DYNAMIC LIGHT TRAIL CONSTRUCTION IN WAVELENGTH DIVISION MULTIPLEXING (WDM) OPTICAL NETWORKS

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Betty Elizabeth Amuge

In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Program:
Computer Science

December 2016

Fargo, North Dakota

North Dakota State University
Graduate School

Title

DYNAMIC LIGHT TRAIL CONSTRUCTION IN WAVELENGTH DIVISION MULTIPLEXING (WDM) OPTICAL NETWORKS

By

Betty Elizabeth Amuge

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Kendall Nygard
Chair

Dr. Jun Kong

Dr. Jacob Glower

Approved:

December 13th 2016
Date

Dr. Brian M. Slator
Department Chair

ABSTRACT

This paper addresses dynamic light trail construction issues in a WDM optical network. This construction was done with an objective to consume a minimum number of resources, such as, the number of free wavelengths, while ensuring that the light trails were always available and had backups especially in cases of failure. A simulation study had been carried out in a previous study in 2010 (27), and this paper took it further by implementing the proposed dynamic light trail routing algorithms using the Java language as well as running additional simulations. The paper then presented numerical results that compared and evaluated the performance of the dynamic light trail scheme versus the light path scheme, in terms of the accepted connections and consumed free wavelength links. In both, consideration without protection and with protection, was done on various well known Internet topologies. Suggestions for further research were also proposed.

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my advisor Dr. Kendall Nygard for his excellent guidance and support during my academic pursuits. His patience and confidence in me has spurred me to great heights. I would like to thank Dr. Jun Kong and Dr. Jacob Glower for serving on my committee; their invaluable advice and support has gone a long way in shaping my research work into what it is today. I would like to especially extend my gratitude to Dr. Weiyi Zhang of AT&T Labs Research in New Jersey USA, for his support, during my research and course of study. My gratitude also goes to Hamza Kintu of Smart Telecom Uganda for all his assistance. A special note goes to all the staff within the department for their vibrant support during my stay here. Finally, I would like to appreciate both my parents Professor and Mrs. Opuda-Asibo and my entire family for their unwavering support and kindness throughout the years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AC……Add Coupler

ACR…….Accepted Connections Ratio

ADM(s)……Add/Drop Multiplexer(s)

CS……Cantor Set Dividing Ratio

CSGM……..Cantor Set Growth Method

DC……Drop Coupler

DLIT…….Dynamic Light Trail

DMUX……..Demultiplexing

Gbit/s…….Giga bits per second

GB……..Giga Byte

GHz……Giga Hertz

GUI…….Graphical User Interface

IEEE……...Institute of Electrical and Electronics Engineers

IP……Internet Protocol

ISP……..Internet Service Provider

ILP……..Integer Linear Programming

JDK…….Java Development Kit

Km……..Kilo metre

LAN(s)……Local Area Network(s)

LP……Light Path

LT……Light Trail

LTE……..Line Terminating Equipment

MAN……Metropolitan Area Network

Mbit/s……...Mega bits per second

MPLS……...Multiprotocol Label Switching

ms……milli second

MUX……Multiplexing

OBS……..Optical Burst Switching

OEO……..Optical-Electrical-Optical

PC……..Personal Computer

RAM……..Random Access Memory

RPR……Resilient Packet Ring

RX…… Receive

TB……..Tera Byte

TDM……...Time Division Multiplexing

TX……...Transmit

WDM……..Wavelength Division Multiplexing

LIST OF SYMBOLS

A……...Number of light trails already created

B……...Hop length of last created light trail

c……convener node

C……...Capacity of each light trail wavelength

e……end node

E……...Set of m links

G……..Directed network

L_1…….Convener node

L_2…….End node

L_{max}……..Maximum hop length

LT_i……Set of light trails on WG_i

M…….Cost of an edge

m_i…….Number of edges on WG_i

N……..Node or hop

N_A……Arbitrary node in a ring of N nodes or Convener node

N_B…… End node

p……Number of times the original light trail of (N-1) nodes has been divided

s……Upstream node or convener node

t…….Downstream node or end node

T……Traffic flow matrix

T_{ij}……...Time averaged flow from node N_i to node N_j

V……Set of n nodes

W……Wavelengths on each link

WG_i…….Wavelength plane corresponding to wavelength λ_i

WG_i'……Auxilliary wavelength plane constructed from WG_i

y……..nodes

x……..edges

λ_i ……Wavelength

ε……..Small cost of short cut edge (c, e)

Λ……Wavelength set

μs……micro second

LIST OF APPENDIX FIGURES

1. INTRODUCTION

1.1. Background

Fiber Optical networks are high-capacity telecommunications networks based on optical technologies and components that provide routing, grooming (21, 28, 29), restoration at the wavelength level, as well as wavelength-based services. They are used in video conferencing, science visualization, real-time medical imaging, high speed computing, and many other applications. Using the wavelength division multiplexing (WDM) technology in optical networks is believed to be a top choice for meeting all these demands made on the network especially for the next-generation backbone network, since it effectively increases the bandwidth of a single link from 10 Mbit/s to over 160 Gbit/s (27). Therefore, a major concern for most optical networks that use WDM is with service continuity and network survivability, which have become the most vital issues to deal with, in seeking the greatest efficiency and optimal performance levels to meet the demands made on the network.

To meet the bandwidth requirements of various applications, the WDM technique, makes use of the large amount of bandwidth in optical fibers, by partitioning the bandwidth into a large number of channels. This allows multiple data streams to be simultaneously transferred along the same piece of optical fiber.WDM systems with 160 or more channels and data rates per channel up to 10 Gbit/s with transmission distances up to 3000 km have to be implemented in the backbone network. The backbone network interconnects diverse networks and has a greater capacity than the networks it interconnects, providing a path for information exchange either in the same building (between subnetworks), or in different buildings over wide areas (between different Local Area Networks (LANs)). The WDM technique consists of nodes interconnected with fiber optic links and should therefore be used in conjunction with wavelength routing. It assigns incoming

optical signals to specific frequencies of light (wavelengths, or lambdas) within a certain frequency band. WDM couples several different colours of light onto one fiber, with each colour being an independent data path.

The first major fibre optical networking communication method is based on lightpaths (6) or optical circuits that are set up in such a way that a complete or entire wavelength is exclusively used by the connection's source and destination node-pair, without allowing any sub-wavelength sharing between nodes along the light path (27). Often the dynamically varying bandwidth requirements that accommodate the ever increasing bandwidth on demand applications prevalent in today's Internet traffic, uses mostly IP centric communication (4, 10, 14, 21, 22, 25, 26, 28, 29). The allocation of a complete wavelength as described above is not an efficient means for bandwidth provisioning. It is therefore not justified, because it leads to the underutilization of the wavelength capacity, and costly network element deployment. Networks using wavelength routing and lightpath communication are thus over-provisioned and as a result, IP centric communication will suffer since it will not be able to get the dynamic guarantees of bandwidth provisioning that it requires. However, in comparison to the light path communication, light trails (5, 11, 13, 14, 17, 18) provide dynamic provisioning and therefore offer a low cost alternative.

The light trail was proposed to bypass the problem of the continuous reconfiguration of switches and to solve the inability of intermediate nodes to use a connection wavelength. In (16), a light trail is described as having multiple nodes that can take part in communication without the need for optical switching (11, 17). It also mentions that the principle of a light trail is like that of an optical bus, in which multiple nodes along the bus can communicate to their downstream nodes, given that no two nodes will transmit at the same time. These light trails exhibit properties such as optical multicasting, sub-wavelength granular support, which is a low-cost for deployment and

allows for dynamic provisioning of bandwidth at the optical layer through a unique node architecture and out-of-band control protocol. There are static and dynamic light trails. This paper examines the dynamic light trails which are discussed in detail under literature review.

1.2. Objectives and Justification

Today's Computer Networks demand for the highest efficient service possible and this is especially the case in wavelength-division multiplexing (WDM) optical networks. However, efficient service should not be achieved at the cost of the underutilization of wavelength capacity which would definitely occur in WDM optical networks. This wastage of resources is evident, since each light path between the source and destination device-pair has an entire exclusively dedicated wavelength that is not to be shared with other devices along the light path. The waste is even more prominent since bandwidth requirements are constantly varying in today's networks. Light trails which are either static or dynamic go a long way into solving this dilemma.

The main issue in the constructing of dynamic light trails in a WDM optical network is to minimize the resources used, such as wavelengths, wavelength channels, the total system travel cost and also the time required for the construction, therefore exploiting the capabilities of each of the nodes in the optical network. This paper, presents the theoretic concepts developed, leading to the development of an efficient algorithm that for; light trail availability and survivability purposes establishing dynamic light trail routing and construction for any new connection request, while a minimum of network resources are used. The design of this construction needs to be presented for a loose virtual topology with a required connectivity property, which reserves a few wavelengths to cope with dynamic traffic demands properly. In addition to this, it examines the issue of network availability and survivability, and since the light trails are dynamic there is the need to address the consistency of these light trails in terms of performance and backup availability

in case of failure. However, efficient dynamic light trail construction in all-optical WDM networks is still a challenging problem, making this study essential. Comparison of the light trail technique to the existing light path technique is necessary to determine performance levels, operation levels, and future improvements that are needed to increase efficiency in WDM optical networks.

The optimization requirement: From multiple nodes, the spatially diverse sub-wavelength flows are groomed in the wavelength bus or light trail by arbitrating the bandwidth in the light trail to accommodate each flow. The light trail represents an adept medium for several nodes to share wavelength bandwidth. For bandwidth arbitration in light trails, an out-of-band control channel is fundamental. In addition to this the out-of-band channel dimensions, sets up, and tears down light trails in the network. This double abstraction is essential to enable the network to support traffic growth. It is important to note that due to the absence of optical switching, the setting up and tearing down of light trail connections is dynamic and therefore significantly time-consuming. This is highlighted in (16), where it mentions that the time required to set up connections is of the order of 10 μs, while the time to set up light trails was three orders of magnitude higher at 2.4 ms. For each new connection request, there is therefore need to minimize the probability of setting up a light trail, or create a topology of light trails on which traffic is readily routed, and these light trails are mapped to nodes and wavelengths across the network. A constrained optimization problem is realized due to the combined problem of considering the temporal aspects of traffic while grooming spatial sub-wavelength flows into light trails.

To be able to set up the dynamic light trails in the best possible manner, according to (16, 19), there is need to minimize; the resources such as the wavelengths used; the provisioning time, and also the need to maximize the possibility that a new connection request will find an available light-trail, thus reducing the probability of needing to create an extra light-trail. Note that light trail

communication is constrained by the bus property, that is, upstream nodes have a higher priority for establishing connections compared to downstream nodes. This leads to a situation in which, if a node X is in the process of sending data over a connection, and an upstream node Y desires to use the light trail at the same time, then node X has to halt its transmission in order to allow upstream node Y to transmit. This leads to queuing delays on nodes in a light trail, depending on their position within the trail with respect to the convener of the light trails.

There is need to construct a system in which we assign optimal dynamic light trails across a network graph such that the traffic latency realized from provisioning connections or due to new light trails will fall within the bounds required by the traffic demands. This will ensure that the optimization process which includes the proper utilization of wavelength capacity, the survivability of the network in cases of failure, and other major characteristics that are vital to WDM optical networks can be achieved.

This paper studies the dynamic light trail construction problem in WDM optical networks and proposes a solution to meet the objectives mentioned above which lead to achieving the set of requirements that are vital to efficiency such as, low-cost deployment, dynamic bandwidth provisioning which would ensure low experienced delay, and fast hand-off. The light trail technology has the details of all these requirements mapped to it. It tries to find a light trail to carry each dynamic connection request that arrives, and does so with an objective of consuming a minimum number of free wavelengths while also considering the survivable connection provisioning in which the light trail routing finds working and backup light trails for the connection request.

Examined in this paper are various issues that are organized as follows: chapter 2 examines works related to this study; chapter 3 discusses the design problem issues related to dynamic light

trail construction, and then examines various test cases used to develop the most effective algorithm for the dynamic light trail construction and assignment; chapter 4 discusses the methodology and implements the dynamic light trail algorithm using the java language and discusses it in detail; chapter 5 presents the simulation results obtained, and also makes comparisons and performance evaluations between the Dynamic light trail routing schemes and the light path routing scheme with respect to various Internet topologies; and finally chapter 6 presents the conclusion in which observations made and future work are discussed.

2. LITERATURE REVIEW

Over the past decade the demand to handle large data streams efficiently and at high speeds has been rapidly and continuously increasing. In an effort to efficiently provide and utilize the bandwidth on demand offered by a single wavelength between multiple nodes or users, several techniques have been proposed, and researched. They include the following.

Gigabit Ethernet, which as defined by the IEEE 802.3-2008 standard, refers to various technologies that transmit Ethernet frames at a rate of a gigabit per second. As an optoelectronic technique, Gigabit Ethernet provides a lightpath connection for IP centric traffic flow between a source and destination node, with the traffic flow oblivious to the intermediate nodes. It therefore does not serve as a suitable technique to efficiently utilize the bandwidth or provide sub-lambda type traffic between multiple nodes.

The Resilient Packet Ring (RPR) (23), is another optoelectronic technique, which is an IEEE 802.17 protocol standard, that was designed to meet the requirements of a packet based Metropolitan Area Network (MAN) and to ensure optimized transport of data traffic over an optical fibre ring type of network. It effectively utilizes the capacity of an optical link when it allows consecutive nodes to communicate in a downstream direction, with the optical signal at each node getting stripped, and electronically processed such that it can free space to carry additional traffic, allowing other local traffic to get catered to as the signal goes on its way to the destination node. The downside of the RPR technique is that expensive high performance electronics will be needed to handle signal stripping at each node and as mentioned in (27), since RPR is a slotted technique, it therefore faces synchronization problems and as a result utilization problems arise. This makes RPR, not a favourable technique to handle the bursty IP traffic common on networks today.

In line with the objective of this paper, the following techniques were examined, because they were proposed as techniques that efficiently use available fiber optic resources and thereby enhance the WDM network's abilities: Optical Burst Switching (OBS) (10, 25, 26); Multiprotocol Label Switching (MPLS) (24) which directs data traffic from one node to another based on the shortest path rather than network addresses, thereby bypassing complicated searches through the routing table; and Traffic Grooming (21, 28, 29) which aims to minimize network costs, such as the cost and usage of line terminating equipment (LTE) or add/drop multiplexers (ADMs), so it groups many small data (for example, a network using both time division multiplexing (TDM) and WDM) flows that are destined for a common node into larger units on the same wavelength to be processed as single entities, allowing them to be dropped by a single optical ADM.

The Light trail technique researched in (14) was then proposed as a way to eliminate; the constant costly reconfiguration of switches; and the intermediate nodes failure to utilize a connection wavelength available to a given source and destination node pair. Frederick, et al (9) took the research on light trails further by proposing a technique that would help to get rid of collisions that were prevalent and common between different connections along light trails. With respect to light trail routing, it is very important to efficiently construct light trails that will carry traffic in WDM networks. A study examining the static light trail routing problem was researched by the authors in (7, 8), and (27) studied dynamic light trail routing, with connection requests coming dynamically, one at a time. In light trail routing, a working light trail requires a protection backup, light trail. It is therefore vital to have a continuous flow of traffic between any end-to-end node pair, especially in case of optical link failure. Baring this in mind, the authors in (20), for static traffic, examined the survivable design in light trail fibre optical networks, and the authors

in (27) also examined the survivable routing provisioning for dynamic traffic. The latter will be expounded on in this paper.

2.1. Understanding the Light Trail Technology

With the exponential surge in data traffic today, it is with greater emphasis that the use of light trail technologies in WDM fiber optic communication is being addressed by researchers and various interested establishments. This is because the use of light trails in fiber optic communications has been recognized as a driving force towards obtaining low cost alternatives through providing dynamic bandwidth provisioning in comparison to light path communication. To minimize the major problems realized in trying to sustain IP centric communication (4, 22) at the optical layer, the authors in (14) proposed and implemented the concept of light-trails (5, 11, 12, 13, 15, 17, 18).

A light trail, as seen in (27) may be defined as a unidirectional dynamic light path between a source node and a destination node, with the communication being from a node on the path to a destination node that is downstream on the path, without performing optical switch configuration (11, 17) at each node. The light trail has been likened to an optical bus with multiple nodes that can communicate to their downstream nodes as indicated in *Figure 1* below, obtained from (16), in which a wavelength is switched between the convener node which is the first node in a light trail, and the last node in the light trail referred to as the end node. In cases when the light trail connection is dynamic, connection management is handled by the out-of-band control protocol which facilitates communication and resolves light trail conflicts. As seen in (12) this protocol handles management of the light trail and dynamic connections, in that it sets up, tears down, and adjusts the dimensions of the set of light trails as needed.

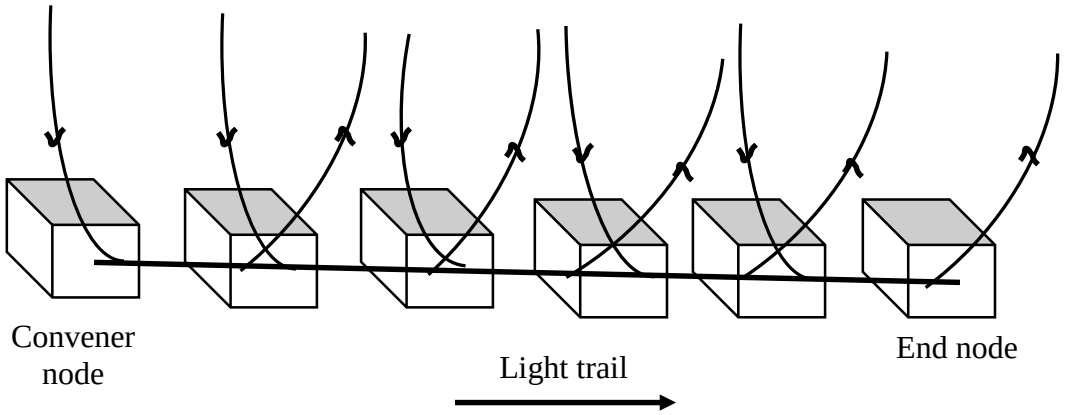

*Figure 1***.** The Concept of a Multi-Point Lightpath or Simply, a Light Trail (16).

According to (27), Light trail nodes contain a local access section for each wavelength and wavelength multiplexing (MUX) and demultiplexing (DMUX) units as illustrated in the light trail node architecture illustration in *Figure 2* below. This local access section has two passive couplers, namely, the drop coupler (DC) which is the first coupler used for dropping the signal, and the add coupler (AC) which is the second coupler used for adding a local signal. These couplers are separated by an optical shutter which is a fast ON/OFF optical switch that can be configured to control the initiation and termination of light trails, by configuring it to be in the ON position at each intermediate node and then when on the desired wavelength at the convener and end nodes, configuring it to be in the OFF position.

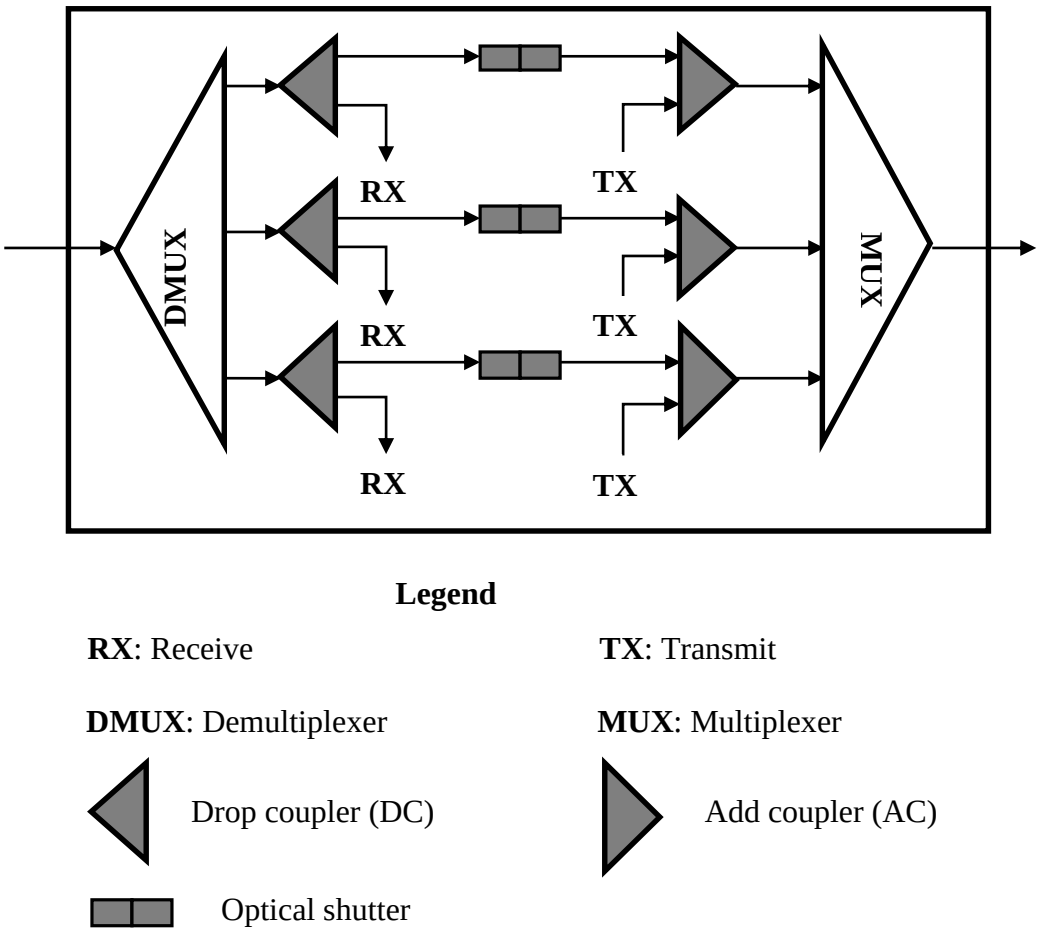

Figure 2. Light Trail Node Architecture (27).

Gumaste and Chlamtac (14) proposed a protocol to establish and dimension light trails also Frederick et al. (9), Gumaste and Chlamtac (14) research was done to eliminate conflicts in communication between opened optical paths found in a light trail.

2.1.1. The Need for the Light Trail

In fiber optics communication, light trail technologies may be used instead of light path technologies because of high efficient performance realized due to high levels of utilization, multicasting and low access delays, as indicated in (5, 14) without the need of fast optical

switching, making it cost effective as well. The reasons why light trail technology is needed can be seen from its properties as was discussed in (27) and they go as follows:

(i) The light trail technology eliminates costly Optical-Electrical-Optical (OEO) switching at intermediate nodes, which are a necessity in traffic grooming. This is done when using the drop and continue function (9, 14) where rather than dropping the entire optical signal at an intermediate node, the intermediate node only uses a sufficient amount of power from the signal to handle local processing and determine whether the data is for it and then the remainder is sent through the optical shutters that are configured to handle optical routing decisions. Optical-Electrical-Optical (OEO) switching refers to a design in which all the input optical signals are converted into electronic signals after they are demultiplexed by demultiplexers and these electronic signals are then switched by an electronic switch module, then are converted back into optical signals by using them to modulate lasers and the resulting optical signals are multiplexed by optical multiplexers onto outlet optical fibers.

(ii) According to the authors in (9, 14), the light trail technologies they presented, provided collision avoidance and re-transmission schemes which are vital for efficient communication. This was necessary, because there could be multiple connections on a light trail, and each connection takes up the light trail exclusively. Then as long as the optical bus is free, each node can then transmit data to any downstream node.

(iii) Light trails can be adjusted or dimensioned to meet certain desired specifications or criteria, giving them the advantage of adaptability in handling varying connection requests as per the proposed protocol in (14).

(iv) Optical signals undergo damage as observed from the optical signal power losses and varying impairments imported by optical fibers and other components. Due to this fact, in comparison to the light path, the light trail cannot be very long. In order to minimize on optical power losses, in studies (5, 8) the expected hop length of a light trail used is 5. This is the same length limitation used in this paper.

2.2. Light Trail Heuristic Algorithms

There are two kinds of heuristic algorithms examined in (16) that deal with light trail construction and these are:

(i) Static light trail assignment heuristic algorithms, in which the traffic matrix is known or predefined for all the network nodes and a quick solution is to be determined. There have been a number heuristic algorithms proposed for 2 static traffic routing schemes like the three examined in (16).

(ii) Dynamic light trail assignment heuristic algorithms in which the traffic matrix is unknown and an adaptable solution is to be determined. This was examined in (27).

This paper addresses the latter - dynamic light trail assignment, algorithms and construction.

2.2.1. Dynamic Light Trail Assignment Algorithms

The light trail routing scheme algorithms proposed for dynamic traffic have been examined in (27) to a great detail, and in (16). The three dynamic assignment algorithms proposed by the authors in (16, 27) are described below.

2.2.1.1. The Cantor Set Growth Method (CGSM)

According to this method in (16), for a ring network with *N* nodes, the authors created two light trails L_1 and L_2 with the constraint of having N-1 hops. The convener node of L_1 is the end node of L_2 and the end node of L_1 is the convener node of L_2. The system they use is able to cover N^2 connections and has two light trails each moving in an opposite direction from the other. They start by placing traffic in the two light trails, giving the condition that, in case the total traffic flow will exceed what the two light trails support, then they can create Cantor segments of each light trail to make new light trails as shown below in *Figure 3*.The idea is to consider the existing light trails as ancestors of the next Cantor segments. The relation between the ancestor and descendant is the Cantor set dividing value (CS) which here is 2. As observed in *Figure 3* below, they start by having one light trail of $N - 1$ hops, and with the increase in the traffic flow they create 2 new light trails each of $(N - 1)/2$ hops and this Cantor set procedure of creating fractals of the previous light trails is repeated until the traffic flow is routed. This scheme has a short coming in that in an effort to facilitate good utilization, it ends up re-routing existing connections. If re-routing was not done, a situation can be reached where a combination of shorter light trails have enough capacity to route a connection between nodes that are far apart but an existing longer light-trail is busy accommodating shorter connections.

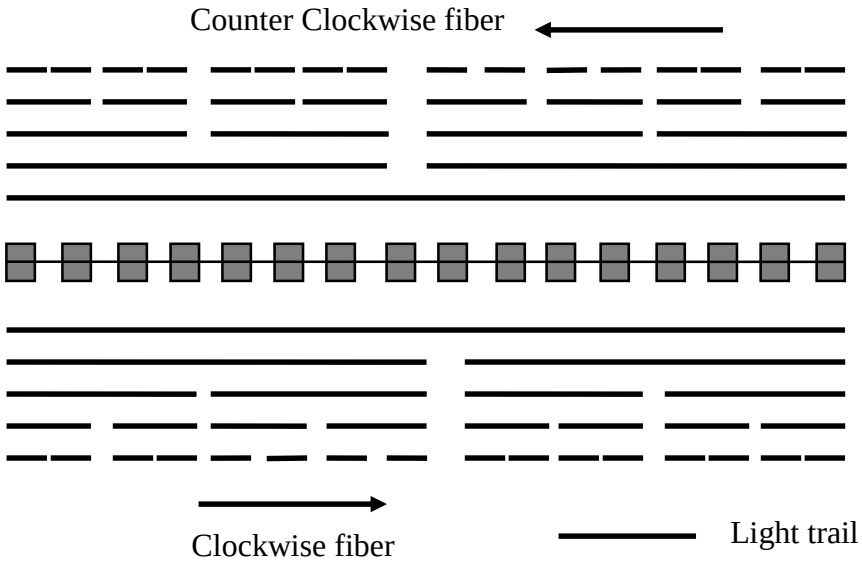

Figure 3. Cantor Set Growth Method (16).

With A as the number of light trails already created, B as the hop length of the last created light trail, CS as the Cantor set dividing ratio, T as the traffic flow matrix, T_{ij} as the time averaged flow from node N_i to Node N_j, and finally with p as the number of times the original light trail (of $N - 1$ nodes) has been divided, then,

While $T \neq 0$

Create CS*A light-trails each of B/CS hops

Route traffic:

For h = 1:p

For each of the h*CS light-trails (with N/CS*h hops)

Do:

If traffic $T_{ij} \in$ of light-trail, and $|i - j| \leqslant N/CS * h$

Assign T_{ij} to this light-trail

$T \leftarrow T - T_{ij}$

End

End

2.2.1.2. Decrement Wrapping

In this method as proposed in (16), N_A is an arbitrary node in a ring of N nodes. A clockwise light-trail of $N - 1$ hops is set up starting at N_A as the convener node to an end node $N_A+(N-1) = N_B$.Then another light trail is set up in the counter clockwise fiber, of N-1 hops, starting from node N_B and ending at node N_A. With C as the capacity for each light trail wavelength, these two light trails can provide complete N^2 connectivity given that the total capacity required by the N^2 connections is less than 2C. On the other hand, if the load is greater than 2C then two light trails are just not sufficient to provide complete N^2 connectivity, hence the need for more light trails. The authors used decrement wrapping to do this, whereby, whenever the set of existing light trails was not sufficient for the traffic flow, a new set of light trails would be built, with the longest in the new set, decremented by one hop from the longest light trail in the previous iteration. The idea is to route longer and shorter connections in longer light trails and shorter light trails respectively, while ensuring that there are enough light trails to cater to very short duration traffic, by providing a large set of very small light trails.This is illustrated below in *Figure 4*. The position of decrementing the next light trail is a random one.

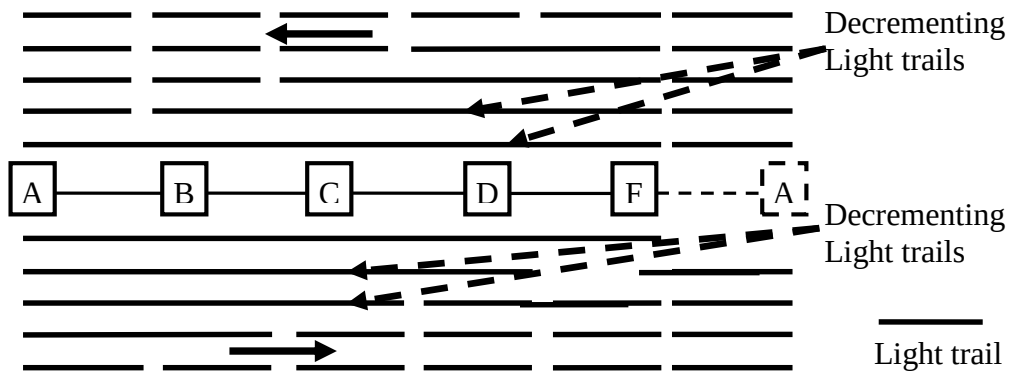

Figure 4. Decrement Wrapping Scheme for Dynamic Assignment (16).

2.2.1.3. The Dynamic Light Trail Routing Algorithm (DLIT)

This algorithm was proposed in (27) where by connection requests come and leave dynamically. The algorithm was further expanded by the authors to enable survivable routing provisioning for the dynamic traffic. The algorithms are displayed in *Figure A1*, *Figure A2*, *Figure A3*, and *Figure A4* (**Appendix A)** and are discussed in detail in chapter 3, where the dynamic light trail design problem is examined.

3. THE DESIGN PROBLEM

This paper focuses on the construction of dynamic light trails in WDM networks, where the connection requests arrive and leave dynamically and also examines the survivable routing provision. It is an expansion of the study in (27). This expansion involves having the algorithms in (27) that is, algorithms 1, 2, 3 and 4 shown in *Figure A1*, *Figure A2*, *Figure A3*, and *Figure A4* respectively (**Appendix A**), converted to the Java language and carrying out tests to compare the LT and the LP performance at much higher wavelengths. The results from these tests are displayed in graphical format.

Although, the works of most of the authors referenced in this chapter were focused on static traffic routing with a known or pre-defined traffic matrix, they were greatly instrumental in laying a strong foundation and enabling the research leading toward achieving objectives stated at the beginning of this paper. Similar ideas and principles were used here, with the exception that in this paper and in (27) they are applied to dynamic traffic flow.

The authors in (19) proposed the use of a tree shaped version of the light trail as they examined developing an efficient light trail based virtual topology for a given network traffic flow. Fang, et al. (8) used a traffic matrix and a directed WDM network in their examination of the static light trail routing problem. Their objective was to satisfy all traffic demands using a minimum number of light trails through: preprocessing the traffic matrix thus obtaining shortest paths between all source-to-destination pairs and satisfying the hop length constraint by dividing every single long hop (that is ≥ 5) into multiple hops and; finding the minimum set of light trails by an Integer Linear Programming (ILP) formulation. Similarly, in (3), the static light trail routing problem was studied, but with an objective to accommodate all the static traffic using a minimum number of wavelengths. A heuristic algorithm and an enhanced ILP formulation were proposed in

(2) to enable static traffic routing using the least number of light trails. Then in their examination of survivable light trail routing for static traffic, the authors in (1) provided a heuristic algorithm that would reduce the number of light trails while providing protection and accepting a maximum number of connections. Keeping in mind the ideas and principles obtained from these authors, the problem statement also in (27) is examined as is shown in the next section of this chapter.

3.1. The Dynamic Light Trail (DLIT) Construction Problem

A possible way to deal with the DLIT construction problem would be to obtain a lightpath that is longer than the maximum hop length (L_{max}) and consumes a minimum number of wavelengths links, then divide it into several parts which will be setup as light trails of lengths that are less than or equal to L_{max}. The major disadvantage realized with this method though is that the existing light trails will not be efficiently re-used, thereby needing more Optical-Electrical-Optical (OEO) conversions especially since each part needs OEO. According to (5, 17), light trail dimensions need time to process, and using this method with the existing light trails getting constantly changed for dynamic traffic, implies that too many light trail dimensions are produced, thus adding an extra load to the network system and leading to inferior light trail routing and construction. Bearing this in mind, in order to preserve most of the light trails for re-use and to lessen OEO conversions, the light trails examined in this paper are kept to a L_{max} of no more than five (5, 8, 27). Below is the problem statement.

Given: a directed network G (V, E, Λ), with V as the set of n nodes, E as the set of m links and Λ as the wavelength set; LT the set of existing light trails; L_{max}; a new connection request (**s, t**) with s as an upstream node and **t** as a downstream node. There are W wavelengths on each link, and an assumption is made that there is no wavelength converter at any node in G. WG_i is the wavelength plane corresponding to wavelength λ_i, and it is composed of: all links in G for which

wavelength λ_i is available; all nodes in G and; all the light trails using wavelength λ_i. LT_i is the set of light trails on the wavelength plane WG_i, it is a light trail in LT and, WG_i' is the auxiliary wavelength plane constructed from WG_i in order to find a light trail for the request on WG_i. For the connection request, the DLIT construction problem seeks a single or multi-hop light trail with the constraint of being no longer than L_{max} that consumes the minimum number of free wavelengths.

3.1.1. The DLIT Connections with Protection

Survivability and protection of a network refers to its ability to maintain an acceptable level of service during network or equipment failure. In this respect therefore, to ensure that little to no information is lost due to link failure, for each connection request received on a WDM optical network, it is vital that a working light trail is found and that it should have a backup light trail.

Accordingly algorithm 4 in *Figure A4* (**Appendix A**) endeavors to find a maximum of W working light trails as indicated by its use of lines 1-29 of Algorithm 1 in *Figure A1* (**Appendix A**). A working light trail is chosen from any one of the k minimum light trail candidates found in CandidateLightTrail, then a backup or link-disjoint protection light trail is found for it. If a backup light trail is found for the working light trail, then both the working and backup light trails are returned and the connection request is accepted, but if there is no backup light trail for this working light trail, it moves to the next working light trail candidate to try find one again. There are two cases in which the connection request is dropped and that is: when there is no candidate working light trail found in the CandidateLightTrail and; if none of the **k** working light trail candidates from the CandidateLightTrail can have a protection or backup light trail.

3.1.2. The DLIT Connections without Protection

Following the proposed DLIT routing algorithm in (27), the same algorithm is used for the dynamic light trail construction without the protection provision, as is shown in Algorithm 1 of *Figure A1* (**Appendix A**). Being that this study is dealing with dynamic traffic, it is assumed that for any new incoming connection request (**s, t**) on each wavelength plane WG_i of the directed network G, a set of existing light trails LT_i has existed, and so the following scenario are considered for each request:

(i) If both **s** and **t** are on an existing light trail with **s** as the source or upstream node and **t** as the target or downstream node, then the connection request is satisfied by reusing the light trail. Open the connection (**s, t**) on the light trail and return. This is shown in lines 5-7 of algorithm 1 in *Figure A1* (**Appendix A**).

(ii) If (i) above is not satisfied, there is need to construct a new light trail for the connection request by doing the following:

  (a) Find a light trail for the new request on each wave length plane.

  (b) Select the light trail with the minimum wavelength consumption and set up the connection.

  (c) In order to find a light trail for the connection request on a wavelength plane WG_i, there is need to first construct an auxiliary wavelength plane WG_i' from WG_i. This is shown in line 2 of algorithm 1 in *Figure A1* (**Appendix A**). All the free edges that are not in any light trail on WG_i are copied into WG_i' and each light trail lt of LT_i is transformed into an edge on WG_i'. There are four different cases examined here as shall be discussed in the next section of this chapter looking at the light trail test cases.

3.1.1.1. Light Trail Test Cases

The four cases examined in (27) for finding a light trail for the connection request on a wavelength plane WG_i include the following:

(i) **Case 1**: Both nodes s and t are on the light trail lt, but t is an upstream node of s. Due to this being the nature of the light trail, it will be ignored, because it cannot be reused.

(ii) **Case 2**: Neither node s nor node t is on light trail lt. This is shown in lines 8-11 of algorithm 1 in *Figure A1* (**Appendix A**), and is illustrated in *Figure 5* and *Figure 6* below. Suppose the convener node of light trail lt is **c** and the end node is **e**. A short cut edge (**c, e**) which is assigned a small cost **ε** and an edge length (being the number of hops of the light trail lt) is therefore added in the auxiliary graph WG_i' (Remember that free edges in WG_i free edges marked by the black arrowed links in are copied into WG_i' and each light trail lt of LT_i is transformed into an edge on WG_i'). Since neither s nor t are on the light trail, the light trail (**c, 1, 2, 3, 4, e**) marked by the dashed arrow links, is shrunk to a new edge (**c, e**) in WG_i', while all the nodes and free edges remain intact.

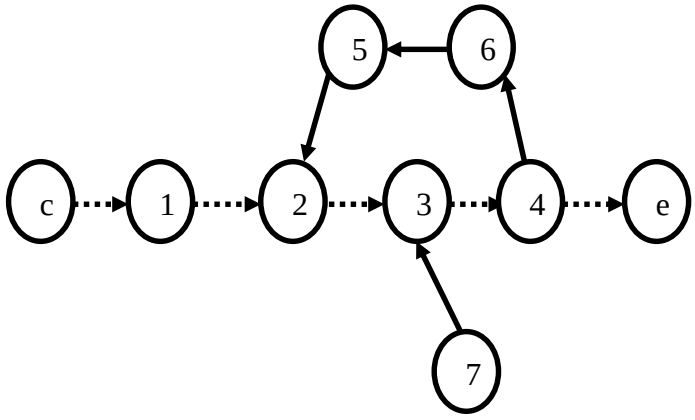

Figure 5. Wavelength Plane Transformation for Case 2 - Original Wavelength Plane WG_i (27).

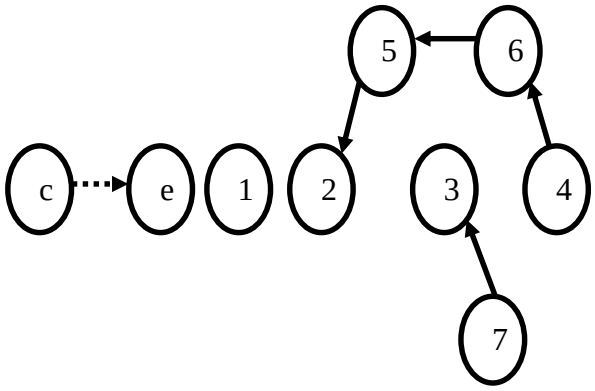

Figure 6. Wavelength Plane Transformation for Case 2 - WG_i' (27).

(iii) **Case 3**: Only node **s** is on light trail **lt**. This is shown in lines 12-15 of algorithm 1 in *Figure A1* **(Appendix A)**, and is illustrated in *Figure 7* and *Figure 8* below. In this case if **s** is the end node of the light trail, then the light trail will be ignored, because it can't be reused. Otherwise, add a shortcut edge **(s, e)**, that has a cost of **ε** and an edge length (being the number of hops of the light trail **lt**) onto WG_i', and if the light trail is reused for the new connection, the end node **e** is expanded to the destination node, ignoring the part from the convener node **c** to **s** on the light trail **lt**. The original lt in WG_i is (**c, 1, s, 3, 4, e**), and if the new edge **(s, e)** on WG_i' is used for the request, then it will be extended to the new end node **t**.

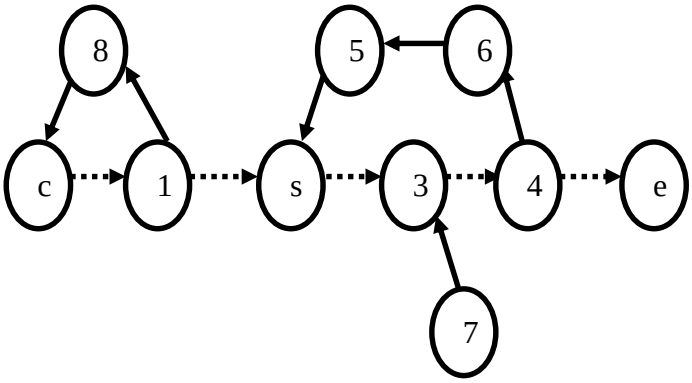

Figure 7. Wavelength Plane Transformation for Case 3 - Original Wavelength Plane WG_i (27).

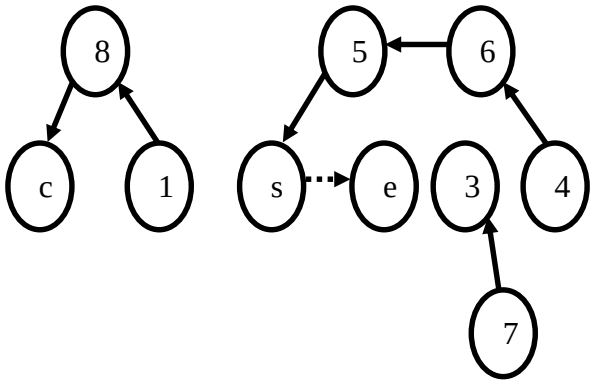

Figure 8. Wavelength Plane Transformation for Case 3 - WG_i' (27).

(iv) **Case 4**: Only node **t** is on light trail **lt**. This is shown in lines 16-19 of algorithm 1 in *Figure A1* **(Appendix A)**, and is illustrated in *Figure 9* and *Figure 10* below. In this case if **t** is the convener node **c** of the light trail lt, then the light trail will be ignored, because it can't be reused for the new request. Otherwise, replace the light trail with a shortcut edge (**c, t**), that has a cost of **ε** and an edge length (being the number of hops of the light trail **lt**) onto WG_i'.

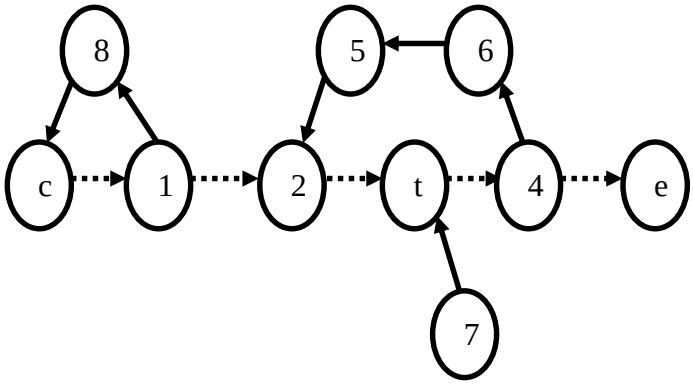

Figure 9. Wavelength Plane Transformation for Case 4 - Original Wavelength Plane WG_i (27).

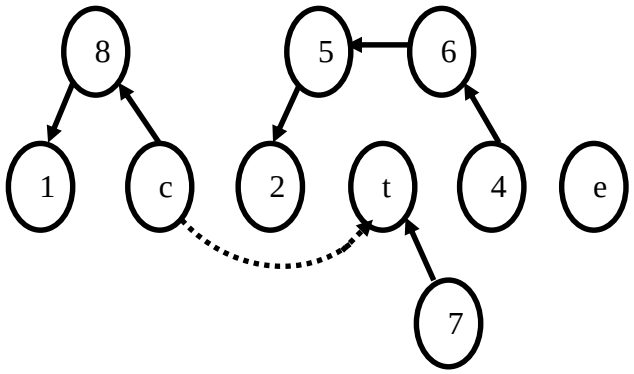

Figure 10. Wavelength Plane Transformation for Case 4 - WG_i' (27).

As shown in lines 21-24 of algorithm 1 in *Figure A1* **(Appendix A)**, after all existing light trails are replaced, then all the free edges of WG_i are copied to WG_i', and to each edge a length = 1 is assigned as well as a cost M, where M >>$\boldsymbol{\varepsilon}$, M ≥ $\boldsymbol{\varepsilon.m_i}$ ($\mathbf{m_i}$ is the number of edges in WG_i'). This ensures that existing light trails will be reused for new requests before consuming any new wavelengths.

At line 25 of algorithm 1 in *Figure A1* **(Appendix A)**, after WG_i' has been constructed for each wavelength plane WG_i, then a light trail for the new connection request on the WG_i' needs to be found. If this light trail is found, it is placed into line 27 of algorithm 1 in *Figure A1* (**Appendix A**), in the set CandidateLightTrail. There are at least W wavelength planes, implying that there are at most W feasible connections in set CandidateLightTrail. According to line 31 of algorithm 1 in *Figure A1* (**Appendix A**), on the wavelength plane, the light trail that is chosen for the connection request is the one that uses the minimum number of wavelengths.

3.1.1.2. Searching for a Feasible Light Trail

In finding this light trail, a path **p** with multiple constraints on WG_i', is found with the path being no more than 5 in length, and consuming a minimum number of wavelengths. In (27), a polynomial time algorithm was used to obtain the optimal solution that would ensure that a

minimum number of free wavelengths was consumed and that the path length was less than or equal to 5. This solution is displayed in Algorithm 2 in *Figure A2* (**Appendix A**)

3.1.1.3. Updating a Light trail

When a light trail for the connection request **(s, t)** is found, there is need to obtain a corresponding light trail in the original wavelength plane WG_l.This is done as indicated in (27) by replacing each shortcut edge by the segment it represents. Algorithm 3 in *Figure A3* (**Appendix A**) was used to find this light trail.

4. METHOD IMPLEMENTED

This chapter describes the approach used in the construction of dynamic light trails in a WDM optical network with the goal to minimize the resources used in the optical fiber network. The testing of the LT and LP algorithms in Java code was done on the Smart Telecom Network in Uganda. Each incoming connection request was examined. There was need to find a working light trail and also to find a link-disjoint backup light trail for the working light trail in case of any single link failure.

4.1. Materials Used in Design and Implementation

The materials used in the design and implementation of the dynamic light trail routing algorithm in WDM optical networks include the following:-

4.1.1. Programming Language

The system and simulation program for the DLIT algorithm was designed in the Java Programming language using the netbeans 8.1 development kit under the Java Development Kit (JDK) platform that is sitted on a Java run time environment virtual machine that was installed on the computer or machine.

4.1.2. Equipment

The server room comprised of physical servers, implying that the server (a Lenovo ThinkPad server) where the application was deployed was used with and comprised of the following hardware specifications:

1. 100GB RAM.
2. Running windows server 2012 as the core operating system.

3. 1000TB of hard disk space.

4. 16 processor cores 3.1 GHz.

In conjunction with the server other equipment like, fibre optic cables and Internet Service Provider (ISP) servers for internet access were used. The fiber network topology and the server network are displayed in *Figure 11* and *Figure 12* below.

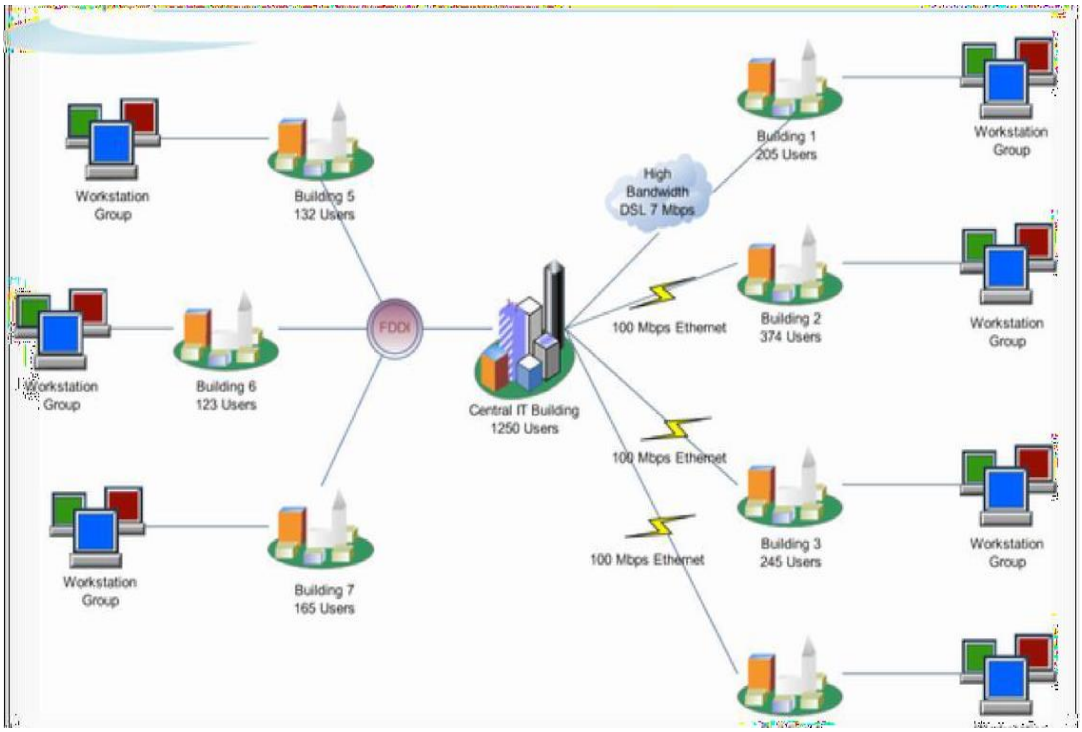

Figure 11. The Server Fiber Topology

Figure 12. The Server Network

4.2. Methods

The need for dynamic light trail routing in WDM optical networks in general, with the goal of minimizing the wastage of resources, necessitated the implementation of the best case algorithm developed as shown in *Figure A1*, *Figure A2*, *Figure A3* and *Figure A4* (**Appendix A)**. To Study the dynamic light trail construction design, the following methods were used.

4.2.1. Observation and Research

The observation was that the resources used, such as wavelengths, wavelength channels, the total system travel cost, and also the time required to deliver the packet were exploiting a lot of the capabilities of each of the nodes in the WDM optical network. This fell in line with the findings of the authors in (27), thus further justifying the need and the importance of this study.

4.2.2. Coding, Testing and Evaluation

Having done the observation and research of the existing ways of how the DLIT routing algorithm could possibly solve the issue of wastage of resources, especially by using an algorithmic design, there was need to implement the algorithmic design by translating the

algorithms 1, 2, 3 and 4 in *Figure A1*, *Figure A2*, *Figure A3* and *Figure A4* (**Appendix A)** to a well-known programming language (in this case the Java programming language was used), and testing it further using higher wavelengths to observe the behaviour of the LT scheme versus the LP scheme on well-known network topologies, namely ARPANET, ItalianNet, and NSFNET and also on some random network topologies. In this section is discussed the way testing and evaluation was achieved.

4.2.2.1. The Java Code

The Java code was done with open source files to enable one to change the simulation parameters later on if desired. The Java translation for the four algorithms is displayed in **Appendix B**. There was need to provide the Java code for the LP heuristic algorithm found in (29), this is displayed in the LightpathAlgorithm class (**Appendix B**), to enable its inclusion in the java code for comparison purposes to the LT algorithm.

The Connection class (**Appendix B**) handles incoming connection requests on each wavelength plane. To create the nodes and the edges as seen in Algorithm 1 of *Figure A1* (**Appendix A**), the Node class and the Edge class (**Appendix B**) were designed. The Graph class **(Appendix B**) is responsible for handling the connections between the nodes using edges for each wavelength plane and so it works with the: Connection class; Node class and; WG class (All found in **Appendix B**).

The DLIT class (**Appendix B**) works with the Connection, Graph and LightTrail classes (All in **Appendix B**) the dynamic light trail routing algorithm is presented in this class, just like in Algorithm 1 of *Figure A1* (**Appendix A**). For each incoming connection request, it handles: finding a feasible light trail for a new connection request on each wavelength plane; constructing

an auxiliary graph for each wavelength plane to update the light trail and; finding working light trail candidates and backup light trails.

The Dijkstra class (**Appendix B**) which requires finding the shortest path between two nodes is an independent algorithm that was written to find a feasible light trail on a wavelength plane or to handle trail finding as displayed in Algorithm 2 of *Figure A2* (**Appendix A**).

The LightWalk class (**Appendix B**) handles updating light trails as seen in Algorithm 3 of *Figure A3* (**Appendix A**).

An additional algorithm was specifically created as seen in the WG class (**Appendix B**), because Algorithm 4 in *Figure A4* (**Appendix A**) was not as straightforward. The step in line 7 of Algorithm 4 in *Figure A4* which requires removing a light trail from current network to find a backup trail is actually lengthy and tricky, because when there are a set of edges to be removed, one does not simply remove them from the graph, but one must also: parse all existing light trails; see if a light trail has edges in common with the edges to be removed, then; "divide" this light trail into subtrails so that the edges in common are removed and the edges not in common remain as shorter light trails. The LightTrail class (**Appendix B**) is vital in this case as well, since it creates light trails and subtrails at different wavelengths.

There was need to code implementations of the well-known network topologies, NSFNET, ARPANET, ItalianNet, and these are displayed in the NsfnetGraph class, ArpanetGraph class, ItalianetGraph class and the RandomGraph class respectively (**Appendix B**). The simulation class (**Appendix B**) is responsible for ensuring that LT and LP simulations are produced for all the network topologies being examined with or without protection, including simulations for the random topologies.

4.2.2.2. Testing

Having developed the Java algorithms for LT and LP, they were tested by deploying them on a server at Smart Telecom Uganda, to ascertain whether the LT algorithm satisfied the objectives it was designed for. The system was deployed in one of the virtual servers on the network for live testing. It was executed to: detect a fibre network connection signal and; to derive various algorithmic graphs which were displayed on the Graphical User Input (GUI) interface.

Before the program was deployed on the server, the Java framework was installed, on which the program would run. During the deployment, the program was first placed in the shared drive from where it would be accessed whenever there was need to do physical deployments on the server.

To be able to see how the system would work there was need to disconnect the fiber optic cable from the server, then start the program, after this the fiber optic cable would be reconnected then the behaviour of the network would be noted via the graphical display.

4.2.2.3. Evaluation

The developed Java algorithms for LT and LP were evaluated by observing the graphical outputs via the GUI.

When the fiber cable was connected on the test server, it was noted that for each incoming connection request, the light trail algorithm helped to generate a simulation on the GUI, but it was also noted that for the disjointed links per interval, there was an inconsistent behaviour in the graphs as the packets kept on increasing.

When a single link failure happened as was experienced, or the loss of light packets as a result of request time outs on the network, the traffic switched from the previously working light trail to the backup light trail quickly, so that there was less information lost. However, there was

also inconsistence in the graphs due to the drops in the packets and in cases when there was no detection of packets, the graphs were not displayed at all. The results of the test are further discussed in the chapter 5.

4.2.2.4. Analysis of Algorithms

Algorithmic analysis is necessary to gauge the efficiency and scalability of the specific algorithm for solving a given problem. The time required to solve a problem or any measure of complexity or the space required is computed as a function of the size of the instance. The complexity theory is therefore interested in how algorithms scale with an increase in the input size.

There is need to prove both upper and lower bounds on the least amount of time needed by the algorithm solving a given problem in order to classify the computation time. These bounds are normally stated using the big O notation. With respect to the time complexity of a problem, in order to provide an upper bound, one needs to show only that a particular algorithm has a specific running time at most. On the other hand, proving lower bounds is a lot more complicated, because lower bounds make a statement about all possible algorithms that solve a given problem, and that includes not only just the algorithms currently known, but any other future algorithms written. To provide a lower bound of time for solving a problem requires showing that no algorithm can have time complexity lower than that specific time.

The Java algorithm displayed in the DLIT class (**Appendix B**) of the java algorithm can obtain an optimal solution in terms of consuming a minimum number of free wavelength links for an incoming connection request (s, t) on a given directed WDM network G (V, E, λ) with W wavelengths on each link, y nodes and x edges.

In the DLIT class of the java algorithm, an auxiliary graph WG_i' is first constructed for each wave length plane WG_i and it takes *O(x)* time to do so, this is due to the fact that Light trails

do not share wavelength links, implying that each edge will be either free or on just a single light trail in the wavelength plane WG_i.

To satisfy the connection request (s, t), the Dijkstra class (**Appendix B**) then finds a path whose length is no more than L_{max}, on the auxiliary wavelength plane WG_i'. In Algorithm 2 of *Figure A2* **(Appendix A)**, the auxiliary wavelength plane $WG_i'^{Lmax}$ has $y \cdot (L_{max} + 1)$ nodes and $O(2 \cdot x \cdot L_{max} + y \cdot L_{max})$ edges (27) where y are nodes in G and x are edges in G. An implementation of the priority queue gives a run time complexity $O(V^2)$, where V is the number of vertices, implying a run time complexity of $O((y \cdot (L_{max} + 1))^2)$. Implementing the priority queue with a Fibonacci heap makes the time complexity $O(E + V \log V)$, where E is the number of edges, implying a time complexity of $O((2 \cdot x \cdot L_{max} + y \cdot L_{max}) + (y \cdot (L_{max} + 1) \log (y \cdot (L_{max} + 1)) = O((x + y) L_{max})$ time to find the shortest path in the auxiliary wavelength plane $WG_i'^{Lmax}$. To obtain a path on each auxiliary wavelength plane would therefore take $O(x + (x + y) L_{max})$ time and to obtain at most W candidate paths would take $O(W \cdot (x + (x + y) L_{max}))$ time.

In the DLIT class of the java algorithm, an optimal light path is obtained in $O(W)$ time, and because the edges on the path are L_{max} at most, then this path is converted to the corresponding LightWalk minLT on WG_i in $O(L_{max})$ time. This implies that the total time for this is $O(W + L_{max})$. After adding up all the times, the time complexity in total is bounded by $O((W + L_{max}) + W \cdot (x + (x + y) L_{max})) = O(x + y)$.

4.3. Graphical User Interface

Presented in *Figure 13* below, is the screen shot of the graphical user interface (GUI) obtained from the source code (when the LightTrailAlgorithmform.java is run). This GUI is used to observe the graphical simulations of the comparisons between the LT and LP schemes, for the

three well-known topologies, NSFNET, ARPANET and ItalianNet, when one is not connected to a physical network. The GUI contains two sections, one showing the simulations with protection (Survivable provisioning consideration) and the other showing the simulation without protection. The buttons are named according to their functions and once one clicks on an individual button, for example, on the button to view the connection graph on the section with either protection or without, at a given wavelength, the graph is automatically generated. Note that, for this GUI the number of wave lengths running from W = 4 to W = 256 have to be provided in order for the system to work. It is also limited to 900 connections.

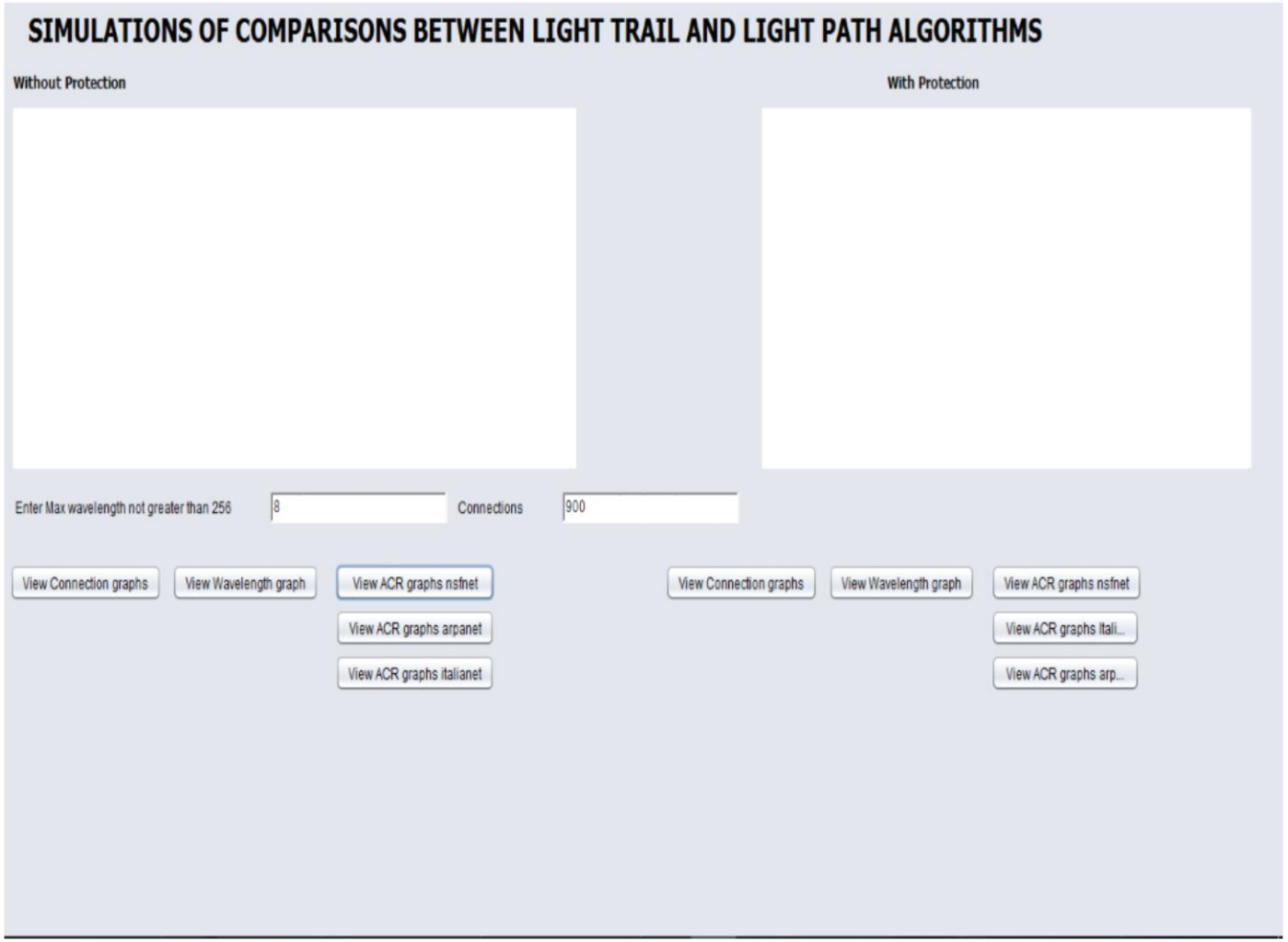

*Figure 13***.** LT versus LP Simulation Graphical User Interface

5. RESULTS

This chapter examines the performance of the DLIT routing algorithm using its Java implementation in the DLIT construction. The testing of the Java algorithms was first carried out on a 2.00 GHz Windows PC with 2GB of memory. However the live testing of the algorithm was done on a 16 processor cores 3.1GHz Lenovo ThinkPad server with 100GB RAM and 1,000TB of hard disk space. The results provided are from live testing of the algorithm.

The performance and numerical results of the light trail present an average of 10 runs and are represented in the figures displayed in this chapter by LT, and those of the light path scheme are represented in the figures also displayed in this chapter by LP. This performance is compared and analysed using: some eight randomly generated topologies with 20 nodes and 40 nodes and; some well-known Internet topologies such as ItalianNet [33 nodes, 67 edges], NSFNET [14 nodes, 21 edges], and ARPANET [20 nodes, 32 edges]. In addition to this, 900 connections are randomly generated for each network topology and the cases considered are those in which the number of wavelengths on each link is 4, 8, 16, 32, 64, 128 and 256 respectively.

The connection requests which are uniformly randomly distributed in [1, 20] are generated at each random source and destination node, and have a life time set to a random integer that is also uniformly distributed in [1, 100]. The evaluation of these numerical results is done by comparing the LT and LP schemes in the following ways:

(i) Without the consideration of protection

(ii) With protection or the survivable connection provisioning considered.

In each of these two cases, the matrices used for evaluating the performance of the LT versus the LP scheme for ItalianNet, NSFNET, ARPANET and some random networks versus the number of wavelengths on each link (W) include the:

(a) Comparison of accepted connections;

(b) Comparison of consumed free wavelength links and;

(c) Examination of the Accepted Connections Ratios (ACR)

5.1. Comparison between LT and LP without Protection

The LT and LP schemes are first compared without the consideration of protection. On each network topology studied, seven cases are examined, and these are cases where by on each link we have, 4 wavelengths (W = 4), 8 wavelengths (W = 8), 16 wavelengths (W = 16), 32 wavelengths (W = 32), 64 wavelengths (W = 64), 128 wavelengths (W = 128) and 256 wavelengths (W=256) respectively. The results are illustrated in the graphs below and discussed in the proceeding sections of this chapter.

5.1.1. Comparison of Accepted Connections without Protection

From examining *Figure 14* below, in the ItalianNet network, with 128 wavelengths on each link, LT had 33.3% more connections in comparison to LP. Within all the networks, with the exception of when the wavelengths on each link are at W = 64 and W = 256, increasing the number of wavelengths per link, increases the number of accepted connections. For example, in the ItalianNet network, the use of the LT scheme with 128 wavelengths on each link results in 350 more connections being accepted than when 4 wavelengths are used on each link on the same network.

5.1.2. Comparison of Consumed Free Wavelength Links without Protection

In *Figure 15* below, for LT and LP, the number of free wavelength links consumed for all accepted connections are compared on the three well-known topologies. It is observed, that in NSFNET, the wavelength consumption of the LT scheme is similar to that of LP in the case of W

= 32 and W = 64 on each link, but for the other cases LT always consumed at least 33.3% less number of free wavelength links than LP did. For ARPANET, with the exception of the wavelength link consumption on each link at W = 32 and W = 128, LT consumed similar or less number of free wavelength links than LP did. While for ItalianNet with the exception of W = 8, W = 16 and W = 128, LT consumed similar or less number of free wavelength links than LP did.

From both *Figure 14* and *Figure 15* below, LT accommodates more network connections with less number of free wavelength links. Also the bigger the size of the network, the more the number of free wavelength links get consumed, but the number of connections are reduced. This is because the bigger the network, the length of the average connection is longer and therefore uses more wavelength links for a single connection.

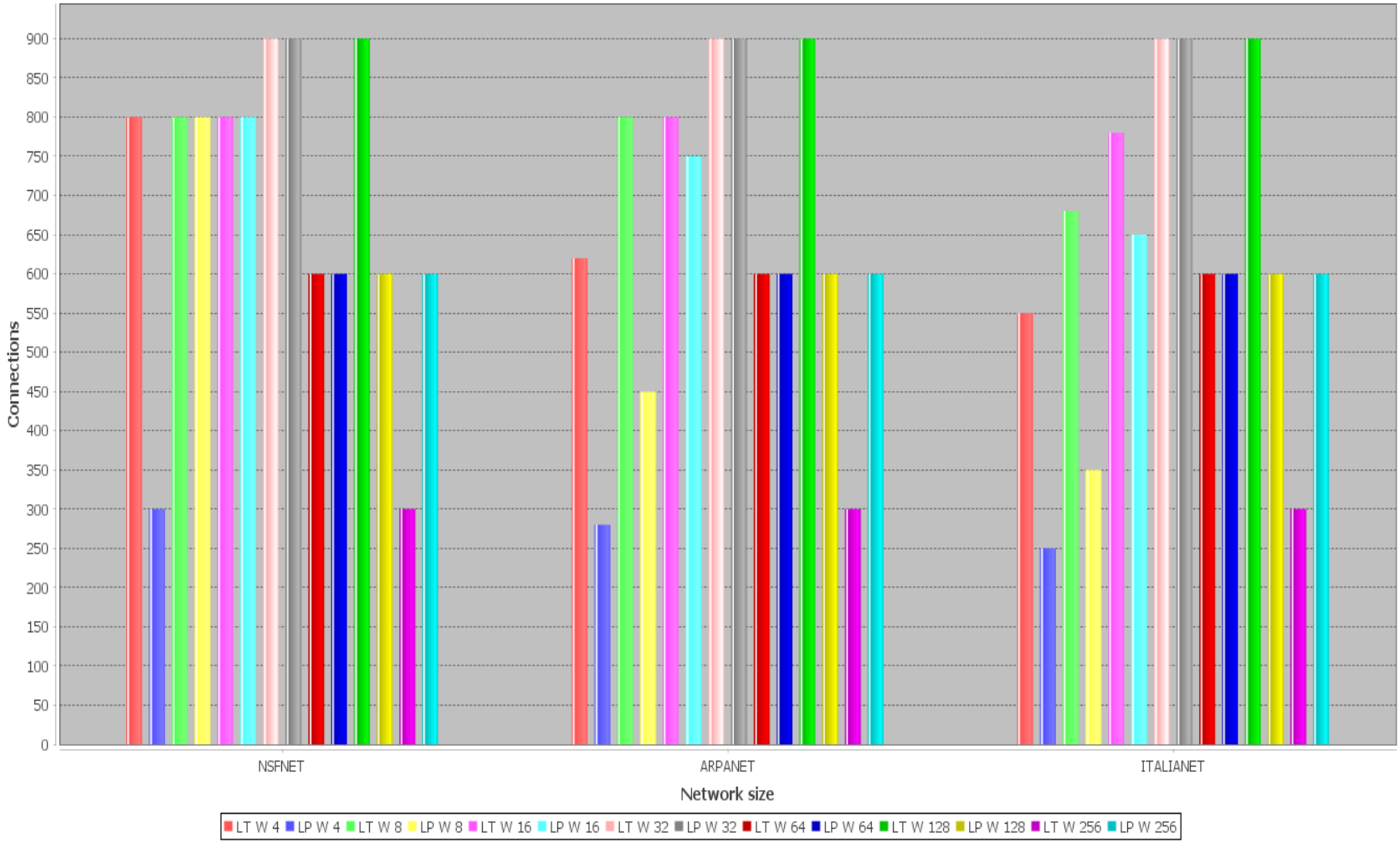

Figure 14. Graph Showing Comparison of Accepted Connections between LT and LP without Protection

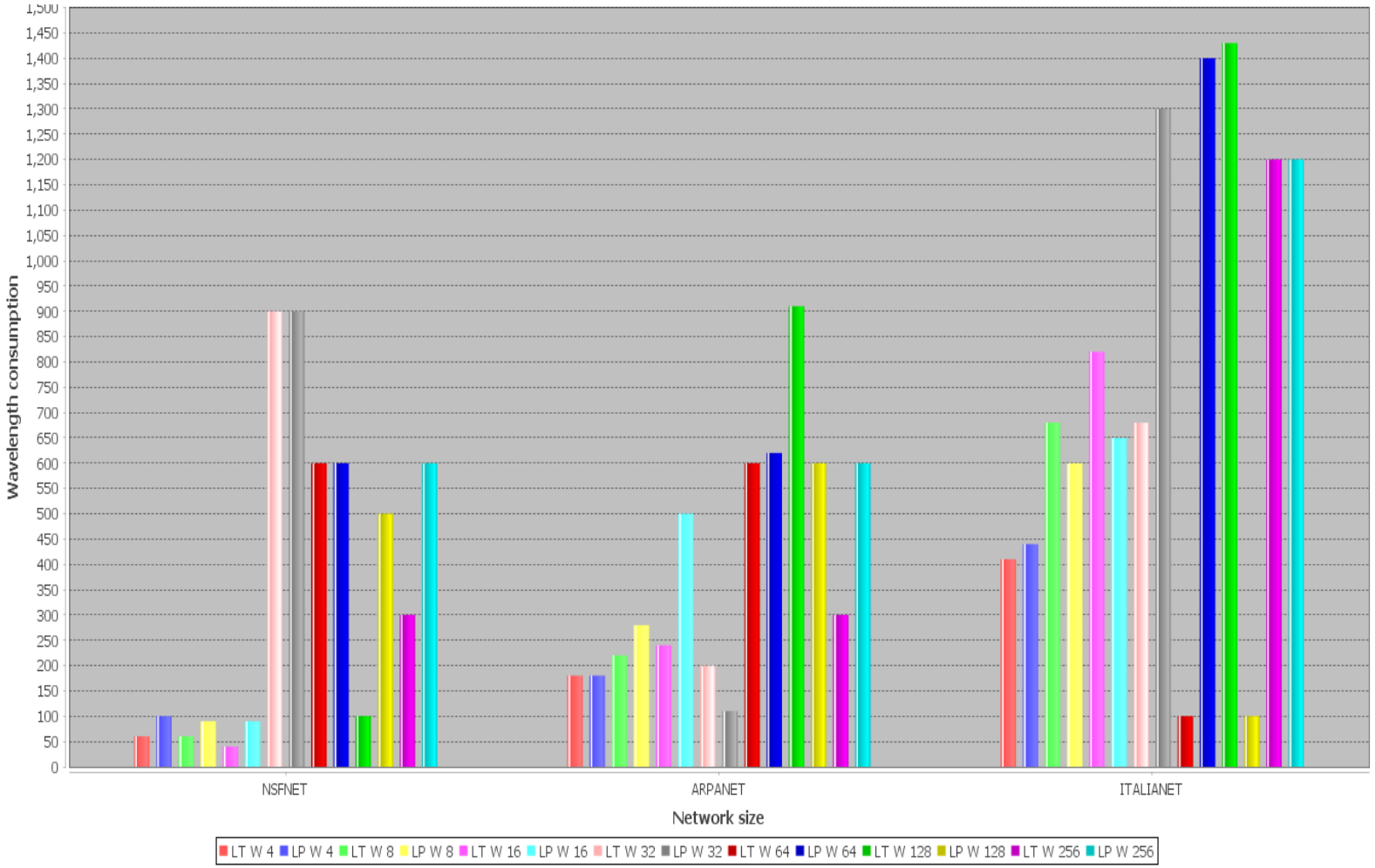

Figure 15. Graph Showing Comparison of Consumed Free Wavelength Links between LT and LP without Protection

5.1.3. Accepted Connections Ratio (ACR) versus the Number of Wavelength Links (W) without Protection

In *Figure 16*, *Figure 17* and *Figure 18* below, we see the impact of the number of wavelengths on each link on the total accepted connections for each of the three well-known topologies. This is provided by the Accepted Connections Ratio (ACR) which is obtained by dividing the total number of accepted connections by the 900 randomly generated connections. As observed from the ACR graphs in *Figure 17* and *Figure 18* below, for both LT and LP the increase in the wavelengths on each link, resulted in the increase in the number of connections accepted. In *Figure 16* below, examining LT, shows that as the wavelength on each link increased, the connections accepted remained high, but constant, while in the case of LP, an increase in the number of wavelengths on each link, for example at W = 32, reduced the number of connections accepted by 20% out of the 900 connections at W = 8 on the same network. This implies that, the number of wavelengths (W) on each link, has more impact on LP than on LT. This is further displayed in *Figure 17* below, where at W = 8, LP accepted 60% out of 900 generated connections and at a higher wavelength of W = 16 it accepted 90% out of the 900 connections.

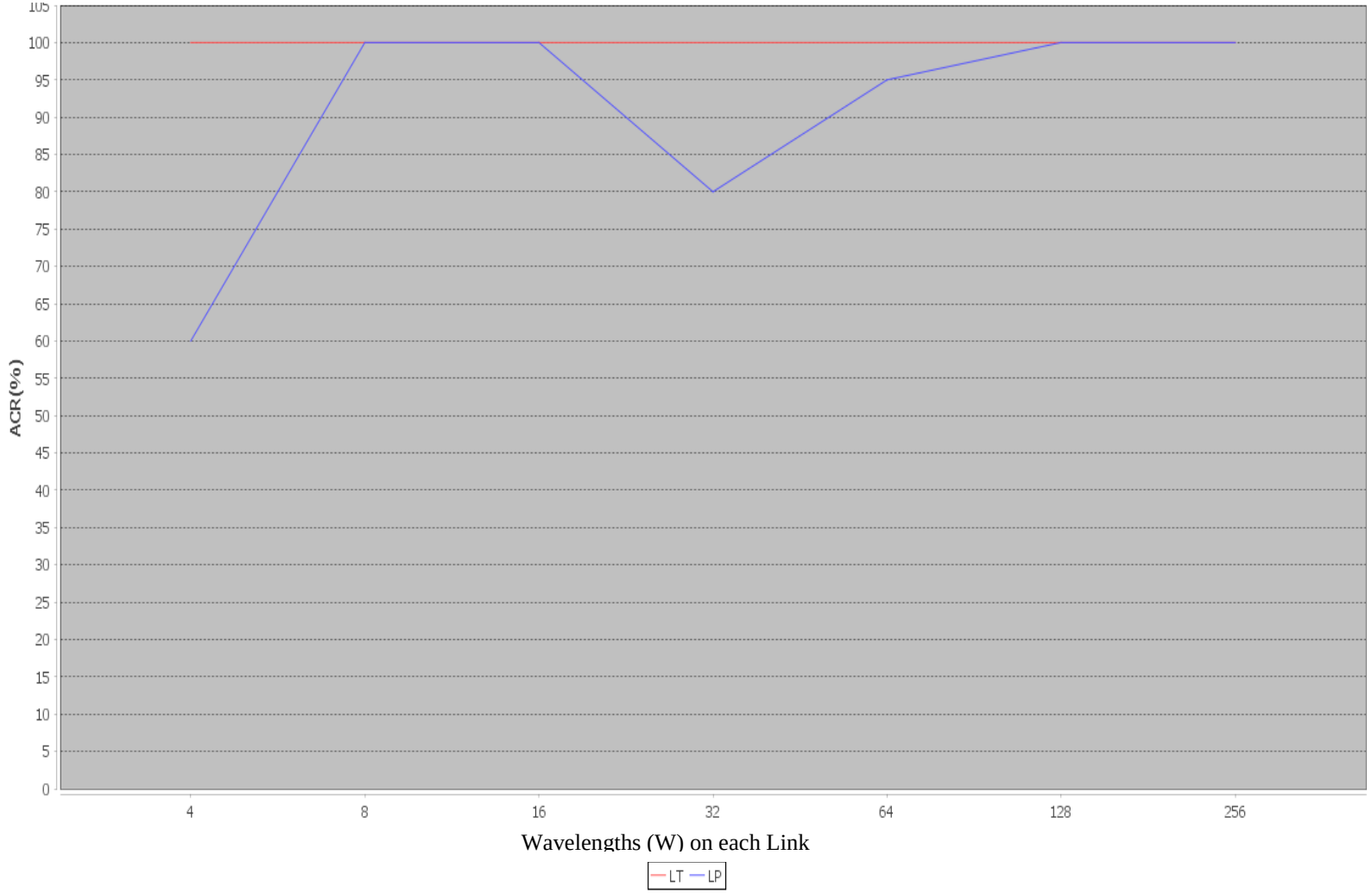

Figure 16. Accepted Connections Ratio (ACR) versus the Wavelengths per Link on NSFNET

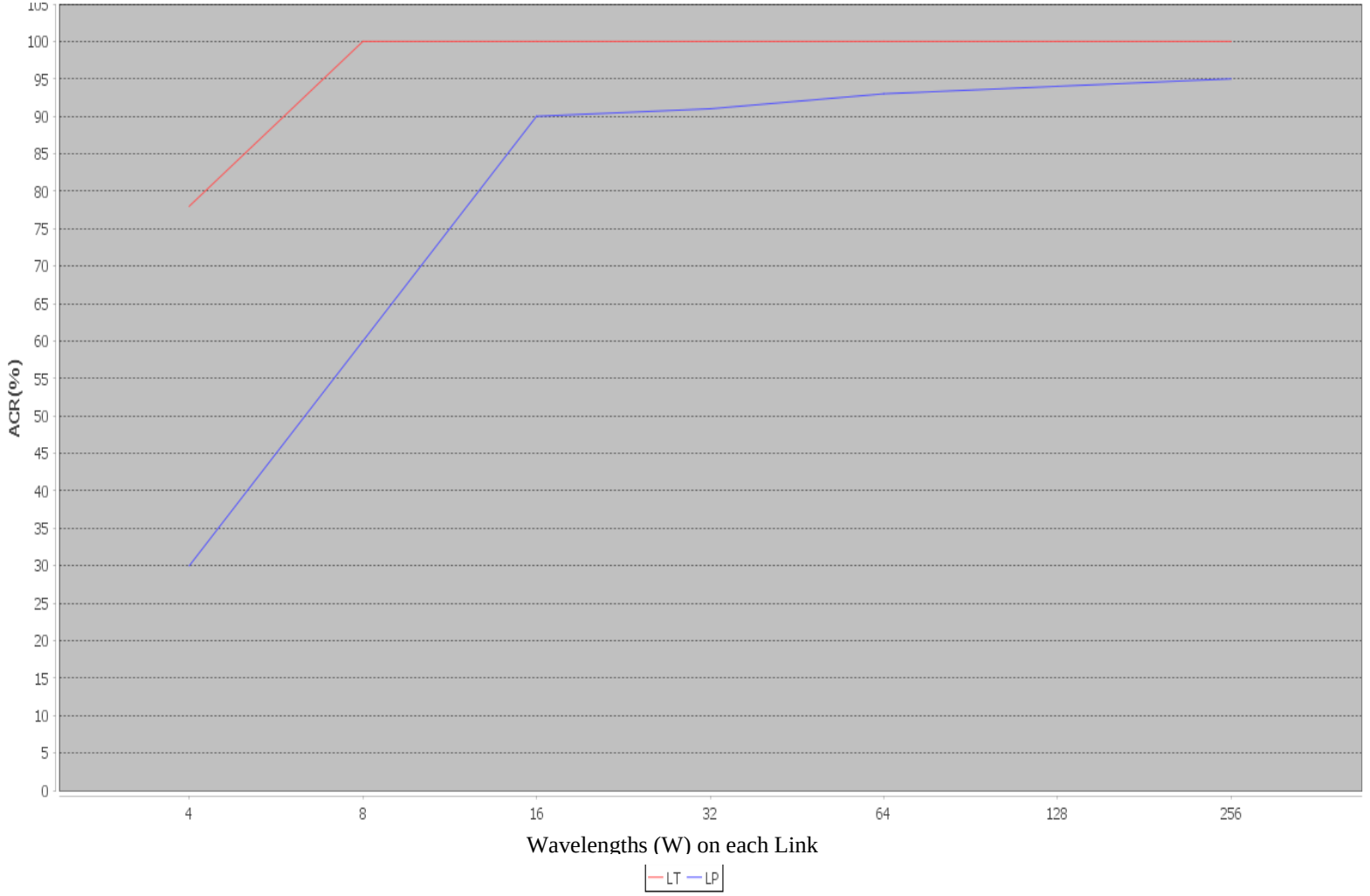

Figure 17. Accepted Connections Ratio (ACR) versus the Wavelengths per Link on ARPANET

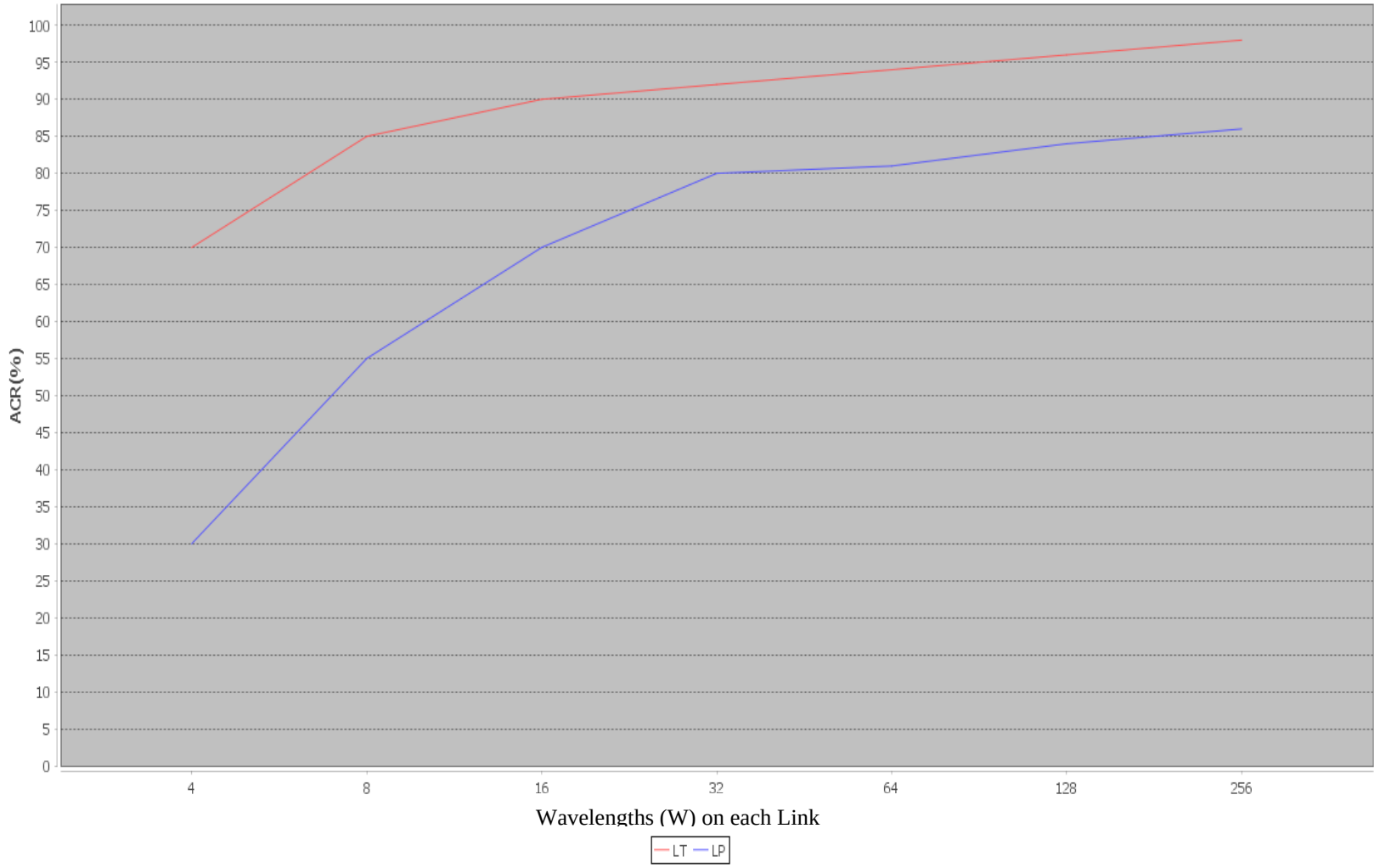

*Figure 18***.** Accepted Connections Ratio (ACR) versus the Wavelengths per Link on ItalianNET

5.2. Comparison between LT and LP with Protection

In this case the survivable connection provisioning for LT is studied, that is, a pair of link-disjoint paths have to be found, and these two paths are: a working path and; a protection path in case of link failure. This is done for both LT and LP as is displayed in *Figure 19* and *Figure 20* below. The shortest working light path for the connection with the minimum free wavelength usage is first computed, followed by the shortest backup path.

5.2.1. Comparison of Accepted Connections with Protection

From *Figure 19* below, more network connections were provided by LT in most cases, in comparison to LP, implying that LT performed better than LP. For all the three well-known topologies, LT still performed better than LP, except at the highest wavelength of 256 on each link, where the LP scheme had more connections than the LT scheme

5.2.2. Comparison of Consumed Free Wavelength Links with Protection

Figure 20 shows that for both ARPANET and ItalianNet, LT performed better than LP in 3 cases out of 7 by consuming less number of wavelength links and performed the same as LP in one case, however LT did not perform well in 3 other test cases. With the exception of one case, LT performed better than LP at higher wavelengths in all the three well-known topologies. For NSFNET, in 6 out of the 7 test cases, LT worked better than the LP, and performed as well as LP in the seventh case. On studying *Figure 20* in conjunction with *Figure 19*, LT also accommodated more connections with fewer wavelength links, in comparison to LP. Therefore implying that LT is better at efficiently utilizing its resources than LP.

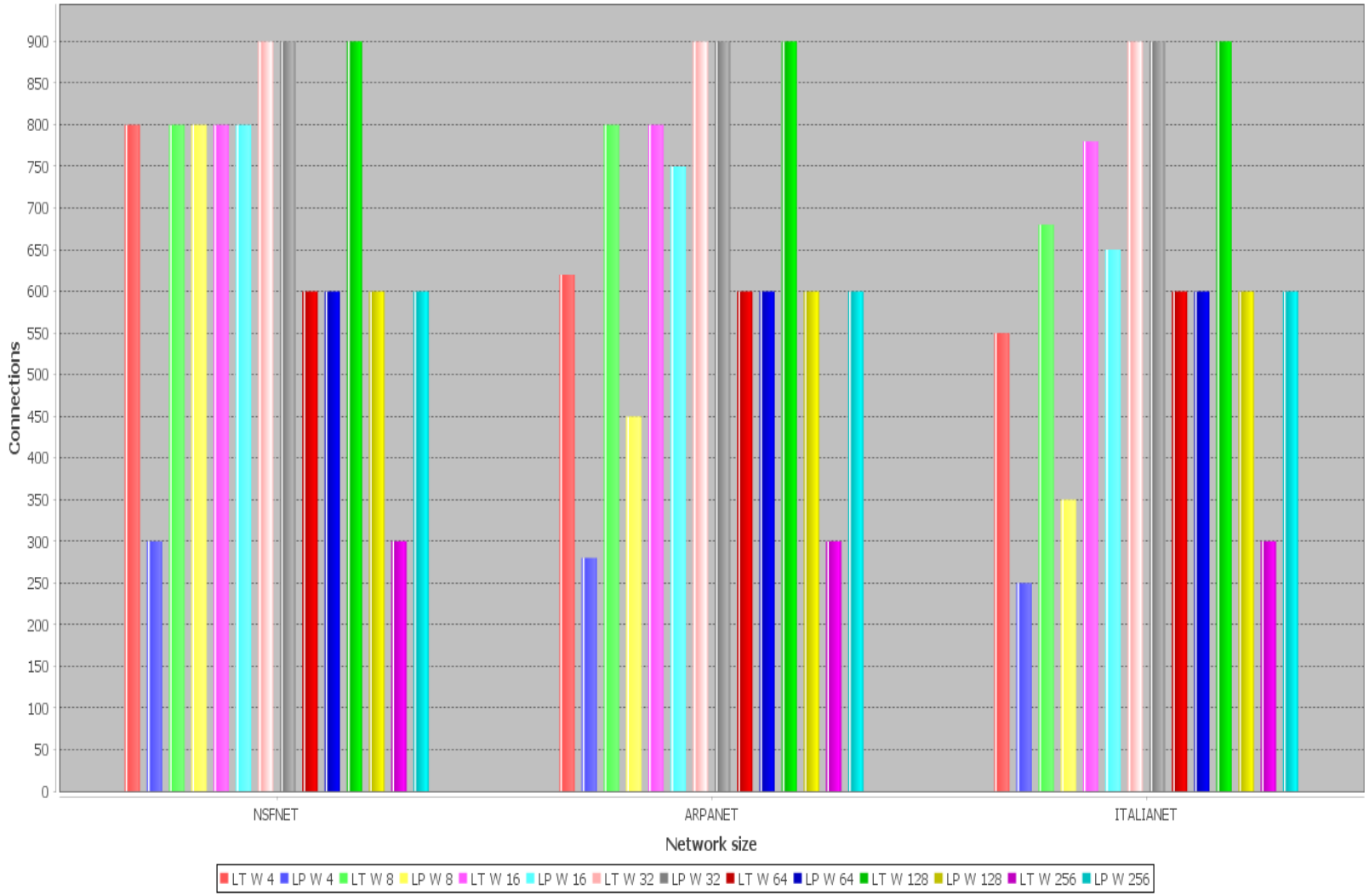

Figure 19. Graph Showing Comparison of Accepted Connections between LT and LP with Protection

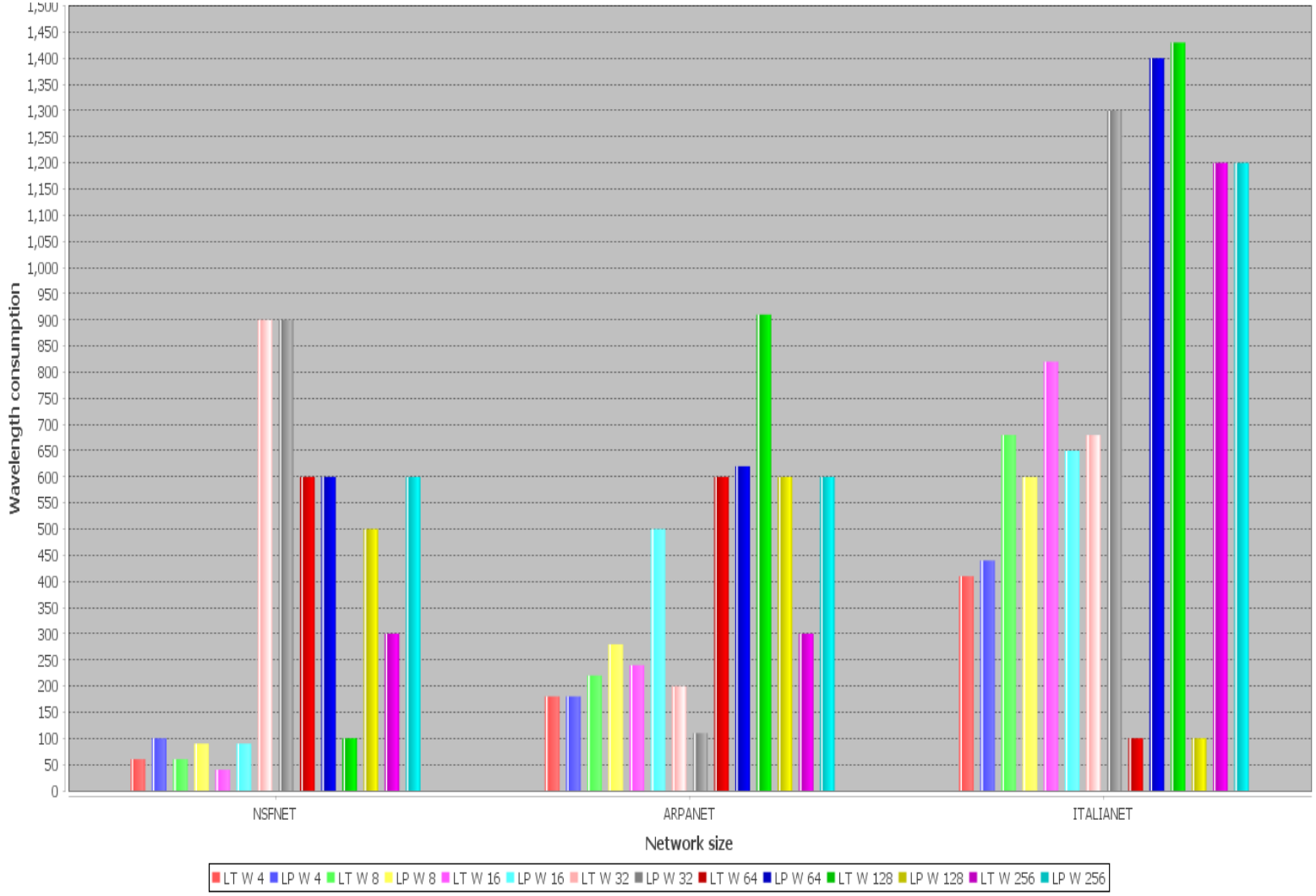

Figure 20. Graph Showing Comparison of Consumed Free Wavelength Links between LT and LP with Protection

5.2.3. Accepted Connections Ratio (ACR) versus the Number of Wavelength Links (W) with Protection

Figure 21, *Figure 22* and *Figure 23* below, show the simulations that determine the impact of the number of wavelength (W) for each link, on the number of accepted connections, by obtaining the Accepted Connections Ratio (ACR) which is computed by dividing the number of accepted connections by the 900 randomly generated connections. Note that from studying all three ACR graphs, LT in comparison to LP, provided more number of connections. Also an increase in the number of wavelength (W) on each link for both LT and LP in NSFNET shown in *Figure 21* and in ARPANET shown in *Figure 22* led to accommodation of more connections. This was also the case for LT in *Figure 23* for ItalianNet, however, there was a decrease in the number of connections accepted by LP out of the 900 generated connections, from 90% at W = 16 to 80% for higher wavelengths.

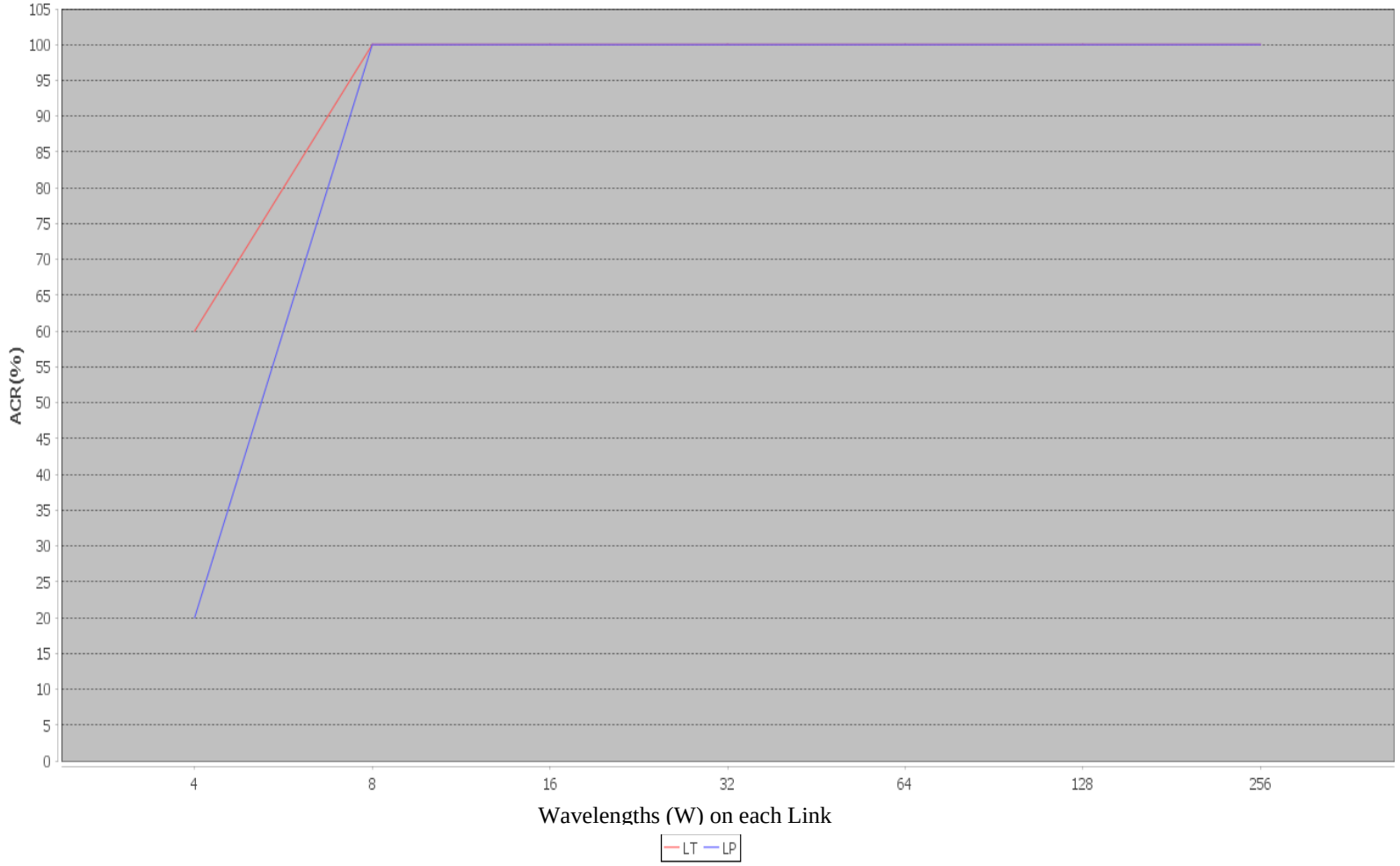

Figure 21. Accepted Connections Ratio (ACR) versus the Wavelengths per Link on NSFNET

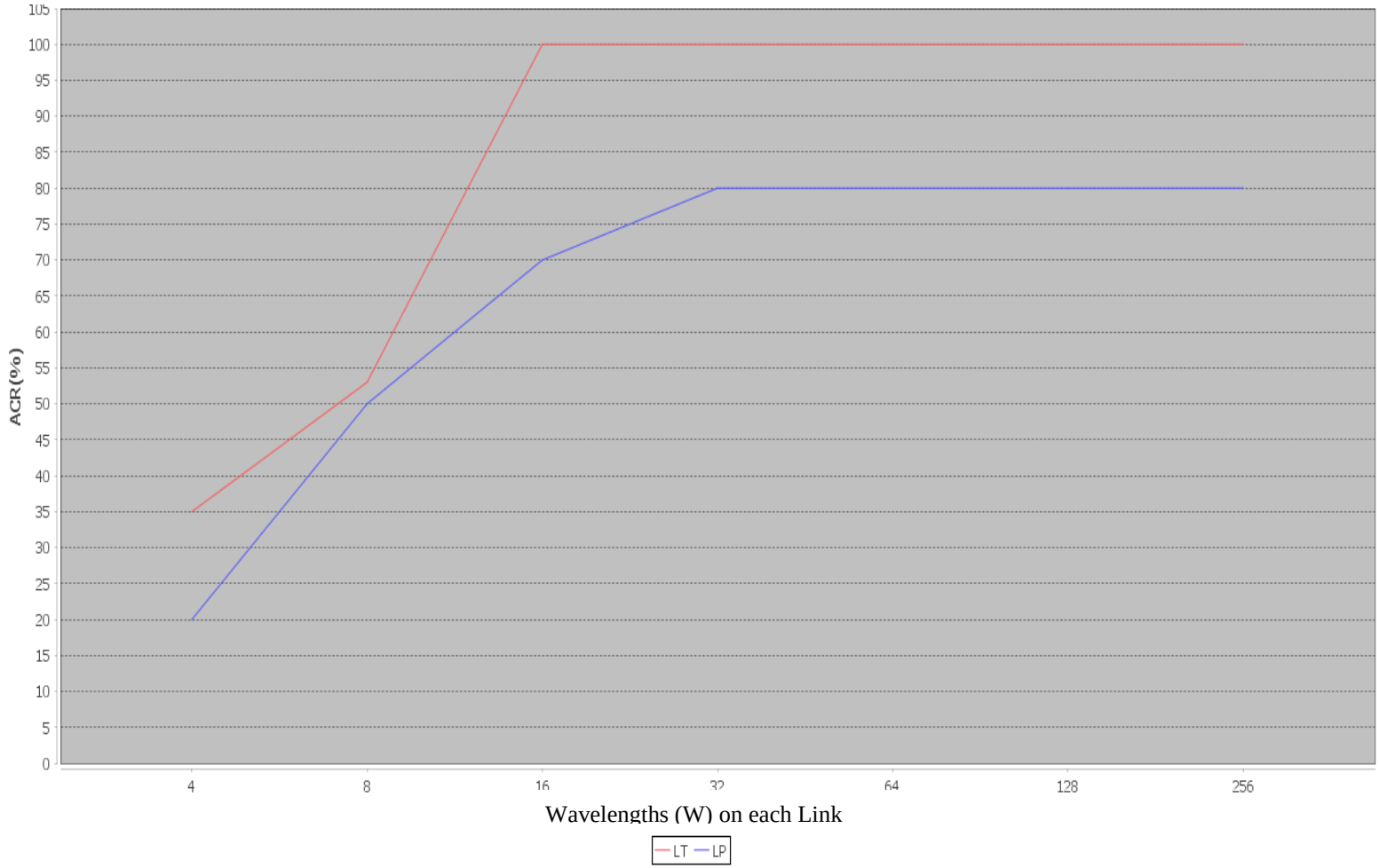

Figure 22. Accepted Connections Ratio (ACR) versus the Wavelengths per Link on ARPANET

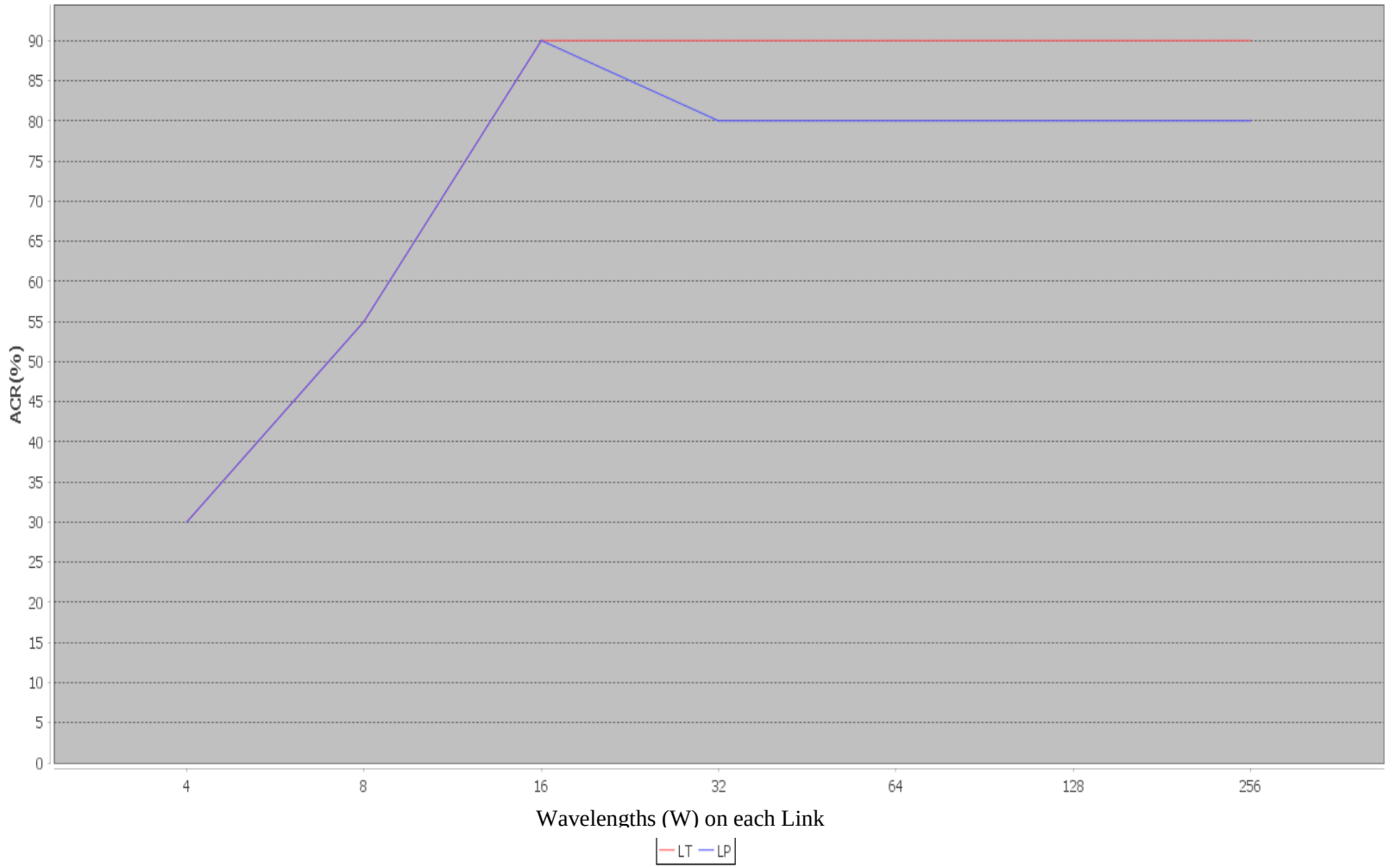

Figure 23. Accepted Connections Ratio (ACR) versus the Wavelengths per Link on ItalianNET

5.3. Comparison between LT and LP on Random Networks

The illustration in *Figure 24* and *Figure 25* below, is for eight different network sizes with their nodes being 20 and 40.

Examining *Figure 24* below in the case of the networks with 40 nodes, LT satisfies as many connections as LP. The same is observed for networks with 20 nodes, however, even more connections are accommodated by LT than by LP at W = 4 on each link.

Figure 25 below shows that LT consumed similar or less wavelengths for each link in comparison to LP, implying that, for the accepted connections, LT is more efficient than LP in utilizing the network resource by using less number of free wavelength links.

The results considered in this chapter, indicate that, in comparison to the LP scheme, the LT scheme performed better and is therefore the favourable option that can accommodate as many or more connections by using less number of wavelength links (W) in a WDM optical network.

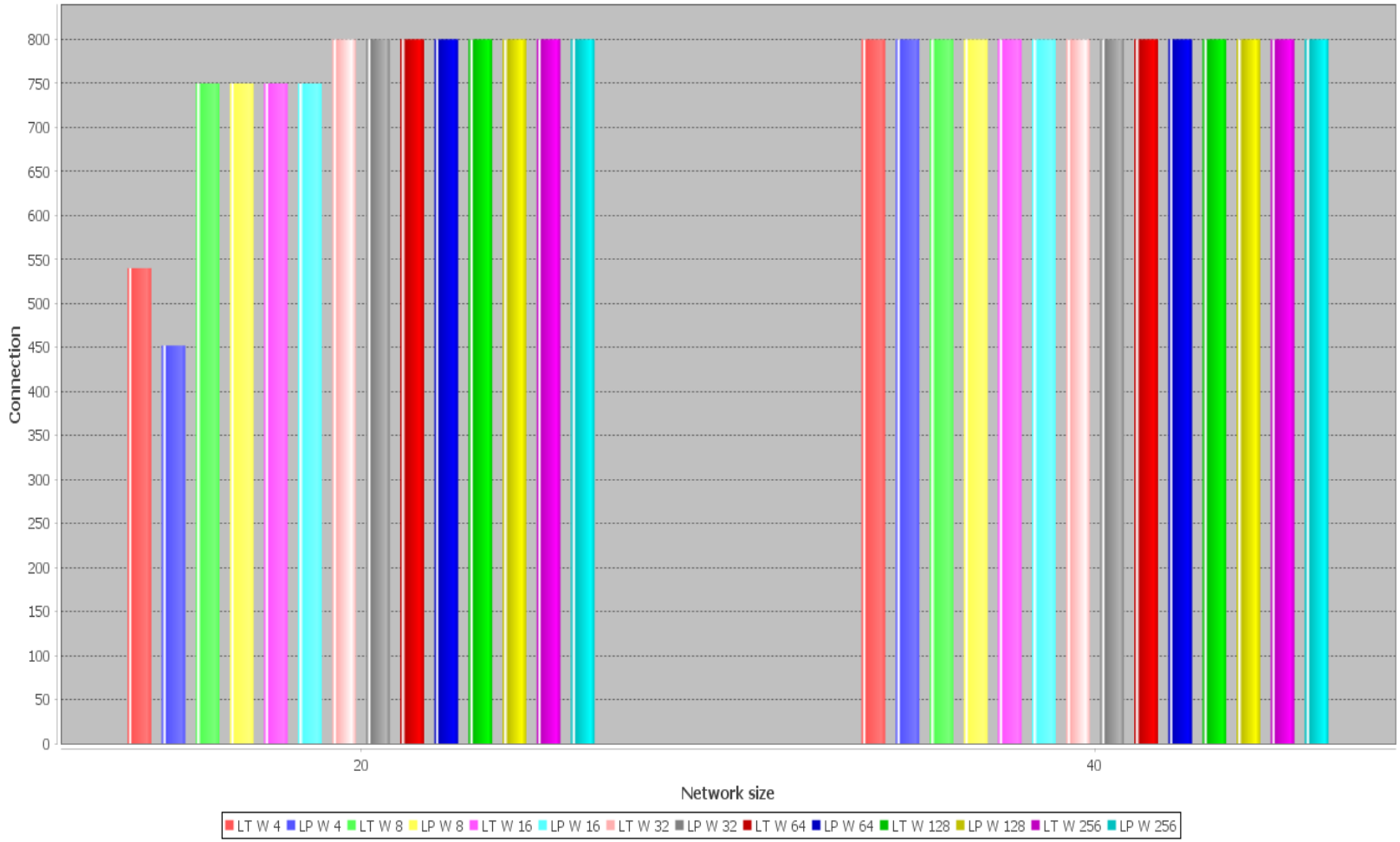

Figure 24. Graph Showing Comparison of Accepted Connections between LT and LP on Random Networks

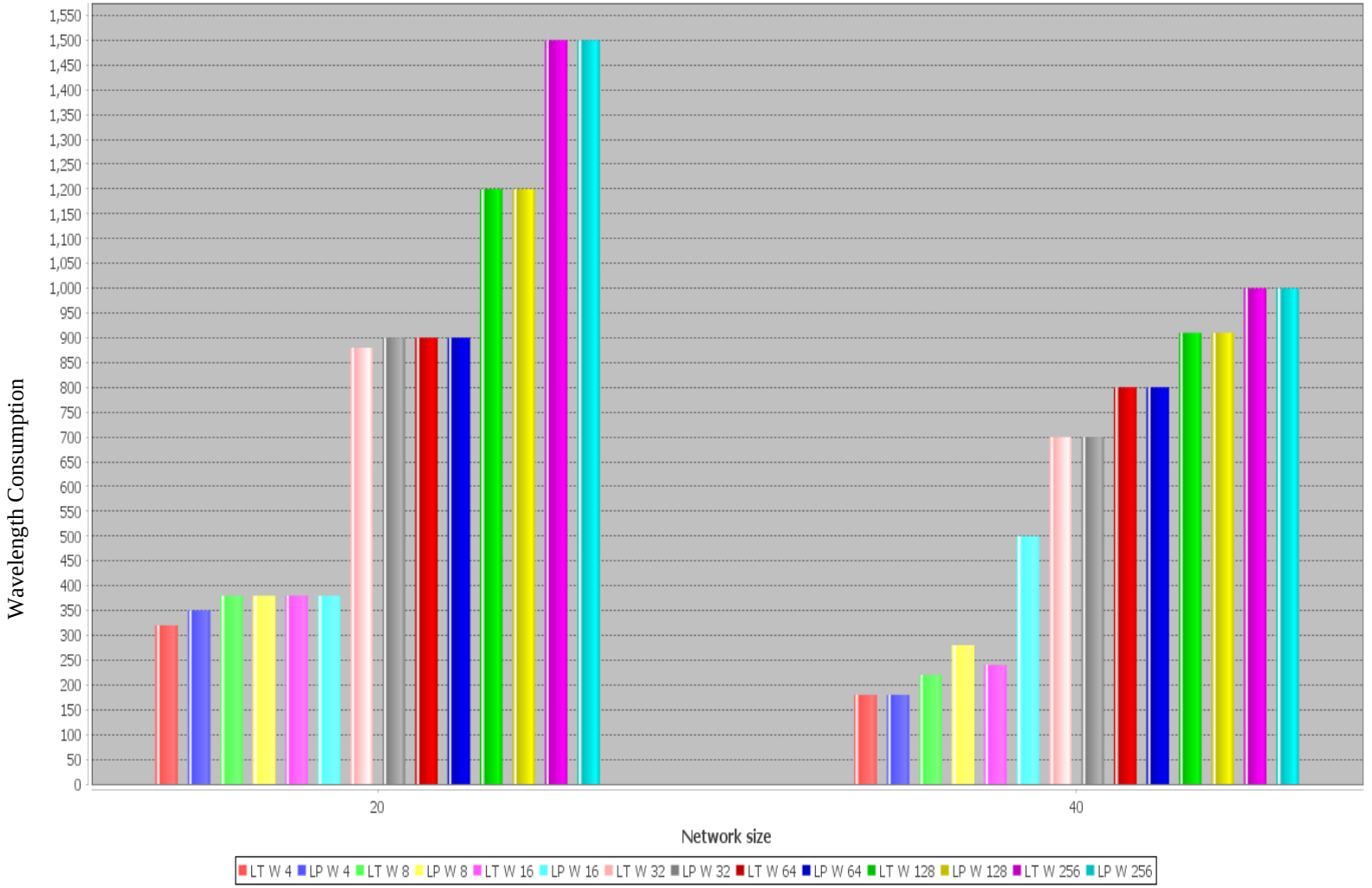

Figure 25. Graph Showing Comparison of Consumed Free Wavelength Links between LT and LP on Random Networks

6. CONCLUSION

In conclusion, this paper provides valuable information about emerging trends in WDM optical networks, which could provide information for optimal selections with respect to achieving efficient service while minimizing the resources using dynamic light trails. This paper studies the construction and constrained optimization of dynamic light-trails in WDM optical networks. Light trails overcome the basic limitation of optical multicasting in optical communication through light paths and therefore represent one of the best methods that enable fast provisioning in optical networks.

This paper presents a routing application program written in the Java programming language that is based on heuristic algorithms, for the dynamic construction of light trails in optical networks and then compares the simulation results of its performance to that of the light path scheme when applied on various well known Internet topologies. It evaluates the light trail scheme versus the light path scheme with and without survivable connections provisioning, using criteria such as: the number of accommodated connections generated; the number of free wavelengths consumed and; the accepted connection ratio (ACR). It then presents this dynamic routing application program as a way forward in achieving efficient dynamic light trail construction.

From the Numerical results it is evident that more benefits can be obtained from the use of an enhanced dynamic light trail construction scheme in comparison to the traditional light path schemes. This underscores the need for further studies that would determine other mechanisms involved in further enhancing and deciphering dynamic light trails of WDM optical networks.These results also point towards the need to continuously re-evaluate the current dynamic light trail technologies which could be further addressed through educational and research

interventions, that are geared towards or target satisfying the current and possible future demands on WDM optical networks.

Furthermore, the results highlight the fact that networks using light path communication and wavelength routing are over-provisioned, leading to expensive network element deployment and them not being able to provide the dynamic guarantees of bandwidth provisioning as required by IP centric communication. Light trails as seen in (11, 13, 14, 17, 18, 27) on the other hand, offer a low cost pragmatic alternative by providing dynamic provisioning as compared to rigid light path communication.

REFERENCES

(1) Ansari, Z., & Tariq , S. (2006). A Heuristic Approach for Dynamically Reducing the Number of Light Trails in Survivable Optical Networks. *IEEE Broadnet'2006*, (pp. 1-8). San Jose, CA.

(2) Ayad, A., Elsayed, K., & Ahmed, S. (2008). Enhanced Optimal and Heuristic Solutions of the Routing Problem in Light Trail Networks. *Springer Photonic Netw. Commun. 15*, (pp. 7-18).

(3) Balasubramanian, S. (2005). Light-trail Networks: Design and Survivability. *IEEE LCN'2005*, (pp. pp 174-181).

(4) Calleghatti, F., Casoni, M., & Raffaelli, C. (1999). Packet Optical Networks for High-speed TCP-IP Backbone. Solutions for Optical Transparent Networking. An Optical Transparent Packet Network. IP over OTP. *IEEE Communication Magazine.*

(5) Chlamtac, I., & Gumaste, A. (2003). Light-Trails: A Solution to IP Centric Communication in the Optical Domain. Berlin, Heidelberg: Springer-Verlag.

(6) Chlamtac, I., Ganz, A., & Karmi, G. (1992). Lightpath Communications: A Novel Approach to High Speed Optical WANs. *IEEE Trans. Commun. 40(7)*, (p. 1152).

(7) Cormen, T., Leiserson , C., Rivest, R., & Stein, C. (2001). Introduction to Algorithms. *2nd Edition.* New York: McGraw Hill.

(8) Fang, J., He, W., & Somani, A. (2004). Optimal Light Trail Design in WDM Optical Networks. *ICC'2004*, (pp. 1699-1703).

(9) Fredrick, M. T., Vanderhorn, N. A., & Somani, A. K. (2004). Light trails: A Sub-wavelength Solution for Optical Networking. *HPSR'2004*, (pp. 175-179).

(10) Ge, A., Callegati, F., & Tamil, L. (2000). On Optical Burst Switching and Self Similar Traffic. *IEEE Commun. Lett. 4(3)*, (pp. 98-100).

(11) Gumaste , A., & Zheng, S. (2005). Next Generation Optical Storage Area Networks: The Light-trails Approach. *IEEE Communications Magazine.*

(12) Gumaste, A., & Chlamtac , I. (2003). Light-trails: A Novel Solution for IP Centric Communications in the Optical Domain. *IEEE Proceedings of HPSR.* Torino, Italy.

(13) Gumaste, A., & Chlamtac , I. (2003). Mesh Implementation of Light-trails: A Solution to IP Centric Communication. *ICCCN'2003*, (pp. 178-183).

(14) Gumaste, A., & Chlamtac, I. (2003). Light-trails: A Novel Conceptual Framework for Conducting Optical Communications. *HPSR'2003*, (pp. 251-256).

(15) Gumaste, A., & Chlamtac, I. (2004). Light-trails: A Solution for Dynamic Optical Communications. *OSA Journal of Optical Networking.*

(16) Gumaste, A., & Palacharla, P. (2007). Heuristic and Optimal Techniques for Light-trail Assignment in Optical Ring WDM Networks. *Computer Communications, 30, Issue 5*, pp. 990-998.

(17) Gumaste, A., Chlamtac, I., & Jue, J. (2004). Light-frames: A Pragmatic Framework for Optical Packet Transport. *ICC'2004*, (pp. 1537-1542).

(18) Gumaste, A., Kuper , G., & Chlamtac, I. (2004). Light-trail Assignment to Optical Mesh Networks. *IEEE 13th LANMAN.* San Francisco.

(19) Gumaste, A., Kuper, G., & Chlamtac, I. (2004). Optimizing Light-trail Assignment to WDM Networks for Dynamic IP Centric Traffic. Local and Metropolitan Area Networks, LANMAN 2004. *13th IEEE International Workshop on Software Technology and Engineering Practice*, (pp. 113-118).

(20) He, W., Fang, J., & Somani, A. (2004). On Survivable Design in Light Trail Optical Networks. *Proceedings of 8th IFIP Working Conference on Optical Network Design and Modeling*, (pp. 155-172).

(21) Modiano, E., & Lin, P. (2001). Traffic Grooming in WDM Networks. *IEEE Commun. Mag. 39*, (pp. 124-129).

(22) Qiao, C. (2001). Labeled Optical Burst Switching: IP over WDM Networks. *IEEE Communication Magazine.*

(23) Ring, A. R. (2001, Oct.). *An Introduction to Resilient Packet Ring Technology.* Retrieved from http://www.rpralliance.org

(24) Semeria, C. (n.d.). Multiprotocol Label Switching: Enhancing Routing in the New Public Network. *Juniper Networks Inc.*

(25) Verma, S., Chaskar, H., & Ravikanth, R. (2000). Optical Burst Switching: A Viable Solution for Terabit IP Backbone. *IEEE Netw. 14*, (pp. 48-53).

(26) Yoo, M., & Qiao, C. (1999). Optical Burst Switching - A New Paradigm for and Optical Internet. *JHSN. 8(1)*, (pp. 69-84).

(27) Zang, W., Kandah, F., Wang, C., & Li, H. (2010). Dynamic Light Trail Routing in WDM Optical Networks. *Photon Netw Commun.* Springer Science + Business Media, LLC 2010.

(28) Zhu, H., Zang, K., & Mukherjee, B. (2003). A Novel Generic Grap Model for Traffic Grooming in Heterogeneous WDM Mesh Networks. *IEEE/ACM Trans. Netw. 11*, (pp. 285-299).

(29) Zhu, K., & Mukherjee, B. (2000). *Traffic Grooming in an Optical WDM Mesh Network.* Davis Computer Science Technical Report CSE-2000-10.

APPENDIX A

Algorithm 1 DLIT($G, \Lambda, s, t, \mathcal{LT}, L_{\max}$)

1: **for** each wavelength plane WG_i **do**
2: Construct a new wavelength plane WG'_i;
3: Add all nodes in WG_i into WG'_i;
4: **for** each light trail lt in $\mathcal{LT}_i$ **do**
5: **if** s and t are both in lt **and** s is an upstream node of t **then**
6: Return light trail lt to satisfy the request (s, t);
7: **end if**
8: **if** neither s nor t is in lt **then**
9: Add an edge eg from the *convener node* of lt to the *end node* of lt in WG'_i;
10: $cost(eg) = \varepsilon, length(eg) = hops\ of\ lt$;
11: **end if**
12: **if** only s is in lt **and** $s \neq$ *end node* of lt **then**
13: Add an edge e from s to the *end node* of lt in WG'_i;
14: $cost(eg) = \varepsilon, length(eg) = hops\ of\ lt$;
15: **end if**
16: **if** only t is in lt **and** $t \neq$ *convener node* of lt **then**
17: Add an edge e from the *convener node* of lt to t in WG'_i;
18: $cost(eg) = \varepsilon, length(eg) = hops\ of\ lt$;
19: **end if**
20: **end for**
21: **for** each free edge eg on WG_i **do**
22: Add edge eg into WG'_i;
23: $cost(eg) = M, length(eg) = 1$;
24: **end for**
25: TrailFinding($WG'_i, cost, length, L_{\max}$)
26: **if** there is a feasible path in WG'_i **then**
27: Add the path into set $CandidateLightTrail$;
28: **end if**
29: **end for**
30: **if** $CandidateLightTrail$ is not empty **then**
31: Choose the minimum cost path P, which uses wavelength l, from $CandidateLightTrail$;
32: Recover P to be a set of edges $MinLT$ on wavelength plane WG_l;
33: UpdateLightTrail($WG_l, MinLT$);
34: Return the found light trail;
35: **else**
36: Drop this connection request;
37: **end if**

Figure A1. Algorithm 1

Algorithm 2 TrailFinding($WG'_i, cost, length, L_{\max}$)

1: Construct a directed graph $WG_i'^{L_{\max}}$ with node set $V^{L_{\max}} = V \times \{0, 1, \ldots, L_{\max}\}$ and edge set $E^{L_{\max}}$.
2: **if** (u, v) is an edge in WG'_i **then**
3: Add edges from u_{l_c} to v_{l_v} such that $l_v = l_c + length(u, v)$, where $0 \le l_c \le L_{\max} - length(u, v)$, in $E^{L_{\max}}$;
4: Assign cost $cost(u, v)$ to all such edges;
5: $E^{L_{\max}}$ also contains *zero-cost* edges from vertex t_{l_c} to $t_{L_{\max}}$;
6: **end if**
7: Compute the shortest paths from s_0 to $t_{L_{\max}}$ in $WG_i'^{L_{\max}}$.
8: **if** there is a path p **then**
9: Return path p;
10: **else**
11: Drop the connection request;
12: **end if**

Figure A2. Algorithm 2

Algorithm 3 UpdateLightTrail($WG_l, MinLT$)

1: $tempLT = \emptyset$;
2: **while** *MinLT is not EMPTY* **do**
3: Pop an edge (u, v) from *MinLT*
4: Mark node u;
5: **if** node v has been marked **then**
6: Insert (u, v) back to the head of *MinLT*;
7: Set up all edges in *tempLT* to be a new light trail *newlt*;
8: Insert *newlt* into set of light trails *foundlightrails*;
9: Remove all light trails in $\mathcal{LT}_i$ which have been expanded into *newlt*;
10: Unmark all marked nodes;
11: Empty set *tempLT*;
12: **else**
13: Insert (u, v) to set *tempLT*;
14: **end if**
15: **end while**

Figure A3. Algorithm 3

Algorithm 4 Survivable LTDesign($G, s, t, \mathcal{LT}, L_{\max}$)

1: Apply Lines 1 - 29 of LTRouting($G, s, t, \mathcal{LT}, L_{\max}$);
2: **if** set $CandidateLightTrail$ is empty **then**
3: block this request;
4: **end if**
5: **for** k minimum working light trails found in set $CandidateLightTrail$ **do**
6: pick up a light trail as the working light trail;
7: remove this working light trail from network G;
8: Apply Lines 1 - 29 of LTRouting($G, s, t, \mathcal{LT}, L_{\max}$);
9: **if** set $CandidateLightTrail$ is not empty **then**
10: pick up the minimum light trail in the set as the backup light trail;
11: Return the working and backup light trails;
12: **else**
13: Recover the working light trail in G;
14: **end if**
15: **end for**
16: Block the connection request (s, t);

Figure A4. Algorithm 4

APPENDIX B

Main Method

```
package lighttrailalgorithm;

import com.jmatio.io.MatFileWriter;
import com.jmatio.types.MLDouble;
import java.util.*;

public class Main {
    /**
     * Each simulation instance is executed given a graph type. Supported graph
     * types are: 'a':Arpanet, 'i':ItaliaNet, 'n': NSFNet,
     * 'r': random graph with fixed parameters as in the paper.
     */
    public static List<Character> graphTypes;
    /**
     * A list of the number of wavelengths allowed per link for each simulation.
     * For example, {4,8,16,32,64,128,256} runs 3 simulations per each graph, with wavelengths
     * 4, 8, 16, 32, 64, 128 and 256 respectively.
     */
    public static List<Integer> supportedWavelengths;
    /**
     * Number of connections in each simulation to be run.
     */
    public static int noConnections = 900;
    /**
     * Number of repetitive trials for each simulation to be run.
     */
    public static int noTrials = 10;
    /**
     * Determines whether the simulations are carried with the survivable
     * connection property or not. Supported values are:
     * 0: connections are not survivable in all simulations,
     * 1: connections are survivable in all simulations,
     * 2: carry out two simulations per graph, one without survivable property
     * and one with.
     */
    public static int s = 0;
```

```
    /**
     * A list of all the simulations that have run in the application. Each
     * simulation object in the list has simulation parameters, a log and
     * simulation results.
     */
    public static List<Simulation> sims = new ArrayList<>();

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

    new Simulation('a', 4, 10, 1, false);

//        readInputCommands(args);
    }

    /**
     * Parses the command line arguments and starts the corresponding
     * simulations then initiates writing results to file, or displays error
     * messages if input commands are not supported.
     * @param args
     */
    public static void readInputCommands(String[] args) {

        if (args.length == 0) {
            // Start simulation with default settings
            graphTypes = new ArrayList<>();
            graphTypes.add('a');
            graphTypes.add('n');
            graphTypes.add('i');
            graphTypes.add('r');
            supportedWavelengths = new ArrayList<>();
            supportedWavelengths.add(4);
            supportedWavelengths.add(8);
            supportedWavelengths.add(16);
```

```
            supportedWavelengths.add(32);
            supportedWavelengths.add(64);
            supportedWavelengths.add(128);
            supportedWavelengths.add(256);
            start();
        } else {
            int i = 0;
            do {
                String command = args[i];
                switch (command) {
                    case "-g":
                        i = ++i;
                        graphTypes = new ArrayList<>();
                        String types = args[i];
                        for (char c : types.toCharArray()) {
                            if (c == 'a' || c == 'n' || c == 'i' || c == 'r') {
                                graphTypes.add(c);
                            } else if (c == ',') {
                                // do nothing - move to next graph type.
                            } else {
                                System.out.println("Error: unsupported graph type");
                                return;
                            }
                        }
                        i = ++i;
                        break;
                    case "-w":
                        i = ++i;
                        String wavelengths = args[i];
                        String noWavelengths = "0";
                        try {
                            for (char c : wavelengths.toCharArray()) {
                                if (c == ',') {
                                    supportedWavelengths.add(Integer.parseInt(noWavelengths));
                                    noWavelengths = "0";
                                } else {
```

```
                noWavelengths = noWavelengths.concat(String.valueOf(c));
            }
        }
    } catch (Exception e) {
        System.out.println("Error: the values after -w sould be integers>0,"
                + " separated by commas");
        return;
    }
    i = ++i;
    break;
case "-c":
    i = ++i;
    try {
        noConnections = Integer.parseInt(args[i]);
    } catch (Exception e) {
        System.out.println("Error: the value after -c sould be an integer>0");
        return;
    }
    i = ++i;
    break;
case "-t":
    i = ++i;
    try {
        noTrials = Integer.parseInt(args[i]);
    } catch (Exception e) {
        System.out.println("Error: the value after -t sould be an integer>0");
        return;
    }
    i = ++i;
    break;
case "-s":
    i = ++i;
    s = Integer.parseInt(args[i]);
    if (s < 0 || s > 2) {
        System.out.println("Error: the value after -s sould be between 0 and 2");
        return;
```

```
                    }
                    i = ++i;
                    break;
                default:
                    System.out.println("Error: unsupported command");
                    return;
            }
        } while (i < args.length);
        start();
    }
}

/**
 * A helper function to start simulation objects and stores them in sims for
 * further processing
 */
public static void start() {
    switch (s) {
        case 0:
            for (int noWavelengths : supportedWavelengths) {
                for (Character type : graphTypes) {
                    sims.add(new Simulation(type, noWavelengths, noConnections, noTrials, false));
                }
            }
            break;
        case 1:
            for (int noWavelengths : supportedWavelengths) {
                for (Character type : graphTypes) {
                    sims.add(new Simulation(type, noWavelengths, noConnections, noTrials, true));
                }
            }
            break;
        case 2:
            for (int noWavelengths : supportedWavelengths) {
                for (Character type : graphTypes) {
                    sims.add(new Simulation(type, noWavelengths, noConnections, noTrials, false));
```

```
                }
            }
            for (int noWavelengths : supportedWavelengths) {
                for (Character type : graphTypes) {
                    sims.add(new Simulation(type, noWavelengths, noConnections, noTrials, true));
                }
            }
            break;
    }

    writeToFile(sims);
}

/**
 * A helper function to write the results of all simulations in MATLAB
 * compatible files (with extension .mat). Each written file identifies the
 * simulation parameters for the contained results for easier processing in
 * MATLAB. The function makes use of the 3rd party package jmatio.
 * @param sims
 */
public static void writeToFile(List<Simulation> sims) {
    for (Simulation sim : sims) {
        String fileName = Character.toString(sim.graphType);
        fileName = fileName.concat("_");
        fileName = fileName.concat(Integer.toString(sim.noWavelengths));
        fileName = fileName.concat("_");
        fileName = fileName.concat(Integer.toString(sim.noConnections));
        fileName = fileName.concat("_");
        fileName = fileName.concat(Integer.toString(sim.noTrials));
        fileName = fileName.concat("_");
        fileName = fileName.concat(Boolean.toString(sim.isSurvivable));
        fileName = fileName.concat(".mat");

        double[] results = {sim.getAvgNoFreeWL(), sim.getAvgNoAccConn()};
        MLDouble mlDouble = new MLDouble("results", results, 1);
        ArrayList list = new ArrayList();
```

```
            list.add(mlDouble);
            try {
                new MatFileWriter(fileName, list);
            } catch (Exception e) {
                System.out.println(e);
                continue;
            }
        }

    }
}
```

Connection Class

```
package lighttrailalgorithm;
import java.sql.Statement;
import java.util.*;

public class Connection {
    public Node src;
    public Node dst;
    public int startTime;
    public int lifeTime;
    public int endTime;
    public int demand;

    public List<LightTrail> ltList;
    public int noUsedFreeWL;

    Connection(Graph g, int sKey, int dKey, int startTime, int lifeTime, int demand){
        src = g.getNode(sKey);
        dst = g.getNode(dKey);
        this.startTime = startTime;
        this.lifeTime = lifeTime;
        this.endTime = startTime + lifeTime -1;
        this.demand = demand;
        this.ltList = new LinkedList<>();
    }

    public void addLT(LightTrail lt){
        ltList.add(lt);
    }

    @Override
    public String toString(){
        String s = "Connection " + startTime
                + " ends @ " + endTime
                + " (" + src.key + ">>" + dst.key + ") -- "
                + " using trails: \n";
        for (LightTrail lt: ltList){
```

```
            s = s.concat(lt.toString() + "\n");
        }
        return s;
    }
}
```

Dijkstra Class

```
package lighttrailalgorithm;

import java.util.*;
import lighttrailalgorithm.Graph;

public class Dijkstra {

    public Graph g;
    public Node src;
    public Node dst;
    public List<Node> solved;
    public List<Node> unsolved;
    public List<Edge> edges;
    public List<Double> minDistance;
    public List<Node> parents;
    public boolean pathFound;
    public boolean terminate;
    public ArrayList<Edge> path;
    public double minCost;

    public Dijkstra(Graph g, Node src, Node dst) {
        this.g = g;
        this.src = src;
        this.dst = dst;
       // runDijkstra();
    }

    public void updateDistance() {
        double minDist = -1;
        Node minNode = null;
        Node minParent = null;
        Edge minEdge = null;

        for (Edge e : edges) {
            }
            // Only for undirected graphs:
```

```
//            if (solved.contains(e.destination) && unsolved.contains(e.source)) {
//
//                 double dist = e.cost + minDistance.get(solved.indexOf(e.destination));
//                 if (dist < minDist || minDist == -1) {
//                     minDist = dist;
//                     minEdge = e;
//                     minNode = minEdge.source;
//                     minParent = minEdge.destination;
//                 }
//             }
        }



    }
```

DLIT Class

```
package lighttrailalgorithm;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import lighttrailalgorithm.Connection;
import lighttrailalgorithm.Graph;
import lighttrailalgorithm.LightTrail;

public class DLIT {

    public static final int lMax = 5;
    public static final double epsilon = 0.001;
    public double M;

    public LinkedList<LightTrail> MainDLIT(Graph g, int noWavelengths, Connection c) {
        M = 10 * g.numberOfEdges;
        createWVPlanes(g, noWavelengths);

        LinkedList<LightTrail> chosenLTs = new LinkedList<>();
        LinkedList<LightTrail> candidateLTs = new LinkedList(findCandidateLTs(g, c));

        if (!candidateLTs.isEmpty()) {
            if (!g.isSurvivable) {
                if (!candidateLTs.getFirst().isLightWalk) {
                    LightTrail lt = candidateLTs.getFirst();
                    chosenLTs.add(lt);
                } else {
                    }
            } else {
                for (LightTrail candidate : candidateLTs) {
                    LightWalk minLT = (LightWalk) candidate;
                    WG chosenWG = g.wgList.get(minLT.wavelength);
                    minLT = recover(minLT, chosenWG, c);
                    Graph g1 = g.copyExtracting(minLT);
```

```
                g1.isSurvivable = false;
               //   Connection c1 ="";// new Connection(g1, c.src.key, c.dst.key, c.startTime, c.lifeTime, c.demand);
                 }
            }
        }
        return chosenLTs;
    }

    private void createWVPlanes(Graph baseGraph, int noWavelengths) {
        List<WG> wgs = baseGraph.wgList;
        for (int i = 0; i < noWavelengths; i++) {
            wgs.add(new WG(i, baseGraph));
        }
    }

    public List<LightTrail> findCandidateLTs(Graph g, Connection c) {
        List<LightTrail> candidateLT = new LinkedList<>();
        for (WG wg : g.wgList) {
            WG wg1 = new WG(wg);
            for (LightTrail lt : wg.ltList) {
                     return candidateLT;
               }
            LightTrail clt = trailFinding(wg1, c);
            if (clt != null) {
                candidateLT.add(clt);
            }
        }
        return candidateLT;
    }
       }
```

Edge Class

```
package lighttrailalgorithm;

public class Edge {

    public Node source;
    public Node destination;
    public double cost;
    public int length;
    public boolean free;

    // For shortcut edges
    public boolean shortcut;
    public LightTrail lt;  //parent LightTrail if this was a shortcut



    public Edge(Node source, Node destination){
        this.source = source;
        this.destination = destination;
        this.free = true;
    }

    @Override
    public String toString(){
        return ("(" + source.key + "//" + source.auxKey
                + "," + destination.key + "//" + destination.auxKey
                + ") -- " + cost);
    }
}
```

Node Class

```
package lighttrail;

public class Node {

    public int key;
    public int auxKey;
    public String name;
    public boolean mark;

    public Node (int key){
    this.key  = key;
    this.name  = Integer.toString(key);
    }

    public Node (int key, String name){
    this.key  = key;
    this.name  = name;
    }

    public Node (int key, int auxKey){
        this.key = key;
        this.auxKey = auxKey;
    }

    @Override
    public String toString(){
        return ("N:" + key + "\\"+ auxKey );
    }
}
```

Graph Class

```
package lighttrailalgorithm;

import java.util.*;
import java.sql.Connection;
import lighttrail.Node;
import lighttrail.WG;

public class Graph {
    public List<Node> nodes;
    public List<Edge> edges;
    public int numberOfNodes;
    public int numberOfEdges;
    public List<WG> wgList;
    int noAcceptedConnections;
    int noUsedFreeWL;
    List<Connection> activeConnections;
    int noActiveLTs;
    List<Connection> processedConnections;
    boolean isSurvivable;
    public Graph() {
        this.nodes = new ArrayList<>();
        this.edges = new ArrayList<>();
        this.numberOfNodes = 0;
        this.numberOfEdges = 0;
        wgList = new ArrayList<>();
        noAcceptedConnections = 0;
        noUsedFreeWL = 0;
        activeConnections = new LinkedList<>();
        noActiveLTs = 0;
        processedConnections = new LinkedList<>();
    }

    public void addNode(Node v) {
        nodes.add(v);
        numberOfNodes++;
    }
```

```
public void addEdge(Edge e) {
    edges.add(e);
    numberOfEdges++;
}

public Node getNode(int key) {
    return nodes.get(key);
}

public Node getNode(int key, int auxKey) {
    for (Node n : this.nodes) {
        if (n.key == key && n.auxKey == auxKey) {
            return n;
        }
    }
    return null;
}

public Node getNode(String name) {
    for (Node v : nodes) {
        if (v.name.equals(name)) {
            return v;
        }
    }
    return null;
}



public void updateNoActiveLTs(){
    int n = 0;
    for (WG wg:this.wgList){
        n = n+wg.wavelength;
    }
    this.noActiveLTs = n;
}
```

```
public boolean contains(String name) {
    boolean b = false;
    if (nodes.isEmpty()) {
        return b;
    }
    for (Node v : nodes) {
        if (v.name.equals(name)) {
            b = true;
            return b;
        }
    }
    return b;
}

public boolean isEmpty() {
    return nodes.isEmpty() && edges.isEmpty();
}

public void print() {
    System.out.println("Resulting graph:");
    System.out.println("---------------");

    System.out.println("Vertices:");
    if (nodes.isEmpty()) {
        System.out.println("No vertices found");
    } else {
        for (Node v : nodes) {
            System.out.println("Node: " + v.key + " \\ " + v.auxKey);
        }
    }

    System.out.println("Edges: ");
    if (edges.isEmpty()) {
        System.out.println("No edges found");
    } else {
```

```
        }
    }

    public Graph copy() {
        return null;
    }
    public void clear() {
        wgList = new ArrayList<>();
        noAcceptedConnections = 0;
        noUsedFreeWL = 0;
        activeConnections = new LinkedList<>();
        noActiveLTs = 0;
        processedConnections = new LinkedList<>();
        //call GC
    }

    public Graph copyExtracting(LightWalk lw) {
        Graph g = new Graph();
        g.nodes = this.nodes;
        g.edges = this.edges;
        g.numberOfNodes = this.numberOfNodes;
        g.numberOfEdges = this.numberOfEdges;
        //removing the edges in lw from g

        g.wgList = this.wgList;
        WG wg = g.wgList.get(lw.wavelength);
        if (!lw.subLTs.isEmpty()) {
            for (LightTrail subLT : lw.subLTs) {
             //   wg.removeLT(subLT);
            }
        }
        lw.edges.stream().map((e) -> {

        return g;
}
```

LightTrail Class

```
package lighttrailalgorithm;

import java.util.*;

public class LightTrail {

    public int wavelength;
    public List<Edge> edges;
    public LinkedList<Node> nodeWalk;
    public int hops;
    public double cost;
    public List<Connection> activeConnections;
    public boolean isLightWalk;
    public Set<LightTrail> subLTs;

    LightTrail(int wavelength) {
        this.wavelength = wavelength;
        this.nodeWalk = new LinkedList<>();
        this.edges = new ArrayList<>();
        this.activeConnections = new ArrayList<>();
        this.subLTs = new HashSet<>();
    }

    public void addSubLT(LightTrail lt){
        this.subLTs.add(lt);
    }

    public void removeSubLTs(){
        if (!subLTs.isEmpty()){
            for (LightTrail subLT : subLTs){
            //    this.addConnections(subLT);
            }
        }
        this.subLTs.clear();
    }
```

```
          public boolean hasNode(Node n) {
    return nodeWalk.contains(n);
}

public boolean hasEdge(Edge e) {
    return edges.contains(e);
}

public void hasEdge(int srcKey, int dstKey) {
    for (Edge e : edges) {

        }
    }

}
```

LightWalk Class

```
package lighttrailalgorithm;

import java.util.*;

public class LightWalk extends LightTrail {

    public List<Edge> directedEdges;

    LightWalk(int wavelength) {
        super(wavelength);
        directedEdges = new ArrayList<>();
        isLightWalk = true;
    }

    LightWalk(int wavelength, List<LightWalk> lwList){
        this(wavelength);
        LinkedList<LightWalk> list = new LinkedList<>(lwList);
        this.hops = list.size();
        for (LightTrail lw : list){
            for (Edge e:lw.edges){
                edges.add(e);
            }
        }

    }


    public void createDirectedEdges(Node src, Node dst){
      //  nodeWalk.clear();
        Node n1 = src;
        for (Edge e: edges){

        }
        //updateNodeWalk();
    }
```

```
    public void addDirectedEdge(Edge e){
        directedEdges.add(e);
    }



    public void addEdge(Edge e) {
        edges.add(e);
    }

    public void addEdges(LightTrail lt) {
        edges.addAll(lt.edges);
    }

}
```

WG Class

```
package lighttrailalgorithm;

import java.util.*;

/**
 * Wavelength Plane class. It extends the Graph class with additional support of
 * Light Trail inclusion.
 */

public class WG extends Graph {
    /**
     * The Wavelength Plane's wavelength.
     */
    public int wavelength;
    /**
     * A list of all Light Trails associated with this Wavelength Plane.
     */
    public List<LightTrail> ltList;

    /**
     * Creates a Wavelength Plane based on the input graph. All nodes and edges
     * in the base graph are copied into the Wavelength Plane. No Light Trails are
     * created.
     */

    WG(int wavelength, Graph baseGraph) {
        this.wavelength = wavelength;
        Graph clone = baseGraph.copy();
        this.nodes = clone.nodes;
        this.edges = clone.edges;
        this.numberOfNodes = clone.numberOfNodes;
        this.numberOfEdges = clone.numberOfEdges;
        this.ltList = new ArrayList<>();
    }

    WG(WG rootWG) {
```

```
        this.wavelength = rootWG.wavelength;
        this.nodes = rootWG.nodes;
    }

    /**
     * Adds the Light Trail lt to the set of current Light Trails associated
     * with this Wavelength Plane. All edges on the Light Trail are marked as
     * non-free for later processing.
     * @param lt
     */
    public void addLT(LightTrail lt) {
        if (!ltList.contains(lt)){
            ltList.add(lt);
            noActiveLTs ++;
        }
        for (Edge e: lt.edges){
            e.free = false;
        }
    }

    /**
     * Removes the Light Trail lt from the set of current Light Trails associated
     * with this Wavelength Plane. All edges on the Light Trail are marked as
     * free for later processing.
     * @param lt
     */
    public void removeLT(LightTrail lt) {
        ltList.remove(lt);
        for (Edge e: lt.edges){
            e.free = true;
        }
        noActiveLTs--;
    }

}
```

ArpanetGraph Class

```
package lighttrailalgorithm;

import lighttrail.Node;

public class ArpanetGraph extends Graph{

    ArpanetGraph(){
//      Creating nodes
        for (int i=0;i<20;i++) {
            nodes.add(new Node(i));
        }
//      Creating edges
        this.numberOfNodes = nodes.size();
     }

    @Override
    public Graph copy(){
        Graph g = new ArpanetGraph();
        return g;
    }

}
```

ItalianetGraph Class

```
package lighttrailalgorithm;

public class ItalianetGraph extends Graph {

    ItalianetGraph() {
//        Creating nodes
        for (int i = 0; i < 33; i++) {
            nodes.add(new Node(i));
        }

//        Creating edges


//      Updating node and edge count
        this.numberOfNodes = nodes.size();
        this.numberOfEdges = edges.size();

    }

    @Override
    public Graph copy(){
        Graph g = new ItalianetGraph();
        return g;
    }
}
```

NsfnetGraph Class

```
package lighttrailalgorithm;

public class NsfnetGraph extends Graph{

    NsfnetGraph(){
//        Creating nodes
          nodes.add(new Node(0,"WA"));
          nodes.add(new Node(1,"CA1"));
          nodes.add(new Node(2,"CA2"));
          nodes.add(new Node(3,"UT"));
          nodes.add(new Node(4,"CO"));
          nodes.add(new Node(5,"NE"));
          nodes.add(new Node(6,"TX"));
          nodes.add(new Node(7,"MI"));
          nodes.add(new Node(8,"IL"));
          nodes.add(new Node(9,"PA"));
          nodes.add(new Node(10,"GA"));
          nodes.add(new Node(11,"NY"));
          nodes.add(new Node(12,"MD"));
          nodes.add(new Node(13,"NJ"));

//        Creating edges

//      Updating node and edge count
        this.numberOfNodes = nodes.size();
        this.numberOfEdges = edges.size();

    }

    @Override
    public Graph copy(){
        Graph g = new NsfnetGraph();
        return g;
    }

}
```

RandomGraph Class

```
package lighttrailalgorithm;

import java.awt.*;
import java.util.*;

public class RandomGraph extends Graph{

    RandomGraph(int n, int width, int height, double alpha, double beta){
        Random rg = new Random();
        ArrayList<Point> points = new ArrayList<>();
        double[][] dist = new double[n][n];

        for (int i=0; i<n; i++){
            nodes.add(new Node(i));
            int x = rg.nextInt(width+1);
            int y = rg.nextInt(height+1);
            points.add(new Point(x,y));
        }
        double M = 0;
        for (int i=0; i<n; i++){
            for (int j=i+1; j<n; j++){
                Point p1 = points.get(i);
                Point p2 = points.get(j);
                double Xdiff = p2.getX()-p1.getX();
                double Ydiff = p2.getY()-p1.getY();
                double d = Math.sqrt(Xdiff*Xdiff + Ydiff*Ydiff);
                dist[i][j] = d;
                if (d>M){
                    M = d;
                }
            }
        }

        for (int i=0; i<n; i++){
            for (int j=i+1; j<n; j++){
                double a = dist[i][j] / (alpha * M);
```

```
                double p = beta * Math.exp(a);
                if (rg.nextDouble() < p){
                  //  edges.add(new Edge(nodes.get(i),nodes.get(j)));
                }
            }
        }

//      Updating node and edge count
        this.numberOfNodes = nodes.size();
        this.numberOfEdges = edges.size();
    }

    @Override
    public Graph copy(){
        Graph g = new Graph();
        for (int i=0; i<numberOfNodes; i++){
            g.nodes.add(new Node(i));
        }
        for(Edge e: edges){
              }
        g.numberOfNodes = g.nodes.size();
        g.numberOfEdges = g.edges.size();
        return g;
    }

}
```

Simulation Class

```
package lighttrailalgorithm;

import java.util.*;

public class Simulation {

    private static int noSimulations = 0;

    /**
     * @return the noSimulations
     */
    public static int getNoSimulations() {
        return noSimulations;
    }

    /**
     * @param aNoSimulations the noSimulations to set
     */
    public static void setNoSimulations(int aNoSimulations) {
        noSimulations = aNoSimulations;
    }
    private int simID;
    public Character graphType;
    public int noWavelengths;
    public int noConnections;
    public int noTrials;
    public boolean isSurvivable;
    private Date d;
    private Graph g;
    private int[] connectionTotals;
    private int[] freeWLTotals;
    private double avgNoAccConn;
    private double avgNoFreeWL;

    Simulation(Character graphType, int noWavelengths, int noConnections, int noTrials, boolean isSurvivable) {
        init(graphType, noWavelengths, noConnections, noTrials, isSurvivable);
```

```
        for (int trial = 0; trial < noTrials; trial++) {
            DLIT worker = new DLIT();
            for (int t = 0; t < noConnections; t++) {
                Connection c = generateRandomConnection(t);
                LinkedList<LightTrail> chosenLTs = worker.MainDLIT(g, noWavelengths, c);
                updateGraph(t, c, chosenLTs);
                printProgress(t);
            }
            connectionTotals[trial] = g.noAcceptedConnections;
            freeWLTotals[trial] = g.noUsedFreeWL;
            printFinal();
            g.clear();
        }
        calculateAvgs();
    }

    private void init(Character graphType, int noWavelengths, int noConnections, int noTrials, boolean isSurvivable) {
        this.simID = ++noSimulations;
        this.graphType = graphType;
        this.noWavelengths = noWavelengths;
        this.noConnections = noConnections;
        this.noTrials = noTrials;
        this.isSurvivable = isSurvivable;
        d = new Date();
        if (graphType == 'a') {
            g = new ArpanetGraph();
        } else if (graphType == 'i') {
            g = new ItalianetGraph();
        } else if (graphType == 'n') {
            g = new NsfnetGraph();
        } else {
            g = new RandomGraph(20, 1000, 1000, 0.15, 0.2); //hardcoded for now
        }
        g.isSurvivable = isSurvivable;
        connectionTotals = new int[noTrials];
        freeWLTotals = new int[noTrials];
```

```
        printInit();
    }
    private Connection generateRandomConnection(int startTime) {

        Random rg = new Random();
        int src = rg.nextInt(g.numberOfNodes);
        int dest;
        do {
            dest = rg.nextInt(g.numberOfNodes);
        } while (dest == src);
        int lifeTime;
        do {
            lifeTime = rg.nextInt(101);
        } while (lifeTime == 0);
        int demand;
        do {
            demand = rg.nextInt(21);
        } while (demand == 0);
        return new Connection(g, src, dest, startTime, lifeTime, demand);
    }
    private void calculateAvgs() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }
    private void printFinal() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }
    private void printProgress(int t) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }
    private void updateGraph(int t, Connection c, LinkedList<LightTrail> chosenLTs) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }
    private void printInit() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }
}
```

LightpathAlgorithm Class

```
package lighttrailalgorithm;

import java.util.Date;
import java.util.LinkedList;
import java.util.Random;

public class LightpathAlgorithm {
    public Node source;
    public Node destination;
    public double cost;
    public int length;
    public boolean free;
    // For shortcut edges
    public boolean shortcut;
    public LightTrail lt;  //parent LightTrail if this was a shortcut
    public LightpathAlgorithm(Node source, Node destination){
        this.source = source;
        this.destination = destination;
        this.free = true;
    }
    @Override
    public String toString(){
        return ("(" + source.key + "//" + source.auxKey
                + "," + destination.key + "//" + destination.auxKey
                + ") -- " + cost);
    }
     public static void setNoSimulations(int aNoSimulations) {
        int noSimulations = aNoSimulations;
    }
    private int simID;
    public Character graphType;
    public int noWavelengths;
    public int noConnections;
    public int noTrials;
    public boolean isSurvivable;
```

```
private Date d;
private Graph g;
private int[] connectionTotals;
private int[] freeWLTotals;
private double avgNoAccConn;
private double avgNoFreeWL;

LightpathAlgorithm(Character graphType, int noWavelengths, int noConnections, int noTrials, boolean isSurvivable) {
    init(graphType, noWavelengths, noConnections, noTrials, isSurvivable);
    for (int trial = 0; trial < noTrials; trial++) {
        DLIT worker = new DLIT();
        for (int t = 0; t < noConnections; t++) {
            Connection c = generateRandomConnection(t);
            LinkedList<LightTrail> chosenLTs = worker.MainDLIT(g, noWavelengths, c);
            updateGraph(t, c, chosenLTs);
            printProgress(t);
        }
        connectionTotals[trial] = g.noAcceptedConnections;
        freeWLTotals[trial] = g.noUsedFreeWL;
        printFinal();
        g.clear();
    }
    calculateAvgs();
}

private void init(Character graphType, int noWavelengths, int noConnections, int noTrials, boolean isSurvivable) {
    int noSimulations = 0;
    this.simID = ++noSimulations;
    this.graphType = graphType;
    this.noWavelengths = noWavelengths;
    this.noConnections = noConnections;
    this.noTrials = noTrials;
    this.isSurvivable = isSurvivable;
    d = new Date();
    if (graphType == 'a') {
        g = new ArpanetGraph();
```

```
        } else if (graphType == 'i') {
            g = new ItalianetGraph();
        } else if (graphType == 'n') {
            g = new NsfnetGraph();
        } else {
            g = new RandomGraph(20, 1000, 1000, 0.15, 0.2); //hardcoded for now
        }
        g.isSurvivable = isSurvivable;
        connectionTotals = new int[noTrials];
        freeWLTotals = new int[noTrials];
        printInit();
    }

    private Connection generateRandomConnection(int startTime) {

        Random rg = new Random();
        int src = rg.nextInt(g.numberOfNodes);
        int dest;
        do {
            dest = rg.nextInt(g.numberOfNodes);
        } while (dest == src);
        int lifeTime;
        do {
            lifeTime = rg.nextInt(101);
        } while (lifeTime == 0);
        int demand;
        do {
            demand = rg.nextInt(21);
        } while (demand == 0);
        return new Connection(g, src, dest, startTime, lifeTime, demand);
    }

    private void calculateAvgs() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }
```

```
    private void printFinal() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    private void printProgress(int t) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    private void updateGraph(int t, Connection c, LinkedList<LightTrail> chosenLTs) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    private void printInit() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }
}
```