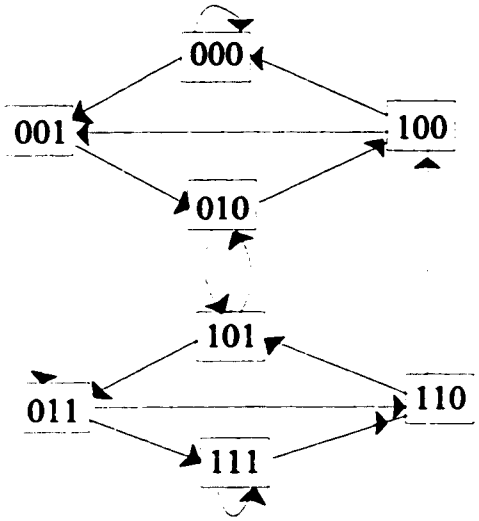


Fig. 2: A (2, 3) De Bruijn graph

A directed De Bruin graph has d^k nodes for some positive integers d and k, each node has d directed edges [31]. It can be considered as the state transition diagram of a k stage d-ary shift register. There is a one-to-one correspondence between all possible states of shift register and the nodes of the De Bruijn graph. There is an edge from node x to node y, if node x can reach node y by one shift. The nodes of the graph can be numbered as 0 through d^k - 1 and node i has arcs directed to node j if $j = id + (q \bmod d^k)$, where $0 \leq q < d$. Using a d-radix representation, any number i can be represented by $i_0 i_1 i_2 \ldots\ldots i_{k-1}$, where $0 \leq k < d$ and i_0 is the most significant digit, i_{k-1} is the least significant digit.

An example of (2,3) De Bruijn graph is shown in fig. 2, where the graph is constructed out of 8 nodes, numbered as 0,1,2,3, ..., 7. Using d-radix representation, the nodes are denoted by 000, 001, 010, 011, 100, 101, 110, 111. The problem to determine a route from any source $s = s_0 s_1 s_2 \ldots s_{k-1}$ to any destination $d = d_0 d_1 d_3 \ldots d_{k-1}$ is explained in [28].

The constant length path from any source $s = s_0\ s_1\ s_2 \ldots s_{k-1}$ to any destination $d = d_0\ d_1\ d_3 \ldots d_{k-1}$ will be : $s_0\ s_1\ s_2 \ldots s_{k-1} \rightarrow s_1\ s_2 \ldots s_{k-1}\ d_0 \rightarrow s_2 s_3 \ldots s_{k-1}\ d_0 d_1 \rightarrow \ldots\ldots\ldots \rightarrow s_{k-1}\ d_0 d_1 d_2 \ldots\ldots d_{k-2} \rightarrow d_0\ d_1\ d_3 \ldots d_{k-1}$. For instance, the constant length path between the source node 001 and the destination node 100 is given by: 001→011 →110 →100. The constant length path is not the longest route.

To determine the shortest route between the same source and destination node, we have to determine j such that, $d_0\ d_1 \ldots . d_{j-1} = s_{k-j}\ s_{k-j+1} \ldots . s_{k-1}$. If such a j exists, the route will be : $s_0\ s_1\ s_2 \ldots s_{k-1} \rightarrow s_1\ s_2 \ldots s_{k-1}\ d_j \rightarrow s_2 s_3 \ldots s_{k-1}\ d_j d_{j+1} \rightarrow \ldots\ldots\ldots \rightarrow s_{k-j}\ s_{k-j+1} \ldots s_{k-1}\ d_j\ d_{j+1}\ d_{j+2} \ldots . \ldots\ldots d_{k-3}\ d_{k-2}\ d_{k-1}$. The shortest path for the source node 001 and destination node 100 is given by : 001 →010 → 100. In this case j exists and the value of j is 1. If such a j does not exist, the first intermediate node will be $s_1\ s_2 \ldots s_{k-1}\ d_0$, the second intermediate node will be $s_2 \ldots s_{k-1}\ d_0\ d_1$,...., the $(k-1)^{th}$ intermediate node will be $s_{k-1}\ d_0\ d_1\ d_3 \ldots d_{k-2}$ and the shortest route and constant length route will be the same. For instance, the shortest path and the constant length path from source node 001 to destination node 000 will be 001→ 010 → 100 → 000.

The diameter of the network is $\log_k N$, where N is the number of nodes in the network. It has an inherent asymmetry in the structure due to the existence of self loops "000" and "111" which are wasted as they carry no traffic [34]. Due to this asymmetry, even if the offered load in the network is fully symmetric, the link loads can be unbalanced resulting in lower throughput. However, for a given diameter, a De Bruijn graph supports more nodes than ShuffleNet [47].

2.2.3 Wavelength Routed Network

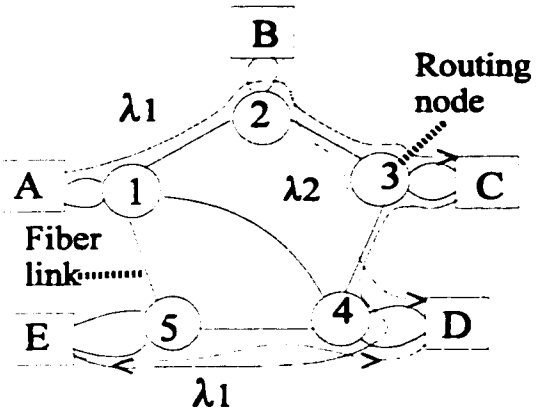


Fig. 3:An all-opticalwavelength-routed network

Wavelength routed networks have been most commonly used in WDM networks [17] and it is considered to be the potential candidate for the next generation of wide-area network [40]. It is composed of some form of wavelength-selective elements at the nodes of the network [8]. Such node (called wavelength router) makes its routing decision based on the input port and the wavelength of the signal passing through it [32]. Wavelength routing is achieved by demultiplexing the different wavelengths from each input port and then multiplexing signals at each output port. Sometimes switches are used between demultiplexers and multiplexers.

For example, an N x N network is shown in fig. 3 [40][41]. Network consists of N tunable laser sources , N (wavelength independent) receivers, and a WDM network that interconnects sources and receivers. If a transmitter is tuned to a particular wavelength, the signal can be routed from that end-node to another end-node of the network which has a receiver tuned to the same wavelength. Such an end-to-end connection is called a light path. It is not generally feasible to provide lightpaths between every pair of nodes due to the physical constraints such as limited

number of wavelengths, limited number and tunability of optical transceivers and light wave dispersions [11].

Since another light path can reuse the same wavelength in some other part of the network, at the same time, N inputs can be interconnected to N outputs by using only N wavelengths instead of N x N wavelengths as expected in broadcast-and-select network [6]. If node A wants to communicate with node C using the wavelength channel $\lambda 1$ all along the way, there must be an unassigned WDM channel for each edge in this communication. Similarly, if node C wishes to communicate with node E using the same wavelength channel $\lambda 1$, there must be an unassigned WDM channel for each edge involved in the communication. Therefore, two simultaneous communications using the same WDM channel are possible as long as they do not coexist on the same fiber. However, for the communication between node B and node D, we can not use the same wavelength $\lambda 1$ since it involves edges that have been already used by other communications.

A wavelength routed network is more scalable than a broadcast-and-select network due to this spatial reuse of wavelengths. It reduces the calculation of optimal rerouting [26]. Another advantage associated with this network is that the energy invested in a light path is not split to irrelevant destinations which supports long distance transmission. Wavelength routing networks can be designed in different ways depending on the components in use and design of the node.

Wavelength router can be nonreconfigurable [33] and reconfigurable [6][40]. In

nonreconfigurable wavelength router, there is no switching between the demultiplexer and the multiplexer and the routes for different signals arriving at any input port are fixed [6]. The wavelengths on each incoming fiber are separated by grating demultiplexer and then the wavelengths from various inputs are recombined to a single output by a multiplexer before it is fed into an outgoing fiber. Different wavelengths from different input ports will be routed to different output ports depending on a "routing matrix". This matrix gives the internal "connections" between the demultiplexers and the multiplexers.

In a reconfigurable wavelength router, there are optical switches between a demultiplexer and a multiplexer and the routes for different signals arriving at any input port may be reconfigured to adapt to changing traffic patterns [6][33]. Such networks are more flexible than passive, nonreconfigurable wavelength routed networks as they have more control over routes. In [3], Banerjee and Mukherjee have presented an exact linear programming formulation to provide a reconfigurable methodology based on constituent lightpaths. The solution to the reconfiguration algorithm generates a virtual topology which minimizes the amount of reconfiguration that is needed to adapt the virtual topology of the new traffic matrix.

A node in a wavelength routed network can be designed in three different ways: electro-optical nodes, all-optical nodes, full-conversion all-optical nodes [15]. Electro-optical nodes convert the signal from optical domain to electrical domain, do the switching in this domain, and reconvert the optical signal at the outputs. This design supports wavelength conversion but does not support transparency. All-optical node separates the wavelengths of incoming signal and sends

them to different switches depending on their wavelengths and finally switches send them to the output ports. This design does not support wavelength conversion and thus wavelength can not be reused in the network. But the advantage is that this design is cost-effective as it does not require expensive transceiver per channel per node [15][32]. Full-conversion all-optical nodes can convert each wavelength from each input to any other wavelength. It is predicted that wavelength routed networks will become a commercially viable solution in few years and very gradually replace high-speed SONET networks [15].

2.3 Faults in optical network

There are mainly three types of faults : channel fault, link fault, and node fault [16].

2.3.1 Channel fault

A channel fault occurs when a single wavelength channel on a link between two nodes has failed due to the failure of the laser or receiver for that wavelength channel, or due to cable disconnections. This fault can be managed by routing the traffic to a spare channel on the same physical link and bypassing the faulty channel.

2.3.2 Link fault

A link fault occurs due to fiber cut or due to noise, jitter etc. (particularly those running over long distance) and can be managed by using a bundle of protection fibers in addition to the working fiber. The performance or survivability is limited by the number of links in each bundle. Another way of solving this problem is by providing a "loopback" mechanism within each node on the same working fiber.

2.3.3 Node fault

A node fault occurs due to power outages or catastrophic failure resulting in an entire WADM node failure. It is complicated to handle since when this fault occurs, a part of the conversion capability of the network as a whole is lost and leaves an open circuit and can result in deadlock of all routing networks. One solution is to introduce redundancy in the internal connections of the networks rather than in the connections of nodes. In the case of multiple failures, the solution is a "firewall" that can prevent deadlock spreading in the network. This can be achieved by a simple device that times out if a packet passing through it stay longer than a specified time.

2.4 Fault-tolerant network architectures

Several fault-tolerant optical architectures are explored in recent few years: automatic protection

switching (APS/DP), dual-homing, self-healing rings (SHRs), and dynamically path rearrangeable mesh architecture. Some architectures e.g. APS and rings have already been implemented in local exchange carrier (LEC) networks. The emerging technologies to implement the above architectures are SONET, ATM, and passive optical technology (optical switching and Wavelength Division Multiplexing(WDM)). This thesis is based on passive optical technology (WDM).

Passive optical technology is a proposed concept for potentially reducing the cost of the survivable network. It is in its infancy because the technology is not totally available. It may restore service very quickly compared to SONET and ATM layer protection. There are several techniques to implement fault-tolerant optical network in passive optical technology: Passive Protected APS, Passive Protected Self-Healing Rings, Passive Protected DCS Mesh Networks.

2.4.1 Passive Protected APS

Conventional (SONET and ATM) 1:1 diversely protection technique is very expensive, since it requires electronics equipments in addition to duplicate fiber facilities. To reduce the protection cost, 1:1 optical diverse protection (1:1/ODP) architecture was proposed in [49]. This architecture employs optical switches for 1:1 fiber cable protection and maintains 1:N electronic protection using a 1:N APS system. It is shown in [49] that the use of 1:1 ODP architecture can reduce the protection cost of the traditional 1:1 DP architecture by about 70 percent. However,

the survivability may be less in case of 1:1 ODP architecture, if multiple simultaneous components fail. This type of passive protected APS systems is commercially available.

2.4.2 Passive Protected Self-Healing Rings

Conventional four-fiber bidirectional self-healing ring (BSHR/4), despite its many advantages, is relatively expensive compared with other SONET ring architectures as it requires duplicate ADMs in each node. By using passive optical technology, the protection cost can be greatly reduced, while retaining the other advantages of BSHR/4. In this architecture, one SONET ADM is required for each working ring, while optical switches and optional optical amplifiers (in case of long protection ring) are needed for protection rings [53]. Optical switches basically eliminate the use of duplicate ADMs. It acts as optical "add-drop" component at the two ends of the failed facility while, in the intermediate ring nodes, it acts as optical "pass through" components. Optical signal carried on the protection ring is added and dropped at two different optical switches according to SONET ADMs on the working ring. The disadvantage of this type of architecture is that it may be less survivable than the conventional four-fiber BSHR in case of simultaneous multiple failures.

2.4.3 Passive Protected DCS Mesh Networks

Existing DCSs and cross-connect technology can not meet SONET two second restoration

objective. It is reported in [51] that significant modifications of existing DCSs are required to achieve the goal which include parallel DCS processing, parallel cross-connect architecture, and ultra fast cross-connect technology. It is suggested in [52] that about 30 percent extra cost will be added due to spare capacity. For these two reasons, this technology is not very useful in case of fast restoration. Passive optical technology can provide a better solution.

One example of a passive protected DCS mesh network has been proposed in [52]. In this network, a passive optical system is used as a protection system whose reconfiguration is performed in the all-optical domain and is controlled by electronic network components. An inexpensive special optical cross-connect system (OCS) at each node is the main component in the passive protected DCS networks. It eliminates the cost associated with the modification mentioned above. It is using a single restoration path approach, while conventional systems (SONET and ATM) are using multiple restoration paths. Another advantage is that it has fast optical switching time (few milliseconds). A new optical protection DCS self-healing architecture that uses WDM technology at the ATM VP layer has been reported in [44].

2.5 Survival route graphs

The notion of survival route graphs is based on graph theory and standard terminologies are defined in [4]. A routing $\rho(x,y)$ provides a fixed path to each ordered pair of nodes in the network [12]. A routing $\rho(x,y)$ is said to be minimal if any route from x to y is assigned to one of

the shortest paths from x to y. Any two nodes communicate with each other via the fixed route

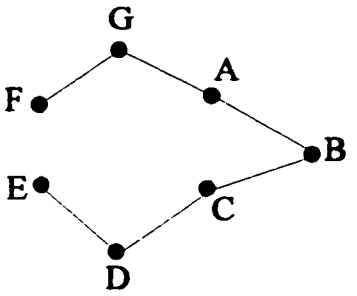


Fig. 4: A network

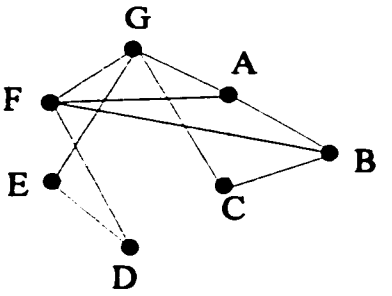


Fig. 5: Survival route graph

$\rho(x,y)$ between them as long as no node or link in this route is faulty. If there happens to be a faulty node or edge in this route, then they can no longer use this route for communications. However, if there exists another node p such that the path from x to p and the path from p to y are both working, then it is possible to send information from x to y via p using the composite path consisting of the path from x to p and then from p to y. This strategy is very useful to recompute an alternate fault-free path [19]. It is very important to make sure that the number of paths to be concatenated is small so that the delay in communication does not become too long. Dolev et al. [12] have formalized this problem in the following way.

Dolev et al. [12] defined the survival route graph as a digraph that contains all the non-faulty nodes of the original digraph with an edge from node x to node y if there is no faults on the path $\rho(x,y)$. For instance, let us consider a network where we consider minimal length routing (shown in fig 4). If edge CD becomes faulty, then all routes passing through that edge will be unusable.

Fig 5 shows the survival route graph where two nodes are connected if and only if the path between them did not pass through the faulty edge CD. Now if station C wants to broadcast a message to all stations, the message will reach A, B, G only and it will not reach D, E or F since node C can send message only along the fixed routes. If G rebroadcast the message, station E and F will get the message. If D wants to get the message, F or E has to rebroadcast the message. In the worst case, the number of rebroadcast needed to reach the message to all stations is the diameter of the survival graph.

The diameter of a survival route graphs gives the upper bound on the number of routes to be required in any communication between two fault-free nodes. If diameter is high, it may cause signal attenuation and network congestion. The fault-tolerance of this network is one less than the node connectivity of the graph and the maximum message routing delay is equal to the diameter of the graph.

In general, which nodes and edges in a graph will become faulty is not known in advance. If we calculate an upper bound d on the diameter of the survival route graph where there are f or fewer faults (f fixed), then a message should be rebroadcast d times to ensure that all stations get the message. Dolev et al.[12] obtained several results on the diameter of this digraph and suggested the networks where the associated survivable route graph has small diameter. It was shown that for any (k+1) node connected graph G and routing ρ, the diameter of the survival route graph $R(G, \rho)/F$ is no greater than max(2k, 4), and for some special class of graphs (e.g. hypercubes), there is a shortest path unidirectional routing ρ such that the diameter of $R(G, \rho)/F$ is 2 and there

is a shortest path bidirectional routing ρ such that the diameter of R(G, ρ)/F is 3. Such networks are called fault-tolerant. They concluded with a list of interesting open problems.

The connection network is represented by a digraph G = (V, E) where V is the set of nodes and E is the set of directed edges. Let k be the node connectivity and $F \subset V \cup E$ be the set of faulty components of the network where F_v is the set of faulty nodes and F_e is the set of faulty edges such that $F = F_v \cup F_e$. Let t be the maximum number of faulty components so that $|F| \leq t$. It is assumed that $t \leq k - 1$ so that the digraph is not disconnected when there is a fault. For any node $u \in V$, let P(u) and S(u) denote the set of predecessors and successors of u in digraph G where, $P(u) = \{ \alpha \mid (\alpha, u) \in E\}$ and $S(u) = \{ \alpha \mid (\alpha, u) \in E\}$.

A set of concentrator nodes of a digraph G is given by a set $M = \{ m_1, m_2, ..., m_p\}$ of its nodes if for each $m_i \in M$, satisfies the following conditions C1 to C5 where $X_i \subseteq P(m_i)$ and $Y_i \subseteq S(m_i)$ $|X_i| = |Y_i| = k$. [45].

C1: $m_i \notin Y_j$, $1 \leq i, j \leq p$

C2: $m_i \notin X_j$, $1 \leq i, j \leq p$

C3: $Y_i \cap Y_j = \phi$, $i \neq j, 1 \leq i, j \leq p$

C4: $X_i \cap X_j = \phi$, $i \neq j, 1 \leq i, j \leq p$

C5: $Y_i \cap X_j = \phi$, $i \neq j, 1 \leq i, j \leq p$

2.6 Related work

Considerable studies have been done using the fixed routing between every pair of nodes [35], [9][20][27]. If any node or edge fails, the route that passes through that node or edge become unusable. However, communication between two nodes is still possible by sending a message along a sequence of survival routes which do not have faulty node or edge.

Ramamoorthy and Ma [37] have discussed about the problems to find the optimally fault-tolerant reconfiguration strategies without any repair in the failed modules and presented a stochastic model and constructed a polynomial-time algorithm for finding optimal reconfiguration strategies which can be very useful in case of long life systems and systems that are maintained periodically (e.g. unmanned spacecraft).

Sengupta et al. [46], have presented a regular digraph topology of interconnections between the nodes of a network which is optimally fault-tolerant and the message routing delay is very small unless the number of faulty nodes reaches its optimal value. This topology is similar to the topology proposed by Pradhan [36]. The difference is that it uses unidirectional links where as in Pradhan [36], bidirectional links are used. The topology proposed in Pradhan [36] is not always optimally fault-tolerant.

Broder et al. [9] have constructed an efficient fault-tolerant routing based on "product route

graph" from two or more constituent "route graphs" and an upper bound of the diameter of the associated survival route graph was calculated. In [35], an efficient routing is defined based on the concept of concentrator nodes, a set of nodes having some specific properties. They defined several routing functions through concentrator nodes and have shown that if the number of concentrator nodes is enough then ρ can be defined such that the diameter of $R(G, \rho)/F$ is not greater than 4 where F denotes node and edge faults.

Manabe et al. [27] have shown the sufficient condition for which ρ could be defined such that the diameter of $R(G, \rho)/F$ will not exceed 6. In [19], it is shown that Kautz and De Bruijn digraphs are good fault-tolerant networks as the diameter of their survival route graphs is two. They have used a minimal length routing.

Sengupta and Elfe in [45] have used the ideas of concentrator nodes for digraphs and defined an efficient routing for which the diameter of the survival route graph will not exceed 3. They have also proved that for all digraph of n nodes if the maximum degree is not more than $0.6n^{1/3}$ then, the concentrator nodes and therefore the routing will always exist.

Dowd [13] introduced a shared-channel, multihop system based on the binary hypercube topology which improves performance and reduces complexity. In this network, each node has p transmitters and p receivers and the total number of nodes. All of the logical links on a common dimensional axis of the hyperbola share a unique wavelength. This network is highly fault-tolerant and cost-effective. This approach provides more efficient resource allocation than

employing a large number of point-to-point channels with large resources to support high traffic.

In [2], Bandyopadhyay and Sengupta have proposed two schemes for designing fault tolerant WDM networks based on survival route graph and the idea of concentrator nodes introduced in [35]. They have employed De Bruijn graph as a physical topology for both schemes. One scheme uses full wavelength conversion while the other uses limited wavelength conversion. The proposed schemes require wavelength conversions only at few nodes of the network, where the fault-tolerant scheme proposed in [16] requires wavelength conversion at each node. Another advantage of this scheme is it can be used in multi hop network with little modification. Gerstel et al. [16] have used ring as a physical topology so their scheme can handle a single fault.

Chapter 3

Design of fault-tolerant WDM network

3.1 Introduction

Our fault-tolerant schemes are basically fault avoidance schemes. A good fault avoidance can be achieved by designing the system properly according to the probability of faults. Our focus is on the design of optimized networks in which the rerouting scheme to avoid fault is as simple as possible. In this fault-tolerant WDM network, all deflector nodes (spare resources) and end-nodes are connected by using De Bruijn graph topology. We have proposed two schemes in this thesis. We initially attempted the design of an all-optical network. However this scheme cannot manage fault in most cases. Our second scheme is a hybrid network of single hop and multi hop networks which can successfully manage faults in most cases. In the absence of fault, any two nodes will communicate with each other using a single-hop scheme and there is no need of electrical buffering (all-optical case). Whenever, there is any kind of fault in the network, two nodes might not be able to communicate with each other in single-hop. In such cases, we require multi-hops and then the network will no longer be an all-optical.

Both of our schemes are based on fixed routing using the notion of a survival route graphs. In a network when a node or link fails, all of the routes that go through the failed components will

become unusable and consequently certain pairs of nodes will be unable to communicate in their normal routes. However, they can still communicate by sending the information along a sequence of survival routes if the network is still connected. In graph model, a communication from x to y is possible if there exists a route $\rho(x, y)$ from x to y that does not involve any faulty node or edge. However, in a wavelength routed optical network, a communication from x to y is possible if there exists a route $\rho(x, y)$ from x to y that does not involve any faulty node or edge and there must be an end-to-end lightpath using a WDM channel for each edge in the route $\rho(x, y)$. In our schemes, we will show how we will create a new lightpath to replace a faulty lightpath.

In our schemes, we have used some special nodes (concentrator, predecessor, and successor nodes) as spare resources. All end-nodes (not special nodes) are called ordinary nodes. A set of nodes that satisfy the following conditions are called concentrator nodes [45].

- There is no edge from one concentrator node to another concentrator node.
- No two concentrator nodes can have the same predecessor node.
- No two concentrator nodes can have the same successor node.
- The predecessor of one concentrator node can not be the successor of another concentrator node.

If a node has an edge to a concentrator node, then the node is called the predecessor node of that concentrator node and if the concentrator node has an to a node, then the node is called the successor node of that concentrator node. A concentrator node can have a fixed number of

predecessor and successor nodes depending on the number of edges of the networks. All special nodes are called deflector nodes. It will be clear later why we call them as deflector nodes.

It can be easily shown, if we choose ii----i as the i^{th} concentrator node in a network of d^k nodes, where d represents the number of edges, k represents the number of digits and $k > 2$, all properties of a concentrator node are satisfied. In this case the predecessor nodes will be ji----i for some $j \neq i$ and the successor nodes will be ii----j for some $j \neq i$.

We have used a random number generator to simulate edge fault in randomly chosen edges in the network to validate the fault tolerant mechanisms of the system and obtained statistics on different parameters.

3.2 Scheme 1

We will discuss about our first scheme for optical networks where the physical topology for the communication is a De Bruijn graph of $N = d^k$ nodes. To test our scheme, we have designed a single hop wavelength routed network and used the idea of survival route graphs for fault avoidance in wavelength routed networks.

3.2.1 Network topology - De Bruijn graph

We have employed the De Bruijn graph architecture since in this architecture, for a given concentrator node, all the node disjoint paths from the source to the predecessor nodes and all the node disjoint paths from the successor nodes to the destination can be defined very easily. Besides this, it inherently allows a number of alternative routes for communication and its survival route graph has a very small diameter, the upper limit of the number of routes to be traversed in any communication between two fault-free nodes, to avoid signal attenuation and network blocking. In this scheme, all the paths for every source-destination pair are constant length paths which reduces the signal dynamics at the receiver and simplifies the receiver implementation [29].

3.2.2 Deflector nodes of the networks

In our scheme, optical networks consist of $N = d^k$ nodes, where d represents the number edges and k represents the number of digits. Out of N, n nodes are end-nodes and the rest are deflector nodes (special nodes). The number of deflector nodes depends on the network size i.e., number of nodes (end-nodes) and links. End nodes are connected to optical routers and can be source or destination. Deflector nodes contain only optical routers and are used to replace a faulty lightpath. Deflector nodes can be predecessor nodes, concentrator nodes, or successor nodes. If networks contain C concentrator nodes and for each concentrator node, there are d predecessor

nodes and d successor nodes, the number of deflector nodes will be C * (2d + 1) and the number of end-nodes will be N - C * (2d + 1).

In this scheme, the connectivity of a digraph G is k -1 and the number of concentrator node is k. Since the maximum number of faulty edge can not exceed k -2, it is guaranteed that there must be at least two concentrator nodes which are not faulty.

3.2.3 Routes for lightpath segments

To determine the routes for lightpath segments, we have to consider the following cases:

- If x is an end node and y is a deflector node, we have to define $\rho(x,y)$ only if y is the predecessor of some concentrator node c_i. In this case, $\rho(x,y)$ is the route which is node disjoint from all other routes $\rho(x,p)$ where p is any predecessor node of the same concentrator node c_i, $p \neq y$.
- If x and y both are deflector nodes, we have to define $\rho(x,y)$ only if x is the predecessor of a concentrator node m_i and y is a successor of the same concentrator node. In this case, $\rho(x,y)$ is the route $x \rightarrow c_\alpha \rightarrow y$.
- If x is a deflector node and y is an end-node, we have to define $\rho(x,y)$ only if x is the successor of some concentrator node c_i. In this case, $\rho(x,y)$ is the route which is node disjoint from all other routes $\rho(s,y)$ where s is any successor node of c_i, $s \neq y$.

3.2.4 Wavelength Allocation for lightpath segments

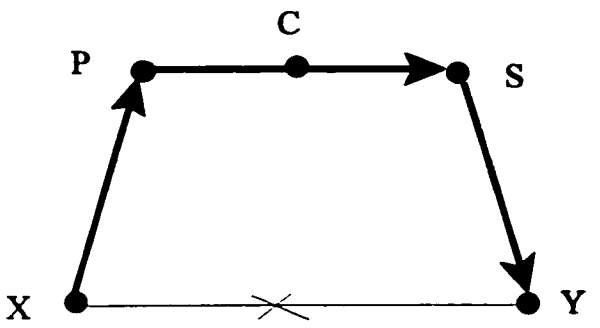


Fig. 6: New route through P and S

For the first and the third cases, for each route ρ(x,y) we assign one lightpath segment per route to make sure that no two lightpath segments share the same wavelength if they share the same edge. In the second case, for each route we assign as many lightpath segments as possible depending on the maximum number of wavelengths per fiber and the wavelengths of lightpaths already assigned to the edges $x \rightarrow c_\alpha$ and $c_\alpha \rightarrow y$ due to the existing lightpaths or lightpath segments. When there is no fault in the network, all wavelengths in this route are marked as "unassigned".

3.2.5 Creation of Lightpaths

To create the lightpaths, the steps are as follows:

Step i) We determine the lightpaths between every pair of ordinary nodes.

Step ii) We determine lightpath segments

- from all ordinary nodes to all predecessor nodes.
- from the j^{th} predecessor node of the concentrator node ii----i to the k^{th} successor node of the same concentrator node for all j, k, and i where $0 \leq i, j, k < d$.
- from all successor nodes to all ordinary nodes.

Step iii) We set the optical routers according to the lightpaths created in step i and the lightpath segments created in step ii. However, the lightpath segment created in step ii will not be used as long as there is no fault in the network.

In our first scheme, we create one lightpath segment for every (end node, deflector node) or (deflector node, end node) combination in step ii. For each (deflector node, deflector node) combination, we create as many lightpath segments as permitted by the maximum number of wavelengths per fiber[2].

If a node or an edge in a network becomes faulty, a number of lightpaths will be unusable. If there are two unusable paths from x to y1 and y2, we cannot use the same predecessor node for the both communications (one from x to y1 and another from x to y2). Since there is one lightpath segment from x to the predecessor node with a specific wavelength, x will use that wavelength for all communication from x to y1. Therefore, once we select one predecessor node for a communication from x to y1, we can not use the same predecessor node for the communication from x to y2. This follows from the fact that a wavelength routed network does not allow two simultaneous communications to share the same WDM channel. Similarly, we

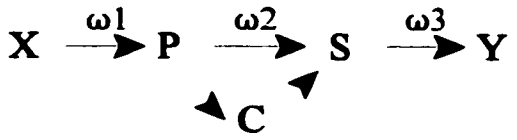
cannot use the same successor node for the communications from different sources (x1 and x2) to the same destination node y. If an end node x(or y) is the source(or destination) of two unusable lightpaths, we have to use a distinct node as the predecessor(or successor) node. However, our scheme allows a number of lightpath segments from a predecessor node to a successor node through a concentrator node. For a given source-destination pair, the number of lightpath segments is three in case of faults.

3.2.6 Adaptive rerouting scheme

If the normal route between a pair of nodes X and Y becomes faulty (fig. 6), we have to find out a successor node S and predecessor node P of some concentrator node C, which is not faulty such that

- there is an unassigned fault-free lightpath segment from X to P with wavelength $\omega 1$
- there is an unassigned lightpath segment from P to S with wavelength $\omega 2$
- there is an unassigned fault-free lightpath segment from S to Y with wavelength $\omega 3$.

It requires two wavelength conversions: one from $\omega 1$ to $\omega 2$ at P and another from $\omega 2$ to $\omega 3$ at S. If we are successful in our search for to the above three paths, we can replace the faulty path from X to Y by a composite lightpath consisting of three lightpath segments as follows:



We will mark the above lightpath segments as "assigned". If we are not successful, we have to try with other predecessor nodes and successor nodes of all concentrator nodes until we can replace the faulty path. If we can replace all the faulty paths in the network, we conclude that we can handle the fault. If $C > d - 2$, it is guaranteed that at least one concentrator node is fault-free. Since all the lightpaths from X to all (d -1) predecessor nodes of a concentrator node are node-disjoint paths and all the lightpaths from all (d -1) the successor nodes of a concentrator node to Y are node-disjoint paths, it is always possible to find fault-free $\rho(X,P)$ and $\rho(S,Y)$ routes. However, it might happen that all the lightpath segments to/from the deflector nodes have been already assigned to replace another faulty path. In this situation, our method will fail to provide an alternative lightpath for the faulty lightpath from X to Y.

3.2.7 Fault injection

A major problem in the development of fault-tolerant systems is the accurate determination of the dependability of the system. Performance of a system can be evaluated by employing benchmark programs. However, the degree of fault-tolerance and reliability of a system cannot be evaluated in this way, since in general, we can not allow a system to run for many years to

observe their behavior during faults. The generally acceptable solution to this problem is to inject the effect of faults in a simulation model or a prototype implementation, and to observe the behavior of the system under the injected faults. We have employed random number generator to simulate edge fault in randomly chosen edges. If any faulty node is special or any faulty edge is coming out from a special node, then all the lightpaths passing through that faulty node or faulty edge will be deleted.

3.2.8 Why scheme 1 will fail in most cases

Let us consider a De Bruijn graph with 4^3 nodes (64 nodes) where the concentrator nodes are 000, 111, 222, and 333. The predecessor nodes of 000 are 100, 200, and 300. The successor nodes of 000 are 001, 002, and 003. Similarly, for other concentrator nodes, there are specific predecessor nodes and successor nodes. For our simulation, let us pick one edge 021 $\rightarrow$212 as faulty edge. Now, we have to determine which source-destination pairs use this edge as an intermediate edge. According to Ajmone Marshan [28], the edge 021 $\rightarrow$212 appears in the following lightpaths:

Source node	Destination node
021	2**
*02	12*
**0	212

* can represent 0,1,2,or 3.

Each of these source-destination pairs represent 4 x 4 values since * can have 4 different values. Three different source-destination pairs represents 48 values. In general, in such cases, the three pairs may not be 3 x16 = 48 values because the same node may appear twice. We have to attempt to replace all these 48 lightpaths. If we succeed in doing that we have managed to handle the faults, otherwise, we have failed to manage the faults.

As an example, let us consider one lightpath from the set source *02 and destination 12*. We can get this by substituting any valid value for a *. One such source is 202 and the destination is 121. To replace this faulty path, an alternate path is shown in fig 7. The normal path is 202 →021 → 212 → 121 which cannot be used since the edge 021 → 212 is faulty. In this diagram concentrator node is 111 and its predecessor node is 211 and successor node is 113.

This alternate path is valid if

① the path 202 →022 → 221 → 211 contains no faulty edge.

② this lightpath has not been already used to take care of some other faulty lightpath since there is only one such lightpath in the network.

③ the path 113 →131 → 312 → 121 contains no faulty edge.

④ this lightpath has not been already used to take care of some other faulty lightpath since there is only one such lightpath in the network.

⑤ the path 211 →111 → 113 contains no faulty edge.

⑥ the number of lightpath does not exceed a specific value.

If we succeed we will attempt to replace the next faulty lightpath, else we have to try the predecessor-successor pair for 111. If we try and fail for all the pairs, we try the next concentrator node 222. If we fail with all concentrator nodes, we will say that we could not handle all faults.

If we use our scheme with De Bruijn graph, the number of non-faulty lightpaths from a given node to all the predecessor nodes in the network is d(d -1) and the number of faulty lightpath is d^{k-1}. Since $kd^{k-1} > d(d-1)$ except for trivial values of k, the number of non-faulty lightpaths is always much less than the faulty lightpaths (48) since the number of communication from an ordinary node (or successor node) to a predecessor node (or ordinary node) is always limited to one. For this reason, in most cases our first scheme will fail to provide non-faulty alternate path.

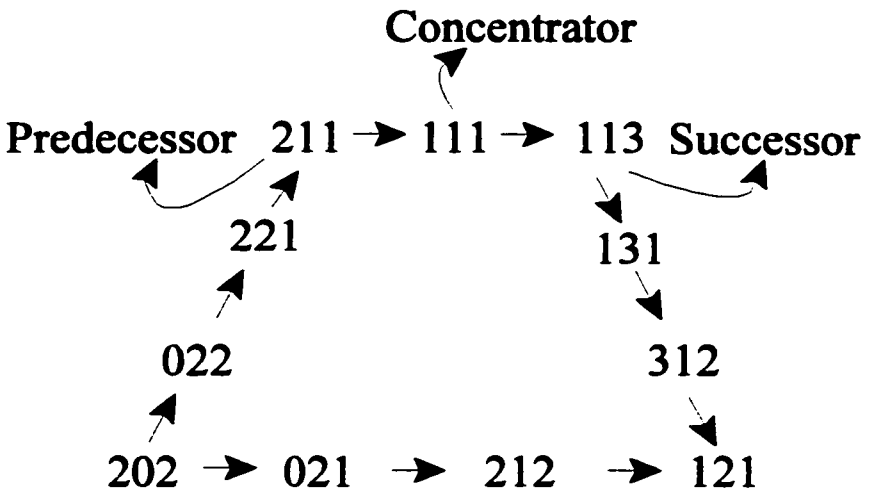


Fig. 7: An alternate Path for source-destination (202, 121)

3.3 Scheme 2

Our first fault-avoidance scheme is a single-hop network (all-optical) and there is no need of electrical buffering. However, our second approach is a hybrid system of single-hop and multihop systems. In the absence of fault, any two nodes will communicate with each other in a single-hop. Whenever, there is any kind of fault in the network, two nodes might not be able to communicate with each other in single-hop. When it is not possible to establish a lightpath between two nodes, they must use a sequence of lightpaths through intermediate nodes to communicate. At each intermediate node, the information coming in on a lightpath must be converted to electrical form, switched electrically and then converted back to optical form and ultimately reach their destinations by using different lightpaths. In other words, in the case of

faults it might require multi-hopping and electrical buffering, and then the network will no longer be an all-optical. The disadvantage of this type of network is its lower throughput. However, since in this scheme, there is no restriction of the maximum number of lightpaths passing through different edges, it can successfully manage multiple failures in most cases.

It will be clear if we give an example. Let us consider a De Bruijn graph of 16 (2^4) nodes where the concentrator nodes are 1111 and 0000. The predecessor node of 1111 (0000) is 0111 (1000) and the successor node of 1111 (0000) is 1110 (0001). According to Ajmone Marshan[28], the following source and destination pairs will use the randomly generated faulty edge 0001 →0011 as an intermediate edge.

Source node	Destination node
0001	1***
*000	11**
**00	011*
***0	0001

* represents 0 or 1.

As an example, let us consider one lightpath from the set source **00 and destination 011*. We can get this by substituting any valid value for a *. One such source is 0100 and the destination is 0110. To replace this faulty path, an alternate path is shown in fig 8. The normal path is 0100 →1000 → 0001 → 0011 →0110 which cannot be used since the edge 0001 → 0011 is faulty. Another such source is 1100 and the destination is 0110. To replace this faulty path, an alternate

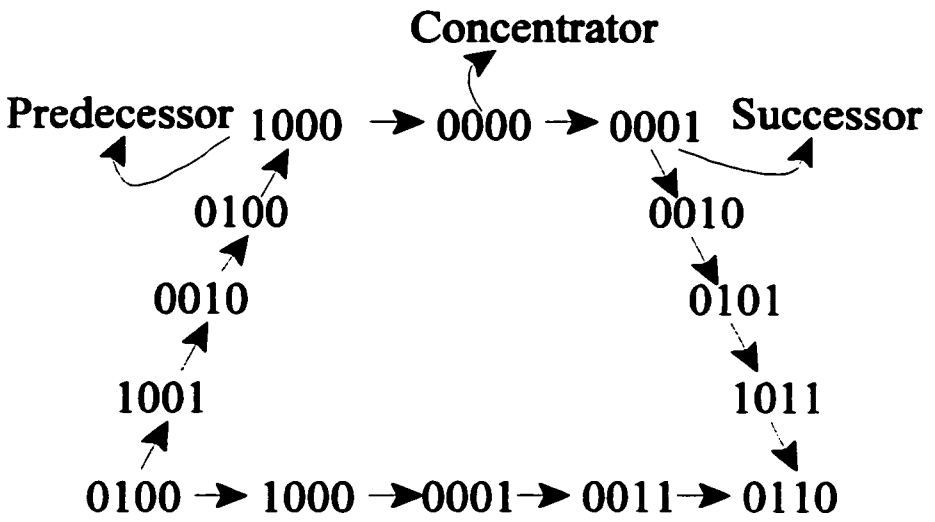


Fig. 8: An alternate Path for source-destination (0100, 0110)

path is shown in fig 9. The normal path is 1100 →1000 → 0001 → 0011 →0110 which cannot be used since the edge 0001 → 0011 is faulty. Both alternate paths are simultaneously sharing the same lightpath segment 0001 →0010 → 0101 → 1011 →0110. Our first scheme cannot handle this situation since it does not allow two or more communications to concurrently use the same lightpath segment. However, our second scheme can successfully provide alternate path in this situation as it supports the fact that two or more communications can concurrently share the same lightpath segment.

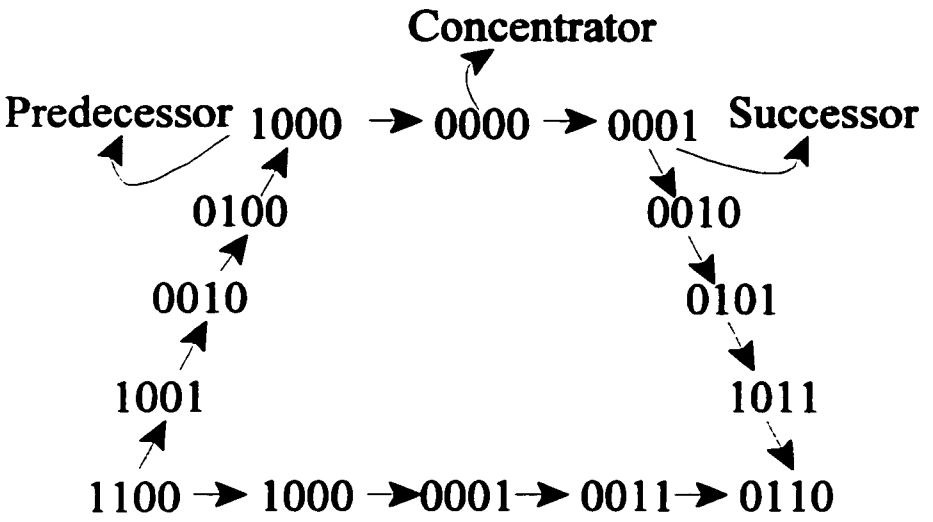


Fig. 9: An alternate Path for source-destination (1100, 0110)

3.3.1 Alternate paths using our second scheme

Whenever there is a faulty edge in the networks, some paths that involve this edge will become unusable. Therefore alternate non-faulty paths are required in order to continue communications in case of faults. In Table1, alternate path passing through special nodes (such as, predecessor, concentrator, and successor nodes) for each affected path is provided. In this case, the size of the network is 27 (3^3) and the faulty edges are 200 →001 and 202 → 020.

Table 1. Alternate paths passing through special nodes in a network of size 27 (3^3) and faulty edges 200 →001 and 202 → 020.

Serial number of affected paths	Source node	Predecessor node	Successor node	Destination node
1	6	8	25	3
2	15	8	25	3
3	6	9	1	5
4	15	9	2	5
5	20	9	1	3
6	20	17	24	6
7	20	18	1	7
8	20	22	12	5
9	6	17	25	19
10	15	17	25	19
11	6	22	14	20
12	15	22	12	20
13	11	4	12	6
14	5	4	12	6
15	23	4	12	6

3.3.2 Few cases when our second scheme will fail

If we can provide alternate path for each path that is affected by the fault, then only we can say our scheme is successful. Each alternate path is a composite path of three paths: path from the source to a predecessor, path from the predecessor to a successor, and path from the successor to the destination. If one of them is faulty, the composite path can not be used as an alternate path to replace the faulty path. In few cases, our second scheme will not be able to provide alternate path for all communications that are affected. In Table2, we have shown a special case when our second scheme will fail to provide alternate path for a particular set of faults in the network.

In this case, the network is a De Bruijn graph of 32 (2^5) nodes where the concentrator nodes are 11111 and 00000. The predecessor node of 11111 (00000) is 01111 (10000) and the successor node of 11111 (00000) is 11110 (00001). According to Ajmone Marshan [28], the following source and destination pairs will use the randomly generated faulty edge 00011 →00111 as an intermediate edge.

Source node	Destination node
00011	1****
*0001	11***
**000	111**
***00	0111*
****0	00111

* represents 0 or 1.

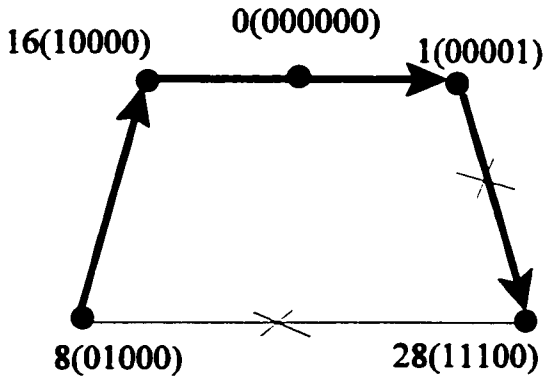


Fig. 10: Invalid alternate path1

A number of lightpaths will become unusable due to the fault, however, we have to replace a subset of the lightpaths where source (or destination) is an ordinary node. Whenever our scheme can not provide alternate path for a single faulty path it will report with an appropriate message and it will not continue checking for alternate paths for the remaining faulty paths. In this example, our scheme can successfully manage first nineteen faulty lightpaths, however, in the following case where source is 8 (01000) and destination is 28 (11100) our scheme can not provide alternate path. Let us investigate why it will fail to provide alternate path.

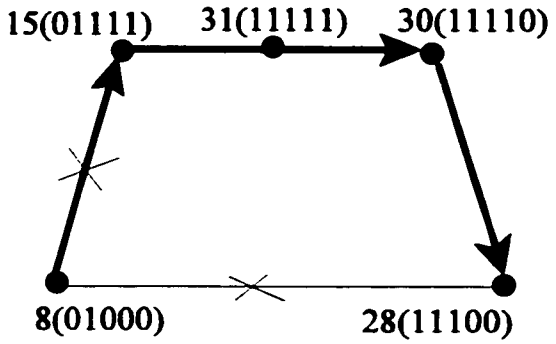


Fig. 11: Invalid alternate path2

The probable alternate paths can be :

- 01000(8) →10000(16) →00001(1) →11100(28) (shown in fig. 10).
- 01000(8) →01111(15) →11110(30) →11100(28) (shown in fig. 11).

However, the first alternate path (shown in fig 10) will not be valid since the lightpath segment from the successor node 1(00001) to the destination 28(11100) uses faulty edge. The second alternate path (shown in fig 11) will also be invalid since the lightpath segment from the source 8 (01000) to the predecessor node 15 (01111) uses faulty edge.

In this case, the number of concentrator node is two and for each concentrator node, there is only one predecessor and one successor node. Therefore it is possible that a situation can occur when there is no non-faulty lightpath segment from the source to the predecessor node or from the successor node to destination node to provide an alternate path.

Table 2. Alternate paths passing through special nodes in a network of size 32 (2^5) and faulty edge 00011 →00111.

Serial number of affected paths	Source node	Predecessor node	Successor node	Destination node
1	3	15	30	24
2	3	15	30	20
3	3	15	30	28
4	3	15	30	18
5	3	15	30	26
6	3	15	30	22
7	3	15	30	17
8	3	15	30	25
9	3	15	30	21
10	3	15	30	29
11	3	15	30	19
12	3	15	30	27
13	3	15	30	23
14	17	15	30	24
15	17	15	30	28
16	17	15	30	26
17	17	15	30	25
18	17	15	30	29
19	17	15	30	27
20	8	WE CAN NOT FIND AN ALTERNATE PATH		28

3.4 Evaluation of our second scheme

We have tested our scheme by randomly generating maximum number of faulty edges for a particular network assuming the network is still connected. We have observed that in a network where the number of edges is greater than two, our scheme can successfully provide alternate path in case of faults. In thousand cases, we did not encounter a single miss. By performing chi-square test we can determine the significance of success rate.

The formula for chi-square is:

$$\text{chi-square } (X^2) = \text{sum of } [\ (O-E)^2 / E\ ]$$

where O = observed frequency

E = expected frequency

Here, null hypothesis is that there will be no difference in the preference for success and failure. Thus, the observed and expected frequencies are as follows:

	Observed frequency	Expected frequency
Success	1000	500
failure	0	500

The calculated chi-square value, obtained from the above formula, is 1000. In order to determine statistical significance of this chi-square value, we consulted a table for critical values. The

chi-square critical value for significance at the 0.001 probability level is 10.828 for one degree of freedom; therefore, the frequency with which success occurs was statistically significant.

However, our scheme can fail to provide alternate path in presence of failure when the number of edges is two. In this case, the number of concentrator node is two and for each concentrator node, there is only one predecessor and one successor node. Therefore it is possible that a situation can occur when there is no non-faulty lightpath segment from the source to a predecessor node or from a successor node to the destination node to provide an alternate path.

3.5 Difference between scheme 1 and scheme 2.

	Scheme 1	Scheme 2
System	Single hop.	Hybrid system of single hop and multi hop.
Type	All-optical.	Not all-optical.
Electronic Buffering	It does not need electronic buffering.	It needs electronic buffering in case of faults.
Throughput	High.	Low (in some cases).
Communication	The number of communication from ordinary (or successor) to predecessor (or ordinary) node is limited to1.	The number of communication from a given ordinary (or successor) to given predecessor (or ordinary) node is not limited.
Bottlenecks - Uniform Distribution	It can never cause bottlenecks in the networks.	It can cause bottlenecks- the communications through different edges are distributed as uniformly as possible to avoid it.
Success	In very few cases, it will provide non-faulty alternate paths.	In most cases, it will provide non-faulty alternate paths.

Chapter 4

Results and Statistics

4.1 Results

4.1.1 Table3 : Distribution of lightpaths

In the absence of faults in a network, any two nodes will communicate directly (in a single hop). However, when there is a fault in the network , they will communicate using 3 hops through the special nodes (predecessor, concentrator, and successor nodes). The number of affected paths will be different depending on the location of the fault, the number of the faults and the size of the network. Therefore, the number of communications passing through the different logical edges (such as, source to predecessor edge, predecessor to successor edge, successor to destination edge) are different for different runs. In Table3, we have divided this number into three groups and shown how many logical edges are belonged to each group. In some cases, the maximum number of communications (load) on edges are very high, however very few communications have used these edges which indicates that very few communications will be affected by this high load. Besides this, the assumption that each node is equally likely to communicate with all others is not true in reality. It is unlikely that all nodes are concurrently communicating with all other nodes.

Table 3A: The distribution of communications through different logical edges in a network of size 27 (3^3) and two randomly generated faulty edges.

Run	Load (number of communications)	Source to predecessor edge	Predecessor to successor edge	Successor to destination edge
1	1 - 2	15	8	9
	3 - 4	0	1	1
	5 - 11	0	0	0
2	1 - 2	6	0	0
	3 - 4	0	2	2
	5 - 11	0	0	0
3	1 - 2	11	6	5
	3 - 4	0	1	1
	5 - 11	0	0	0
4	1 - 2	7	6	7
	3 - 4	0	0	0
	5 - 11	0	0	0
5	1 - 2	7	7	7
	3 - 4	0	0	0
	5 - 11	0	0	0
6	1 - 2	13	2	8
	3 - 4	0	2	1
	5 - 11	0	1	0
7	1 - 2	4	3	3
	3 - 4	0	0	0
	5 - 11	0	0	0
8	1 - 2	17	4	11
	3 - 4	0	4	0
	5 - 11	0	0	0
9	1 - 2	3	3	3
	3 - 4	0	0	0
	5 - 11	0	0	0
10	1 - 2	9	2	8
	3 - 4	0	2	0
	5 - 11	0	0	0

Table 3B: The distribution of communications through different logical edges in a network of size 81 (3^4) and two randomly generated faulty edges.

Run	Load (number of communications)	Source to predecessor edge	Predecessor to successor edge	Successor to destination edge
1	1 - 5	104	2	61
	6 - 15	5	7	4
	16 - 40	0	3	1
2	1 - 5	92	5	62
	6 - 15	2	5	3
	16 - 40	0	2	1
3	1 - 5	103	0	68
	6 - 15	5	10	4
	16 - 40	0	2	1
4	1 - 5	67	4	59
	6 - 15	5	6	0
	16 - 40	0	2	1
5	1 - 5	78	2	76
	6 - 15	3	8	3
	16 - 40	0	2	0
6	1 - 5	107	2	75
	6 - 15	5	5	5
	16 - 40	0	5	1
7	1 - 5	104	4	44
	6 - 15	0	6	4
	16 - 40	0	2	0
8	1 - 5	106	1	73
	6 - 15	6	7	4
	16 - 40	0	4	1
9	1 - 5	92	5	39
	6 - 15	0	4	5
	16 - 40	0	3	1
10	1 - 5	122	4	64
	6 - 15	3	4	4
	16 - 40	0	4	1

Table 3C: The distribution of communications through different logical edges in a network of size 64 (4^3) and two randomly generated faulty edges.

Run	Load (number of communications)	Source to predecessor edge	Predecessor to successor edge	Successor to destination edge
1	1 - 2	9	8	8
	3 - 5	0	0	0
	6 - 11	0	0	0
2	1 - 2	8	8	8
	3 - 5	0	0	0
	6 - 11	0	0	0
3	1 - 2	32	17	25
	3 - 5	0	3	2
	6 - 11	0	0	0
4	1 - 2	16	14	16
	3 - 5	0	0	0
	6 - 11	0	0	0
5	1 - 2	9	0	0
	3 - 5	0	2	2
	6 - 11	0	0	0
6	1 - 2	0	0	0
	3 - 5	0	0	0
	6 - 11	0	0	0
7	1 - 2	18	17	16
	3 - 5	0	0	0
	6 - 11	0	0	0
8	1 - 2	17	15	16
	3 - 5	0	0	0
	6 - 11	0	0	0
9	1 - 2	24	12	16
	3 - 5	0	1	2
	6 - 11	0	1	0
10	1 - 2	18	5	5
	3 - 5	0	3	3
	6 - 11	0	0	0

Table 3D: The distribution of communications through different logical edges in a network of size 625 (5^4) and three randomly generated faulty edges.

Run	Load (number of communications)	Source to predecessor edge	Predecessor to successor edge	Successor to destination edge
1	1 - 5	833	19	562
	6 - 60	32	58	45
	61 - 120	0	3	0
2	1 - 5	746	6	590
	6 - 60	46	71	41
	61 - 120	0	3	0
3	1 - 5	749	31	416
	6 - 60	32	42	40
	61 - 120	0	3	0
4	1 - 5	695	16	596
	6 - 60	47	61	29
	61 - 120	0	3	0
5	1 - 5	973	2	497
	6 - 60	48	74	60
	6 1- 120	0	4	0
6	1 - 5	722	11	482
	6 - 60	31	61	39
	61 - 120	0	4	0
7	1 - 5	981	0	635
	6 - 60	48	76	67
	61 - 120	0	4	0
8	1 - 5	945	12	629
	6 - 60	48	64	46
	61 - 120	0	4	0
9	1 - 5	877	16	649
	6 - 60	48	60	38
	61 - 120	0	4	0
10	1 - 5	933	10	582
	6 - 60	48	66	54
	61 - 120	0	4	0

4.1.2 Table 4 : Maximum number of communications through different edges.

In Table 4, we have shown the maximum number of communications passing through each logical edge for different runs. It can be useful to calculate the maximum delay we can expect. The number of communications may be as high as one hundred in some cases. However, this the worst case scenario in terms of queue length where we have assumed that each node is concurrently communicating with other nodes. In an actual network, any node can concurrently communicate only with a limited number of nodes and therefore the value of the maximum number of communications through different edges would be much less than what we observed in this thesis. Therefore in real life, the delay would be much less than what we can predict from this thesis.

Table 4A: Maximum number of communications through different edges in a network of size 27 (3^3) and two randomly generated faulty edges.

Run	Maximum number of communications through		
	Source to predecessor edge	Predecessor to successor edge	Successor to destination edge
1	2	3	3
2	1	3	3
3	2	4	3
4	1	2	1
5	1	1	1
6	2	7	4
7	1	2	2
8	2	4	4
9	1	1	1
10	2	5	3

Table 4B: Maximum number of communications through different edges in a network of size 81 (3^4) and two randomly generated faulty edges.

Run	Number of communications through		
	Source to predecessor edge	Predecessor to successor edge	Successor to destination edge
1	9	27	22
2	6	34	16
3	6	35	22
4	6	30	20
5	6	21	13
6	6	37	18
7	6	25	14
8	7	27	19
9	6	33	18
10	6	40	18

Table 4C: Maximum number of communications through different edges in a network of size 125 (5^3) and three randomly generated faulty edges.

Run	Number of communications through		
	Source to predecessor edge	Predecessor to successor edge	Successor to destination edge
1	1	14	8
2	2	13	6
3	1	7	6
4	2	11	5
5	1	12	6
6	2	11	6
7	2	8	6
8	1	14	8
9	1	9	9
10	2	7	6

Table 4D: Maximum number of communications through different edges in a network of size 256 (4^4) and two randomly generated faulty edges.

Run	Number of communications through		
	Source to predecessor edge	Predecessor to successor edge	Successor to destination edge
1	7	39	29
2	8	29	23
3	7	53	29
4	7	56	27
5	7	40	23
6	7	54	24
7	7	28	23
8	7	30	24
9	7	39	28
10	7	60	29

4.2 SYSTAT

4.2.1 Organizing our data for analysis

The first task for analyzing data is to present the data in a form acceptable to SYSTAT for processing. SYSTAT uses data organized in rows and columns. The rows are called cases, and the columns are called *variables*. When data are arranged in rows and columns like this and is stored in a file, it is called a *'cases-by variables'* or *'rectangular data file.'*

SYSTAT accepts numerical and categorical data. The numbers stored in numeric variables can have up to 15 digits. We can use a negative sign (-) for negative numbers. Categorical data stored in character variables can have up to12 characters and can include alphabets and other symbols(e.g., $)&*/( ). If we include numbers within character values they will be treated as characters. Also, upper and lower case character values are differentiated (e.g., JUNIOR is not the same as junior).

We must assign a unique name for each variable. Variable names may contain up to eight characters or numbers, and must begin with a letter. The names of character variables must end with a dollar sign; the dollar sign does not count as one of the eight letters. Variable names, unlike character values, are not case sensitive. Once we have entered a variable name, we may change it, but we cannot change its type from character to numeric or vice versa. If we forget to

put a $ sign at the end of a character variable name, we must type the correct name as a new variable in a new column and later delete the incorrect variable.

When a numeric data is missing, we have to enter a period (.) to flag the position where the value is missing. When a character data is missing enter a blank space surrounded by single or double quotation marks. These quotation marks will not show up in the spread sheet. Note that arithmetic involving missing values propagates missing values.

4.3 Statistics

4.3.1 Frequency Distributions and Histograms

Network size 81 (3^4), number of randomly generated faulty edges = 2.

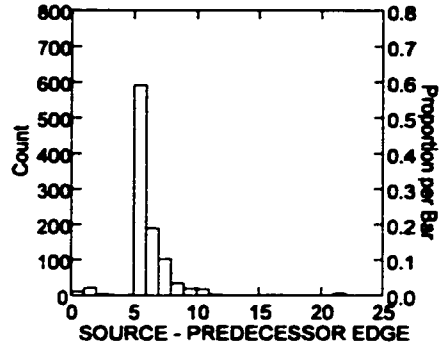


Fig. 12: Distribution of the upper limit of the communications through source to predecessor edge

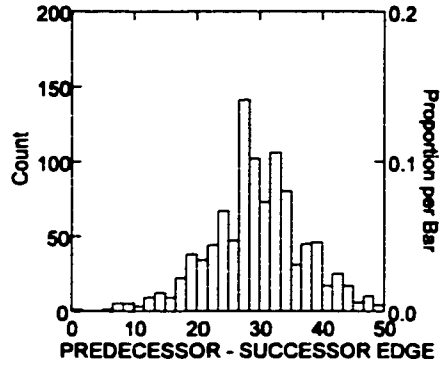


Fig. 13: Distribution of the upper limit of the communications through predecessor to successor edge

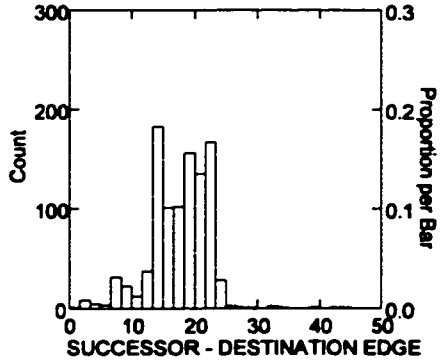


Fig. 14: Distribution of the upper limit of the communications through successor to destination edge.

In general, a frequency distribution can represent the variable x over the entire population or over a sample (subset) of the population. When the distribution represents the entire population, such characteristics are called parameters. When the distribution represents a sample of the population, such characteristics are called statistics.

A histogram is simply a bar graph of a frequency distribution. For each class, a rectangular bar is drawn whose base is the class (on the horizontal axis) and whose height is the frequency (or relative frequency).

Plotting the Data

We have plotted histograms of the distribution of the upper limit of the number of communications through different edges (Figs. 12-14). Fig 12 shows a right skewed distribution of the upper limit of the communications through the source to predecessor edge for one thousand data points. It is observed that in almost 600 out of 1000 cases, the upper limit of the number of communications is very close to 6. The distribution is unimodal having a single peak at 6 so that the frequencies at first increase and then decrease. The mean of this distribution is 6.627 which is a good measure of central tendency for roughly symmetric distributions. However, it can be misleading in skewed distribution as the mean can be greatly influenced by the extreme scores [55]. Therefore, other measures of central tendency such as median may be more informative. The median of the distribution is 6 which indicates that the one half of the

scores occurred above this value and one half of the scores occurred below this value. Median is less sensitive to the extreme scores. The variance of the data points is 3.493 which gives the spread out of the distribution. The standard deviation is 1.869 which implies that the scores of the distributions are not far from the mean. As the score in a distribution becomes more heterogeneous, more "spread out" and different, the value of the standard deviation becomes larger.

Fig. 13 shows the distribution of the upper limit of the number of communications through predecessor to successor edge in one thousand cases. Except an outlier, the distribution is normal where roughly two-third of the scores lie within a distance of one standard deviation of the mean, 95% lie within two standard deviations of the mean [54]. For a normal distribution, the percentage of results that are expected with certain limits can be predicted.

Fig. 14 shows a left skewed distribution of the upper limit of the number of communications through successor to destination edge in one thousand cases. The mean of this distribution is 17.828 and the standard deviation is 4.558 which shows that the scores of the distributions are not far from the mean.

4.3.2 Statistical analysis

Network size 81 (3^4), number of randomly generated faulty edges = 2.

	Histogram 1	Histogram 2	Histogram 3
N of cases	1000	1000	1000
Minimum	0.000	0.000	0.000
Maximum	22.000	49.000	44.000
Range	22.000	49.000	44.000
Median	6.000	30.000	18.000
Mean	6.627	29.674	17.828
Standard Dev	1.869	7.535	4.558
Variance	3.493	56.783	20.777
Skewness	2.937	-0.162	-0.198

Chapter 5

Critical summary and Future directions

5.1 Summary of new results

- Extensive theoretical work has been done on survival route graph. However, no one has so far attempted to apply survival route graph in an optical network. This is the first attempt to realize the applicability of survival route graph in fault-tolerant optical networks.
- In this thesis, initially we have attempted to design a completely all-optical single hop fault-tolerant network. The number of communications passing through the source to predecessor and successor to destination edges is limited to one in our scheme. In this situation, the number of affected lightpaths becomes greater than the number of unaffected lightpaths. As a result, our first scheme was successful to provide non-faulty lightpaths in a few cases.
- Then we have proposed another scheme which is a hybrid network of single hop and multi hop. In this scheme, two nodes will communicate in a single hop (all-optical) in absence of fault. Whenever, there is any kind of fault in the network, two nodes might not be able to communicate with each other in single-hop, it might require multi-hopping and then the network will no longer be an all-optical. This scheme will be capable of

providing non-faulty alternate path in case of fault in most cases and in very few cases it will fail since there is no restriction on load. It is fault-tolerant against multiple faults. However, since this scheme is no longer all-optical, throughput will be lower.

- Since there is no restriction on the number of communications (load) through logical edges in this scheme, high demand on certain links can cause overloading, with particular paths or nodes being heavily used while others remain under-utilized. Such a network is said to be unbalanced and can cause intolerable restoration delay for some communications. It can create bottlenecks in the network. Our adaptive rerouting algorithm introduces additional computational requirements to distribute the load uniformly throughout the network to improve this delay. However, the advantage of this approach is that we need to uniformly distribute the communications only once when any edge in the network happens to be faulty.
- If we have some source destination pairs such that communication between them is critically important, we can ensure that such a pair can always communicate with each other even at the expense of disallowing communication between some other source destination pairs. In the scheme outlined in this thesis, we are taking each source destination pair that cannot communicate using their normal path due to faults and are replacing each faulty path by a fault free path. In a situation where we fail to provide a fault-free path for even a single faulty path, we have concluded that the scheme fails. In a variation of this scheme, it is quite possible to modify the scheme so that certain source destination pairs are guaranteed to be able to communicate even if some other source destination pairs may no longer be allowed to communicate. This would involve

assigning some priorities to different source destination pairs so that we can guarantee that "important" source destination pairs can communicate at the risk of losing the ability of some other "unimportant" source destination pairs.

- We note that our scheme assumes that some external agent is monitoring the network at intervals of time and that, each time the network has some new faults, our scheme is used to determine alternative paths for each faulty path in the entire network. Every time the scheme is used, we are allotting alternative paths for each faulty path in the network. Therefore it is quite possible that the fault avoidance scheme used at a given point in time may be completely reformulated to handle a new set of faults at a subsequent point in time. The time interval at which such reformulation takes place depends on the network manager who decides how frequently network testing will take place.
- From different tables in the results section it is clear that when the number of faulty edges is very high, the number of communications through different edges will also be high. The communications that will use the same logical edge have to wait until the earlier communications have already completed their journey through the logical edge. As the number of communications increases, so does the queue length. In some cases, queue length can reach up to one hundred.
- In this thesis we have assumed that all nodes are equally likely to communicate with other nodes. However, in real time it is very unlikely that a node is concurrently communicating with all other nodes. Therefore in this thesis we have considered the worst case scenario in terms of queue length.

5.2 Conclusions

- Originally we planed to design an all-optical fault-tolerant network. We proposed that scheme and wrote the code for that in C, however when we tested that scheme by randomly generating faults, it failed in most cases.
- Our second scheme, hybrid system of single hop and multi hop system is fault-tolerant against multiple failures and can manage faults in most cases.
- Since in this scheme there is no restriction on the number of communications (load) through each logical edge, high demand on certain links can cause intolerable restoration delay for some communications. To improve this delay, we have added extra code in our rerouting algorithm to distribute the load throughout the network as uniformly as possible.
- From different tables we observed that for some communications, the maximum number of queue lengths can be very high. However, we should remember this number will be much less in real network since there each node will concurrently communicate with a limited number of nodes. Therefore we do not need to provide alternate paths for all communications that are affected by the faults.
- However, not all networks and services will be benefitted from this approach. Our scheme will be very useful for networks and services that require higher level of survivability but can endure delay due to rerouting.

5.3 Future directions

We have proposed two schemes : we have investigated our first scheme and it did not work out the way we expected. We have proposed another scheme and did a preliminary study. We have measured the queue length of an alternate path in case of faults. However, we could not estimate the average communication time due to lack of time. One could construct a simulator and study simulation.

In our scheme, we have assumed that each source node is equally likely to communicate with each destination node. However, in reality, different source nodes have different probabilities of communications with different destination nodes. Another limitation of our scheme is it does not work for any number of nodes and it works only for integer multiple of nodes. One can design a network that will work for any number of nodes.

Other important issues in rerouting, such as fault detection, fault diagnosis, and fault recovery, are not addressed in this thesis. A system can not be fully fault-tolerant without fault detection, fault diagnosis, and fault recovery.

Appendix A

C CODE FOR FAULT-TOLERANT OPTICAL NETWORK

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#define NUM_NODES 27  /* This is d^k  */
#define NUM_DIGITS 3    /* This is k  */
#define NUM_EDGES 3   /* This is d  */
#define NUM_BITS 4
#define NUM_FAULTY_EDGES 2  /* To be changed  */
#define BIT_MASK 15

static long table [32];
int first_time, Z = 0, rprev = 0;
int ordinary_to_pred_nodes_table[(NUM_NODES - NUM_EDGES*
            (2*NUM_EDGES -1))*NUM_EDGES*(NUM_EDGES-1)][3],
   succ_to_ordinary_nodes_table[(NUM_NODES - NUM_EDGES*
            (2*NUM_EDGES -1))*NUM_EDGES*(NUM_EDGES-1)][3];

int pred_to_succ_nodes_table[NUM_EDGES*(NUM_EDGES-1)*(NUM_EDGES-1)][3],
   faulty_nodes[NUM_FAULTY_EDGES],
   faulty_edges[NUM_FAULTY_EDGES],
   ordinary_nodes[NUM_NODES - NUM_EDGES*(2*NUM_EDGES - 1)],
   predecessor_nodes[NUM_EDGES*(NUM_EDGES-1)],
   successor_nodes[NUM_EDGES*(NUM_EDGES-1)],
   concentrator_nodes[NUM_EDGES];


/* This defines initial values in the network  */

void initialise_network();

void update_single_path(int table[][3],
                int size,
                int source_node_number,
                int destination_node_number);

unsigned long int suffix(unsigned long int node_id, int suffix_length,
```

```
                int num_digits_in_node_id);

unsigned long int prefix(unsigned long int node_id, int prefix_length,
                int num_digits_in_node_id);

unsigned long int convert_int_to_node_id( unsigned long int number,
                                int d);
void shift_stack_array(int stack_array[][20], int k);

void update_paths(int source_node_no,
                int conc_node_no,
                int pred_node_no,
                int succ_node_no,
                int destination_node_no);

unsigned long int convert_node_id_to_int(unsigned long int node_id,
                                int d, int mask);

int source_destination_are_ordinary(int source, int destination);

void determine_paths_passing_thru_faulty_edge(int faulty_node_no,
                                int faulty_edge_no);
int identify_predecessor_nodes(int digits[]);

int identify_concentrator_nodes(int digits[]);

int identify_successor_nodes(int digits[]);

double randgen();

void init_rand_gen ();

void add_1(int stack_array[][20], int d, int i);

int get_source_node_no(int stack_array[][20], int k);

int get_destination_node_no(int stack_array[][20], int k);

void process_if_source_or_destination_is_special(int source,
                int destination);


int look_up_path_table(int table[][3],
```

```
                int size,
                int source_node_number,
                int destination_node_number);

void delete_path(int table[][3], int size_of_table, int source,
            int destination);

int available_path1( int source_node_no, int pred_node_no,
                int conc_node_no);

int available_path2( int conc_node_no, int pred_node_no,
                int succ_node_no);

int available_path3( int conc_node_no,
                int  succ_node_no, int destination_node_no);

void find_alternate_path_for_faulty_path(int source_node_no,
                        int destination_node_no);

int min_lightpath_no_in_path2(int pred_node);

/*=============================================================================
 FUNCTION    : identify_predecessor_nodes
 PURPOSE     : Checks whether the node is a predecessor  or not.

 PARAMETERS  : digits[].
 RETURN VALUE : If the node is a predecessor, the function will return 1, otherwise it
                        will return 0.
=============================================================================*/

int identify_predecessor_nodes(int digits[])
{
  int i, flag = 1;
  if (identify_concentrator_nodes(digits)) return 0;
  for (i = 1; i < NUM_DIGITS - 1; i++)
   if (digits[i] != digits[i - 1]) flag = 0;
  return flag;
}

/*=============================================================================
 FUNCTION    : identify_concentrator_nodes
 PURPOSE     : Checks whether the node is a concentrator  or not.
```

```
 PARAMETERS   : digits[].
 RETURN VALUE : If the node is a concentrator, the function will return 1, otherwise it
                will return 0.
================================================================================*/

int identify_concentrator_nodes(int digits[])
{
  int i, flag = 1;
  for (i = 1; i < NUM_DIGITS; i++)
   if (digits[i] != digits[i - 1]) flag = 0;
  return flag;
}

/*================================================================================
 FUNCTION    : identify_successor_nodes
 PURPOSE     : Checks whether the node is a successor  or not.

 PARAMETERS   : digits[].
 RETURN VALUE : If the node is a successor, the function will return 1, otherwise it
                will return 0.
================================================================================*/

int identify_successor_nodes(int digits[])
{
  int i, flag = 1;
  if (identify_concentrator_nodes(digits)) return 0;
  for (i = 2; i < NUM_DIGITS; i++)
   if (digits[i] != digits[i - 1]) flag = 0;
  return flag;
}

/*================================================================================
FUNCTION    : determine_paths_passing_thru_faulty_edge
PURPOSE     : 1. Finds which paths will be affected by this faults.
              2. It eliminates all duplicate source and destination pairs and then checks
                 whether the affected source-destination pairs are ordinary or special nodes.
              3. If they are ordinary, then this function will call
                 "find_alternate_path_for_faulty_path" function to provide alternate patsh for
                 these  affected ordinary paths.

 PARAMETERS   : faulty_node_no and faulty_edge_no.
 RETURN VALUE : none.
================================================================================*/
```

```
void determine_paths_passing_thru_faulty_edge(int faulty_node_no,
                                              int faulty_edge_no)
{
  int mask = 15,k, j, i, source , destination, unique_source_destination ;
  unsigned int node_id;
  int stack_array[2][20], source_destination_array[1458][2];

  for(i=0; (i<1458) ; i++)
    {
          source_destination_array[k][0] = 0;
          source_destination_array[k][1] = 0;
    }

  node_id = convert_int_to_node_id(faulty_node_no, NUM_EDGES);
  j=0;
  for (i = 0; i < NUM_DIGITS; i++)
  {
   stack_array[0][NUM_DIGITS - i - 1] = 0;
   stack_array[1][NUM_DIGITS - i - 1] = node_id & mask;
   node_id = node_id >> 4;
  }
  stack_array[0][NUM_DIGITS] = 0;
  stack_array[1][NUM_DIGITS] = faulty_edge_no;
  for (i = NUM_DIGITS + 1; i < 2 * NUM_DIGITS; i++)
  {
   stack_array[0][i ] = -1;
   stack_array[1][i ] = 0;
  }
  stack_array[0][2 * NUM_DIGITS ] = -1;
  stack_array[1][2 * NUM_DIGITS ] = 0;

  for (i = 0; i < NUM_DIGITS; i++)
  {
     while (stack_array[1][2 * NUM_DIGITS] == 0)
     {
      source = get_source_node_no(stack_array, NUM_DIGITS);
      destination = get_destination_node_no(stack_array, NUM_DIGITS);
      unique_source_destination = 1;
      for(k=0; (k<1458 && (unique_source_destination == 1)); k++)
       if ((source_destination_array[k][0] == source) &&
          (source_destination_array[k][1] == destination))
         unique_source_destination = 0;
       else unique_source_destination = 1;
```

```
        if ((source != destination) &&
           (source_destination_are_ordinary(source, destination)) &&
           (unique_source_destination))
        {
           printf("source = %d, destination = %d\n",
                source, destination);
           find_alternate_path_for_faulty_path(source, destination);
           /*printf("source = %d, destination = %d\n",
                source, destination);*/
           source_destination_array[j][0] =source;
           source_destination_array[j][1] = destination;
           j++;
        }
        add_1(stack_array, NUM_EDGES, 0);
      }
      shift_stack_array(stack_array, NUM_DIGITS);
    }
}


void add_1(int stack_array[][20], int d, int i)
{
    if (stack_array[0][i] == 0)
    {
       add_1(stack_array, d, i+1);
       return;
    }
    if (stack_array[1][i] == d-1)
    {
       stack_array[1][i] = 0;
       add_1(stack_array, d, i+1);
       return;
    }

   stack_array[1][i]++;
}

/*=============================================================================
FUNCTION    : get_source_node_no.
PURPOSE     : 1. Shifts(left) four digits of source_node_id and adds the value of
                 stack_array[1][i] from i = 0 to i = k -1 in a loop to extract the
                 value of source_node_id. from the stack.
              2. Finally converts the source_node_id into an integer.
```

```
 PARAMETERS   : stack_array[][20] and NUM_DIGITS (k).
 RETURN VALUE : source_node_no in terms of integer.
=============================================================*/

int get_source_node_no(int stack_array[][20], int k)
{
   unsigned  int source_node_id = 0;
   int i=0;
   for (i = 0; i < k; i++)
        source_node_id  = (source_node_id << 4) |
                    (stack_array[1][i]);
   return (convert_node_id_to_int(source_node_id,
         NUM_EDGES, BIT_MASK ));
}

/*=============================================================
FUNCTION     : get_destination_node_no.
PURPOSE      : 1. Shifts(left) four digits of destination_node_id and adds the value of
                  stack_array[1][i] from i =  k to i  =  2*k -1 in a loop to extract the
                  value of destination_node_id.from the stack.
               2. Finally converts the destination_node_id into an integer.

 PARAMETERS   : stack_array[][20] and NUM_DIGITS (k).
 RETURN VALUE : destination_node_no in terms of integer.
=============================================================*/

int get_destination_node_no(int stack_array[][20], int k)
{
   unsigned  int destination_node_id = 0;
   int i;
   for (i = k; i < 2*k; i++)
       destination_node_id  = (destination_node_id << 4) |
                      (stack_array[1][i]);
   return (convert_node_id_to_int(destination_node_id, NUM_EDGES,
                       BIT_MASK ));
}

/* =============================================================

 FUNCTION     : shift_stack_array
 PURPOSE      : 1. All values in stack_array[][20] are temporarily stored in temp_stack[][20].
                2. Put -1 and 0 in the first column of stack_array[][0].
```

3. Values of the temp_stack[][20] will be stored in stack_array[][20] from its second column after giving a right shift.
4. To indicate the end, let us put -1 and 0 at stack_array[0][2*k] and stack_array[1][2*k].

PARAMETERS : stack_array[][20] and NUM_DIGITS (k).
RETURN VALUE : none.

==========*/

```
void shift_stack_array(int stack_array[][20], int k)
{
  int temp_stack[2][20], i;
  for (i = 0; i < 20; i ++)
   {
    temp_stack[0][i] = stack_array[0][i];
    temp_stack[1][i] = stack_array[1][i];
   }
  stack_array[0][0] = -1;
  stack_array[1][0] = 0;
  for (i = 1; i < 2 * k; i++)
   {
    stack_array[0][i] = temp_stack[0][i - 1];
    stack_array[1][i] = temp_stack[1][i - 1];
   }

  stack_array[0][2 * k] = - 1;
  stack_array[1][2 * k] = 0;
}
```

/* ==========

FUNCTION : initialise_network()
PURPOSE : 1. Store all concentrator, predecessor, successor, and ordinary nodes in proper arrays.
2. Store all ordinary nodes in the first and their corresponding predecessor nodes in the second column of ordinary_to_pred_nodes_table[][2]. The third column of the array is initialised to zero to ensure that the lightpaths from ordinary to predecessor nodes are "unassigned".
3. Store all predecessor nodes in the first and their corresponding successor nodes in the second column of pred_to_succ_nodes_table[][2]. The third column of the array is initialised to zero to ensure that the lightpaths from predecessor to successor nodes are "unassigned".
4. Store all successor nodes in the first and their corresponding ordinary nodes

in the second column of succ_to_ordinary_nodes_table[][2]. The third column of the array is initialised to zero to ensure that the lightpaths from successor to ordinary nodes are "unassigned".

PARAMETERS : none

RETURN VALUE : none

==========*/

```
void initialise_network()
{
  int i, mask, j, k, num_concentrator_nodes = 0,
     num_predecessor_nodes = 0,
     num_successor_nodes = 0, num_ordinary_nodes = 0,
     digits[NUM_DIGITS],
     ordinary_to_pred_nodes_table_index = 0,
     succ_to_ordinary_nodes_table_index = 0,
     pred_to_succ_nodes_table_index = 0;

  unsigned long int node_id, succ_node_id, pred_node_id;

  for (i = 0; i < NUM_NODES; i++)
   {
    mask  = 15;
    node_id = convert_int_to_node_id(i, NUM_EDGES);
    /*printf("node_id = %u\n", node_id);*/
    for (j = 0; j < NUM_DIGITS; j++)
     {
      digits[j] = node_id & mask;
      node_id = node_id >> 4;
     }

    if (identify_concentrator_nodes(digits))
     {
      concentrator_nodes[num_concentrator_nodes] = i;
     /* printf("concentrator_nodes[%d] = %d\n",
           num_concentrator_nodes,
           concentrator_nodes[num_concentrator_nodes]);*/
      num_concentrator_nodes++;
     }

    else if (identify_predecessor_nodes(digits))
     {
       predecessor_nodes[num_predecessor_nodes] = i;

       num_predecessor_nodes++;
```

```
    }
    else  if (identify_successor_nodes(digits))
      {
        successor_nodes[num_successor_nodes] = i;
        num_successor_nodes++;
      }
      else
       {
         ordinary_nodes[num_ordinary_nodes] = i;
         num_ordinary_nodes++;
       }


}

for ( i = 0; i< num_ordinary_nodes; i++)
{
 for (j = 0; j < num_predecessor_nodes; j++)
  {

  ordinary_to_pred_nodes_table
     [ordinary_to_pred_nodes_table_index][0] =
     ordinary_nodes[i];
  /*printf("o_to_p[%d][0]=%d\n",
     ordinary_to_pred_nodes_table_index,
     ordinary_to_pred_nodes_table
     [ordinary_to_pred_nodes_table_index][0]);*/

  ordinary_to_pred_nodes_table
     [ordinary_to_pred_nodes_table_index][1] =
     predecessor_nodes[j];
  /*printf("o_to_p[%d][1]=%d\n",
     ordinary_to_pred_nodes_table_index,
     ordinary_to_pred_nodes_table
     [ordinary_to_pred_nodes_table_index][1]);*/

  ordinary_to_pred_nodes_table
     [ordinary_to_pred_nodes_table_index][2] = 0;
     ordinary_to_pred_nodes_table_index++;
  }

  for (j = 0; j < num_successor_nodes; j++)
   {
```

```
      succ_to_ordinary_nodes_table
        [succ_to_ordinary_nodes_table_index][0] =
        successor_nodes[j];
      succ_to_ordinary_nodes_table
        [succ_to_ordinary_nodes_table_index][1] =
        ordinary_nodes[i];
      /*printf("succ_to_o[%d][0]=%d\n",
        succ_to_ordinary_nodes_table_index,
        succ_to_ordinary_nodes_table
        [succ_to_ordinary_nodes_table_index][0]);*/

      succ_to_ordinary_nodes_table
        [succ_to_ordinary_nodes_table_index][2] = 0;
        succ_to_ordinary_nodes_table_index++;
  }
}

 for (i = 0; i < num_predecessor_nodes; i++)
   for (j = 0; j < num_successor_nodes; j++)
     {
       pred_to_succ_nodes_table
         [pred_to_succ_nodes_table_index][0] =
         predecessor_nodes[i];
       pred_node_id = convert_int_to_node_id
         ( predecessor_nodes[i], NUM_EDGES);
       succ_node_id = convert_int_to_node_id
         ( successor_nodes[j], NUM_EDGES);
       if (suffix(pred_node_id, NUM_DIGITS -1, NUM_DIGITS)==
          prefix(succ_node_id,
          NUM_DIGITS - 1, NUM_DIGITS))
        {
           pred_to_succ_nodes_table
           [pred_to_succ_nodes_table_index][1] =
           successor_nodes[j];

          /*printf("p_to_s[%d][0]=%d\n",
             pred_to_succ_nodes_table_index,
             pred_to_succ_nodes_table
           [pred_to_succ_nodes_table_index][0]);
           printf("p_to_s[%d][1]=%d\n",
             pred_to_succ_nodes_table_index,
             pred_to_succ_nodes_table
```

```
            [pred_to_succ_nodes_table_index][1]);*/
               pred_to_succ_nodes_table
            [pred_to_succ_nodes_table_index][2] = 0;
            pred_to_succ_nodes_table_index++;
        }
    }
}

/*=============================================================================
 FUNCTION    : convert_int_to_node_id.
 PURPOSE     : Convert the number from integer to node_id.
 PARAMETERS  : number and d (NUM_EDGES).
 RETURN VALUE : node_id

=============================================================================*/

unsigned long int convert_int_to_node_id( unsigned long int number,
                                          int d)
{

   unsigned long int result = 0;
   int remainder[20];
   int quotient, i = 0;
   while (number != 0)
     {
       remainder[i] = number%d;
       i++;
       number /= d;
     }
    for (i = i-1; i>= 0; i--)
     {
        result  <<= NUM_BITS;
        result = result | remainder[i];
     }
   return result;
}

/*=============================================================================
 FUNCTION    : convert_node_id_to_int.
 PURPOSE     : Convert the node_id to an integer
 PARAMETERS  :node_id, d (NUM_EDGES), and mask.
 RETURN VALUE : number
=============================================================================*/
```

```
unsigned long int convert_node_id_to_int(unsigned long int node_id,
                                         int d, int mask)

{  int i;
   int digits[20], digit_no = 0;
   unsigned long int result = 0;
   while (node_id != 0)
      {
        digits[digit_no] =  node_id & mask;
        digit_no++;
        node_id >>= NUM_BITS;
      }
   for ( i =  digit_no-1; i>= 0; i--)
         result = result * d + digits[i];
   return result;
}

/*=============================================================================
 FUNCTION     : prefix
 PURPOSE      :

1.       If prefix_length is less than 1 or the num_digits_in_node_id, it will give an error
         message.
2.       Otherwise, it will return the digits of the node_id from the left most digit up to the
         prefix_length.

 PARAMETERS   : node_id, prefix_length, and num_digits_in_node_id..
 RETURN VALUE : the digits of the node_id from the left most digit up to the prefix_length.

=============================================================================*/

unsigned long int prefix(unsigned long int node_id, int prefix_length,
                         int num_digits_in_node_id)
{
 if ((prefix_length > num_digits_in_node_id) |  (prefix_length < 1))
   printf("prefix is being called with incorrect parameter %x; %d;  %d\n",
           node_id, prefix_length, num_digits_in_node_id);
   return (node_id >> ((num_digits_in_node_id -
          prefix_length)*NUM_BITS));
}
```

```
/*=============================================================================
 FUNCTION    : suffix
 PURPOSE     : 1. If suffix_length is less than 1 or the num_digits_in_node_id, it will give an error
                  message.
               2. Otherwise, it will return the digits of the node_id from the right most digit up to
                  the suffix_length.
 PARAMETERS  : node_id, suffix_length, and num_digits_in_node_id..
 RETURN VALUE : the digits of the node_id from the right most digit up to the suffix_length.

=============================================================================*/

unsigned long int suffix(unsigned long int node_id, int suffix_length,
                         int num_digits_in_node_id)
{
  unsigned long int mask=0;
  int i;
  if ((suffix_length > num_digits_in_node_id) |  (suffix_length < 1))
    printf("suffix is being called with incorrect parameter\n");
  for (i = 0; i < suffix_length; i++)
    mask = (mask<<NUM_BITS) | BIT_MASK;
  return (node_id & mask);
}

/*=============================================================================
 FUNCTION    : randgen
 PURPOSE     : Function to generate random numbers.
 PARAMETERS  : none
 RETURN VALUE : A random integer value.
=============================================================================*/
double randgen ()

{

int i;

 if ( first_time )
   {
    for ( i = 0; i < 32; ++i )
      {
       table [i] = ( 321 * rprev + 123 ) % 100000;
       rprev     = table [i];
      }
    Z = 0;
```

```
    first_time = 0;
   }

 i = ( 32*Z ) / 100000;
 Z = table [i];
 table [i] = ( 321 * rprev + 123 ) % 100000;
 rprev = table [i];
 return (((double) Z)/ 100000);
}


/*=============================================================================
 FUNCTION    : init_rand_gen
 PURPOSE     :  Discard a few random numbers generated by randgen()
                  (To ensure randomization).

 PARAMETERS  : none
 RETURN VALUE : none

=============================================================================*/

void init_rand_gen ()
{

int j;

   for (j=0; j  < rand()% 100000; j++)
      randgen ();
}

/*=============================================================================
 FUNCTION    : process_if_source_or_destination_is_special
 PURPOSE :
```

1. If the source is a predecessor node and the destination is a successor node, it calls delete_path. If the source matches with the first column and the destination matches with the second column of the same row of the pred_to_succ_nodes_table[][3], the third column of that row will be -1 to ensure that the lightpath between the predcessor and the successor node is deleted.
2. If the source is an ordinary node and the destination is a predecessor node, it calls delete_path. If the source matches with the first column and the destination matches with the second column of the same row of the ordinary_to_pred_nodes_table[][3], the third column of that row will be -1 to ensure that the lightpath between the ordinary and the predecessor node is deleted.

3. If the source is a successor node and the destination is an ordinary node, it calls delete_path. If the source matches with the first column and the destination matches with the second column of the same row of the succ_to_ordinary_nodes_table[][3], the third column of that row will be -1 to ensure that the lightpath between the successor and the ordinary node is deleted.

PARAMETERS : source and destination.
RETURN VALUE : none

==========*/

```
void process_if_source_or_destination_is_special(int source,
                                                 int destination)
{
  unsigned int node_id_source, node_id_destination, mask = 15;
  int j, digits_source[NUM_DIGITS], digits_destination[NUM_DIGITS];

  node_id_source = convert_int_to_node_id(source, NUM_EDGES);
  node_id_destination = convert_int_to_node_id(destination, NUM_EDGES);
  for (j = 0; j < NUM_DIGITS; j++)
    {
      digits_source[j] = node_id_source & mask;
      digits_destination[j] = node_id_destination & mask;
      node_id_source = node_id_source >> 4;
      node_id_destination = node_id_destination >> 4;
    }
  if (identify_predecessor_nodes(digits_source) &&
     identify_successor_nodes(digits_destination))
          delete_path(pred_to_succ_nodes_table,
                  NUM_EDGES*(NUM_EDGES-1)*(NUM_EDGES-1),
                  source,
                  destination);
    else if (identify_predecessor_nodes(digits_destination))
          delete_path(ordinary_to_pred_nodes_table,
                  (NUM_NODES - NUM_EDGES*(2*NUM_EDGES -1))
                  *NUM_EDGES*(NUM_EDGES-1),
                  source,
                  destination);
       else if (identify_successor_nodes(digits_source))
          delete_path(succ_to_ordinary_nodes_table,
                  (NUM_NODES - NUM_EDGES*(2*NUM_EDGES -1))
                  *NUM_EDGES*(NUM_EDGES-1),
                  source,
```

```
                destination);

}

/*=============================================================================
 FUNCTION    : delete_path
 PURPOSE     : If the source matches with the first column and the destination matches with
               the  second column of the same row of the table[][3], the third column of this
               row  will be -1 to ensure that the lightpath beween the source and the destination
               is  deleted.

 PARAMETERS  : table[][3], size_of_table, source, and destination.
 RETURN VALUE : none

=============================================================================*/

void delete_path(int table[][3], int size_of_table, int source,
          int destination)
{

  int row_no;
  for (row_no = 0; row_no < size_of_table; row_no++)
   {
    if (table[row_no][0] == source)
       if (table[row_no][1] == destination)
       {
         table[row_no][2] = -1;
         return;
       }
   }
}


/*=============================================================================
 FUNCTION    : generate_faulty_edges
 PURPOSE     : 1. Randomly generates a number of faulty edges.
               2. Calls randgen() to generate a random number.
 PARAMETERS  :none.
 RETURN VALUE :none

=============================================================================*/

void generate_faulty_edges()
```

```
{

 int i, faulty_node_edge_pair;
 for (i = 0; i < NUM_FAULTY_EDGES ; i++)
 {
   faulty_node_edge_pair = NUM_NODES * NUM_EDGES * randgen();
   faulty_nodes[i] =  faulty_node_edge_pair % NUM_NODES;
   faulty_edges[i] =  faulty_node_edge_pair / NUM_NODES;
 }
}

int delete_paths_passing_thru_faulty_edge(int d,
                                          int k,
                                          int faulty_node_no,
                                          int faulty_edge_no)
{
 int mask = 15,  i, source, destination ;
 unsigned int node_id;
 int stack_array[2][20];

 node_id = convert_int_to_node_id(faulty_node_no, d);
 for (i = 0; i < k; i++)
 {
  stack_array[0][k - i - 1] = 0;
  stack_array[1][k - i - 1] = node_id & mask;
  node_id = node_id >> 4;
 }
 stack_array[0][k ] = 0;
 stack_array[1][k ] = faulty_edge_no;
 for (i = k+1; i < 2*k; i++)
 {
  stack_array[0][i ] = -1;
  stack_array[1][i ] = 0;
 }
 stack_array[0][2*k ] = -1;
 stack_array[1][2*k ] = 0;

 for (i = 0; i < k; i++)
 {
   while (stack_array[1][2*k ] == 0)
   {
     source = get_source_node_no(stack_array, k);
     destination = get_destination_node_no(stack_array, k);
```

```
      process_if_source_or_destination_is_special(source,
                destination);
      add_1(stack_array, d, 0);
     }
     shift_stack_array(stack_array, k);
  }
}

/*=============================================================================
 FUNCTION    : find_alternate_path_for_faulty_path.
 PURPOSE     : 1. Find whether the lightpath from the source to a predecessor node is minimum
                  or not.
               2. Find a successor node for which the lightpath from the predecessor to the
                  successor is minimum and the number of lightpath from that successor to
                  destination is not deleted. If unsuccessful, repeat 1 & 2.
               3. Update all the paths from source to predecessor, predecessor to successor, and
                  successor to destination.

 PARAMETERS  : source_node_no and destination_node_no
 RETURN VALUE : none.

=============================================================================*/
void find_alternate_path_for_faulty_path(int source_node_no,
                                         int destination_node_no)
{
 int i, j, no_lightpath, row_no, min_lightpath_pred_node = 0,
    current_index, keep_looking_for_succ, row_index, found_succ,
    index_for_pred_minimum, index_for_succ_minimum, min_lightpath_succ_node = 0, found =
0,
    min = 9999, starting_row_no = -1, num_lightpath_pred_list[NUM_EDGES * (NUM_EDGES
- 1)],
    num_lightpath_succ_list[NUM_EDGES -1];
 unsigned long int pred_node_id, succ_node_id, pred_node_no,
    succ_node_no, conc_node_no;

 for (row_no = 0; (row_no < ((NUM_NODES - NUM_EDGES*(2*NUM_EDGES -1))
          * NUM_EDGES * (NUM_EDGES-1))) &&
          (found == 0)
          ; row_no++)
  if (ordinary_to_pred_nodes_table[row_no][0] == source_node_no)
    found = 1;
  row_no--;
  if (found == 0)
```

```
  {
      printf ("WARNING! check ordinary_to_pred_nodes_table;\n");
      printf("entry not found!! %d and %d\n", source_node_no, destination_node_no);
      exit(0);
  }
for (i = row_no; i < row_no + (NUM_EDGES * (NUM_EDGES - 1)); i++)
   num_lightpath_pred_list[i - row_no] = ordinary_to_pred_nodes_table[i][2];

found = 0;
while (found != 1)
  {
    for (current_index = 0, min = 9999;
       current_index < NUM_EDGES * (NUM_EDGES - 1);
       current_index++)
     {
       no_lightpath = num_lightpath_pred_list[current_index];
       if ((no_lightpath != -1) && (min > no_lightpath))
          {
              min = no_lightpath;
              index_for_pred_minimum = current_index;
          }
     }
    if (min == 9999)
      {
        printf("WARNING!!! WHY CAN WE NOT FIND AN ALTERNATE PATH????\n");
        exit(0);
      }

    min_lightpath_pred_node = ordinary_to_pred_nodes_table
                                [index_for_pred_minimum + row_no][1];
    printf("min_pred = %d\n",  min_lightpath_pred_node);
    found_succ = 0;
    for (row_index = 0; (row_index <
          NUM_EDGES * (NUM_EDGES -1) * (NUM_EDGES -1)) && (found_succ == 0)
        ; row_index++)
    if (pred_to_succ_nodes_table[row_index][0] == min_lightpath_pred_node)
     found_succ = 1;

   row_index--;

    for (i = row_index; i < row_index + (NUM_EDGES - 1); i++)
    num_lightpath_succ_list[i - row_index] = pred_to_succ_nodes_table[i][2];
```

```
keep_looking_for_succ = 1;
while (keep_looking_for_succ == 1)
{
 for (current_index = 0, min = 9999; current_index < NUM_EDGES -1;
      current_index++)
 {
   no_lightpath = num_lightpath_succ_list[current_index];
   if ((no_lightpath != -1) && (min > no_lightpath))
      {
          min = no_lightpath;
          index_for_succ_minimum = current_index;
       }
 }
 if (min < 9999)
  {
   min_lightpath_succ_node =
             pred_to_succ_nodes_table[index_for_succ_minimum
                                      + row_index][1];
   printf("min_succ = %d\n", min_lightpath_succ_node);
   pred_node_id = convert_int_to_node_id(min_lightpath_pred_node,
                                NUM_EDGES);
   succ_node_id = convert_int_to_node_id(min_lightpath_succ_node,
                                NUM_EDGES);
   pred_node_no = prefix(pred_node_id, 1, NUM_DIGITS);
   succ_node_no = suffix(succ_node_id, 1, NUM_DIGITS);
   conc_node_no = suffix(prefix(pred_node_id, 2, NUM_DIGITS), 1,
                  NUM_DIGITS);
   if (look_up_path_table(succ_to_ordinary_nodes_table,
          (NUM_NODES - NUM_EDGES*(2*NUM_EDGES -1))
          *NUM_EDGES*(NUM_EDGES-1),
           min_lightpath_succ_node,
           destination_node_no) != -1)

     {
       update_paths(source_node_no,
                 conc_node_no,
                 pred_node_no,
                 succ_node_no,
                 destination_node_no);
       found = 1;
       keep_looking_for_succ = 0;
     }
    else num_lightpath_succ_list[index_for_succ_minimum] = 9999;
```

```
            }
            else keep_looking_for_succ = 0;
        }
      if (found == 0) num_lightpath_pred_list[index_for_pred_minimum] = 9999;

  }
}

/*=============================================================================
 FUNCTION     : update_paths.
 PURPOSE      : 1. Converts pred_node_no and succ_node_no into pred_node_number
                   and  succ_node_number.
                2. Calls update_single_path with ordinary_to_pred_nodes_table and updates the
                   path between the ordinary and predecessor nodes.
                3. Calls update_single_path with pred_to_succ_nodes_table and updates the path
                   between the predecessor and the successor nodes.
                4. Calls update_single_path with succ_to_ordinary_nodes_table and updates the
                   path between the successor and the ordinary nodes.

 PARAMETERS   : source_node_no, conc_node_no,  pred_node_no, succ_node_no,
                   and destination_node_no
 RETURN VALUE : none.

=============================================================================*/


void update_paths(int source_node_no,
                  int conc_node_no,
                  int pred_node_no,
                  int succ_node_no,
                  int destination_node_no)
{
 unsigned int pred_node_id = 0, succ_node_id = 0;
 int i, pred_node_number, succ_node_number;
 pred_node_id = pred_node_no;
 for ( i = 1; i < NUM_DIGITS; i++ )
 {
   succ_node_id = succ_node_id << NUM_BITS | conc_node_no;
   pred_node_id = pred_node_id << NUM_BITS | conc_node_no;
 }
 succ_node_id = succ_node_id << NUM_BITS | succ_node_no;
 pred_node_number = convert_node_id_to_int(pred_node_id, NUM_EDGES,
                    BIT_MASK);
```

```
  succ_node_number = convert_node_id_to_int(succ_node_id, NUM_EDGES,
                     BIT_MASK);

   /*printf("source_node_no = %d, destination_node_no = %d\n",
   source_node_no, destination_node_no);*/

  update_single_path(ordinary_to_pred_nodes_table,
                (NUM_NODES - NUM_EDGES*(2*NUM_EDGES -1))
                *NUM_EDGES*(NUM_EDGES-1),
                source_node_no,
                pred_node_number);
  update_single_path(pred_to_succ_nodes_table,
                NUM_EDGES*(NUM_EDGES-1)*(NUM_EDGES-1),
                pred_node_number,
                succ_node_number);
  update_single_path(succ_to_ordinary_nodes_table,
                (NUM_NODES - NUM_EDGES*(2*NUM_EDGES -1))
                *NUM_EDGES*(NUM_EDGES-1),
                succ_node_number,
                destination_node_no);
}



/*=============================================================================
 FUNCTION    : update_single_path
 PURPOSE     : If the source_node_number matches with the first column and the
               destination_node_number matches with the  second column of the same row of
               the table[][3], the value of the third column of this row  will be incremented by
               1 to ensure that the number of lightpath between the source and the destination
               is  increased by 1.

 PARAMETERS  : table[][3], size_of_table, source_node_number, and destination_node_number.
 RETURN VALUE : none.

=============================================================================*/


void update_single_path(int table[][3],
                int size_of_table,
                int source_node_number,
                int destination_node_number)
{
```

```
  int row_no;
  for (row_no = 0; row_no < size_of_table; row_no++)
   {
    if (table[row_no][0] == source_node_number)
       if (table[row_no][1] == destination_node_number)
       {
         table[row_no][2] ++;
         return;
       }
   }
  /* printf("num_of_lightpath = %d\n", table[row_no][2]);
    printf("=====================================");
    printf("\n");*/
}

/*=============================================================================
 FUNCTION    : look_up_path_table
 PURPOSE     : If the source_node_number matches with the first column and the
               destination_node_number matches with the  second column of the same row of
               the table[][3], the value of the third column of this row  will be returned.

 PARAMETERS  : table[][3], size_of_table, source_node_number, and destination_node_number.
 RETURN VALUE : returns the number of lightpaths for a specific source and destination.

=============================================================================*/

int look_up_path_table(int table[][3],
                int size_of_table,
                int source_node_number,
                int destination_node_number)
{
  int row_no;
  for (row_no = 0; row_no < size_of_table; row_no++)
   {
    if (table[row_no][0] == source_node_number)
       if (table[row_no][1] == destination_node_number)
           return table[row_no][2];
   }
}

/*=============================================================================
 FUNCTION    : source_destination_are_ordinary
 PURPOSE     : 1. It converts source and destination to source_id and destination_id and checks
```

```
                the digits of the source_id and destination_id to make sure whether they are any
               special nodes or not.
                  2. If both the source and the destination are ordinary, it returns 1, otherwise
                     returns 0.
  PARAMETERS   :source and destination
  RETURN VALUE : returns 1 if source and destination both are ordinary, else returns 0.

================================================================================*/

int source_destination_are_ordinary(int source, int destination)
{
  unsigned int node_id_source, node_id_destination, mask = 15;
  int j, digits_source[NUM_DIGITS], digits_destination[NUM_DIGITS];

  node_id_source = convert_int_to_node_id(source, NUM_EDGES);
  node_id_destination = convert_int_to_node_id(destination, NUM_EDGES);
  for (j = 0; j < NUM_DIGITS; j++)
    {
      digits_source[j] = node_id_source & mask;
      digits_destination[j] = node_id_destination & mask;
      node_id_source = node_id_source >> 4;
      node_id_destination  = node_id_destination >> 4;
    }
  if (identify_predecessor_nodes(digits_source)) return 0;
  if (identify_successor_nodes(digits_source)) return 0;
  if (identify_concentrator_nodes(digits_source)) return 0;
  if (identify_successor_nodes(digits_destination)) return 0;
  if (identify_predecessor_nodes(digits_destination)) return 0;
  if (identify_concentrator_nodes(digits_destination)) return 0;
  return 1;
}

main(int argc, char *argv[])
{
 int i, j, repeat, max_pred, max_pred_succ, max_succ,
    num_lightpath_through_pred, num_lightpath_from_pred_to_succ,
    num_lightpath_from_succ_to_ordinary, o_p_1_value,
    o_p_2_value, o_p_3_value, p_s_1_value, p_s_2_value, p_s_3_value,
    s_o_1_value, s_o_2_value, s_o_3_value;
 int result[1000][3];

 i=0, max_succ = 0;
```

```
for(i = 0; i < atoi (argv[1]); i++)
  init_rand_gen();

for ( repeat=0; repeat < 1000; repeat++)
{
 i=0;
 initialise_network();
 generate_faulty_edges();
 printf("faulty_edges[%d]=%d\n", i, faulty_edges[i]);
 printf("faulty_nodes[%d]=%d\n", i, faulty_nodes[i]);
 printf("faulty_edges[%d]=%d\n", i+1, faulty_edges[i+1]);
 printf("faulty_nodes[%d]=%d\n", i+1, faulty_nodes[i+1]);

 for(i = 0; i < NUM_FAULTY_EDGES; i++)
  {
    delete_paths_passing_thru_faulty_edge(NUM_EDGES,
                                NUM_DIGITS,
                                faulty_nodes[i],
                                faulty_edges[i]);
/*     for (j = 0; j < 12; j++)
      if (pred_to_succ_nodes_table[j][2] != 0)
              printf(" i = %d, j = %d, p-s-value = %d\n",
                  pred_to_succ_nodes_table[j][0],
                  pred_to_succ_nodes_table[j][1],
                    pred_to_succ_nodes_table[j][2]);
    for (j = 0; j < 72; j++)
      if (ordinary_to_pred_nodes_table[j][2] != 0)
              printf(" i = %d, j = %d, o_p_value = %d\n",
                        ordinary_to_pred_nodes_table[j][0],
                        ordinary_to_pred_nodes_table[j][1],
                       ordinary_to_pred_nodes_table[j][2]);
    for (j = 0; j < 72; j++)
      if (succ_to_ordinary_nodes_table[j][2] != 0)
              printf(" i = %d, j = %d, o_p_value = %d\n",
                    succ_to_ordinary_nodes_table[j][0],
                    succ_to_ordinary_nodes_table[j][1],
                    succ_to_ordinary_nodes_table[j][2]); */
  }
 for(i = 0; (i < NUM_FAULTY_EDGES); i++)
  determine_paths_passing_thru_faulty_edge(faulty_nodes[i],
                                faulty_edges[i]);
```

```
/*==========================================================
 To collect the maximum lightpath in ordinary_to_pred_nodes_table.
==========================================================*/

 max_pred = 0;
 for (i=0; i < (NUM_NODES - NUM_EDGES*
          (2*NUM_EDGES -1))*NUM_EDGES*(NUM_EDGES-1) ; i++)
 {
  num_lightpath_through_pred = ordinary_to_pred_nodes_table[i][2];
  if (num_lightpath_through_pred > max_pred)
    max_pred = num_lightpath_through_pred;
 }
 result[repeat][0] = max_pred;
 printf("max_lightpath from ordinary to pred = %d\n", result[repeat][0]);

/*==========================================================
 To collect the maximum lightpath in pred_to_succ_nodes_table.
==========================================================*/

 max_pred_succ = 0;
 for (i=0; i < NUM_EDGES*(NUM_EDGES-1)*(NUM_EDGES-1) ; i++)
 {
  num_lightpath_from_pred_to_succ = pred_to_succ_nodes_table[i][2];
  if (num_lightpath_from_pred_to_succ > max_pred_succ)
    max_pred_succ = num_lightpath_from_pred_to_succ;
 }
 result[repeat][1] = max_pred_succ;
 printf("max_lightpath from pred to succ is =%d\n", result[repeat][1]);


/*==========================================================
 To collect the maximum lightpath in succ_to_ord_nodes_table.
==========================================================*/

 max_succ = 0;
 /*for (i=0;i< (NUM_NODES - NUM_EDGES*
          (2*NUM_EDGES -1))*NUM_EDGES*(NUM_EDGES-1); i++)
 printf("ordinary_to_succ= %d\n", succ_to_ordinary_nodes_table[i][2]);*/

 for (i=0; i < (NUM_NODES - NUM_EDGES*
          (2*NUM_EDGES -1))*NUM_EDGES*(NUM_EDGES-1); i++)
 {
  num_lightpath_from_succ_to_ordinary =
```

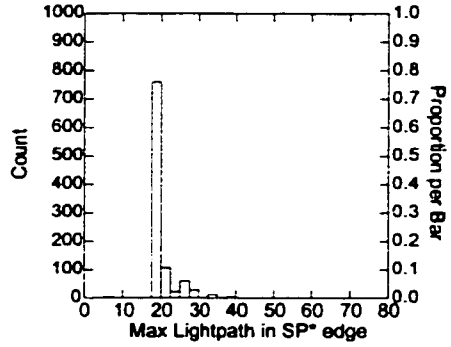
```
        succ_to_ordinary_nodes_table[i][2];
  if (num_lightpath_from_succ_to_ordinary > max_succ)
    max_succ = num_lightpath_from_succ_to_ordinary;
 }
 /*printf("max_succ = %d\n", max_succ);*/
 result[repeat][2] = max_succ;
 printf("max_lightpath from succ to ordinary node is =%d\n",
        result[repeat][2]);


/*=============================================================================
To find the number of light paths  from ord to pred with different loads

=============================================================================*/

      o_p_1_value = 0;
      o_p_2_value = 0;
      o_p_3_value = 0;
      for (j = 0; j < (NUM_NODES - NUM_EDGES*
            (2*NUM_EDGES -1))*NUM_EDGES*(NUM_EDGES-1); j++)
       {
        if ((ordinary_to_pred_nodes_table[j][2] >= 1) &&
           (ordinary_to_pred_nodes_table[j][2] < 6))
          o_p_1_value = o_p_1_value + 1;
        else if ((ordinary_to_pred_nodes_table[j][2] >= 6) &&
              (ordinary_to_pred_nodes_table[j][2] < 61))
          o_p_2_value = o_p_2_value + 1;
        else if ((ordinary_to_pred_nodes_table[j][2] >= 61) &&
              (ordinary_to_pred_nodes_table[j][2] < 121))
          o_p_3_value = o_p_3_value + 1;
       }

      printf("o_p_1_value = %d, o_p_2_value = %d, o_p_3_value = %d\n",
           o_p_1_value, o_p_2_value, o_p_3_value);


/*=============================================================================
 To find the number of light paths  from predecessor to successor nodes with different loads
=============================================================================*/
      p_s_1_value = 0;
      p_s_2_value = 0;
      p_s_3_value = 0;
```

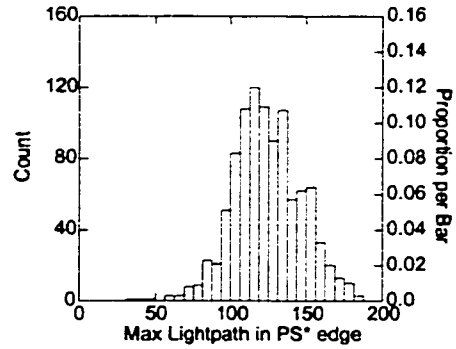
```
for (j = 0; j < (NUM_EDGES*(NUM_EDGES-1)*(NUM_EDGES-1)); j++)
 {
 if ((pred_to_succ_nodes_table[j][2] >= 1) &&
    (pred_to_succ_nodes_table[j][2] < 6))
  p_s_1_value = p_s_1_value + 1;

 else if ((pred_to_succ_nodes_table[j][2] >= 6 ) &&
       (pred_to_succ_nodes_table[j][2] < 61 ))
  p_s_2_value = p_s_2_value + 1;

 else if ((pred_to_succ_nodes_table[j][2] >= 61) &&
       (pred_to_succ_nodes_table[j][2] < 121))
  p_s_3_value = p_s_3_value + 1;
 }

printf("p_s_1_value = %d, p_s_2_value = %d, p_s_3_value = %d\n",
      p_s_1_value, p_s_2_value, p_s_3_value);


/*=========================================================================
To find the number of light paths  from succ to ord with diffrent loads

=========================================================================*/

s_o_1_value = 0;
s_o_2_value = 0;
s_o_3_value = 0;

for (j = 0; j < (NUM_NODES - NUM_EDGES*
       (2*NUM_EDGES -1))*NUM_EDGES*(NUM_EDGES-1); j++)
 {
  if ((succ_to_ordinary_nodes_table[j][2] >= 1) &&
     (succ_to_ordinary_nodes_table[j][2] < 6))
    s_o_1_value =  s_o_1_value + 1;
  else if ((succ_to_ordinary_nodes_table[j][2] >= 6) &&
        (succ_to_ordinary_nodes_table[j][2] < 61))
    s_o_2_value =  s_o_2_value + 1;
  else if ((succ_to_ordinary_nodes_table[j][2] >= 61) &&
        (succ_to_ordinary_nodes_table[j][2] < 121))
    s_o_3_value =  s_o_3_value + 1;
 }

printf("s_o_1_value = %d, s_o_2_value = %d, s_o_3_value = %d\n",
      s_o_1_value, s_o_2_value, s_o_3_value);
```

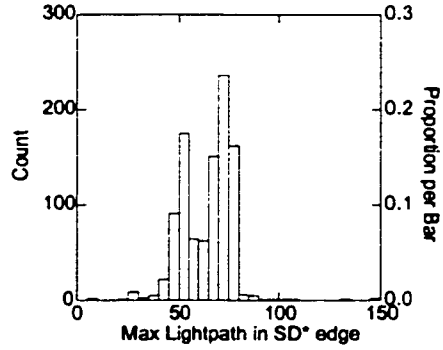
```
    }
}
```

Appendix B

The distributions of the upper limits of the number of communications through different logical edges: source to predecessor edge (SP* edge), predecessor to successor edge (PS* edge) and successor to destination edge (SD* edge) are shown in different diagrams.
Network size = 243 (3^5), number of randomly generated faulty edges = 2.







Statistics:

	SP* edge	PS* edge	SD* edge
N of cases	1000	1000	1000
Minimum	3.000	36.000	6.000
Maximum	77.000	187.000	149.000
Range	74.000	151.000	143.000
Median	20.000	123.000	67.000
Mean	20.913	124.227	64.509
Standard Dev	3.643	22.669	12.758
Variance	13.271	513.899	162.759
Skewness	4.866	-0.009	0.138

Appendix C

Glossary of terms and abbreviations

All-optical network: A type of network in which messages are transmitted and processed entirely in the optical domain, ensuring a high bit transmission rate.

Asynchronous transfer mode(ATM): The proposed mode of operation of the emerging broadband integrated services digital network. All information to be transmitted - voice, data, image, video - is first fragmented into small, fixed-sized frames known as cells. These are switched and routed using packet switching principles - also known as cell or fast - packet switching.

Bandwidth: The difference between the highest and lowest sinusoidal frequency signals that can be transmitted across a transmission line or through a network. It is measured in hertz (Hz) and also defines the maximum information-carrying capacity of the line or network.

Broadcast: A means of transmitting a message to all devices connected to a network. Normally, a special address, the broadcast address, is reserved to enable all the devices to determine that the message is a broadcast message.

Concentrator node: A set of nodes in a network that satisfy some specific conditions are called concentrator nodes. These specific conditions are: there is no edge from one concentrator node to another concentrator node, no two concentrator nodes can have the same predecessor node, no two concentrator nodes can have the same successor node and the predecessor of one concentrator node can not be the successor of another concentrator node.

Fault-tolerant network: A type of network that provides alternate path in the presence of faults. The basic principle behind a fault-tolerant system is to provide the system with redundant resources beyond the minimum requirement for normal operation. These spare resources help avoid faults. Fault-tolerance in a network can be studied in terms of network reliability and survivability.

Frequency-division multiplexing (FDM): A technique to derive a number of separate data channels from a single transmission medium, such as a coaxial cable. Each data channel is assigned a portion of the total available bandwidth.

Lightpaths: End-to-end circuit switched communications traversing one or more fibers and use one WDM channel per fiber.

Logical topology: A graph where the node of the graph represents end-node of the network and two nodes A and B are connected by a directed edge from A to B if there is a lightpath from the end-node A to the end-node B.

Multihop network: A type of network in which messages are routed from the source to the destination node in several hops and are processed at the intermediate points, that is, there are several electro-optical conversions resulting in a low bit transmission rate.

Network reliability: The ability of a network to perform its function, the lower the probability that a network will fail to perform its function, the more reliable it is.

Network survivability: The ability of a network to maintain an acceptable level of performance in the presence of network failure by employing various restoration techniques.

Optical fiber: A type of transmission medium over which data is transmitted in the form of light waves or pulses. It is characterized by its potentially high bandwidth, and hence data-carrying capacity, and its high immunity to interference from other electrical sources.

Physical topology: It provides the physical connections between every pairs of nodes in the network. A network is defined by a physical and a logical topology.

Predecessor node: If a node in a network has an edge to a concentrator node, then the node is called the predecessor node of that concentrator node. A concentrator node can have a fixed number of predecessor nodes depending on the number of edges of the network.

Single-hop network: A type of network in which messages are routed from the source to the destination node in one hop without being processed at intermediate points, that is, there is no electro-optical conversion. It requires a significant amount of dynamic coordination between the source and the destination nodes.

Special nodes (Deflector nodes): The nodes in a network those are not end-nodes are called special nodes. Special nodes can not be a source or a destination node. They contain only optical routers and are used only to replace a faulty lightpath.

Survivable route graph: A digraph that contains all the non-faulty nodes of the original digraph with an edge from node x to node y if there is no faults on the path of the routing from x to y. The notion of survivable route graphs is based on graph theory.

Successor node: If a concentrator node of a network has an edge to a node, then the node is called the successor node of that concentrator node. A concentrator node can have a fixed number of successor nodes depending on the number of edges of the network.

Topology: It defines how the nodes of a network are connected. It can be of two types: physical topology and logical topology.

Wavelength-division multiplexing(WDM): It is one promising approach that can be used to exploit the huge bandwidth of optical fiber. In this type of multiplexing, the optical transmission

spectrum is divided into a number of nonoverlapping wavelength (frequency) bands, with each wavelength supporting a single communication channel operating at peak electronic speed. Thus, by allowing multiple WDM channels to coexist on a single fiber, we can tap into the huge fiber bandwidth.

Wavelength routed network: It is composed of some form of wavelength-selective elements at the nodes of the network . Such node (called wavelength router) makes its routing decision based on the input port and the wavelength of the signal passing through it. Wavelength routing is achieved by demultiplexing the different wavelengths from each input port and then multiplexing signals at each output port.

Bibliography

[1] F. Ayadi, J.F. Hayes, and M. Kavehrad, "Optical-fiber networks: Single-hop and multihop systems", Can. J. Elect. & Comp. Eng., 20(1), p. 5-14, 1995.

[2] S. Bandyopadhyay and A. Sengupta, "Strategies for fault-tolerant all-optical routing in WDM-networks", submitted.

[3] D. Banerjee and B.Mukherjee, "Wavelength-Routed Optical Networks: Linear Formulation, Resource Budgeting Tradeoffs, and a Reconfiguration study", IEEE INFOCOM'97: The Conference on Computer Communications, 1997.

[4] C. Berge, "Graphs and Hypergraphs", North Holland, New York, 1973.

[5] B. Bollobas, Extremal Graph Theory, Academic Press, 1978.

[6] M.. S. Borella, J. P. Jue, D. Banerjee, B. Ramamurthy, and B. Mukherjee, "Optical Components for WDM Lightwave Networks", Proceedings of the IEEE, 85(8), p. 1274-1307, August, 1997.

[7] C. A. Brackett, "Dense Wavelength Division multiplexing Networks: principles and Applications", IEEE Journal on Selected Areas in Communications, 8(6), p. 948-964, August, 1990.

[8] C. A. Brackett, "Is There an Emerging Consensus on WDM Networking?", Journal of Lightwave Technology, 14(6), p. 936-941, June, 1996.

[9] A. Broder, D. Dolev, M. Fischer, and B. Simons, "Efficient Fault-Tolerant Routing in Networks", Information and Computation, 75, p. 52-64, 1987.

[10] M. S. Chen, N. R. Dono, and R. Ramaswami, "A Media-Access Protocol for Packet-Switched Wavelength-Division Metropolitan Area Networks", IEEE Journal on Selected Areas in Communications, 8(6), p. 1048-1057, August, 1990.

[11] I. Chlamtac, A. Farago, and T. Zhang, "Lightpath (Wavelength) Routing in large WDM Networks", IEEE Journal on Selected areas in Communications, 14(5), p. 909-913, June, 1996.

[12] D. Dolev, J. Y. Halpern, B. Simons, and H. Raymond Strong, "A New Look at Fault-tolerant Network routing", Information and Computation, 72, p. 180-196, 1987.

[13] P. W. Dowd, "Wavelength Division Multiple Access Channel Hypercube Processor Interconnection", IEEE Transactions on Computers, 41(10), p. 1223-1241, October, 1992.

[14] M. Eisenberg and N. Mehravari, "Performance of the Multichannel Multihop Lightwave Networks Under Nonuniform Traffic", IEEE Journal on Selected Areas in Communications, 6(7), p. 1063-1078, August, 1988.

[15] O. Gerstel, "On the Future of Wavelength Routing Networks", IEEE Network, p. 14-20, November/December, 1996.

[16] O. Gerstel, R. Ramaswami, and G. H. Sasaki, "Fault Tolerant multiwavelength Optical Rings with Limited Wavelength Conversion", INFOCOM '97, p. 1-27, 1997.

[17] P.E. Green, Jr., "Fiber Optic Networks", Prentice Hall, 1993.

[18] P.E.Green, "Optical Networking Update", IEEE Journal on Selected Areas in

Communications, 14(5), p. 764-779, June, 1996.
[19] N. Homobono and C. Peyrat, "Fault-Tolerant Routing in Kautz and De Bruijn Networks", Discrete Applied mathematics, 24, p. 179-186, 1989.
[20] M. Imase and Y. Manable, "Fault Tolerant Routing in K-connected Networks", Information Processing Letters, 28, p. 171-175, 1988.
[21] M. I. Irshid and M. Kavehrad, "A WDM Cross-Connected Star Topology for Multihop lightwave Networks", Journal of Lightwave technology, 10(6), p. 828-835, June, 1992.
[22] F. Jia and B. Mukherjee, "The Receiver Collision Avoidance (RCA) Protocol for a Single-Hop WDM Lightwave Networks", Journal of Lightwave Technology, 11(5/6), p.1053-1065, May/June, 1993.
[23] F. Jia, B. Mukherjee, and J. Iness, "Scheduling Variable-Length Messages in a Single-Hop Multichannel Local Lightwave networks", IEEE/ACM Transactions on Networking, 3(4), p. 477-488, August, 1995.
[24] J. P. Labourdette and A. S. Acampora, "Logically Rearrangeable Multihop Lightwave Networks", IEEE Transactions on Communications, 39(8), p. 1223-1230, August, 1991.
[25] T. Landegem, P. Vankwikelberge, and H. Vanderstraeten, "A Self-Healing ATM Network Based on Multilink Principles", IEEE Journal on Selected Areas in Communications, 12(1), p. 139-148, January, 1994.
[26] K. Lee and V. O. K. Li, "A Wavelength Rerouting algorithm in Wide-Area All-Optical Networks", Journal of Lightwave Technology, 14(6), p.1218-1229, June, 1996.
[27] Y. Manabe, M. Imase, and T. Soneoka, "Reliable and Efficient Fixed Routings on Digraphs", The Transactions of the IEICE, E 71(12), p. 1212-1220, December, 1988.
[28] M. A. Marsan, A. Bianco, E. Leonardi, and F. Neri, "A Comparison of Regular Topologies for All-Optical Networks", Proceedings of the 12th Annual Joint Conference of the IEEE Computer and Communications Societies on Networking: Foundations for the Future., 1, p. 36-47, March,1993.
[29] M. A. Marsan, A. Bianco, E. Leonardi, and F. Neri, "Topologies for Wavelength-Routing All-Optical Networks", IEEE/ACM Transactions on Networking, 1(5), p. 534-546, October, 1993.
[30] B. Mukherjee, "WDM-Based Local Lightwave networks, Part I: Single-Hop Systems", IEEE Network, 6(3), p. 12-27, May, 1992.
[31] B. Mukherjee, "WDM-Based Local Lightwave networks, Part II: Multihop Systems", IEEE Network, 6(4), p. 20-32, July, 1992.
[32] B. Mukherjee, D. Banerjee, S. Ramamurthy, and A. Mukherjee, "Some Principles for Designing a Wide-Area WDM Optical Network", IEEE/ACM Transactions of Networking, 4(5), p. 684-696, October, 1996
[33] B. Mukherjee, "Optical Communication Networks", McGraw-Hill, July, 1997.
[34] B.Mukherjee, "Networks Architectures for Local Employing Optical Wavelength Division Multiplexing", In F. E. Froehlich and A. Kent, The Froehlich/Kent Encyclopedia of Telecommunications, 12, Marcel Dekker, Inc., New York, USA., p. 353-413, 1996.
[35] D. Peleg and B. Simons, "On Fault Tolerant Routings in General Networks", Information and Computation, 74, p. 33-49, 1987.
[36] D.K. Pradhan, "Fault-tolerant multiprocessor link and bus network architecture", IEEE

Transaction of Computer, C-34(1), p. 33-45, Jan.,1985.

[37] C. V. Ramamoorthy and Y. W. Eva Ma, "Optimal Reconfiguration Strategies for Reconfigurable Computer Systems with no Repair", IEEE Transactions on Computers, c-35(3), p. 278- 280, March, 1986.

[38] B. Ramamurthy, J. Iness, and B.Mukherjee, "Minimizing the Number of Optical amplifiers Needed to Support a Multi-Wavelength Optical LAN/MAN", IEEE INFOCOM'97: The Conference on Computer Communications, Kobe, Japan, p. 261-268, April, 1997.

[39] R. Ramaswami, "Multiwavelength lightwave networks for computer communication", IEEE Communication Magazine, 31, p. 78-88, February, 1993.

[40] R. Ramaswami and K. N. Sivarajan, "Design of Logical Topologies for Wavelength-Routed All-Optical Networks", Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'95), p. 1316-1325, June, 1995.

[41] R. Ramaswami and K. N. Sivarajan, "Routing and Wavelength Assignment in All-Optical Networks", IEEE/ACM Transaction Networking, 3(5), p. 489-500, October, 1995.

[42] R. Ramaswami and K. N. Sivarajan, "Design of Logical Topologies for Wavelength-Routed Optical Networks", IEEE Journal on Selected Areas in Communications, 14(5), p. 840-851, June, 1996.

[43] R. Ramaswami and K. N. Sivarajan, "Optical Networks: A Practical Perspective", Morgan Kaufmann Publishers, Inc., San Francisco, California, 1998.

[44] K. Sato, S. Okamoto, and H. Hadama, "Networks Performance and Integrity Enhancement with Optical Path Layer Technologies", IEEE Journal on Selected Areas in Communications, 12(1), p. 159-170, January, 1994.

[45] A. Sengupta and C. D. Elfe, "On Fault-Tolerant Routing in Interconnection Digraphs", Proc. 1992 International Conference on Parallel Processing, p. III-212 - III-215, August, 1992.

[46] A. Sengupta, A.Sen, and S.Bandyopadhyay, "On an Optimally Fault-TolerantMultiprocessor Network Architecture", IEEE Transactions on Computers, C-36(5), p. 619-623, May,1987.

[47] K. N. Sivarajan and R. Ramaswami, "Multihop lightwave Networks Based on De Bruijn Graphs", Proceedings of the IEEE INFOCOM '91, 3, p. 1001-1011, April, 1991.

[48] K. N. Sivarajan and R. Ramaswami, "Lightwave Networks Based on de Bruijn Graphs", IEEE/ACM Transactions on Networking, 2(1), p. 70-79, February, 1994.

[49] T-H. Wu, and S. Habiby, "Strategies and Technologies for Planning a Cost Effective Survivable fiber Network Architecture Using Optical Switche"s, IEEE Journal of Lightwave technology, 8(2), p. 152-159, February, 1990.

[50] T-H. Wu, "Fiber Network Service Survivability", Artech House, 1992.

[51] T-H. Wu, H. Kobrinski, D. Ghosal, and T.V. Lakshman, "The Impact of SONET Digital Cross-Connect System Architecture on Distributed Restoration", IEEE Journal on Selected Areas in Communications, 12(1), p.149-158, January, 1994.

[52] T-H. Wu, "Passive Protected Self-Healing Mesh Network Architecture and Applications", IEEE Transaction on Networking, p. 40-52, February, 1994.

[53] T-H. Wu, "Emerging Technologies for Fiber Network Survivability", IEEE Communications magazine, p. 58-74, February, 1995.

[54] L.Wilkinson, "SYSTAT 7.0 for windows: statistics", SPSS INC., Chicago, IL., 1997.

[55] B. J. Winer, D. R. Brown, and K. M. Michels, "Statistical Principles in Experimental Design", McGraw-Hill, Inc., 1991.

VITA AUCTORIS

Sanchita Mal-Sarkar was born in Krishnanagar, West Bengal, India. She received her B.Sc. with honors, majoring in Physics from Krishnanagar Government College (Calcutta University), and M.Sc.(Physics) from Banaras Hindu University (India). She is pursuing her graduate studies at the University of Windsor in Computer Science and is expecting completion of her Master's in April, 1999. Her thesis, directed by Dr. S. Bandyopadhyay, deals with the design and simulation of fault-tolerant optical networks based on survival route graphs.