SHORTEST PATHWAY AND TIME DETERMINATION IN A WIRELESS PACKET SWITCH NETWORK SYSTEM

BY

OFEM, OFEM AJAH
REG. No. MTH/Ph.D/08/002

A Ph.D DISSERTATION CARRIED OUT IN THE
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF CALABAR-CALABAR
NIGERIA

SUBMITTED TO

GRADUATE SCHOOL
UNIVERSITY OF CALABAR
CALABAR, NIGERIA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
AWARD OF DOCTOR OF PHILOSOPHY (Ph.D)
IN COMPUTER SCIENCE

JULY, 2016

CERTIFICATION

I, Ofem, Ofem Ajah with registration number MTH/Ph.D/08/002 of the Department of Computer Science, hereby certify that this dissertation on the "shortest pathway and time determination in a wireless packet switch network system in the University of Calabar" is original, and has been written by me. It is a record of my research work and has not been presented before in any previous publication.

Ofem, Ofem Ajah
(*Student/Candidate*)

Signature.......[illegible].......

Date....22/07/2016....

DECLARATION

We declare that this thesis entitled "shortest pathway and time determination in a wireless packet switch network system in the University of Calabar" by Ofem, Ofem Ajah (Reg. Number. MTH/Ph.D/08/002), was carried out under our supervision, has been found to have met the regulations of the University of Calabar. We therefore recommend the work for the award of Doctor of Philosopher (Ph.D) in Computer Science.

Dr. Azom E. Edim (Chief Supervisor) *Qualification*: B.Sc., M.Sc., Ph.D. *Status*: Senior Lecturer	Signature [illegible] Date 22/07/2016
Prof. Joseph O. Esin (Supervisor) *Qualification*: B.Sc., M.A. ED.D. *Status*: Professor of Research	Signature [illegible] Date 22/07/2016
Dr. Azom E. Edim (Head of Department) *Qualification*: B.Sc., M.Sc., Ph.D. *Status*: Senior Lecturer	Signature [illegible] Date 22/07/2016
Dr. Rufus Okoro (Graduate Sch. Representative) *Qualification*: B.Sc., M.Sc., Ph.D *Status*: Reader	Signature [illegible] Date 20/7/16
Prof. Stephen O. Olabiyisi (External Examiner) *Qualification*: B.Tech, M.Tech., M.Sc., Ph.D. *Status*: Professor of Computer Science	Signature [illegible] Date 22/07/16

ACKNOWLEDGEMENTS

Let me start by thanking the Almighty God for providing His sufficient grace that sustained me throughout the period of this programme. May His Name continue to be praised forever more, Amen.

I sincerely acknowledge with utmost gratitude the immense contributions of my supervisor, Dr Edim, E. A. and Co-supervisor, Prof. Joseph O. Esin, I appreciate their guidance, encouragement, useful criticisms and suggestions which enhanced the completion of this research.

I would like to thank my mentor: Prof. Zolt Lipcsey and Prof. Okecha for their constant encouragement and stimulating advise. Not forgetting my colleagues, Dr. (Mrs.) Eteng, I. E., Dr. F. U. Ogban, Dr. Arikpo I., Dr. E. E. Williams, Dr. Akpan, S. S., Dr. Ele, B. I., Mr. Ele, S. I., Engr. Osahon, O., Mr. Isamo, E., Dr. C. Onwukwe., Dr. Effang, O., Dr. Ernest, E. and others, who have been of great moral inspiration to this research work.

My deepest and immeasurable gratitude goes to my wife Mrs. Glory Ofem for her encouragement. I wish to register my thanks to my children, Master Ofem Ofem, Joseph Ofem and Elisha Ofem. My appreciation goes to my course mate, Mr. Essien Eyo, whose contributions served as a motivating factor in writing this thesis.

ABSTRACT

The problem of finding the shortest pathway and time between two nodes is a well known problem in network analysis. Optimal routing has been widely studied for interconnection networks. This research considers the problem of finding the shortest pathway and time in a wireless packet switch network system in the University of Calabar environment, its theoretical approach, implementation and application. Firstly, the research problem is to design and analyze a wireless packet switch network system. Some routing policies used for packet transmission in a network can be a hindrance to efficient transmission of packets in the network, hence, the research aim is the problem of network routing and packet transmission delay, with an objective of determining the shortest path d_{ij} and time t_{ij} taken by a packet to traverse from a given source node to a given destination (sink) node through an interconnected communication links. The methodology adopted for this research was a formation of a modified Dijkstra's algorithm which uses a comparison addition model in determining the shortest path and time in the network system. The modified Dijkstra's algorithm finds one shortest path in a network with time dependent costs of link in order of $O(n + m)$ time, where n is the number of nodes and m is the number of links in the network. The second algorithm designed for the purpose of this research was the open shortest path first (OSPF) which uses the concept of Dijkstra's in transmitting packets in a network. It also runs in the order of $O(n + m)$ time. Both algorithms were presented in the context of wireless packet switch network system. Three routes were considered in the research. Two experiments were carried out in the network known as Tracet and Ping test. From the simulated results in the wireless packet switch network, it was found that Route 1 shows a distance of 120m with a speed of 7.06m/s and time range of 0.017s to the destination and back through the Internet Control Message Protocol (ICMP Echo). Route 2 shows a distance of 110m with a speed of 6.88m/s and time range of 0.016s to the destination and back through the Internet Control Message Protocol (ICMP Echo). Finally, route 3 shows a distance of 100m with a speed of 7.69m/s and time range of 0.013s to the destination and back through the Internet Control Message Protocol (ICMP Echo). This indicates that, distance is directly proportional to time, and inversely proportional to the speed taken for data to move to and fro in the packet switch. The researcher recommends that the designed of an efficient algorithm for the transmission of packets in a network must fulfilled the following conditions; (i) minimize the utilization of memory core (ii) minimizes the run time (iii) reduce rate of packet lost (iv) maximizes the efficiency of the system (v) minimizes system design and maintenance cost. (**Word Count: 478**).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CPM	-	Critical Path Method
CPU	-	Central Processing Unit
EON	-	European Optical Network
HOL	-	Head-Of-Line
ICCRG	-	Internet Congestion Control Research Group
IP	-	Internet Protocol
LAN	-	Local Area Network
LFS	-	Layer First Searching
OSPF	-	Open Shortest Path First
QDA	-	Quantum Dijkastra algorithm
QoS	-	Quality of Service
SANs	-	System Area Networks
SP	-	Shortest Path
SSSP	-	Single Source Shortest Path
SSTDSP	-	Single Source Time-Dependent Shortest Path
STP	-	Spanning-Tree Protocol
PL	-	Permanent Label
TL	-	Temporary Label
FIFO	-	First-In-First-Out
STDSP	-	Single Time Dependent Shortest Path
IS-IS	-	Intermediate System to Intermediate System
GPS	-	Global Positioning System

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

Packet switch networks are dynamically changing systems that transport packets of data from different sources to different destinations through a network of interconnected communication links. The state of communication links can change, becoming more, less congested or failing completely. Modern telecommunication traffic is performed through packet switching networks: information is broken into small pieces (typically 1000 bytes in length, or less) known as packets. Each packet contains a portion of data as well as some control information. The control information includes the information required to be able to route the packet through the network and deliver it to its destination node. At each node, the routing information is examined and the packet is passed on to the next node. The routing policy plays significant role in determining the overall network performances. Ideal algorithm comprises finding the "optimal" path(s) between source and destination node, enabling high speed data transmission and avoiding a packet loss (Rowe and Schuh, 2005).

The shortest path problem can be formulated as one of finding a minimal cost path that contains the designated source and destination nodes.

The apparent solution to shortest path problem is locating a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. Routing protocols such as Open Shortest Path First (OSPF) uses the Dijkstra's algorithm in calculating the shortest path in a network. It works first, by constructing a shortest path tree then the routing table is populated with the resulting best paths.

Shortest path and time determination in packet switch network system has been a subject of extensive research, resulting in a number of algorithms for various conditions and constrains. In a packet switch network system, communication between two hosts generally are grouped into the following classification; the transmitting host delivers to a node a block of data, called a packet, which are addressed to the destination host (David, 2012). Furthermore, Anupama and Anuj (2014) opined that such architecture insists routers to broadcast the local resource status and the local topology information to all routers. They went ahead to explain that, one manner of providing Quality of Service (QoS) in routers is to apply traffic prioritization. The idea is to classify the traffic to a multiple level of priority queues. The priorities are assigned on packet peculiarities; the protocol uses packet type, source and destination networks. Enhancements are done by subdividing the link capacity into different classes. The traffic is assigned to each class and the routers serve each class with different priority (Anupama and Anuj, 2014). However, the traffic prioritization improves the QoS by class of traffic on a given link, but that link is chosen by the shortest path routing mechanism, which is independent of the QoS requirements. The optimal routing algorithm must keep the delays low as the flow control increases. Hence, the routing increases the throughput and restricts the delay for the packet, during high traffic conditions. The average delay per packet is reduced also at steady or low traffic conditions.

The problems associated with routing in packet-switched networks have been identified and most widely studied by communications network researchers. A large body of literature reviewed for this study presents the theory and experience gained from several decades of progress. One of the earliest research networks was the original ARPANET experiment (Morley, 2013). During the early years of Arpanet development, researchers interested in routing were concerned with issues that included low bandwidth error-prone links, unreliable electronics, and limited processing power and storage capacity. Substantial effort was expended to establish the insight required to understand and solve the resulting routing problems.

Accordingly, Anupama and Anuj (2014) stated that, the function of a routing algorithm is to guide packets through the communication network to their correct destination. Routing algorithm, initially inspired by the Hopfield neural network (Hopfield, 1982), is designed to find the optimal link between source and destination node taking into account not only the shortest path but also several in-node constraints, including link bandwidth, incoming flow and flow statistics, addressed to avoiding the packet loss (Hopfield, 1985).

In the early years of 21st century, the communication by packet flow in large-scale computer networks in an information-oriented society has become much more important in our daily life than ever before. According to Silla and Duato (2000), adaptive routing algorithms tend to select the route of packets dynamically, have been widely studied to make the best use of bandwidth in interconnection networks of massively parallel computers and system area networks (SAN). In the context of SAN operations proposed simple methods to support adaptive routing in InfiniBand switches (Martinez *et al.*, 2004).

As indicated in Internetworking Technology Handbook (ITH) (2002) in today's Internet Protocol (IP) networks, routing protocols are responsible for building a path that carries a data packet to its destination. Each router in the network has to send the packet to its next hop, independently from what other routers are doing at the time, on rules based only on its own knowledge base. These routing tables are built based on topological and traffic information, send and capture from information being received from other routers. Routing forms an integral part of the communications subnetwork. The routing algorithm is a part of the network layer which is responsible for deciding on which outbound queue an incoming packet should be transmitted. It guides packets through the communication network to their correct destinations. If the network uses datagrams internally, this decision must be made for every arriving data packet. An ideal routing algorithm should strive to find an optimum path for packet transmission within a specified time so as to satisfy the Quality of Service (QoS) (Ahn *et al.*, 2001). There are several search algorithms for the shortest path (SP) problem such as the breadth-first search algorithm, the Dijkstra algorithm and the Bellman-Ford algorithm, to name a few. Since these algorithms can solve shortest path (SP) problems in polynomial time (Ahn *et al.*, 2001) (for example if the number of steps required to complete the algorithm for a given input is for some non-negative integer, and complexity of the input is said to fast), they will be effective in fixed infrastructure wireless or wired networks. But, they exhibit unacceptably high computational complexity for real-time communications involving rapidly changing net work topologies (Ahn *et al*, 2001). In majority of the current packet-switching networks, form of SP computation is employed by routing algorithms in the network layer (Ahn *et al.*, 2001). Specifically, the network links are weighted, the weights reflecting the link transmission capacity, the congestion of networks and the estimated transmission status such as the queuing delay of head-of-line (HOL) packet or the link failure.

1.2 Statement of the Problem

The purpose of this research is to design and analyze a wireless packet-switch network system in the University of Calabar community. Most routing policies used for packet transmission in a network can be a hindrance to efficient transmission of packets in the network. Finding the optimal routing algorithm is a major challenge. The solution to the routing algorithm change will help to keep the delays low, increase the throughput and restrict the delay in packet transmission and congestion in the network. Finding the shortest path and considering other in-node constraints will increase network efficiency and avoid packet loss.

This research study analyses four different algorithms and their efficiencies and complexities in relation to packet transmission in the network, and will attempt to bring up a more efficient algorithm to solve the routing problem in a packet switch network. To achieve this, the researcher decided to modify the data structure for the Dijkstra algorithm. The modified Dijkstra's algorithm uses the linked-list priority queue (Anupama and Anuj, 2014), in order to design a network system with the higher Quality of Service (QoS) and maximizes system efficiency through the minimal path weight.

1.3 Aim and Objectives of the Study

The aim of this study is to develop a more efficient algorithm to solve the network routing and packet transmission delay, The specific objectives are to:

a) design and analyze a wireless packet switch network system within unical.

b) determine and identify the shortcomings of the existing shortest path, time determination techniques and proposed routing algorithms to overcome it.

c) design an algorithm in solving the high time complexity (delay) and low accuracy.

d) evaluate different algorithms in order to solve the shortest path problem, as well as to understand the different algorithms in terms of graph

e) explain the general concepts and the implementations of Dijskra's Algorithm, Floyd-Warshall Algorithm, Bellman-Ford Algorithm, and Layer First Searching (LFS) Algorithm design.

1.4 Scope of the Study

In this research, four algorithms that were adequately reviewed were: Bellman Ford, Floyd Warshall, Dijkstra and Layer First Search. It was found that all of them are suitable algorithms for solving a single source shortest path problems in a polynomial time. However, due to universal acceptability of Dijkstra's algorithm in solving shortest path problems with weight costs $d_{ij} \geq O$, its data structure was modified, and the modified Dijkstra's algorithm used in this research prove to be more efficient than all the other algorithms. The modified Dijkstra's algorithm was used to investigate the shortest pathway and time in a wireless packet switch network system.

The focus of the study is to investigate the shortest pathway and time determination in a wireless packet switch network system. The scope of the study will comprise a network system of six nodes where each node denotes a switch. The universe of the nodes will include node 1, node 2, nodes 3, node 4, nodes 5 and node 6 located at different geographical sites within the University of Calabar heterogeneous environment. In this study, we shall design and analyse a complete wireless network system, run the system with some ping messages from source to destination nodes and determine the shortest distance and time it takes the message (packets) to traverse from the source node to the destination node. The protocols suit for the message in the network uses the Dijkstra's algorithm modified algorithm known as Open Shortest Path First (OSPF).

1.5 Significance of the Study

The University of Calabar does not have a network that links its various institutions together. Developing this network and an algorithm that will handle the routing of message will be of immense benefit to the institution. This design can be adopted and expanded to cover the entire organization. It can also serve as a basis for other institutions in implementing a network for their daily operations.

The following importance are achieved in optimizing the path and time for messages to traverse from source to destination node for example;

a) faster and timely communication of data among staff, student and other users who will be using the network;

b) the network would enhance the sharing of hardware and software system resources e.g. printers and data files, research materials, and more.

c) with the aid of this network system e-library and e-learning network system can be developed

d) Optimize the path and time for messages to traverse from source to destination node.

e) improve the efficiency of the network system, which will lead to improved quality of service (QoS) in the network.

1.6 Definition of terms

Shortest Path: A path between two nodes in a networking system such that the sum of the weights of its constituent edges is minimized.

Route: It is a single link between two nodes that are integral part of a larger network which are identified as tangible routes including roads and rails.

Routing information: Consists of the unit of update used by a given routing algorithm to communicate current link, node or path status information. This information is then used by the algorithm to compute routes and build the local routing table

Network operating system: A network operating system is a computer operating system that is designed primarily to support workstation, personal computer, and in some instances, older terminal that are connected on a local area network (LAN).

Packet switch: A packet switch is a node in a network which uses the packet-switching paradigm for data communication. Packet switches can operate at a number of different levels in a protocol although the exact technical details differ fundamentally. However, they all perform the same function; they store and forward packets.

Bandwidth: In computer networks, bandwidth is used as a synonym for data transfer rate, the amount of data that can be carried from one point to another in a given time period (usually a second). Network bandwidth is usually expressed in bits per second (bps); modern networks typically have speeds measured in the millions of bits per second (megabits per second, or Mbps) or billions of bits per second (gigabits per second, or Gbps). Also bandwidth as the range of frequencies -- the difference between the highest-frequency signal component and the lowest-frequency signal component -- an electronic signal uses on a given transmission medium. Like the frequency of a signal, bandwidth is measured in hertz (cycles per second). This is the original meaning of bandwidth, although it is now used primarily in discussions about cellular networks and the spectrum of frequencies that operators license from various governments for use in mobile services.

Throughput: Throughput is a measure of how many units of information a system can process in a given amount of time. It is applied broadly to systems ranging from various aspects of computer and network systems to organizations. Related measures of system productivity include , the speed with which some specific workload can be completed, and response time, the amount of time between a single interactive user request and receipt of the response.

Latency (Delay): The amount of time it takes a packet to travel from source to destination. And when a data packet is transmitted and returned back to its source, the total time for the round trip. Latency also refers to time interval or delay when a system component is waiting for another system component to do something.

Propagation time: is the amount of time it takes for the head of the signal to travel from the sender to the receiver. It can be computed as the ratio between the link length and the propagation speed over the specific medium.

Transmission time: is the amount of **time** from the beginning until the end of a message transmission.

Queuing time: is the time a job waits in a queue until it can be executed.

State of a node: This is the ordered pair of its distance value d_{ij} and its status label.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

There is a wealth of literature on variations of the shortest path problem however, despite such intense research, very few of the results beyond the classical algorithms of Dijkstra, Bellman-Ford, Floyd-Warshall, and min-Plus Matrix multiplication work with real-valued edge lengths using only comparisons and additions.

Previous experimental studies of shortest path algorithms focus on very restricted classes of inputs, where the edge lengths were assumed to be uniformly distributed, relatively small integers. This approach may be preferable for a specific application. However, any algorithm implemented for a more general use must be robust. By robust, it means that it makes no assumptions on the distribution of inputs, and minimal assumptions on the programming interface to the input (in the case of the shortest path problems this leads naturally to the comparison-addition model); the algorithm is the best robust SSSP and APSP algorithm for positively-weight sparse directed graphs.

As explained in the chapter one, chapter two provides a review of the extensive literature that exists in the area of routing algorithms which are responsible for forwarding the data packets over routes to provide optimal performance, as relevant to this research study.

In the area of network routing, OSPF routing protocol is required to maintain the status of all the routes in the network. A router runs a special routing algorithm to compute routes to all known destinations. A routing algorithm mainly consists of the following two parts - an initialization step and a recurring step that is repeated until the algorithm terminates. The recurring steps involve updating the minimum distance of each router for all destinations until the algorithm converges to correct shortest path distances. The routing algorithms differ in the way by which the updating step is implemented.

Earlier researchers have performed various operations on Dijkstra's algorithm to determine the shortest path between the nodes and had obtained good results in their research for the specified number of nodes. But the results were limited to the number of nodes fixed at the time.

Fuhao *et al.* (2016) introduced the classical Dijkstra algorithm in detail, and described the useful process of implementation of the algorithm and drawbacks of the algorithm: it describes the adjacent node algorithm which is a better optimization algorithm based on Dijkstra algorithm. This algorithm makes correlation with each node in the different network topology and information, and avoids the use of co-related matrix that contains huge infinite value, and making it more reliable and suitable analysis of the network for mass data. It is proved that this algorithm can save a lot of memory space and is more reliable to the network with huge nodes, but in this research they found that as node grew larger this approach gets slow in searching nodes. Liu and Chen (2016) used heap sort for unvisited nodes in geography network to improve the efficiency and reliability of Dijkstra algorithm but again it is necessary each time to arrange the heap(sorting) when node is inserted, thus making the process slow.

In Nikita *et al.* (2016) studies, introduced the Dijkstra algorithm in detail, and illustrated the disadvantage of implementation of the algorithm. They applied the algorithm on directed weighted graph to find shortest path between two nodes, they worked on non-negative nodes. Nikita and others also, discussed about how they can improve Dijkstra algorithm in order determining the shortest path according to weight by increasing some number of nodes. Most of the algorithms were modified to find out useful result using Dijkstra's algorithm. They named Thorup's algorithm, adjacent node algorithm, a heuristic genetic algorithm, augmented shortest path, and improved better version of the Dijkstra's algorithm and a graph partitioning based algorithm. But this algorithm was very complex to sort out the Dijkstra's problem (Ravi *et al.*, 2016).

In this research they used Critical Path Method (CPM) to find out critical activities on the critical path so that resources may be use in less time to find out the result. To find out the critical path, three parameters such as latest event time, earliest event time, and slack time for each of its activities are found. They modified Dijkstra's algorithm for critical path method to find latest event time, slack time, earliest event time for each of its activities in a project network (Charika *et al.*, 2016). They read out how to select a path with the minimum cost in terms of expected end-to end delay in a network. They worked on the transmission delay and queuing delay in buffer.

Paramita (2014) formulated the quantum algorithm for the Dijkastra's shortest path algorithm and introduced as Quantum Dijkastra algorithm (QDA) which gives fruitful result in quantum network and circuit design, which is first of its kind. Implementing QDA they obtained good result but again the major problem was whenever they inserted new node it must be optimized using QDA and they apply Dijkastra's shortest path.

2.2 Shortest Pathway, Time Dependent and Router Algorithm

2.2.1 Shortest pathway

In the Internet environment, the routers compute the flow transmissions according to the shortest path algorithm. This algorithm is efficient in finding optimal route, according to the link weights presenting the traffic load on them (Anupama and

Anuj, 2014). Furthermore, according to them, the limitation of this algorithm is that it cannot route the flow along alternative paths. In common network structure there are several paths between the source and destination nodes. Also, the Open Shortest Path First (OSPF) protocol routes according to the shortest path criteria, does not estimate and apply alternative routing to available paths. Thus Quality of Services (QoS) is not supported only by shortest path management. The optimal routing under (QoS) requirements is a complex problem for implementation (Dijkstra, 2012; Bellman-Ford, 2012). Such architecture insists routers to broadcast the local resource status and the local topology information to all routers. One manner of providing QoS in routers is to apply traffic prioritization. The idea is to classify the traffic to a multiple levels of priority queues. The priorities are assigned on packet peculiarities: the protocol uses packet type, source and destination networks. Enhancements are done by subdividing the link capacity into different classes. The traffic is assigned to each class and the routers serve each class with different priority.

Dijkstra, (2012) and Bellman-Ford, (2012) went further to say that, the traffic prioritization improves the QoS by class of traffic on a given link, but that link is chosen by the shortest path routing mechanism, which is independent of the QoS requirements. The optimal routing algorithm must keep the delays low as the flow control increases. The routing increases the throughput and restricts the delay for the packet, during high traffic conditions. The average delay per packet is reduced also at steady and low traffic conditions. Open Shortest Path First (OSPF) is a well known real-world implementation of Dijkstra algorithm used in network routing. In real networks, particularly in Ethernet networks, the Spanning-Tree Protocol (STP) runs on the network before the OSPF (Beaubrun and Pierre in Anupama and Anuj, 2014).

According to Beaubrun, Pierre in Anupama and Anuj (2014), in a general way, a spanning tree of a graph is a sub-graph which is also a tree that contains all the nodes. In other words, in a network environment, where redundant links are common, the STP causes these links to appear closed for the operation of the network elements, as to eliminate the appearance of duplicate messages, such as Neighbour discovery messages. Rings are a particular interesting class of topologies networks, designed to allow an additional level of connectivity for each node (there are now two possible paths to the destination node instead of one), with the cost of a single additional link. Rings are common elements in existing and planned networks, including European Optical Network (EON) and the NSF net. Short version of EON (known as termed COST 239) and the NSF net network, – in both figures, among others, several four node ring sub networks, can be detected, including Amsterdam, Berlin, Prague, Luxembourg for EON and Pittsburgh, Princeton, Boston and Ithaca for NFS net.

2.2.2 Time dependent

Time-dependent network is provided by Gao and Chabini (2006) who propose a two-dimensional taxonomy of STDSP problems according to arc cost dependency and information access. Uncertainty is a fundamental property of transportation networks which evolve continually due to varying travel demand, traffic capacity, and individual behaviour.

Over the past decades, the concept of strategic scenario based traffic assignment has emerged as a promising method to systematically incorporate these uncertainties into transport models (Hamdouch *et al*., 2004; Marcotte *et al*., 2004; Waller *et al*., 2013; Dixit *et al*., 2013). In strategic traffic assignment, the travel demand is modeled as a random variable which can take a finite number of values (scenarios). In the process each scenario corresponds to a representation of the network state. However, in a strategic traffic assignment, users minimize their expected travel time while recognizing the underlying probability distribution for the scenarios and the network travel times in each scenario. A strategic dynamic traffic assignment problem requires that users be able to find the Shortest Path (SP) in a time-dependent network across a set of stochastic demand scenarios.

The case where delay functions are continuous may potentially result in a non-polynomial complexity of the TDSP problem in First-In-First-Out (FIFO) networks. This result was recently proven for piecewise linear delay functions by Foschini *et al.* (2011) who settled a conjecture established by Dean (2004). In this context, Waller and Ziliaskopoulos (2002) showed that polynomial time recourse algorithms can be obtained in the case where arc costs exhibit a limited spatial temporal dependency.

Furthermore, Rey, Dixit, and Waller (2013), investigates the efficiency of routing algorithm in stochastic, time-dependent networks with respect to different probability structures on arc costs as well as different node waiting policies. The main contribution of stochastic is to characterize the single source time-dependent shortest path (SSTDSP), which is a variant of the STDSP problem where time-dependent arc costs are organized by scenarios, and each scenario has an associated probability of occurrence. The SSTDSP is considered as the least expected time-path across a finite set of stochastic scenarios, in which the link travel times are stochastically dependent. Notably, during departure, it is assumed that only one scenario can be realized. In this vein, there was a complete stochastic dependency of the link travel times in contrast to the more general STDSP, where the random variables representing the link travel times are assumed to be independent.

Results of findings confirm that the assumptions in the modeling of stochastic, time-dependent networks can have a significant impact on the computational efficiency of routing algorithms.

2.2.3 Router algorithm

The routing algorithm is described by Moy in Shree and Garcia-Luna-Aceves (2014) as Open Shortest Path First (OSPF) that relies on broadcasting complete topology information among routers and organizes the internet hierarchically to cope with the overhead incurred with topology broadcast. It also guides packets information stored as small strings of bits through the communication subset to their correct destinations. Reasons for the complexity of routing algorithms includes: coordination between the nodes in the network, failures of the links and nodes; congestion of traffic links. Two types of algorithms are used for routing in networks shortest path routing algorithms and Bellman-ford algorithm based on other measures. The efficiency of a routing algorithm depends on its performance, during congestions in the network. The routing algorithms must perform route choice and delivery of messages. The performance of the routing is assessed according to the throughput in the network quantity of data transfer and the average packet delay quality of service.

Route planners and associated features are increasingly popular among web users (Delling, 2009). Several web sites provide easy-to-use interfaces that allow users to select a starting and a destination point on a map, and a path between the two points satisfying one or more criteria is computed. The possible criteria are, for example minimize travel time, total path length or estimated travel cost. Similar capabilities are found in Global Positioning System (GPS) devices and as these usually have a limited amount of memory and Central Processing Unit (CPU) power, several devices now use different kinds of wireless connections in order to query a web service, which computes the desired path using more sophisticated algorithms than those available on the portable device (Delling, 2009).

Goldberg and Harrelson (2005) and Ikeda *et al.* (2004), users are typically interested in the *fastest* path to reach their destination, such as the shortest path in terms of travel time. However, only static information is taken into account when computing this kind of shortest paths, while it is well known that the travel time over a road segment depends on its congestion level, which in turn is dependent on the time instant at which the road segment is traversed (Goldberg and Kaplan, 2008). This implicitly requires complete knowledge of both real-time and forecast traffic information over the whole road network, so that we are able to compute the traversal time of a road segment for each time instant in the future. Apparently, assumption is obviously unrealistic; nevertheless, several statistical models exist which are able to predict to a certain degree of accuracy the evolution of traffic (Schultes, 2005). This type of analysis is made possible by traffic sensors (electromagnetic loops, cams). such development are positioned at strategic places of the road network and constantly monitor the traffic situation, providing both high-level information such as the congestion level of a highway and low-level information such as the travel time in seconds over a particular road segment (Sanders and Schultes, 2005).

Pyrga *et al.* (2005) opined that, using a large database of historical traffic information and statistical analysis tools *speed profiles* can be computed for the different road segments, including cost functions that associate the most probable travel speed (and thus travel time) over a road segment with the time instant at which the segment is traversed. However, Mohring *et al.* (2005), typically put that, there are several classes of these speed profiles, one class of profiles for weekdays and another one for holidays. A road network such that the travel time over a road segment depends on the time instant at which the segment is traversed is called *time-dependent* (Nannicini *et al.,* 2008). One practical problem arises: as road networks increases, traffic sensors cannot cover all road segments. Furthermore, Nannicini *et al.* (2008) that in real-world scenarios, only a small part of the road network is constantly monitored, while the remaining part is uncover due to speed profiles. The monitored part of the road network corresponds to the most important road segments, (1) freeways (2) intersections (3) highways. For long distance paths, the traffic congestion status of these segments is the most important for determining the total travel time, and is also the most significant from a user's point of view: it is reasonable to assume that a car driver who asks for the fastest path to reach the destination wants to avoid traffic jams on highways and freeways impact roads, which have a large influence on the total travel time, and to avoid congestions at local level near the departure the destination point are less important, as well as more difficult (if not impossible) (Liberti *et al.*, 2008) to foresee. In a realistic situation, only a part of the road network is provided with real-time and forecast traffic information, while the remaining part is associated with static travel times.

According to Lavor *et al.* (2006), this scenario further need to be examined due to the fact that the speed profiles are not accurate traffic information available. Indeed, it is clear that dynamic network information, as detected by the traffic sensors, gives the best estimation of travel times for the time instant at which it is gathered. Several predictive models for short and mid-term traffic forecasting exist, which are beyond the scope of this work and will not be discussed here; these models are based on the real-time information and capitalize on the temporal and spatial locality of traffic jams, so that they are able to predict congestions with a larger degree of accuracy with respect to speed profiles, which only take into account historical data

(Kerner, 2004; Hansen *et al.*, 2006). Naturally, the historical speed profiles are not the only source of traffic information and tend to provide a good estimation of long term traffic dynamics, but for short and mid-term forecasting more accurate dynamic data is available. Therefore, the verge process that associate travel times to road segments and the time at which the segment is traversed should ideally be *dynamic*, should be based on historical speed profiles, but they should be frequently updated in order to take into account both real-time traffic information and short and mid-term traffic forecastings. It is contended that the time required for each shortest path computation is much shorter than the time interval at which real-time traffic information traffic forecastings. Cornuejols *et al.* (2008) stated that updated must be on a duly bases so that computation can always be carried out before the cost functions are modified. This is realistic in industrial applications, since a shortest path should be computed very quickly no more than a second, whereas traffic information is typically updated every few minutes (Cornuejols *et al.*, 2008).

2.2.3.1 Common problems in routing network

1. The total path for delivering the packet is not defined in advance, rather each node decides which line to use in forwarding the packet to the next node.
2. Also, an instantaneous measurement of queue length does not accurately predict the average delay because there is a significant real time fluctuation in queue lengths at any traffic level. Certain variation may occur due to the high average delay of packet on Central Processing Unit (CPU).

According to Beaubrun and Pierre in Anupama and Anuj (2014), the three defects are reflection of a single point, namely that the length of an output queue is only one of many factors that affect a packets delay. The above mentioned routing algorithms is use for the betterment of using fewer network resources, operates on more realistic estimates of networks conditions, reacts faster to important network changes and does not suffer for long term loops and oscillations.

2.2.4 Congestion

Network management and control is a complex problem, which is becoming even more difficult with the increased demand to use the Internet for time/delay-sensitive applications with differing Quality of Service (QoS) requirements for example, voice over IP, video streaming, Peer-to-Peer, interactive games (Andreas, 2006).

Notably, there is a limit to how much control can be accomplished from the edges of the network of such an end-to-end implicit feedback based congestion control. Some additional mechanisms are needed particularly in the routers to complement the endpoint congestion control methods. Hence, the need for router control has recently led to the concept of active queue management. The problem of network congestion control remains a critical issue and a high priority; despite the many years of research efforts and the large number of different control schemes proposed, there are still no universally acceptable congestion control solutions.

The Internet Congestion Control Research Group (ICCRG) report of (2006) noted that congestion is a complex process to define. Despite the many years of research efforts in congestion control, currently there is no agreed definition. One may refer to the ongoing discussion between the active members of the networking community as to give the right definition for congestion.

The status of network congestion is a state of degraded performance from the perspective of a particular user. A network is reported to be congested from the perspective of a user if that user's utility has decreased due to an increase in network load. The user experiences long delays in the delivery of data, perhaps with heavy losses caused by buffer overflows. There is degradation in the quality of the delivered service, with the need for retransmissions, there is a drop in the throughput, which traffic is due to retransmissions in that state not much useful traffic is carried. In the region of congestion, queue lengths, hence queuing delays, grow at a rapid pace-much faster than when the network is not heavily loaded (Keshav 2001).

Pitsillides and Sekercioglu (2000) defined network-centric congestion, as a network state in which performance degrades due to the saturation of network resources, such as communication links, processor cycles, and memory buffers. For example, if a communication link delivers packets to a queue at a higher rate than the service rate of the queue, then the size of the queue will grow. If the queue space is finite then in addition to the delay experienced by the packets until service, losses will also occur. It is observed that congestion is not a state resource shortage problem, but rather a dynamic resource allocation problem.

Networks need to serve all users requests, which may be unpredictable and bursty in behaviour. However, network resources are finite, and must be managed and distributed among all users. Congestion will occur, if the resources are not managed effectively. The optimal control of networks queues are a well-known, studied, and notoriously difficult problem, even for the simplest of case (Hassan and Sirisena, 2001; Andrews and Slivkins, 2006).

2.2.4.1 Congestion control

The aim of congestion control is to facilitate incoming and outgoing traffic entry into a telecommunications network to avoid congestive collapse. The process is in an attempting to avoid oversubscription of any of the processing link capabilities of the intermediate nodes and networks and taking resource reducing steps, such as the rate of sending packets. It should not be confused with flow control, which prevents the sender from overwhelming receivers. Congestion control is a critical issue in the Internet Protocol (IP) networks systems. The majority of research proposals can be found in the literature to provide means of avoiding and controlling the congestion.

2.2.4.2 Modern theory of congestion control

The modern theory of congestion control was pioneered by Frank Kelly (1979), who applied microeconomic theory and convex optimization theory to describe how individuals controlling their own rates can interact to achieve an "optimal" network-wide rate allocation. Examples of "optimal" rate allocation are max-min fair allocation and Kelly's suggestion of proportionally fair allocation, although many others are possible.

The mathematical expression for optimal rate allocation is as follows. Let x_i be the rate of flow i, C_l be the capacity of link l, and r_{li} be 1 if flow i uses link l and 0 otherwise. Let x, $c$and R be the corresponding vectors and matrix. Let $U(x)$ be an increasing, strictly concave function, known as the utility, which measures how much benefit a user obtains by transmitting at rate x. The optimal rate allocation then satisfies

$$\max_{x} \sum_{i} U(x_i)$$

such that $Rx \leq c$

The Lagrange dual of this problem decouples, so that each flow sets its own rate, based only on a "price" signalled by the network. Each link capacity imposes a constraint, which gives rise to a Lagrange multiplier, p_l. The sum of these Lagrange multipliers, $y_i = \sum_{l} p_l r_{li}$, is the price to which the flow responds.

Congestion control then becomes a distributed optimisation algorithm for solving the above problem. The majority of current congestion control algorithms and modelled in this framework, with p_l being either the loss probability or the queueing delay at link l.

2.2.4.3 Congestion control principles

Delling (2009) classify most congestion control approaches into two categories: approaches for congestion avoidance and approaches for congestion recovery. Congestion avoidance mechanisms allow a network to operate in the optimal region of low delay and high throughput, thus, preventing the network from becoming congested. In contrast, the congestion recovery mechanism allows the network to recover from the congested state of high delay and losses, and low throughput. Even if a network adopts a strategy of congestion avoidance, congestion recovery schemes would still be required to retain throughput in the case of abrupt changes in a network that may cause congestion.

Both types of approaches are basically resource management problems. They can be formulated as system control problems, in which the system senses its state and feeds this back to its users who adjust their control (Chiu and Jain, 1989). This simple classification only provides a very general picture of common properties between separating groups of approaches.

A number of taxonomies of congestion control were considered. A detailed taxonomy for congestion control algorithms is proposed by Bellman-Ford (2012), which focuses on the decision-making process of individual congestion control algorithms. The main categories introduced by the Bellman-Ford (2012) taxonomy are open loop and closed loop:

Open loop: These are the mechanisms in which the control decisions of algorithms do not depend on any sort of feedback information from the congested spots in the network, that is, they do not monitor the state of the network dynamically.

Closed loop: These are the mechanisms that make their control decisions based on some sort of feedback information to the sources. With the provision of feedback, these mechanisms are able to monitor the network performance dynamically. The feedback involved may be implicit or explicit. In the explicit feedback scheme, feedbacks have to be sent explicitly as separate packets (Ramakrishnan, Floyd, and Black, 2001).

If there is no necessity of sending the feedback explicitly, the scheme is an implicit feedback scheme. Some examples of such implicit feedbacks are time delays of acknowledgment timeouts, and packet loss (Jacobson, 1988; Stevens, 1997).

The feedbacks are further categorised into binary and "full" feedback. A single bit in the packet header is used as a binary feedback mechanism (Ramakrishnan, Floyd, and Black, 2001). "Full" feedback incorporates more than one bit in the packet header and to send "full" information about the status of the network also known as the round-trip time (Katabi, Handley, and Rohrs, 2002) and explicit multi-bit "full" feedback scheme.

2.3 Shortest Pathway Algorithm for High Time Complexity and Low Accuracy

It is over fifteen years since Keshan (2001) outlined the need for a scientific basis of software measurement. Fenton theory is a prerequisite for any useful quantitative approach to software engineering, although little attention has been received from practitioners and researchers. Measurement is the process that assigns numbers and symbols to attributes of real-world entities. Naturally, empirical studies of software measurements lack a forecast system that combines measurements and parameters in order to make quantitative predictions.

In one accord, Wei *et al.* (2010), presented a new approach to software engineering based on recent advances in complex networks. Furthermore, parameter global statistics provide explanation of the phrase betweenness centrality to fulfill the needs of researchers, the inter-reaction of network from the overall perspective. Consequently, betweenness centrality according to them, is the times of a node being traveled in all the shortest paths of the software network and it reflects the influence of the node in the whole network software system.

2.3.1 Traditional dijkstra algorithm

The complexity of betweenness centrality comes from calculating the shortest path between each two nodes in network (Wei *et al.*, 2010), and the time complexity of Dijkstra is O(n3). The existing algorithms based on the shortest path contain Dijkstra, Floyd-Warshall, and Johnson. Dijkstra is the most popular and classic algorithm.

However, they opined that, the idea of Dijkstra is to abstract the network into a graph, put the isolated nodes the nodes with out-degree and in-degree being both 0 in set V and the nodes having been traveled in set S, then calculate the shortest path from vi to any node in the graph. Move the node vk with the shortest path from V-S to S till V-S is empty.

2.3.2 Traditional Layer First Searching (LFS) algorithm

According to Wei *et al.* (2010), Dijkstra it has a three-cycle and find out the shortest path between each two nodes which, then add 1 to between centrality of the nodes being on the path. The application range is limited to its high time complexity and the time consumption is unavailable when it is applied into large network of thousands of nodes.

As identified by researchers such as Fortz and Thorup (2002) Retvari and Cinkler (2004) Soltani *et al.* (2002), the length of shortest path between each two nodes wouldn't exceed a constant. For example, Pioro *et al.* (2002) noted that the average length is 15.21. Based on this method, Layer First Searching (LFS) is proposed. Starting from a node in the network, put the connected nodes with the shortest path of 1 into array, and then put that of 2 into the array and so on till that of n in the array but there's no connected node with shortest path of n+1. Add 1 to the nodes on the paths which just have been found. The time complexity of LFS is the summation of length of all the shortest path between each two nodes, that is O(V2). Compared with Dijkstra, the time complexity of LFS is reduced obviously.

The preparation of layer first searching is similar to Dijkstra: abstract the network into a graph, set up an adjacent list which makes single-link lists for all non-isolated nodes in the graph and the i-list contains the nodes directly connected to the non-isolated node vi. Each node is composed by two parts: neighboring nodes field adjvex and linking field nextarc. The neighboring nodes field marks where the nodes connected to node vi are in the graph and the linking field marks the next node. Each single-link has a head node which is composed of data field data and linking field (firstarc). Data field marks the number of vi in the graph and liking filed marks the first node that is connected to vi (Wei *et al.*, 2010). LFS only has one-cycle, and travels nodes in the net-work one by one, then find out the shortest path from starting node to the other nodes. The time complexity of LFS is O(V2) and the space complexity is T(V2). De-pending on the process stated above, LFS has great advantages both in time complexity and in space complexity.

Consequently, Wei *et al.* (2010) asserted that, betweenness centrality helps researcher to master the changes of the system from the overall perspective in software network. The existing betweenness centrality algorithm has high time complexity, but low accuracy. Therefore, Layer First Searching (LFS) algorithm is proposed that is low in time complexity and high in accuracy. LFS algorithm searches the nodes with the shortest to the designated node, then travels all paths and calculates the nodes on the paths, at last get the times of each node being traveled which is betweenness centrality. The time complexity of LFS algorithm is O(V2).

2.4 Description and Evaluation of Shortest Path Algorithms

According to Kairanbay and Hajar (2013), the shortest path problem is a process of finding the shortest path or route from a starting point to a final destination. Generally, in order to represent the shortest path problem graphs is used to illustrate the process. It was further demonstrated that, a graph is a mathematical abstract object, which contains sets of vertices and edges. Edges connect pairs of vertices. Along the edges of a graph it is possible to walk by moving from one vertex to other vertices. Depending on whether not one can walk along the edges by both sides and by only one side determines if the graph is a directed graph or an undirected graph. In addition, lengths of edges are often called weights, and the weights are normally used for calculating the shortest path from one point to another point. In the real world, it is possible to apply the graph theory to different types of scenarios. For example, in order to represent a map a graph can be used, where vertices represent cities and edges represent routes that connect the cities. If routes are one-way, then, the graph will be directed; otherwise, it will be undirected. There exist different types of algorithms that solve the shortest path problem.

In a study conducted by Li, Qi, and Ruan (2008), an efficient algorithm named Li-Qi (LQ) was proposed for the single source shortest path (SSSP) problem with the objective of finding a simple path of the smallest total weights from a specific initial or source vertex to every other vertex within the graph. The ideas of the queue and the relaxation form the basis of this newly introduced algorithm; the vertices may be queued several times, only the source vertex and relaxed vertices are being queued (Li, Qi, and Ruan, 2008).

2.4.1 Dijkstra algorithm

Dijkstra's algorithm is a procedure for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later (Frana, 2010; David, 2012).

According to David (2012), algorithm exists in many variants. The original variant found the shortest path between two nodes, and more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest path tree. For a given source node in the graph, the algorithm finds the shortest path between that node and every other nodes (Melhorn *et al.*, 2008). It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. If the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path algorithm is widely used in network routing protocols, here Intermediate System to Intermediate System (IS-IS) and Open Shortest Path First (OSPF). It is also employed as a subroutine in other algorithms such as Johnson's algorithm.

Algorithm

Let the node at the starting be known as the initial node. Let the distance of node *Y* be the distance from the initial node to *Y*. Dijkstra's algorithm will assign initial distance values and will try to improve step by step.

Assign to every node a tentative distance value and set it to zero for the initial node and to infinity for all other nodes.

a) Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.

b) For the current node, consider all of its unvisited neighbors and calculate the *tentative* distances. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. If the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* (through *A*) will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

c) Upon completion considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.

d) If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has completed.

e) Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", repeat step 3.

Description

Attempt to locate the *shortest path* between two intersections on a city map: a *starting point* and a *destination*. Dijkstra's algorithm initially marks the distance from the starting point to every other intersection on the map with *infinity*. This is done not to imply there is an infinite distance, but to note that those intersections have not yet been visited; some variants of this method simply leave the intersections' distances *unlabeled*. Each iteration, select the *current intersection*. For the first iteration, the current intersection will be the starting point, and the distance to it the intersection's label will be *zero*. For subsequent iterations after the first, the current intersection will be the *closest unvisited intersection* to the starting point this will be easy to find.

From the current intersection, *update* the distance to every unvisited intersection that is directly connected to it. This is done by determining the *sum* of the distance between an unvisited intersection and the value of the current intersection, and relabeling the unvisited intersection with this value the sum, if it is less than its current value. In effect, the intersection is relabeled if the path to it through the current intersection is shorter than the previously known paths. To facilitate shortest path identification, use pencil to mark the road with an arrow pointing to the relabeled intersection if you label/relabel it, and erase all others pointing to it. After you have updated the distances to each neighboring intersection, mark the current intersection as *visited*, and select the unvisited intersection with lowest distance from the starting point – or the lowest label—as the current intersection. Nodes marked as visited are labeled with the shortest path from the starting point to it and will not be revisited or returned.

Continue this process of updating the neighboring intersections with the shortest distances, then marking the current intersection as visited and moving onto the closest unvisited intersection until you have marked the destination as visited. Once you have marked the destination as visited as is the case with any visited intersection you have determined the shortest path to it, from the starting point, and can *trace your way back, following the arrows in reverse*; in the algorithm's implementations, this is usually done after the algorithm has reached the destination node by following the nodes' parents from the destination node up to the starting node; that's why we keep also track of each node's parent.

This algorithm makes no attempt to direct "exploration" towards the destination as one might expect. Rather, the sole consideration in determining the next "current" intersection is its distance from the starting point. This algorithm thereby expands outward from the starting point, interactively considering every node that is closer in terms of shortest path distance until it reaches the destination. When understood in this way, it is clear how the algorithm necessarily finds the shortest path. However, it may also reveal one of the algorithm's weaknesses: its relative slowness in some topologies (https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Algorithm).

Rule

The algorithm performs several rules as stated by Dijkstra (2010):

Rule1: A graph of the network is built network and the adjacency matrix a [i, j] with the weight of links is defined. For the case when a direct link between node Vi and Vj is missing, the weight of the link is assumed as infinity. The source and the destination nodes are noted as NS and NT.

Rule2: A status record set is established for every node with three fields:

The first field that shows the previous node, named "predecessor" field. The second filed is named "Length" field and it shows the sum of weights from source to that node. The last field named "Label" filed, shows the status of the node. Each node can have one status mode: "Permanent" or "Tentative".

Rule3: Initialization of the status record set for all nodes and setting all "Length" to Infinity, and all "Label" as tentative.

Rule 4: Labelling node NS as t node and marking its "Label" as "Permanent". When a label changes to permanent, it never changes again. T node rules as a current chosen node.

Rule5: For all tentative nodes, directly linked to t node, status record set is updated.

Rule6: From all the tentative nodes, choose the one whose weight to NS is less and set it as t node.

Rule7: If this node is not the destination NT, then, go to step 5.

Rule8: If this node is NT, then extract its previous node from status record set and do this until return to NS. The nodes show the best route from NS to NV

Notation:

Di = Length of shortest path from node 'i' to node 1.

di,j = Length of path between nodes i and j .

Algorithm

Each node j is labelled with Dj, which is an estimate of cost of path from node j to node 1. Initially, let the estimates be infinity, indicating that nothing is known about the paths. We now iterate on the length of paths, each time revising our estimate to lower values, as we obtain them. Actually, we divide the nodes into two groups ; the first one, called set P contains the nodes whose shortest distances have been found, and the other Q containing all the remaining nodes. Initially P contains only the node 1. At each step, we select the node that has minimum cost path to node 1.

This node is transferred to set P. At the first step, this corresponds to shifting the node closest to 1 in P. Its minimum cost to node 1 is now known. At the next step, select the next closest node from set Q and update the labels corresponding to each node using :

Dj = min [Dj , Di + dj,i]

Finally, after N-1 iterations, the shortest paths for all nodes are known, and the algorithm terminates.

Traditional method

The complexity of betweenness centrality comes from calculating the shortest path between each two nodes in network, and the time complexity of Dijkstra is O(n3). The existing algorithms based on the shortest path contain Dijkstra, Floyd-Warshall, and Johnson. Dijkstra is the most popular and classic algorithm.

The idea of Dijkstra is like this. Abstract the network into a graph, Put the isolated nodes (the nodes with out-degree and in-degree being both 0) in set V and the nodes having been traveled in set S, then calculate the shortest path from vi to any node in the graph. Move the node vk with the shortest path from V-S to S till V-S is empty.

① Initialization: Set up a two-dimensional array a to mark whether the shortest path has been found out between the two nodes. Set up a none-dimensional array to store the betweenness centrality. Set up a adjacency ma-trix arcs with 1 if there exist edge between the nodes, else with ∞. V is a set of all nodes and S is a set of all marked nodes. The value from vi to vj is initializes as follows:

D[j]=arcs[vi][j] vj $\in$V

Then put vi into S.

② Pick up vk which satisfies

D[k]=min{D[j] | vj∈V-S}

vk is the end point of the shortest path starting from vi. Put vk into S.

③ Calculate the length of the shortest path from vi to each accessible node in set V-S

D[k]+arcs[k][m]<D[m]

and revalue the D[m] as

D[k]+arcs[k][m]=D[m]

④ Add 1 to betweenness centrality of nodes if only they are on the shortest path from vi to any node in the graph. Then mark these nodes being travelled in the two dimensional array a. Repeat □, □ n-1 times. At last, it gets all the shortest paths from vi to the other nodes in the graph.

⑤ It is the end.

Repeat the process for n times and get the shortest path between each two nodes. Since each time contains a two-cycle, the time complexity of Dijkstra is O(V3). The space complexity is T(V2).

2.4.2 Floyd-Warshall algorithm

In computer science, the Floyd–Warshall algorithm is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles).

According to Kenneth (2003), a single execution of the algorithm will find the lengths (summed weights) of the shortest paths between *all* pairs of vertices, though it does not return details of the paths themselves. Versions of the algorithm can also be used for finding the transitive closure of a relation R, or (in connection with the Schulze voting system) widest paths between all pairs of vertices in a weighted graph.

Algorithm

The Floyd–Warshall algorithm compares all possible paths through the graph between each pair of vertices. It is able to do this with $\Theta(|V|^3)$ comparisons in a graph. This is remarkable considering that there may be up to $\Omega(|V|^2)$ edges in the graph, and every combination of edges is tested. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is optimal.

Consider a graph G with vertices V numbered 1 through N. Further consider a function shortestPath(i, j, k) that returns the shortest possible path from i to j using vertices only from the set $\{1,2,...,k\}$ as intermediate points along the way. Now, given this function, our goal is to find the shortest path from each i to each j using only vertices 1 to $k + 1$.

For each of these pairs of vertices, the true shortest path could be either

(1) a path that only uses vertices in the set $\{1, ..., k\}$

or

(2) a path that goes from i to $k + 1$ and then from $k + 1$ to j.

We know that the best path from i to j that only uses vertices 1 through k is defined by shortestPath(i, j, k), and it is clear that if there were a better path from i to $k + 1$ to j, then the length of this path would be the concatenation of the shortest path from i to $k + 1$ (using vertices in $\{1, ..., k\}$) and the shortest path from $\{k + 1\}$ to j (also using vertices in $\{1, ..., k\}$).

If $w(i, j)$ is the weight of the edge between vertices i and j, we can define shortestPath($i, j, k + 1$) in terms of the following recursive formula: the base case is

According Vaibhavi and Chitra (2014), Bellman-Ford algorithm uses relaxation to find single source shortest paths on directed graphs. And it is also contain negative edges. The algorithm will also detect if there are any negative weight cycles (such that there is no solution). With specific reference to distances on a map, there is no any negative distance. The basic structure of Bellman-Ford algorithm is similar to Dijkstra algorithm. It relaxes all the edges, and does this |V| - 1 time, where |V| is the number of vertices in the graph (David, 2012). The cost of a path is the sum of edge weights in the path. This algorithm return value that negative cycle is present or not and also return shortest path. This algorithm find shortest path in bottom up manner (Greeks for Greeks, 2011).

Algorithm

Like Dijkstra's Algorithm, Bellman–Ford is based on the principle of relaxation, in which an approximation to the correct distance is gradually replaced by more accurate values until eventually reaching the optimum solution. In both algorithms, the approximate distance to each vertex is always an overestimate of the true distance, and is replaced by the minimum of its old value with the length of a newly found path. However, Dijkstra's algorithm greedily selects the minimum-weight node that has not yet been processed, and performs this relaxation process on all of its outgoing edges; by contrast, the Bellman–Ford algorithm simply relaxes *all* the edges, and does this |V| - 1 times, where |V| is the number of vertices in the graph. In each of these repetitions, the number of vertices with correctly calculated distances grows, from which it follows that eventually all vertices will have their correct distances. This method allows the Bellman–Ford algorithm to be applied to a wider class of inputs than Dijkstra. Bellman–Ford runs in $O(|V| \cdot |E|)$ time, where $|V|$ and $|E|$ are the number of vertices and edges respectively.

In comparison to Dijkstra's algorithm, the Bellman-Ford algorithm acknowledges the (Bellman-Ford, 2012) edges with negative weights. According to his studies revealed that a graph can contain cycles of negative weights, which will generate numerous number of paths from the starting point to the final destination, where each cycle will minimize the length of the shortest path. Taking into consideration, this fact let's assume that our graph does not contain cycles with negative weights .The array d[b] will store the minimal length from the starting points to other vertices. The algorithm consists of several phases, where in each phase it needs to minimize the value of all edges by replacing d[b] to following statement d[a] + c; a and bare vertices of the graph, and c is the corresponding edge that connects them. This algorithm iterates on the number of edges in a path to obtain the shortest path. Since the number of hops possible is limited (cycles are implicitly not allowed), the algorithm terminates giving the shortest path.

Algorithm :

```
Initial condition :    D[ i, 0] = infinity, for all i ( i !=1 )
Iteration :            D[i, h+1] = min { di,j + D[j,h] }
over all values of j
Termination :          The algorithm terminates when
D[i, h] = D [ i, h+1] for all i .
```

2.4.4 Layer First Searching algorithm

It is over fifteen years since Norman Fenton outlined the need for a scientific basis of software measurement. Such a theory is a prerequisite for any useful complexity and high time in accuracy. LFS algorithm searches the nodes with the shortest to the designated node, then travels through all paths and calculates nodes on the paths and at last, get the times of each node being traveled which is betweenness centrality. The time complexity of LFS algorithm is O(V2).

Algorithm

Accordingly, Dijkstra, it has a three-cycle and with ability to find out the shortest path between each two nodes which, then add 1 to between centrality of the nodes being on the path. However, application range is limited for its high time complexity and the time consumption is unavailable when it is applied into large network of thousands of nodes.

Fortz and Thorup (2002), Retvari and Cinkler, (2004), and Soltami *et al.* (2002) stated that the length of shortest path between each two nodes wouldn't exceed a constant. For example, Pioro *et al.* (2002) noted that the average of the length is 15.21.which lead to the integration of LFS Algorithm process. Starting from a node in the network, place the connected nodes with the shortest path of 1 into array, then place that of 2 into the array till that of n in the array inducting that, there is no connected node with shortest path of n+1. Add 1 to the nodes on the paths that has just have been found. The time complexity of LFS is the summation of length of all the shortest path between each two nodes, that is O (V2). Compared with Dijkstra, the time complexity of LFS is reduced obviously.

The preparation of LFS algorithm is similar to Dijkstra: abstract the network into a graph, set up an adjacent list which makes single-link lists for all non-isolated nodes in the graph and the i-list contains the nodes directly connected to the non-isolated node vi. Each node is composed of two parts: neighboring nodes field (adjvex) and linking field (nextarc). The neighboring nodes field marks where the nodes connected to node vi are in the graph and the linking field marks the next node. Each single-link has a head node which is composed of data field (data) and linking field (firstarc). Data field marks the number of vi in the graph and liking filed marks the first node that is connected to vi.

① Initialization: Set up a two-dimensional array c initialized maximum to store the length of the shortest path between each two nodes in the graph, then set up a one-dimensional array bc initialized 0 to store the betweenness centrality of each node. V is the number of non-isolated nodes in the graph.

② Set up array a and b. a is used to restore the end node of the shortest path. b is used to restore the number of nodes with the same starting node and the same length and put the starting point into a and put la=0 into b. At last, set the relative element 0 in array c.

③ Judge whether it is the end of array a. If it is, turn to

⑧; else turn to④.

④ Check out the number of nodes with the length of shortest path la, and set it to be n.

⑤ Travel the next node in array a and the find out all child nodes which are connected to this node (parent node) directly. If the value from starting node to this node in array c is bigger than la, set the value from the starting node to this node and the value from this node to starting node in array c to be la+1, then put the id of parent node and the id of child node into a. The same nodes being put into an m times means that there are m shortest paths from the starting node to this node. Add 1 to num which is the number of nodes on layer la+1. Because the id of parent node can be found by the id of child node, the shortest paths from the starting node to the other nodes in array a can been found, then add 1 to the nodes on each shortest path.

⑥ repeat ⑤ n-1 times.

⑦ put num into array b, la=la+1, turn to □.

⑧ repeat ②-⑦ V-1 times.

⑨ It is the end.

According to the description above, the time complexity of LFS is the summation of all the shortest path in the network, that is O(V2). And V is the number of non-isolated nodes in the network. The space complexity is T(V2) which is equal to that of Dijkstra.

TABLE 1

The comparison between the time complexity and space complexity of the algorithms.

Algorithm	Time complexity	Space complexity
Dijkstra	0(3)	T(V2)
Layer First Searching	0(2)	T(V2)

Performance evaluations

Dijkstra takes breadth-first method to travels all the nodes in the software network, find out all the shortest paths, obtain the nodes on the shortest path, (Wang, 2001; Pioro and Medhi, 2004; Ben-Ameur and Gourdin, 2003) and calculate betweenness centrality. Dijkstra has a three- cycle which makes the time complexity is so high that it is a fatal shortcoming when applied into large-scale software network.

Layer First Searching (LFS) only has one-cycle, and travels nodes in the network one by one, then find out the shortest path from starting node to the other nodes. The time complexity of Layer First Searching is O(V2) and the space complexity is T(V2). Depending on what mentioned above, LFS has great advantages both in time complexity and in space complexity.

Accordingly Wei *et al.* (2010) says that, with the development of computer science, the computer memory becomes bigger and bigger which can satisfy all kinds of demands and no longer need to be considered. However, they get satisfying result with the help of HP computer which is composed of Core Duo 6300 CPU, 1.86GHz Frequency, DDR2 667 1GB Memory and Windows XP SP2 Operation System. Wei *et al.* (2010) went further to say that, in order to verify that LFS has great advantages in time complexity, a samples of twenty-two(22) software of different sizes are selected and sorted ascending which can justify whether LFS is valid. The comparison of time consumption between Dijkstra and Layer First Searching is shown as follows: DT is the time cost by calculating betweenness centrality of the software by Dijkstra, and WT is that by LFS. Time units are seconds. DT/WT which marks advantages of LFS in time consumption is the ratio of DT and WT.

TABLE 2

The comparison of time consumption between Dijkstra and LFS

software	number of nodes	Number of edges	number of non-isolated nodes	DT (s)	WT(s)	DT/WT
Waimea	116	193	86	0.359	0.032	11.22
Kicad	212	300	180	0.609	0.110	5.54
ktorrent	263	335	217	3.313	0.250	13.25
rhythmbox	366	342	252	8.349	0.531	15.72
Filezilla	431	577	358	5.500	0.563	9.77
licq	574	633	491	12.110	1.282	9.45
freemind	713	933	562	53.172	1.812	29.34
Espgs	1339	1271	955	150.094	7.063	21.25
abiword	1300	2124	1167	384.391	11.531	33.34
ArgoUML	2031	2217	1731	747.093	31.718	23.55
kdegraphics	2014	3498	1749	1036.781	44.688	23.20
Mysql-5.0.56	3132	3837	2182	1685.828	54.453	30.96
Mysql_6.0.6	3793	5368	2889	3131.318	104.531	29.96
Kdepim	3518	4447	3008	2933.594	136.047	21.56
Koffice	4580	5892	3883	4853.296	185.891	26.11
Linux	7343	6045	4238	6756.612	296.313	22.80
resin	5076	7875	4261	10613.281	389.072	27.28
node	5418	11451	5418	13693.187	553.985	24.72
Firefox	9261	15533	5781	18167.438	725.530	25.04
Firefox	7100	48236	7100	24267.391	942.793	25.74
Mozilla	8354	13878	7195	37298.863	1315.750	28.35
Firefox	10115	17469	8830	120818.089	3152.580	38.32

Source: Research on LFS Algorithm in Software Network (Wei, W. *et al*, 2010).

2.5 Conceptual Implementations of Described Shortest Path Algorithms

Dijkstra's Algorithm implementation

According to Kairanbay and Hajar (2013), for each vertex within a graph they assign a label that determines the minimal length from the starting point *s* to other vertices *v* of the graph. In a computer they went further, to declare an array *d[]*. Their algorithm works sequentially, and in each step it tries to decrease the value of the label of the vertices. However, Their algorithm stops when all vertices have been visited. The label at the starting point *s* is equal to zero (*d[s]=0*); however, labels in other vertices *v* are equal to *infinity* (*d[v]=∞*), which means that the length from the starting point *s* to other vertices is unknown. In a computer they used a very big number in order to represent *infinity*. Kairanbay and Hajar (2013) added that, for each vertex v^{th} have to identify whether it has been visited or not. In order to do that, they continue to declare an array of *Boolean* type called *u[v]*, where initially, all vertices are assigned as unvisited (*u[v] = false*).

The Dijkstra's algorithm consists of *n* iterations (Kairanbay and Hajar, 2013). If all vertices have been visited, then the algorithm finishes; otherwise, from the list of unvisited vertices they choose the vertex which has the minimum (smallest) value at its label (At the beginning, they choose a starting point *s*). After that, Kairanbay and Hajar (2013) considered all neighbors of this vertex (Neighbors of a vertex are those vertices that were having common edges with the initial vertex). For each unvisited neighbor, their consideration was on a new length, which is equal to the sum of the label's value at the initial vertex v (*d[v]*) and the length of edge *l* that connects them. If the resulting value is less than the value at the label, then they changed the value in that label with the newly obtained value (Jordan, 2005).

$d[\,neighbors\,] = min\,(\,d[\,neighbors\,],\, d[\,v\,] + l\,)$ (1)

Accordingly, after they considered all of the neighbors, Kairanbay and Hajar (2013) assign the initial vertex as visited (*u[v]* = *true*). After repeating this step *n* times, all vertices of the graph was visited and the algorithm finishes or terminates. The vertices that are not connected with the starting point was remained by being assigned to *infinity*. Consequently, in order for them to restore the shortest path from the starting point to other vertices, they need to identify array *p []*, where for each vertex, where $v \neq s$, the number of vertex *p[v]* was store, which penultimate vertices in the shortest path. In other words, a complete path from *s* to *v* was equal to the following statement (Chamero, 2006).

$P = (s, \ldots, p[p[p[v]]], p[p[v]], p[v], v)$ (2)

2.5.1 Floyd-Warshall algorithm implementation

According to Kairanbay and Hajar (2013) explanation the graph *G* were considered, where vertices were numbered from *1* to *n*. The notation *dijk* means the shortest path from *i* to *j*, which also passes through vertex *k*. Obviously if there was exists edge between vertices *i* and *j* it will be equal to *dij0*, otherwise it can assigned as *infinity*. However, for other values of *dijk* there can be two choices: (1) If the shortest path from *i* to *j* does not pass through the vertex *k* then value of *dijk* will be equal to *dijk-1*. (2) If the shortest path from *i* to *j* passes through the vertex *k* then first it goes from *i* to *k*, after that goes from *k* to *j*. In this case the value of *dijk* will be equal to *dikk-1* + *dkjk-1*. And in order to determine the shortest path we just need to find the minimum of these two statements (Shiana, 2014):

dij0 = the length of edge between vertices i and j (3)

dijk = min (dijk-1, dikk-1 + dkjk-1)

2.5.2 Bellman-Ford algorithm implementation

In comparison to Dijkstra's algorithm, the Bellman-Ford algorithm admits or acknowledges the edges with negative weights. That is why, a graph can contain cycles of negative weights, which will generate numerous number of paths the starting point to the final destination, where each cycle will minimize the length of the shortest path. Taking into consideration this fact let's assume that our graph does not contain cycles with negative weights. The array *d[]* will store the minimal length from the starting point *s* to other vertices. The algorithm consists of several phases, where in each phase it needs to minimize the value of all edges by replacing *d[b]* to following statement *d[a] + c*; *a* and *b* are vertices of the graph, and *c* is the corresponding edge that connects them. And in order to calculate the length of all shortest paths in a graph it requires $n - 1$ phases, but for those vertices of a graph that are unreachable, the value of elements of the array will remain by being assigned to *infinity* (Haugardy, 2010).

2.6 Review of Shortest Path Algorithms

Xi, Qi and Wei (2006) used a heuristic method for computing the shortest path from one point to another point within traffic networks. They proposed a "new dynamic direction restricted algorithm obtained by extending the Dijkstra's algorithm. Li, Qi and Ruan (2008) proposed an algorithm (Li-Qi) to solve the single source shortest path (SSSP) problem with the objective of finding a simple path of the smallest total weights from a specific initial or source vertex to every other vertex within the graph. The ideas of the queue and the relaxation form the basis of the algorithm. Layer First Searching (LFS) can solve a series of problems brought by the traditional algorithms of Dijkstra, Floyd-Warshall and Bellman-ford. It has advantages both in time complexity and accuracy which are so important in practical research work that may result in disaster conclusion without it. LFS improve the efficiency and the accuracy to calculate the betweenness centrality, which ensures the further research to be continued smoothly. Kairanbay and Jani (2013) then compared four (4) algorithms namely: Dijkstra, Floyd-Warshall, Bellman-Ford, LFS Algorithms by using pre-defined test cases and automated checking systems available in websites to find out that all could solve the SSSP at different time complexity.

TABLE 3

Algorithm of different time complexity

Algorithm	**Time complexity**
Dijkstra	$n^2 + m$
Bellman-Ford	n^3
Floyd-Warshall	Nm

The loop-free path-finding algorithm (LPA) has been shown to maintain loop-free routing tables. LPA obtains correct routing tables after topological and link-cost changes faster and with less processing and communication overhead than link-state algorithms and prior loop-free routing algorithms based on vectors of distances. The limitation of LPA and prior routing algorithms based on routing trees is that it requires the routers to maintain "host routes" than would be needed in a traditional distance-vector algorithm.

Routing information maintained at each router has to be updated frequently to adapt to changes in the topology and congestion of the internetwork. In an internetwork with a flat routing structure, the size of the routing tables grow linearly with the number of destinations in the network. Due to this, the routing information that is required to be maintained at a node may become excessive in terms of storage and CPU utilization. The information exchanged among nodes may prove to be expensive in terms of channel bandwidth since updates need to be exchanged frequently in order to maintain up to date network state information. Accordingly, aggregation of routing information becomes a necessity in any type of routing protocol.

For routing in large networks, the aggregation of routing information is achieved through a hierarchical partitioning of the network. The main idea of hierarchical routing is to maintain exact routing information regarding nodes very close to it and less detailed information regarding nodes that are farther away from it. The goal of maintaining hierarchy of information is to reduce the size of the routing database maintained at each router so that the exchange of topology information among the routers can be minimized. The objective of doing so is to obtain a reasonable compromise among the size of routing tables, number of updates required to maintain such tables and the speed with which updates are propagated.

However, in order to accomplish a workable system model for suitable system development, the researcher will present a new approach to routing algorithms which is call *Layer First Searching* (LFS), used in large system based on recent advances in complex networks. As a typical parameter and an important global statistics, betweenness centrality can meet he needs of researchers to know the inter-reaction of software network from overall perspective.

According to Wei *et al.* (2010) betweenness centrality is the times of a node being traveled in all the shortest paths of the software network and it reflects the influence of the node in the whole network. Betweenness centrality is an indicator of a node's centrality in a network. It is equal to the number of shortest paths from all vertices to all others that pass through that node. A node with high betweenness centrality has a large influence on the transfer of items through the network, under the assumption that item transfer follows the shortest paths. The concept finds wide application, including computer and social networks, biology, transport and scientific cooperation. Development of *betweenness centrality* is generally attributed to sociologist Linton Freeman (Barthélemy, 2004). The idea was earlier proposed by mathematician J. Anthonisse, but his work was never published (Newman, 2010).

In calculating betweenness and closeness centralities of all vertices in a graph, it is assumed that graphs are undirected and connected with the allowance of loops and multiple edges. When specifically dealing with network graphs, often graphs are without loops or multiple edges to maintain simple relationships (where edges represent connections between two people or vertices). In this case, using Brandes' algorithm will divide final centrality scores by 2 to account for each shortest path being counted twice (Ulrik, 2004).

Another algorithm generalizes the Freeman's betweenness computed on geodesics and Newman's betweenness computed on all paths, by introducing a hyper-parameter controlling the trade-off between exploration and exploitation. The time complexity is the number of edges times the number of nodes in the graph (Amin, 2010).

The concept of centrality was extended to a group level as well (Puzis, 2010). Group betweenness centrality shows the proportion of geodesics connecting pairs of non-group members that pass through a group of nodes. Brandes' algorithm for computing the betweenness centrality of all vertices was modified to compute the group betweenness centrality of one group of nodes with the same asymptotic running time (Puzis *et al.*, 2009). Betweenness is a centrality measure based on shortest paths, widely used in complex network analysis. It is computationally-expensive to exactly determine betweenness; currently the fastest-known algorithm by Brandes requires $O(nm)$ time for unweighted graphs and $O(nm + n2 \log n)$ time for weighted graphs, where n is the number of vertices and m is the number of edges in the network. These are also the worst-case time bounds for computing the betweenness score of a single vertex (David *et al.,* 2006).

Betweenness is also used as the primary routine in popular algorithms for clustering and community identification (David *et al.,* 2006) in real-world networks. For instance, the Girvan-Newman (Puzis *et al.*, 2009) algorithm iteratively partitions a network by identifying edges with high betweenness scores, removing them and recomputing centrality scores. Betweenness is a global centrality metric that is based on shortest-path enumeration. Consider a graph $G = (V;E)$, where V is the set of vertices representing *actors* or *nodes* in the complex network, and E, the set of edges representing the relationships between the vertices. The number of vertices and edges are denoted by n and m respectively. The graphs can be directed or undirected.

In prior work, we explored high performance computing techniques (Bader and Madduri, 2006) that exploit the typical small-world graph topology to speed up exact centrality computation. We designed novel parallel algorithms to exactly compute various centrality metrics, optimized for real-world networks. We also demonstrate the capability to compute exact betweenness on several large-scale networks (vertices and edges in the order of millions) from the Internet and social interaction data; these networks are three orders of magnitude larger than instances that can be processed by current social network analysis packages. *Fast centrality estimation* is thus an important problem, as a good approximation would be an acceptable alternative to exact scores. Currently the fastest exact algorithms for shortest path enumeration-based metrics require n shortest-path computations; however, it is possible to estimate centrality by extrapolating scores from a fewer number of path computations. Using a random sampling technique, Eppstein and Wang (2001) show that the closeness centrality of all vertices in a weighted, undirected graph can be approximated with high probability in $O($ log n^2 2 (n log n $+m$)) time, and an additive error of at most ^{2}¢G (^{2} is axed constant, and ¢G is the diameter of the graph). However, betweenness centrality scores are harder to estimate, and the quality of approximation is found to be dependent on the vertices from which the shortest path computations are initiated from (in this paper, we will refer to them as the set of *source vertices* for the approximation algorithm). Recently, Brandes and Pich (2007) presented centrality estimation heuristics, where they experimented with different strategies for selecting the source vertices. They observe that a random selection of source vertices is superior to deterministic strategies. In addition to exact parallel algorithms, we also discussed parallel techniques to compute approximate betweenness centrality in (Bader and Madduri, 2006), using a random source selection strategy.

Among several variants of the SP algorithms there is a group of algorithms which could be applied to solve the present issue, but the solution would not be efficient. An obvious group of algorithms is the one that gives a more general solution than needed and their solution would be redundant. A good example of a group giving a redundant solution is the 'all pairs' group of the SP algorithms: only one pair of nodes would be used from the set of all pairs.

Matrix algorithms are not of a good use for sparse networks. Matrix algorithms are memory consuming and for sparse networks time consuming. Bellman (2012) designed an algorithm for networks with also negative link lengths. The concern about negative link length made the Bellman algorithm more general than Dijkstra algorithm, but it also made it less efficient for the networks without negative links. Since the report deals with road networks (of which the inherent feature is the nonnegative link lengths) only, the study of the Bellman algorithm will not be of much interest for the report.

Another example of an algorithm providing a solution more general than needed is the algorithm devised by Cai *et al* (1997). This algorithm is an enhanced Dijkstra algorithm. The algorithm processes a network to whose links two attributes are ascribed: cost of traversing and time of traversing. The algorithm searches for the cheapest path (the shortest path in terms of the cost) which satisfies an extra condition: the overall time of such a path (the time required to traverse the links of the path) does not exceed a given T.

CHAPTER THREE
METHODOLOGY AND SYSTEM ANALYSIS

3.1 Introduction

This chapter starts with finding the shortest path for transmitting packets and critically understanding the existing models and algorithm of shortest pathways with a view to proposing a new algorithm that will be devoid of the identified limitations.

3.2 Research Design

According to De Vaus (2006), research design refers to the overall strategy designed to integrate the different components of the study in a coherent and logical way, thereby, ensuring effective executing of the research problem. In addition, it also constitutes the benchmark for the collection, measurement, and analysis of data.

The research design adopted for this research study was modified Dijkstra's algorithm with open shortest path first (OSPF) router protocol suite. Research problem determines the type of design use in a study. The modified Dijkstra's algorithm was used because its has high acceptability in terms of overall performance in solving shortest path problem in a single source, one-to-all sparse network system. While the OSPF provides quality of service (QoS) and high link utilization with minimal losses, and bounded queue fluctuations and delays.

The research design applied in this study is as follows:

a) Model formulation

b) Model specification

c) Model development

d) System analysis of the modified Dijkstra's algorithm model

e) Model validation

f) Algorithm formulation

g) Instrument/Algorithm development

h) Modified Dijkstra's algorithm

i) Reliability test

j) Network design

k) Experimental design

l) Evaluation

m) Network validation

3.2.1 Model formulation

Apparently, the research problem is to determine the shortest pathway and time in a wireless packet switch network system in the University of Calabar. To solve this problem and to give the best solution for increasing number of nodes, the researcher decided to replace the data structure with the priority queue link list having the capacity to store *N* nodes with some constraints containing the value of predecessor (the node preceding a current node) , status (a label, which is either temporary or permanent) and distance.

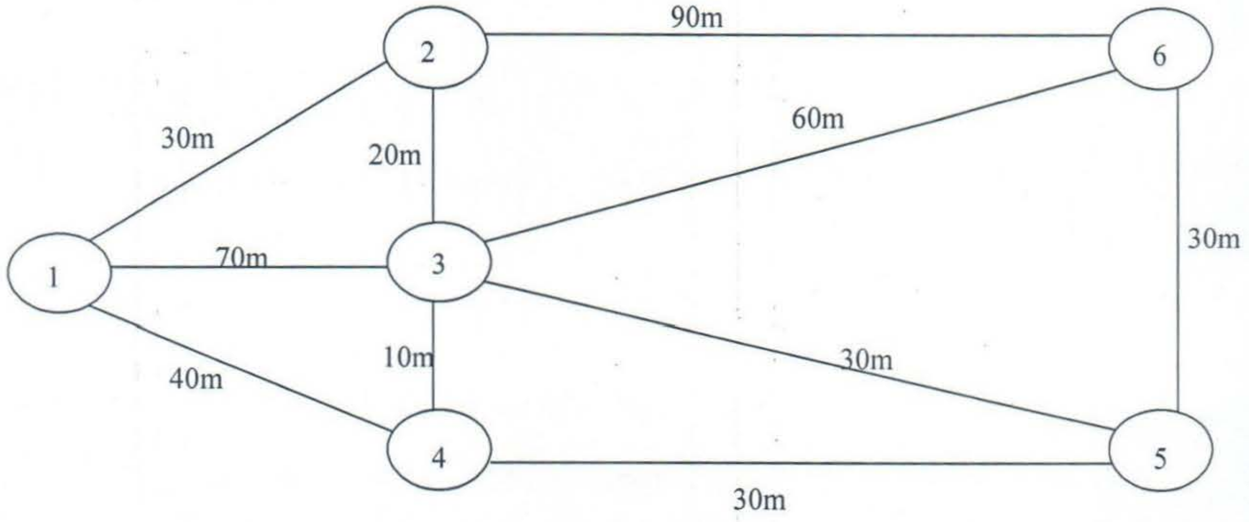
Problem: Given a graph $G = \{V, E\}$ where V is the set of nodes (or vertices) in G and E the set of edges in G, if $n = |V|$ is the number of vertices in G, then the size of the adjacency matrix required to store G in a computer memory is n x n (or n^2):

$$m = \begin{pmatrix} m_{11} & m_{12} & m_{13} \ldots m_{1n} \\ m_{21} & m_{22} & m_{23} \ldots m_{2n} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ m_{n1} & m_{n2} & m_{n3} & m_{nn} \end{pmatrix}$$

It is required to represent *G* in a computer memory with an adjacency matrix of size $N<n^2$.

Solution

Consider a graph *G* having $n = 6$ nodes or vertices as shown.



G requires a 6 x 6 (or 36) adjacency matrix to represent it in a computer memory. This representation is shown below.

$$m = \begin{pmatrix} 0 & 30 & 70 & 40 & 0 & 0 \\ 30 & 0 & 20 & 0 & 0 & 90 \\ 70 & 20 & 0 & 10 & 30 & 60 \\ 40 & 0 & 10 & 0 & 30 & 0 \\ 0 & 0 & 30 & 30 & 0 & 30 \\ 30 & 90 & 60 & 0 & 30 & 0 \end{pmatrix}$$

Notice that this adjacency matrix is a symmetric matrix that is $M^T = M$ thus, $m_{ij} = m_{ji}$ $\forall\, ij \in \{1,2,3,4,5,6\}$ and $m_{ij} = 0,\ i = j$. Hence, if all duplicate is remove, starting from $i = j$ downward, as follows;

30	70	40	0	0
20	0	0	90	
10	30	60		
30	0			
30				

Thus, the size of this new matrix is the sum of the elements in all the rows i.e.

$$5 + 4 + 3 + 2 + 1 = 15.$$

Then, in general,

$$n - 1 + n - 2 + n - 3 + \ldots 2 + 1$$

Hence, the size of the new matrix is

$$= {}^{1}/_{2}\,(n-1)\,(n)$$

and this agree with the preceding result if $n = 6$ is substituted.

Note: n is the number of nodes.

Clearly $N < n^2$.

Theorem: N is less than 50% of n^2 that is, $N < 50\%\ n^2$ or $N < \frac{n^2}{2}$

Proof:

Since the computer requires $n \times n = n^2$ elements for the storage in the memory and the number of elements of the new matrix is $\frac{n}{2}(n-1)$, the ratio

$$\frac{n(n-1)}{2}.\frac{1}{n2}$$

$$=\frac{(n-1)}{2n} = \frac{1}{2} - \frac{1}{2n} < \frac{1}{2}$$

⇨ the number is $n > 2$ less than 50%. i.e 36 memory core = 100%

⇨ 1memory core = ${}^{100}/_{36}$, :.

⇨ 15 memory core = ${}^{100}/_{36} \times {}^{15}/_{1} = 41.67\%$

⇨ Thus, % reduction = 100 – 41.67 = <u>58.33%</u>

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
30	70	40	0	0	20	0	0	90	10	30	60	30	0	30

Observe that each element has been isolated with a faint line while each row separated by thick line as shown above. But matrix m is a 2-D array m_{ij}, hence, we need to provide an interface for this new one dimensional array so as to make it appear multi-dimensional.

Row R_m and Column C_{ij}.

2-Dimensional array is divided into rows and columns. In the case of matrix *mij*, i^{th} = row and j^{th} = column numbers. Hence, we need to find a formula R_m for the row *m* and C_{ij} for the column *C*.

$r_1 = 1$	1st row begins at element 3
$r_2 = n - 1 + 1$	2nd row begins at element 2 => 6-1+1

separated by thick line as shown above. But matrix m is a 2-D array m_{ij}, hence, we need to provide an interface for this new one dimensional array so as to make it appear multi-dimensional.

Row R_m and Column C_{ij}.

2-Dimensional array is divided into rows and columns. In the case of matrix mij, i^{th} = row and j^{th} = column numbers. Hence, we need to find a formula R_m for the row m and C_{ij} for the column C.

$r_1 = 1$	1st row begins at element 3
$r_2 = n - 1 + 1$	2nd row begins at element 2 => 6-1+1
$r_3 = n - 1 + n - 2 + 1$	3rd row beings at element 1 => 6-1+6-2+1
$r_4 = n - 1 + n - 2 + n - 3 + 1$	4th row begins at element 3 => 6-1+6-2+6-3+1
⋮	⋮ " " " " " "

r_m = n-1+n-2+n-3+...+n-(m-1)+1

$$r_m = (m - 1)\, n - (2+3+4+ \ldots + m - 1) + 1$$

$$r_m = (m - 1)\, n - \left[\frac{(m-1)\,(m-2)}{2}\right] + 1$$

Thus,

$$r_m = \frac{2n(m-1) - m(m+1)}{2} + 1$$

Equation (2) is the required formular for the row number m

If $m = I$, we have $r_i = \frac{(i-1)(2n-i)}{2} + 1$

Column

Given a pair of vertices i, j, where $i<j$, then the column where the weight of the graph G is stored is given by.

$$C_{ij} = j - i - 1 \tag{3.1}$$

$i<j$

Therefore, given two vertices (or nodes) *i* and *j*, the edge from *i* to *j* or *j* to *i* is given by

$$Aij = \mathrm{r_i} + \mathrm{C}_{ij} \tag{3.2}$$

Equation (4) is the formular that turns *A* (a single dimension array) into *Aij* (a multidimensional array).

Proof:

We must show that $Aij = Aji \quad \forall\, ij \in \{1,2,3,4,5,6\}$

$M_{12} = 3$,

$A_{12} = \mathrm{r}_1 + \mathrm{C}_{12} \Rightarrow ((1-1)(2(6)-1))/2 + 1 + (2-1-1)$

$A_{12} = 0 + 1 + 2 - 2 = 1$ check the index 1 in *A*

$M_{35} = 3$,

$A_{35} = \mathrm{r}_3 + \mathrm{C}_{35} \Rightarrow ((3-1)(2(6)-1))/2 + 1 + (5-3-1)$

$A_{35} = 12 + 1 = 13$ ... check the index 3 in *A* □

3.2.2 Model specification

Model specification or requirements is the process of understanding and defining what components are required for the network system and identifying the constraints on the network system's operation and development. Model requirement is particularly critical stage of a model development as errors at this stage inevitably lead to later problems in the system design and implementation.

In this research, the model specification is to develop a model which is capable of determining the shortest pathway and time for a packet to traverse the network from source node to destination node through a network of interconnected communication links in the University of Calabar. In achieving this model requirement, some tools are needed for the development of the network system.

These required tools are:

1. Measuring tapes for the measurement of physical distances between the nodes in the network.
2. A map of the unical environment which enhance the researcher to locate the network mask that provided the network physical infrastructures.
3. Two HP laptops used as the sending and receiving terminal in the network.
4. The state of each node, predecessor, distance (*dij*) and status. With the aid of these set of network system requirements, the researcher was able to obtain the physical distances *dij* between any node and its predecessor in the networks which served as the input data to the network system that receive the data, processed the received data transmit the packets of the received data and finally produced the output of the packet transmission through the shortest pathway. Also determined the time taken for the packet to traversed the network system from source to destination node through interconnected network of communication links. From the simulated network the shortest pathway and time for the packet of data to traverse from a source node to a destination node is immediate known through the help of the developed modified Dijkstra's algorithm.

3.2.3 Model development

A Comparison-Addition Model (CAM) for the modified Dijkstra's algorithm was developed in this research for the determination of the shortest path in a wireless packet switch network system in the University of Calabar.

In developing this model, a topological survey was made for the network system to determine the following parameters:

(1) number of nodes in the network

(2) physical topology of the network

(3) the link (edge) distance between a node in the network and its predecessor

(4) the source node

(5) the destination node

(6) the status of a node (either temporary or permanent)

The above named parameters or variables from the network system were used to developed the model using the following keynotes:

(i)	TL	= temporary label of a node
(ii)	PL	= permanent label of a node
(iii)	□	= permanent label of a node
(iv)	O	= temporary label of a node
(v)	*	= permanent label of a node
(vi)	n	= a node in the network i.e n_1 = node 1, n_2 = node 2, n_3 = node 3, n_4 = node4, n_5 = node5 and n_6 = node 6.
(vii)	d_{ij}	= transit or distance cost between node *i* and *j* in the network

The model was developed using six different level of priority queues to classify the packet traffic to a multiple level of priorities. The priorities are assigned on packet peculiarities. The protocol uses packet type, source and destination networks. The link capacity is divided into different classes (levels). The traffic is assigned to each class and the routers serve each class with different priority.

At level 1.

TLn_1, TLn_2 and TLn_3 are given distances d_{ij} = their distance costs from the source node to the node in question, since they are directly reachable from the source node. n_5 and n_6 are not directly reachable from the source node and their distance are $TLn_5 = TLn_6 = +\infty$

At level 2.

Compare TLs for n_2, n_3, n_4 and make the smallest of them a PL. No TL update is required at this level. Only one comparison test was needed at this level to take a decision.

At level 3

Update TLs for node 6 and 3 that are directly reachable from node 2, compare TLs for n_3, n_4 and n_6 and make the smallest of them the new (current) PL. Two comparison test was required at this level to take a decision.

At level 4

Update TL for n_5, compare TLs for n_3, n_5 and n_6 and make the smallest of them a PL. One comparison test was needed here to take a decision.

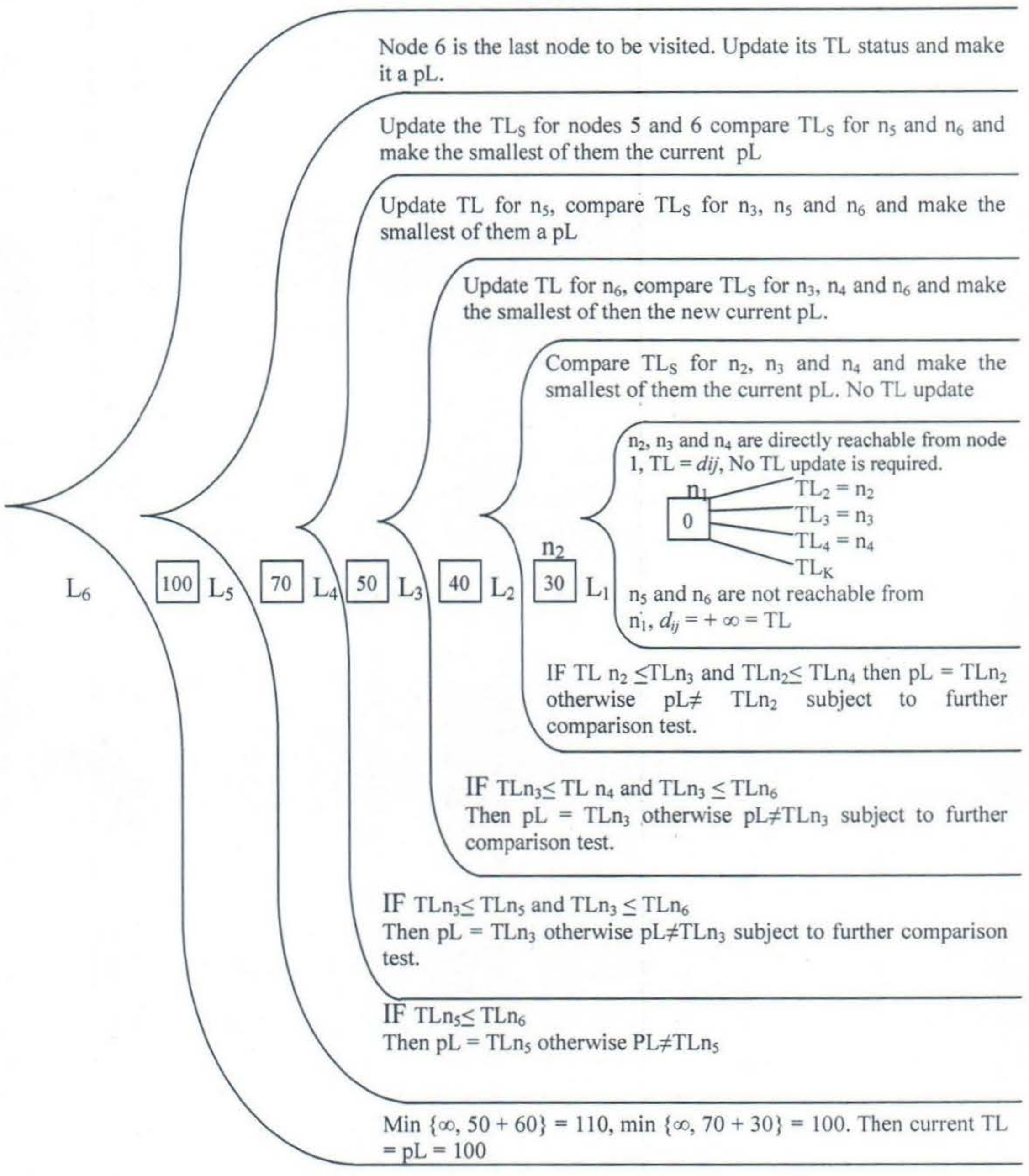
At level 5

Update the TLs for node 5 and 6, compare TLs for node 5 and 6 and make the smallest of them the current PL. One comparison test was needed here to take a decision.

At level 6

Node 6 was the last node to be visited, update its TL status and make it a PL.

Comparison–Addition Model (CAM) for the modified Dijstra's algorithm



Node 6 is the last node to be visited. Update its TL status and make it a pL.

Update the TL_S for nodes 5 and 6 compare TL_S for n_5 and n_6 and make the smallest of them the current pL

Update TL for n_5, compare TL_S for n_3, n_5 and n_6 and make the smallest of them a pL

Update TL for n_6, compare TL_S for n_3, n_4 and n_6 and make the smallest of then the new current pL.

Compare TL_S for n_2, n_3 and n_4 and make the smallest of them the current pL. No TL update

n_2, n_3 and n_4 are directly reachable from node 1, TL = *dij*, No TL update is required.

n_5 and n_6 are not reachable from n_1, $d_{ij} = +\infty = TL$

IF TL $n_2 \leq TLn_3$ and $TLn_2 \leq TLn_4$ then pL = TLn_2 otherwise pL$\neq$ TLn_2 subject to further comparison test.

IF $TLn_3 \leq TL\ n_4$ and $TLn_3 \leq TLn_6$
Then pL = TLn_3 otherwise pL$\neq TLn_3$ subject to further comparison test.

IF $TLn_3 \leq TLn_5$ and $TLn_3 \leq TLn_6$
Then pL = TLn_3 otherwise pL$\neq TLn_3$ subject to further comparison test.

IF $TLn_5 \leq TLn_6$
Then pL = TLn_5 otherwise PL$\neq TLn_5$

Min $\{\infty, 50 + 60\} = 110$, min $\{\infty, 70 + 30\} = 100$. Then current TL = pL = 100

The generalized comparison test condition is as follows:

If $TLn_2 \leq TLn_3$, $TLn_2 \leq TLn_4$, …$TLn_2 \leq TLn_K$ Then

PL = TLn_2 otherwise PL$\neq$ TLn_2

Subject to further comparison test.

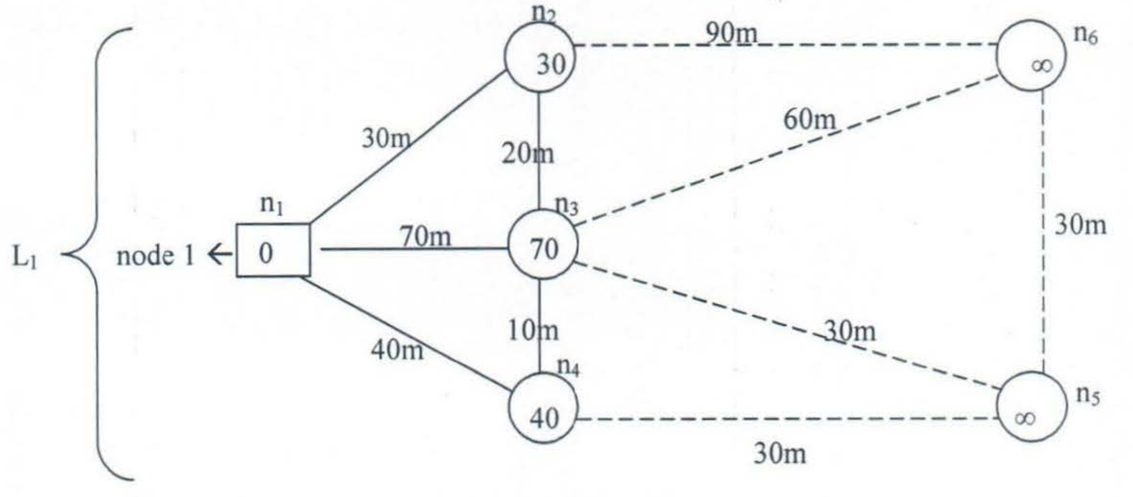
3.2.4 System analysis of the modified Dijkstra's algorithm model

The model of this research, is designed using the linked-list priority queue data structure. The simulation is done at different levels of priorities. The priority levels are as follows:

At level 1:

Node 1 ← $\boxed{0}$

Reachable nodes from node 1 are nodes 2, 3 and 4. The temporary labels for these nodes are their direct distances from node 1 to the node in question. No temporary label distance is updated here from upper boundary to a lower boundary. Nodes that cannot be reached directly from node 1 have ∞ as their temporary label. In this model, the researcher used full line for edges that are reachable from the source nodes and broken lines for edges that cannot be reached directly from the source node. Thus, the node model at this level is:



At level 2:

Compare the temporary labels of n_2, n_3 and n_4 and make the smallest of them the current permanent label. No temporary label is updated at this stage.

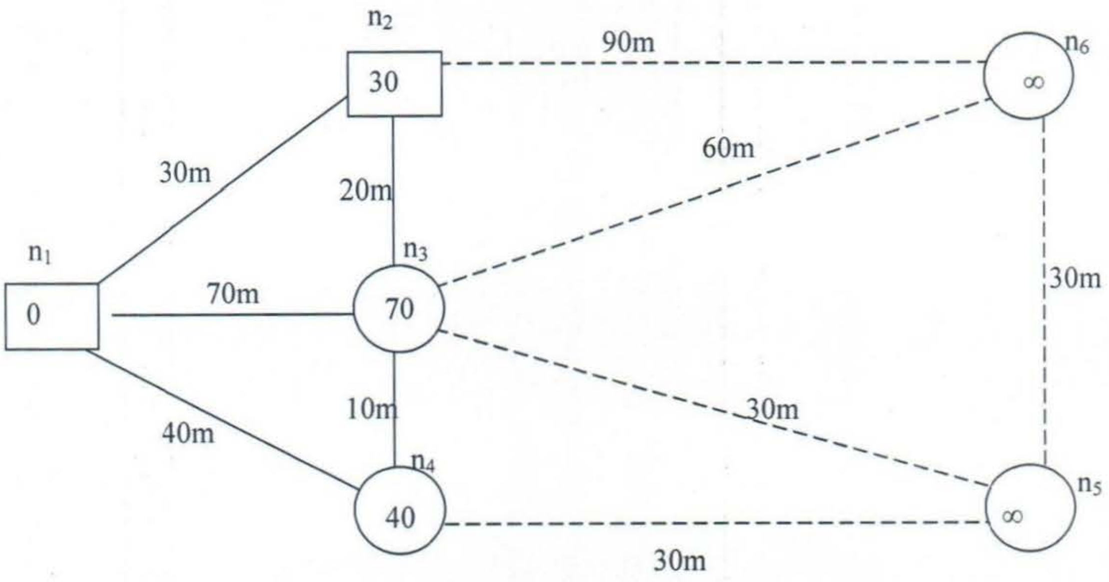
Thus, if $n_2 \leq n_3$ and $n_2 \leq n_4$ then

$pL = n_2$ else $pL \neq n_2$

:. Since $30 \leq 70$ and $30 \leq 40$

$n_2 \leftarrow \boxed{30}$

Thus, the logical structure of the model at level 2 becomes:



At level 3:

Update the temporary labels for the nodes that are directly reachable from node 2; compare their TL_s and make the smallest TL a permanent label.

The new TL for node 3 = min {∞, 30 + 20} = 50

min {∞, 0 + 70} = 70

:. new TL for node 3 = 50 Since 50 < 70

and new TL for node 6 = min {∞, 30 + 90} = 120

now compare the TL_s and make the smallest of them a permanent label.

If $n_3 \leq n_4$ and $n_3 \leq n_6$ then

$pL = n_3$ otherwise $pL \neq n_3$

:. Since the IF statement = False

Another comparison was tested with n_4

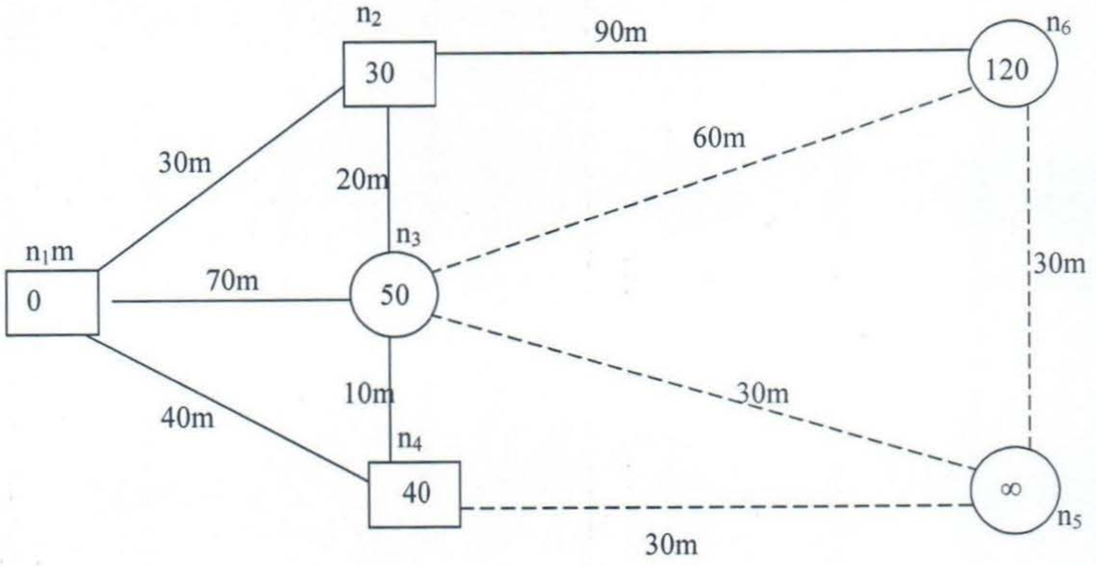
IF $n_4 \leq n_3$ and $n_4 \leq n_6$ then

$pL = n_4$ Else $pL \neq n_4$

Since $40 \leq 50$ and $40 \leq 120$ then

$n_4 \leftarrow \boxed{40}$

Thus, the logical structure of the model at level 3 becomes:



At level 4:

Update temporary label for node 5:

TL(5) = min $\{\infty, 40 + 30\} = 70$

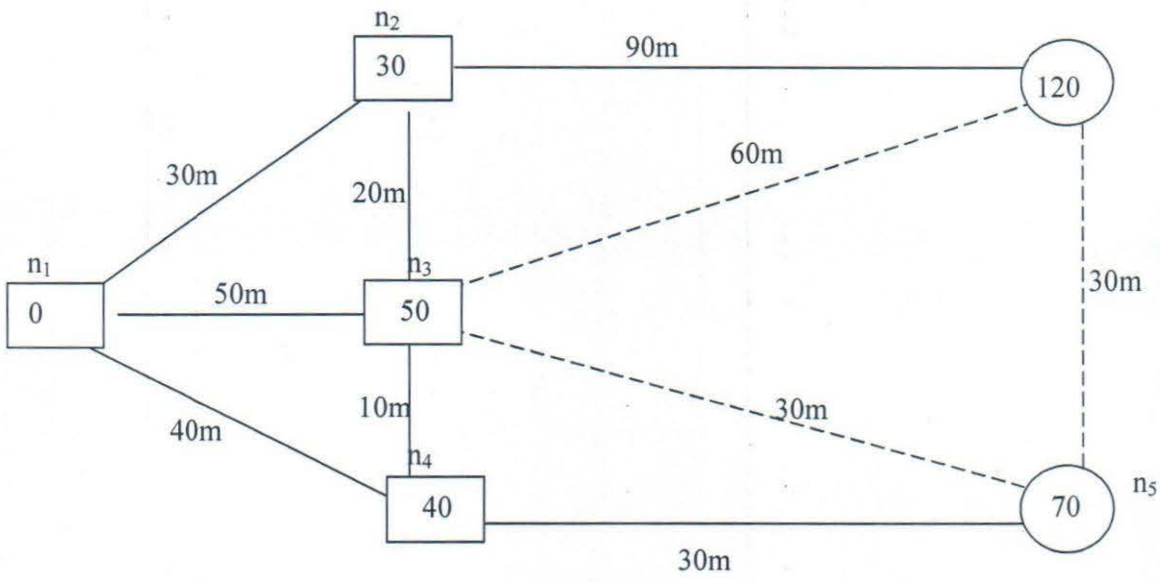
Compare TL_S for n_3, n_5 and n_6 and make the smallest a permanent label.

If $n_3 \leq n_5$ and $n_3 \leq n_6$ then $pL = n_3$

Else $pL \neq n_3$ since $50 \leq 70$ and $50 \leq 120$

$n_3 \leftarrow \boxed{50}$

Thus, the logical structure of the model at level 3 becomes:



At level 5:

Update temporary label for node 5 and 6

n_5: min$\{\infty, 50 + 30\} = 80$

min $\{\infty, 40 + 30\} = 70$

TL for $n_5 = 70$

n_6: min $\{\infty, 30 + 90\} = 120$

min $\{\infty, 50 + 60\} = 110$

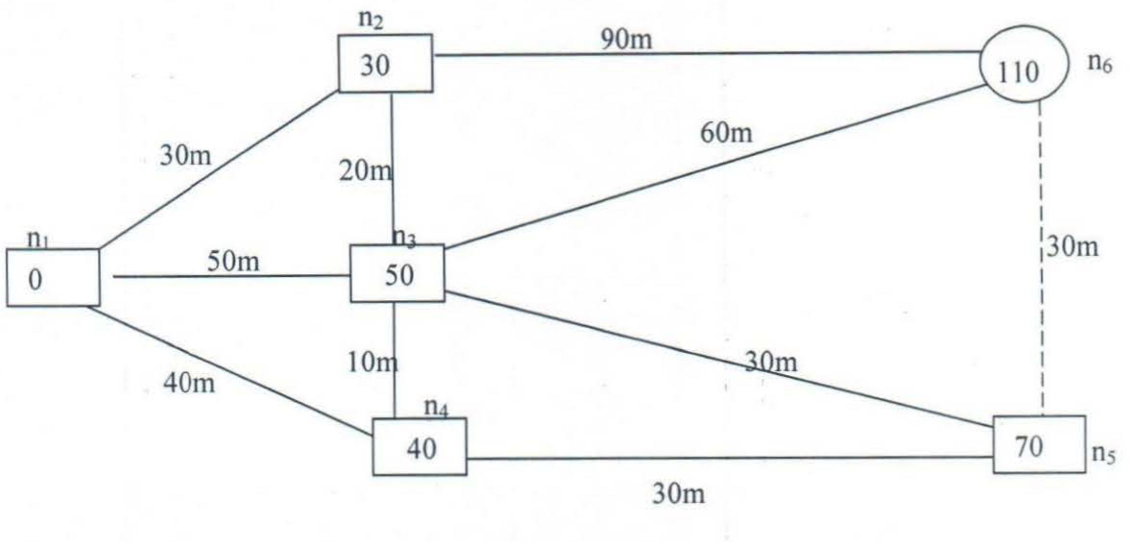
TL for $n_6 = 110$

Now compare TL_S for node 5 and 6 and make the smallest a pL.

IF $n_5 \leq n_6$ then

$n_5 \leftarrow \boxed{70}$

Thus, the logical structure of the model at level 5 becomes:



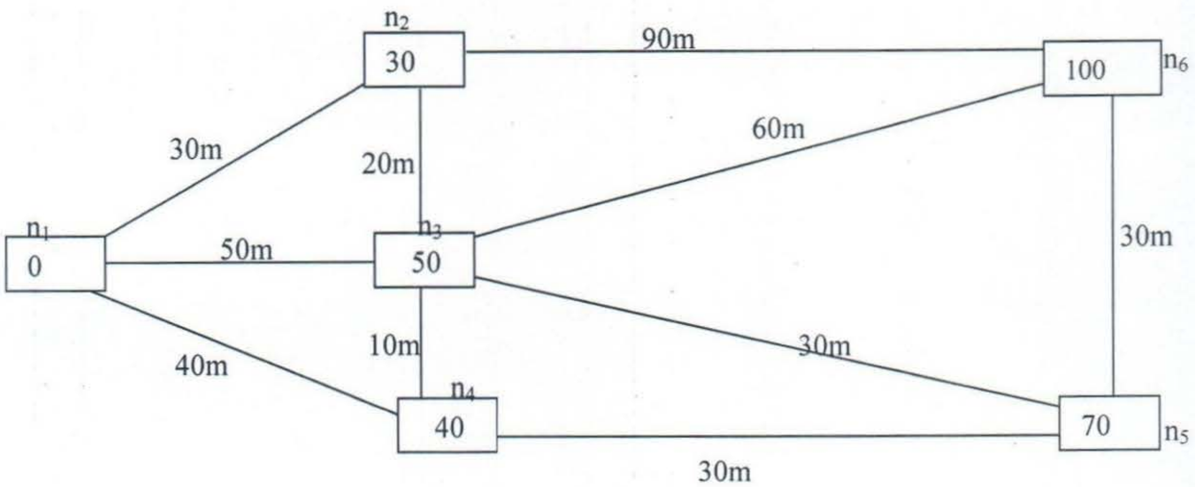
At level 6:

Node 6 is the last node to be visited. Update the temporary label of node 6 and make it permanent.

n_6: min $\{\infty, 50 + 60\} = 110$

min $\{\infty, 70 + 30\} = 100$

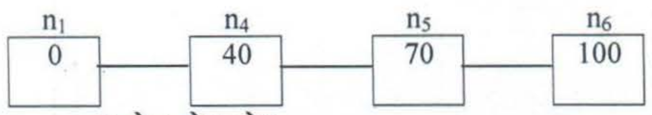
:. $TLn_6 = 100 = PL$



Note: *Distance were measured in meters'*

Thus, node 6 has a new TL of 100 and it is automatically made permanent. The algorithm converges at this point. The shortest path *dij* of the network mode is 100 and the path length or link whose *dij* = 100 is the shortest path of the network. From the logical simulation of the wireless packet switch network model in the

University of Calabar, the shortest path for the packets to traverse from the source node to the destination node is as follows:



$n_1 \rightarrow n_4 \rightarrow n_5 \rightarrow n_6$

The generalized condition for the comparison test is as follows:

IF $TLn_2 \leq TLn_3$

$TLn_2 \leq TLn_3 \ldots, TLn_2 \leq TLn_K$

Then $PL = TLn_2$ otherwise $PL \neq TLn_2$

Subject to further comparison test.

3.2.5 Model validation

This section of the research validates the practical efficiency of a new shortest path algorithm for a wireless packet switch network system, which was modeled as an undirected graph. The modified Dijkstra's algorithm outperforms the traditional Dijkstra's algorithm for all pairs shortest path problem, and more generally path problem, for the challenges of computing single source shortest path from *w*(i) different sources. The term model validation or, more generally, verification and validation (V & V) is intended to show that the model conforms to its specification and that it meets the expectation of the system.

However, the validation of the developed model was done at different levels before testing the entire model to determined the shortest pathway and time that a packet will take to traverse the wireless packet switch network from source to destination. At the end of the validation test for the developed model, it was found that the model was an adequate instrument for the determination of the shortest pathway and time in a wireless packet switch network of six nodes.

3.2.6 Algorithm formulation

The formulation of the modified Dijkstra's algorithm follow the steps, having abstracted the wireless packet switch network system as a graph structure $G = (V, E)$. In graph theory, the shortest path algorithm can be used to find the path between source node to destination node such that the sum of the weights of its constituent edges is minimized.

General description

Basically, the shortest path from a given nodes to other nodes in a wireless packet switch network is a one-to-all single source problem. The modified Dijkstra's algorithm solves this class of problem, by finding the shortest path from a given source nodes to a given destination node in the network. Node S is called the starting node or an initial node. The algorithm starts by assigning some initial values for the distance from node S and to every other node in the network, the modified Dijkstra's algorithm operate in steps, where at each step the algorithm improves the distance value. At each step, the shortest distance from node S to another node is determined.

The algorithm characterizes each node by its state. The state of a node consists of two features: distance (d_{ij}) and status label. The distance value of a node is a scalar representing an estimate of its distance from node S. Status label is an attribute specifying whether the distance value of a node is equal to the shortest distance to node S or not. The status label of a node is permanent if its distance value is equal to the shortest distance from node S. Otherwise the status label of a node is temporary. The algorithm maintains a step by step updates of the states of the nodes. At each step one node is designated as current.

Notation

d_l denotes the distance of a node l

p or t denotes the status label of a node, where

P stands for permanent and

t stands for temporary

d_{ij} is the distance weight cost for traversing link (i,j) as given by the problem.

The state of a node l is the ordered pair of its distance value d_l and its status label.

The procedure for the modified Dijkstra's algorithm is given as follows:

1. Initially make the source node permanent and make it the current working node. All other nodes are made temporary.
2. Examine all the temporary neighbours of the current working node after checking the condition for minimum weight, relabel the require node(s).
3. From all the temporary nodes, find out the nodes which have the minimum value of distance and make it permanent and this became the current working node.
4. If there is a tie in step 2 and 3, choose any one but exactly once.
5. Repeat step 2, 3, and 4 until destination node is made permanent.

3.2.7 Instrument for Algorithm development

Specifically, the aim of this research is to design and analyze a wireless packet switch network system in the University of Calabar environment with the objective of determining the shortest distance (path), d_{ij} and time, t_{ij} for network packets to traverse from the source node to the sink node (destination). The problem domain is a direct implementation of a network routing concepts.

The instrument used for the development of the algorithm are:

1. Source node
2. Destination node
3. four intermediate nodes
4. 2 HP laptops
5. Measuring tapes
6. UNICAL scale map
7. Status of each node, predecessor and distance

This research implements a totally different concept to find out shortest path using scaled map of the University of Calabar environment. The map which the researcher used was scaled and returns correct distance between the nodes in the network model. Here the researcher used linked list to store and traverse N nodes or vertices. The researcher labeled each node with distance, predecessor and status. Distance of node represents the shortest distance of that node from the source node, and predecessor of node represents the node which proceeds the given node in the shortest path from source. Status of a node can be permanent or temporary.

3.2.8 Reliability test

The wireless packet switch network system was tested using a modeled undirected weighted network graph (V, E, d) with node set V, edge set E, and the weight set d specifying d_{ij} for the edges $(i,j) \in E$. This solves only the problems with non-negative costs, i.e, $dij \geq 0$ for all $(i, j) \in E$.

In this research, two experiment were carried out: tracet and ping test. The ping test was carried out in three different routes of the network, including the shortest path route. The network was simulated with 30MB data for each route. From the output (result) it was found that the shortest path route took the smallest time interval for the packet to traverse from the source node to the destination node and back.

The research problem is one-to-all, single source shortest path algorithm. The weight on the link are also referred as cost. The modified Dijkstra's algorithm solves only the problems with non-negative costs, i.e, $dij \geq 0$ for all $(i, j) \in E$. Having abstracted the network model as a graph G, the modified Dijkstra's algorithm can find a shortest path from a given node S to other nodes in the network (one-to-all shortest path problem). The modified Dijkstra's algorithm solves this problem better. It finds the shortest path from a given node S to all other nodes in the network. Node S is called a starting node or an initial (current) node.

The modified Dijkstra's algorithm starts by assigning some initial values for the distances from node S and to every other node in the network. The algorithm operates in steps, where at each step the algorithm improves (update) the distance values. At each step, the shortest distance from node S to another node is determined.

The algorithm characterizes each node by its state. The state of a node consists of two features:

i) distance value and

ii) status label

Distance value of a node is a scalar representing an estimate of its distance from node S. Status label is an attribute specifying whether the distance value of a node is equal to the shortest distance to that node or not. The status label of a node is permanent if its distance value is equal to the shortest distance from node S otherwise the status label of a node is temporary. The algorithm maintains a step-by-step updates of the status of the nodes. At each step, one node is designated as current. *dij* denotes the distance value of a node P or t represented as □ and O which denotes the status label of a node, where P stands for permanent and t stands for temporary. *dij* is the costs of traversing link (*ij*) as given by the network model. The state of a node is the ordered pair of its distance value *dij* and its status label.

The algorithm steps:

Step 1 (Initialization)

a) Assign the zero distance value to node *S*, and label it as permanent [The state of node *S* is (*0*, *P*)].

b) Assign to every reachable node from *S*, a distance value of *dij* and label them as temporary [The state of these nodes are (*dij*, *t*)].

c) Assign to every node not reachable from *S* a distance value of ∞ and label them as temporary [The state of these node are (∞, *t*)].

d) Designate the node *S* as the current node.

Step 2

Distance value update and current node designation update.

Let *i* be the index of the current node.

a) The algorithm find the set *J* of nodes with temporary labels that can be reached from the current node *i* by a link (*i,j*). Update the distance value of these nodes. For each $j \in t$, the distance value *dj* of node *j* is updated as follows:

new $dj = \min \{dj, di + dij\}$

where *dij* is the cost link (*i,j*) as given in the network problem.

b) Determine a node *j* that has the smallest distance value *dj* among all nodes $j \in J$,

find *j** such that;

$\min dj = dj^*$

$j \in J$

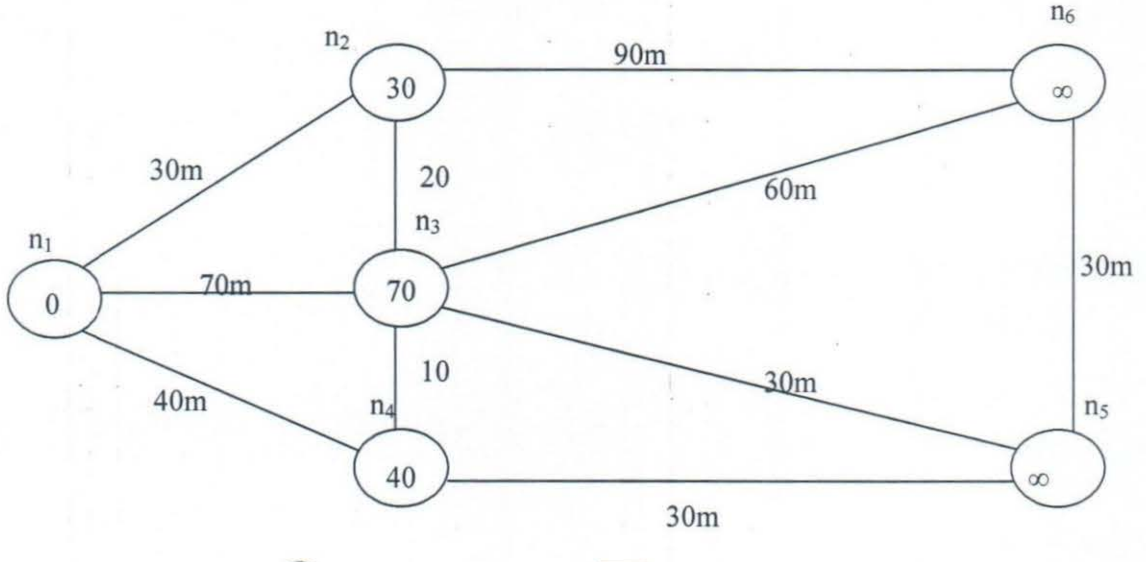
c) Change the label of node *j** to permanent and designate this node as the current node.

Step 3 (termination criterion)

If all nodes that can be reached from node S have been permanently labeled, then stop, which indicates completion. If the temporary labeled node cannot be reach from the current node, then all the temporary labels become permanent (for directed graphs), which signify completion, otherwise go to step 2.

3.2.9 Network design

There are several algorithms to determine and find the shortest path. This research study is compatible with modified Dijkstra's algorithm. The network design for a wireless packet switch network system in the University of Calabar, was abstracted as a graph. The algorithm works efficiently for both directed and undirected network system. The network design is as follows;



*Temporary label = ○, Permanent label = □

Each node is labeled as d_{ij} distance of a node represents the shortest distance of that node from the source node, and predecessor of node represents the node which precedes the given node in the shortest path from the source node. The status of a node may be permanent or temporary. In the context of this research, keynote □ is used for permanent label of a node and ○ for temporary label of a node. Making a node permanent means that it has been included in the shortest path. Temporary node can be relabeled if required but once a node is made permanent, it cannot be relabeled.

The procedure of the algorithm is as follows:

1. Initially make the source node permanent and make it the current working node. All other nodes are made temporary.
2. Examine all the temporary neighbours of the current working node and after checking the condition for minimum weight, re-label the required nodes.
3. From all the temporary nodes, find out the nodes which have the minimum value of distance and make it permanent and this becomes the current working node.
4. If there is a tie in steps 2 and 3, choose any one but exactly once.
5. Repeat step 2, 3 and 4 until destination node is made permanent.

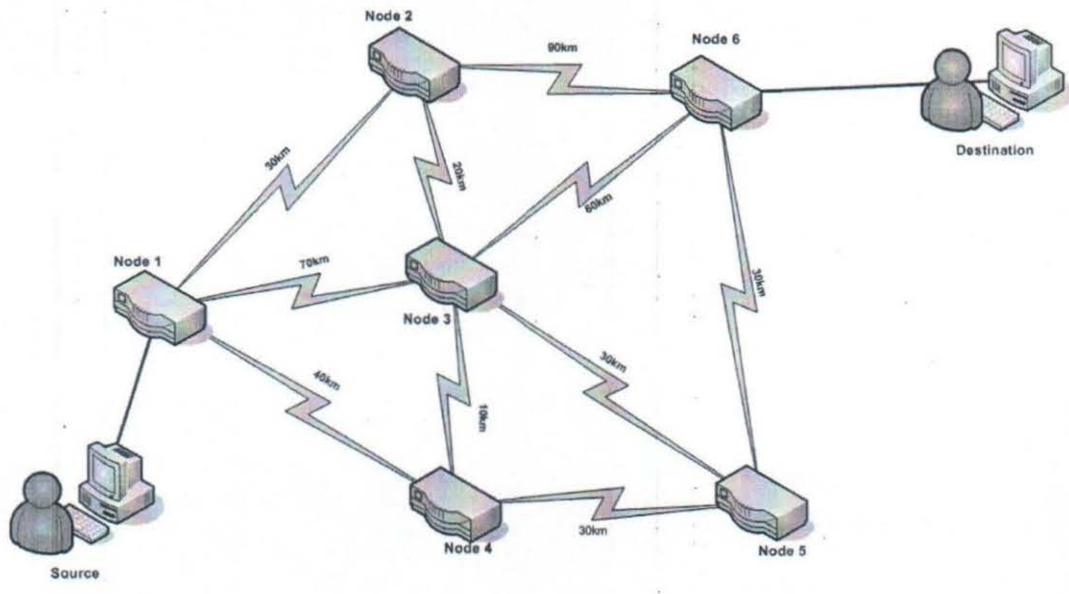


Figure 1: Network diagram showing the source to the destination node

3.2.10 Experimental design

In this research, two experiments were carried out in the network known as Tracet and Ping test. However, Tracet experiment allows the researcher to choose alternative route for the packets to traverse from the source node to the destination (sink) node through a network of interconnected communication links. The Ping test experiment was design to determine the average time taken for a packet to traverse in the chosen route from a source node to the destination (sink) node through a network of interconnected communication links.

3.2.11 Evaluation

In this research, the algorithm evaluates the performance of the recent undirected wireless packet-switch network system in the University of Calabar environment. The algorithm is a robust, comparison-addition based algorithm for solving undirected SSSP from multiple specified sources (MSSP). Abstract was made in the wireless packet switch network in the University of Calabar to a model of a graph *G*. In this model the following assumptions were made:

i) *dij* are non-negative edge weights costs.

ii) The graph is undirected network.

iii) Node 1 is designated as the source node.

iv) Node 6 is designated as the destination node.

v) Nodes that can be reached directly from the source node have *dij* as their temporary labels.

vi) Nodes that cannot be reached directly from the source node have $+\infty$ as their temporary labels from the source node.

vii) Temporary labels are updated base on the current position of the current visited node.

3.2.12 Network validation

From the topological analysis of the wireless packet switch network, it was found that the shortest weight cost was 100m and the shortest path was Node 1 → Node 4 → Node 5 → Node 6. The test tool employed was the ping test which uses an ICMP ECHO from source to destination and back. Therefore, the time recorded for a ping response is twice the distance covered. The ICMP ECHO shortest path was;

Node 6 → Node 5 → Node 4 → Node 1

Route 1 – uses 1 → 2 → 6, dij = 120, t = 0.017s

Route 2 – uses 1 → 4 → 3 → 6, dij = 110, t = 0.016s

Route 3 – uses 1 → 4 → 5 → 6, dij = 100, t = 0.013s

From the above results obtained from the network simulation, it was found that the shortest path took the smallest time for the packet to traverse from source to destination and back thus, validating the efficiency of the network system.

3.3 Research methodology

Irony and Rose (2005) asserted that, methodology is the systematic and theoretical analysis of the methods applied to a field of study. It consists of the theoretical analysis of the body of methods and principles related with a branch of knowledge. It also includes concepts such as paradigm, theoretical model, phases and quantitative or qualitative technique. Creswell (2003) defined methodology as the analysis of the principles of method, rules and hypothesis employed by a discipline. Creswell went further to state that, methodologies go beyond the approach of incorporating guidance for business analysis, project planning and management, project processes (examples: estimation, metrics, risk management), quality assurance, testing, role and responsibilities, reuse and architectural design.

Methodology adopted for this study is outlined below:

a) General research information

b) Comprehensive site study

c) Analyzing various requirements

d) Collate all relevant data

e) Modification and identification of variables

f) Implementation of the algorithm

g) Modified OSPF (protocol suit for the modified Dijkstra algorithm)

h) Application

3.3.1 General research information

A successful wireless packet switch network system is produced in sequence of stages that are typically managed by separate teams of developers. The stages are as follows:

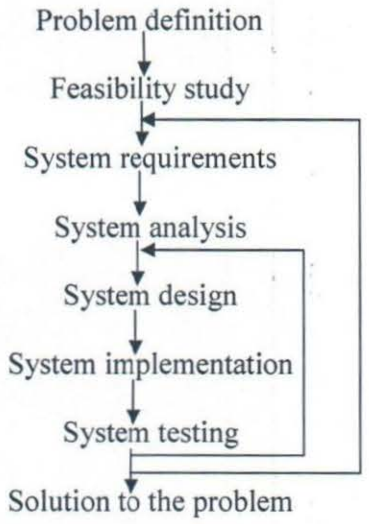


Figure 2: Hierarchical stages to research formulation

The first stage is a recognition of the problem to be solved (the shortest path and time determination in a wireless packet switch network system in the University of Calabar environment) with a view of determining the shortest pathway, *dij*, and time *tij* taken by a packet to traverse in the network from the source node to the sink(destination) node.

The second stage deals with the availability and feasibility of tools that aid the development of a successfully wireless network system in the University of Calabar environment.

This stage is to determine the shortest path and time taken by packet to traverse from a source node to a destination node, thus it solves the optimal routing problem in the network. This requirement document should have enough detail to be used as a standard when the network system is tested.

In the fourth stage, a thorough analysis is done before any effort or resources are spent designing and implementing the network system. This could include a survey of comparable networks system already available and a cost benefit analysis of the value of spending the anticipated resources. Once a decision has been made to proceed, the network designer, then works from the requirement document to design the network system. Thus, including the specification of all the networks components and their interrelationships. It may also require the specification of specialized algorithms that would be implemented in the network system.

The system implementation consists of network engineers running the developed system to produce the results.

The system testing team attempts to ensure that the resulting network system satisfies the requirements document. Failure at this point may require a design or even some fine tuning of the requirements. Those eventualities are represented by the two feedback loops as shown in figure 1. System testing occurs at several levels. Individuals route have to be tested separately and their success at working together must be verified (Tracet). Finally the entire network system is tested against the requirement document. One final aspect of network development, is the maintenance process. After the network has been designed, its designers remains obliged to maintain it with corrected tools and major alterations. Any major alteration would follow the same life cycle steps.

3.3.2 Comprehensive site study

The University of Calabar - also known as UNICAL - is a federal university situated in Calabar, Cross River State, South-southern Nigeria. It is one of Nigeria's second generation universities. The University of Calabar was a campus of the University of Nigeria until 1975. The architecture was designed by John Elliott. It was established by decree to fulfill this traditional mandate, its motto "Knowledge for Service".

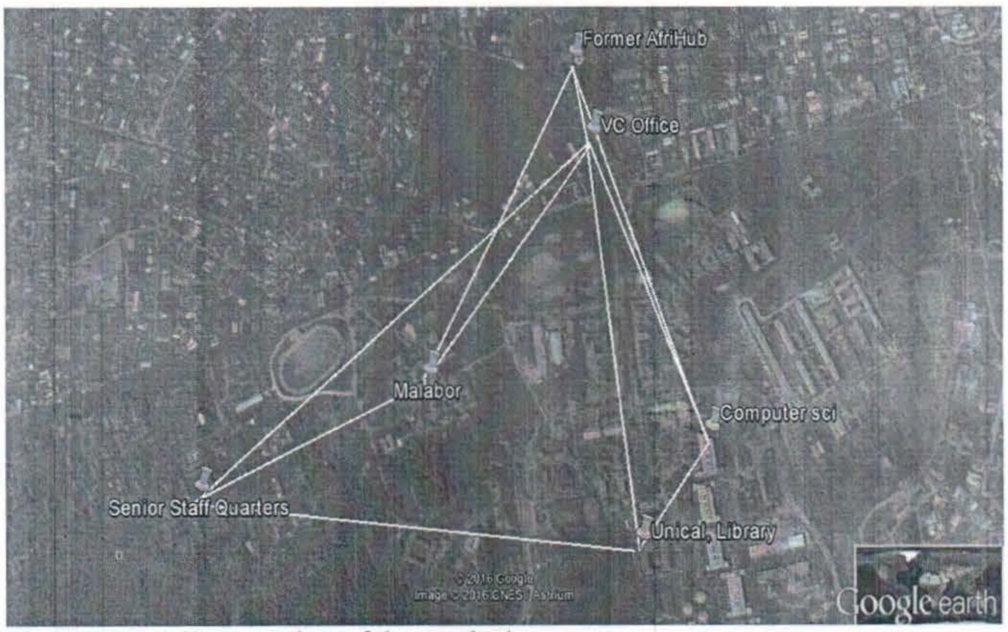


Figure 3: Google snap shot of the test bed

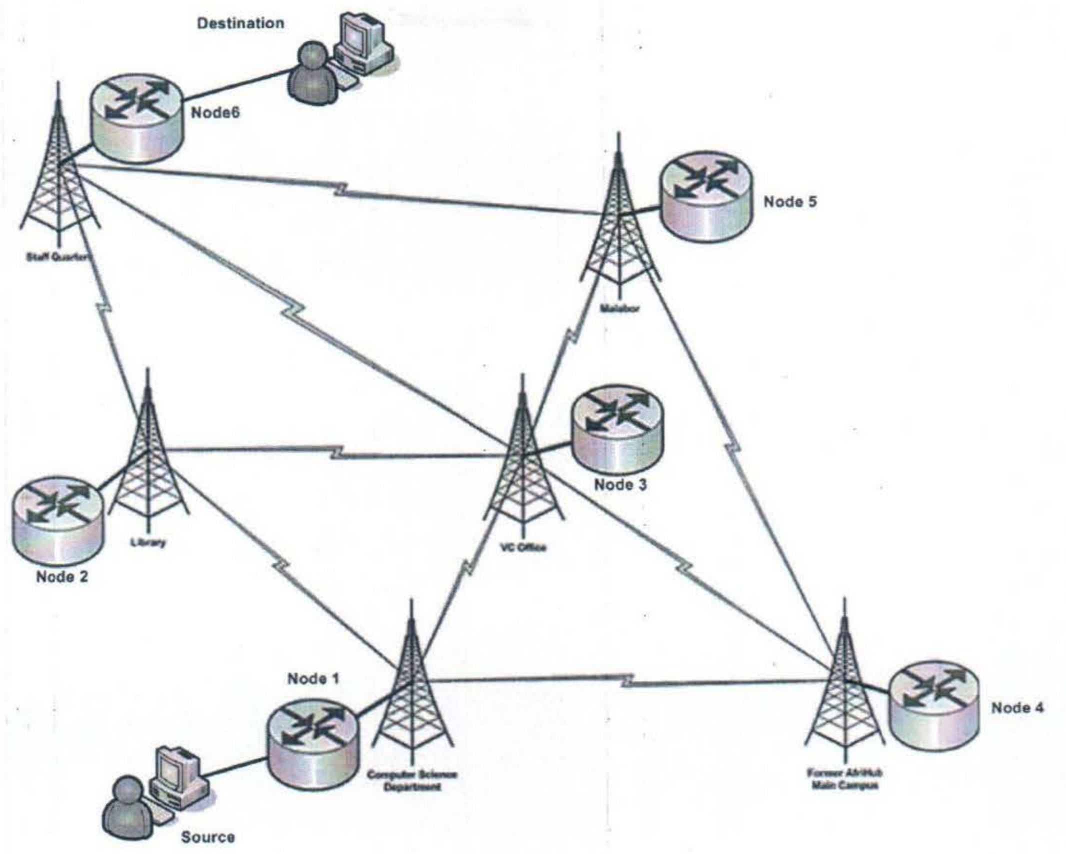


Figure 4: Test-Bed (Network Design Diagram)

Six Nodes was considered in this study namely; Computer Science dept, Former AfriHub building (Main Campus), VC Office, Unical Library, Malabor (Female Hostel) and Staff Quarters. Each node was connected as shown in figure 2 above to implement full mesh connectivity.

Connection between any two nodes will have an IP subnet mask of 255.255.255.252 i.e. X.X.X.X/30 implying that there were only two valid hosts per network. The modified Dijkstra's script was written and configured on each node as shown in the pseudo code above. Each output was installed on the routing table of the router (node).

3.3.3 Analyzing various requirements

The main experimental platform used by the researcher was a CPU with a RB750: AR7240 400MHz RB750GL: AR7242 400MHz and 64MB onboard NAND memory chip. The main memory allowed us to test graphs with millions of vertices. For the purposes of this research, the following specification was used:

Memory	RB750: 32MB DDR SDRAM onboard RB750GL: 64MB SDRAM onboard
Boot loader	RouterBOOT
Data storage	64MB onboard NAND memory chip
Ethernet	RB750: Five 10/100 Mbit/s Ethernet ports with Auto-MDI/X RB750GL: Five 10/100/1000 Gigabit Ethernet ports with Auto-MDI/X
LEDs	Power, NAND activity, 5 Ethernet and 1 wireless LEDs
Power at the device	DC power jack (5.5mm outside and 2mm inside diameter, female, pin positive plug) accepts 8-30V DC
Power over Ethernet	Ether 1 requires 8-30V DC (non 802.3af), to compensate for losses, it's recommended to use 12V or more
Power consumption	Up to 6W
Dimensions	113x89x28mm. Weight without packaging and cables: 129g
Temperature	Indoor device. Operational temperature: -20°C to +50°C
Humidity	Operational: up to 70% relative humidity (non-condensing)
RouterOS	RouterOS v5, Level4 license

3.3.4 Collate all relevant data

The researcher collated relevant data from the output of ping and tracet test in research work as well as interview with engineers/ICT staff of the University.

3.3.5 Modification and identification of variables

In this research work, two known quantities will be considered, Bandwidth and Speed. Bandwidth can best be defined as the amount of data that passes a medium per given time (Sec). While speed can be defined as the distance a packet travels with time (sec).

The mathematical computation of both definitions can be expressed as follows:

Bandwidth (Bw) = data (bytes) per sec

$$Bw = bps \tag{3.3}$$

The unit of measurement can also be expressed in order of their magnitude such as:

Kilobytes per seconds = 1kbps = 1024bps

Megabytes per seconds = 1mbps = 1024kbps

I Gigabyte per seconds = 1024mbps

In measuring the bandwidth across a link/route, the lowest bandwidth on that link is regarded as the bandwidth of the link. Take for example, figure 4 has three (3) routers/hops between the source and destination. In between each device is recorded the distance and relative bandwidth. The bandwidth of the link between the source and destination is 1kbps since it's the lowest bandwidth of the link.

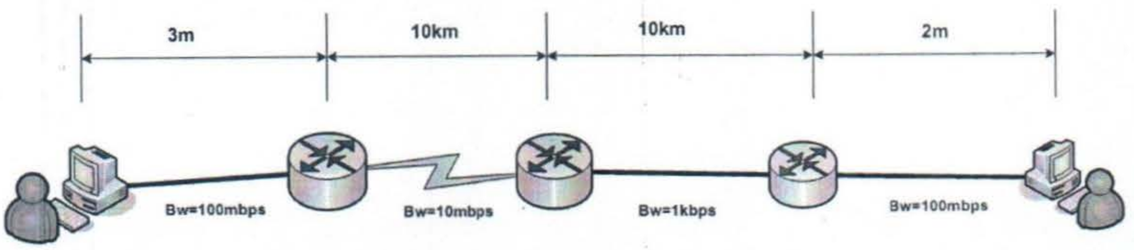


Figure 5: Speed and Bandwidth measurement

The speed can be represented mathematically as:

Speed, S = Total distance covered per sec

$$= m/s \tag{3.4}$$

From the illustration above (figure 4), it therefore implies that the speed attained for a packet to transverse from source to destination is equal to 20,005m per sec.

The test tool employed is the ping test from source to destination. A ping test uses an ICMP ECHO to the destination and back. This implies that, for every reply recorded on the source, the packet has gone to and back. Therefore, the time recorded for a ping response is for twice the distance covered. Hence, equation two (3.5) will be modified to :

$$= 2m/s \tag{3.5}$$

The variable of interest in this work is time. Hence, making time(s) in equation i the subject;

$$\text{Time (s)} = \text{Bandwidth/data} \tag{3.6}$$

Substituting equation (iv) into equation (iii)

$$\text{Speed, S} = \frac{2m}{\left(\frac{Bandwidth}{data}\right)} \tag{3.7}$$

$$= \frac{2m\ x\ data}{Bandwidth} \tag{3.8}$$

3.3.6 Implementation of the algorithm

The basic design concept of modified Dijkstra is similar to Dijkstra: the network will be abstract into a graph, and a set up of an adjacent list which makes single-link lists for all non-isolated nodes in the graph. The i-list containing the nodes will be directly connected to the non-isolated node vi. Each node will compose two parts: neighboring nodes field (adjvex) and linking field (nextarc). The neighboring nodes field will marks where the nodes connect to node vi in the graph and the linking field will also marks the next node. Each single-link will have a head node which is composed of data field (data) and linking field (firstarc). Data field will mark the number of vi in the graph and the linking filed will marks the first node that is connected to vi.

Initialization

Set up a two-dimensional array c initialized maximum to store the length of the shortest path between each two nodes in the graph, then set up a one-dimensional array bc initialized 0 to store the betweenness centrality of each node. V is the number of non-isolated nodes in the graph.

Set up array a and b. a is used to restore the end node of the shortest path. b is used to restore the number of nodes with the same starting node and the same length and put the starting point into a and put la=0 into b. At last, set the relative element 0 in array c.

3.3.7 Modified OSPF (protocol suit for the modified Dijkstra algorithm)

This problem is related to the spanning tree one. The graph representing all the paths from one vertex to all the others must be a spanning tree - it must include all vertices. There will also be no cycles as a cycle would define more than one path from the selected vertex to at least one other vertex. For a graph,

G = (V,E) Where
- **V** is a set of vertices and
- **E** is a set of edges.

The modified Dijkstra's algorithm keeps two sets of vertices:

S the set of vertices whose shortest paths from the source have already been determined *and*

V-S The remaining vertices.

The other data structures needed are:

D array of best estimates of shortest path to each vertex

pi An array of predecessors for each vertex

The basic mode of operation is:

1. Initialise **d** and **pi**,
2. Set **S** to empty,
3. While there are still vertices in **V-S**,
   i. Sort the vertices in **V-S** according to the current best estimate of their distance from the source,
   ii. Add **u**, the closest vertex in **V-S**, to **S**,
   iii. Relax all the vertices still in **V-S** connected to **u**

Pseudo code

```
dist[s] ←o                             (distance to source vertex is zero)
for all v ∈ V-{s}
    do dist[v] ←∞                      (set all other distances to infinity)
S←∅                                    (S, the set of visited vertices is initially empty)
Q←V                                    (Q, the queue initially contains all vertices)
while Q ≠∅                             (while the queue is not empty)
do  u ← mindistance(Q,dist)            (select the element of Q with the min. distance)
   S←S U{u}                            (add u to list of visited vertices)
   for all v ∈ neighbors[u]
       do if dist[v] > dist[u] + w(u, v)       (if new shortest path found)
             then    d[v] ←d[u] + w(u, v)      (set new value of shortest path)
                                               (if desired, add traceback code)
return dist
```

The basic design concept of modified Dijkstra is similar to Dijkstra: the network will be abstract into a graph, and a set up of an adjacent list which makes single-link lists for all non-isolated nodes in the graph. The i-list containing the nodes will be directly connected to the non-isolated node vi. Each node will compose two parts: neighboring nodes field (adjvex) and linking field (nextarc). The neighboring nodes field will marks where the nodes connect to node vi in the graph and the linking field will also marks the next node. Each single-link will have a head node which is composed of data field (data) and linking field (firstarc). Data field will mark the number of vi in the graph and the linking field will marks the first node that is connected to vi.

Initialization

Set up a two-dimensional array c initialized maximum to store the length of the shortest path between each two nodes in the graph, then set up a one-dimensional array bc initialized 0 to store the betweenness centrality of each node. V is the number of non-isolated nodes in the graph.

Set up array a and b. a is used to restore the end node of the shortest path. b is used to restore the number of nodes with the same starting node and the same length and put the starting point into a and put la=0 into b. At last, set the relative element 0 in array c.

Judge whether it is the end of array a. If it is, turn to; else turn to. Check out the number of nodes with the length of shortest path la, and set it to be n. Travel the next node in array a and the find out all child nodes which are connected to this node (parent node) directly. If the value from starting node to this node in array c is bigger than la, set the value from the starting node to this node and the value from this node to starting node in array c to be la+1, then put the id of parent node and the id of child node into a. The same nodes being put into an m times means that there are m shortest paths from the starting node to this node. Add 1 to num which is the number of nodes on layer la+1. Because the id of parent node can be found by the id of child node, the shortest paths from the starting node to the other nodes in array a can be found, then add 1 to the nodes on each shortest path. repeat n-1 time; put num into array b, la=la+1, turn to. Repeat V-1 times. It is the end.

According to the description above, the space complexity is T(V2) which is equal to that of Dijkstra (Wei *et al.*, 2010).

3.3.8 Application

According to Wei *et al.* (2010), Dijkstra has a three-cycle and find out the shortest path between each two nodes will be add 1 to between centrality of the nodes being on the path. The application range is limited for its high time complexity and the time consumption is unavailable when it is applied into large network of thousands of nodes. As put by different researchers Fortz and Thorup (2002), Retvari and Cinkler (2004), and Soltani *et al.* (2002), the length of shortest path between each two nodes would not exceed a constant. Based on this method, OSPF is proposed.

Open Shortest Path First (OSPF) is a well known real-world implementation of DA used in network routing. In real networks, particularly in Ethernet networks, the Spanning-Tree Protocol (STP) (Beaubrun and Pierre, 1997) runs on the network before the OSPF. In a general way, a spanning tree of a graph is a sub-graph which is also a tree that contains all the nodes. In other words, in a network environment, where redundant links are common, the STP causes these links to appear closed for the operation of the network elements, as to eliminate the appearance of duplicate messages, such as e.g. Neighbour discovery ages.

CHAPTER FOUR

RESULTS AND ANALYSIS

4.1 Introduction

This chapter is concern with interpretation, discussion and the results of the design remote network and its application for this study. The connection of PC as node in a diagrammatic form was done.

4.2 Description

A node in the network is referred to as a layer three (3) device on the OSI reference model. Each node is tasked with translating a packet from a directly connected network to another device in a remote network. A directly connected network is a network that has one of its interfaces connected to a node while a remote network is a network that the node has to learn of from another node. Hence in the diagram below, PC-A is directly connected to Node-A while PC-B is remotely connected to Node-A.

The node translates packet received from any of its interfaces by matching the destination address of the received packet with network addresses on its routing table. If the network address on the destination field matches any network address on its routing table, the packet is sent to the specific interface attached to the route else the packet is dropped. Each node can also be called a base station in long range wireless communication. A single base station can cover an area with radius of up to 5km (3miles) while providing 67Mbps throughput; using the most robust modulation and coding scheme and the lowest frequency bandwidth this coverage area increases to 51km (31.8miles) radius, but at a reduced throughput of 400kbps (Wei, *et al.*, 2008).

From these findings, lagrange's interpolation formula was used to obtain estimated bandwidths associated to different distance

$$\frac{y-y_a}{y_b-y_a}=\frac{x-x_a}{x_b-x_a} \qquad (4.1)$$

Six (6) nodes were placed with the help of wireless radio devices at different locations as shown in the diagram below (see figure 6). The objective was to find out if the nodes could compute the cumulative shortest path to the destination from the source.

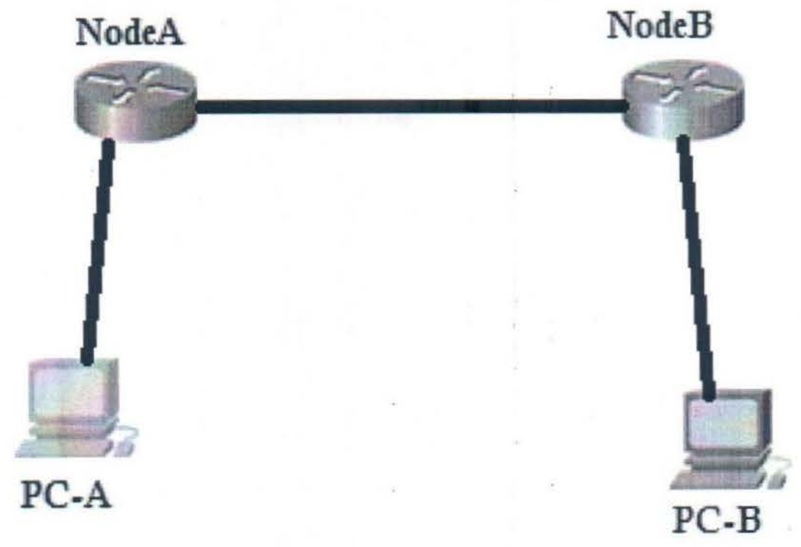


Figure 6: Diagram of a connected remote network

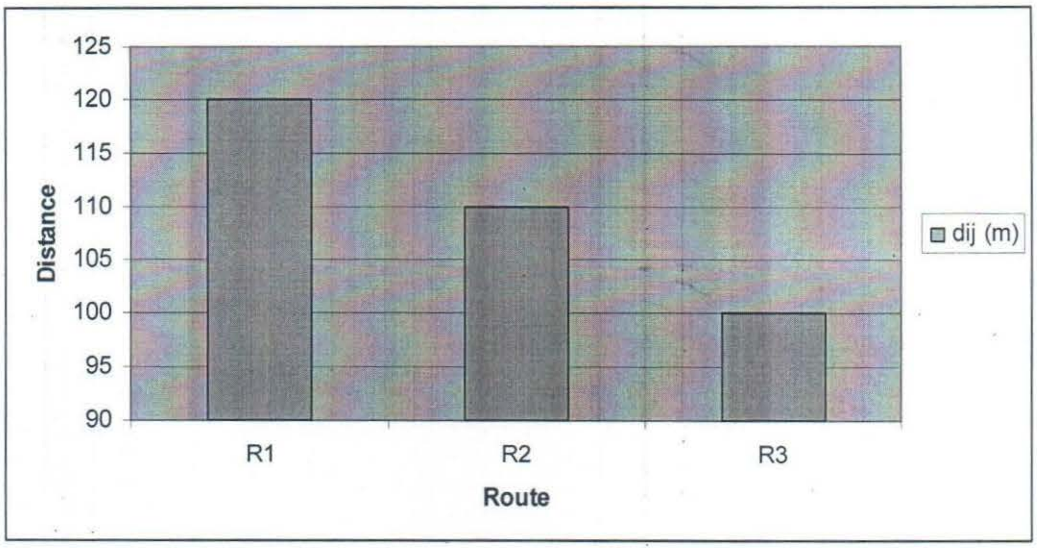


Figure 7: Graph showing Route against Distance, d_{ij} (m)

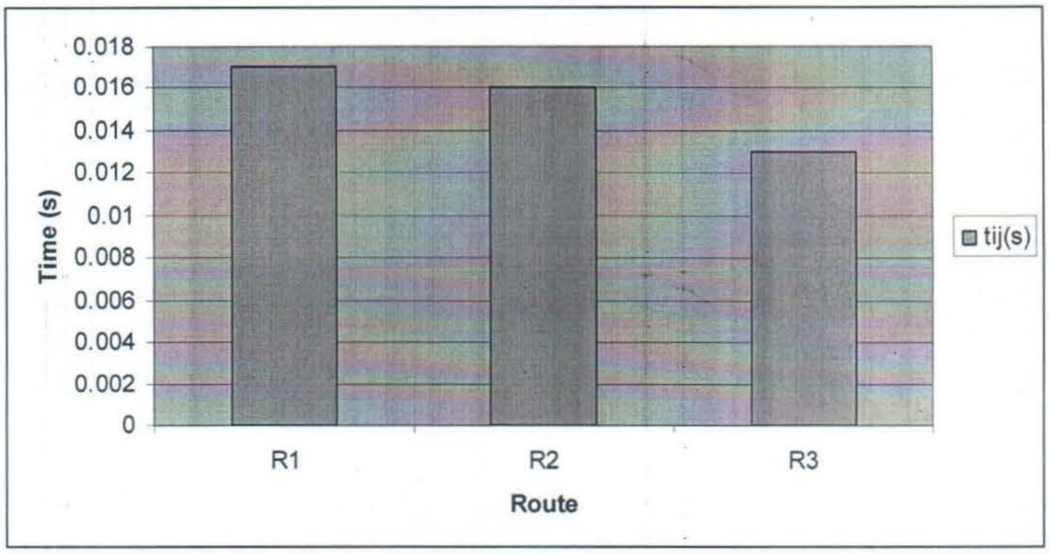


Figure 8: Graph showing Route against Time, t_{ij} (s)

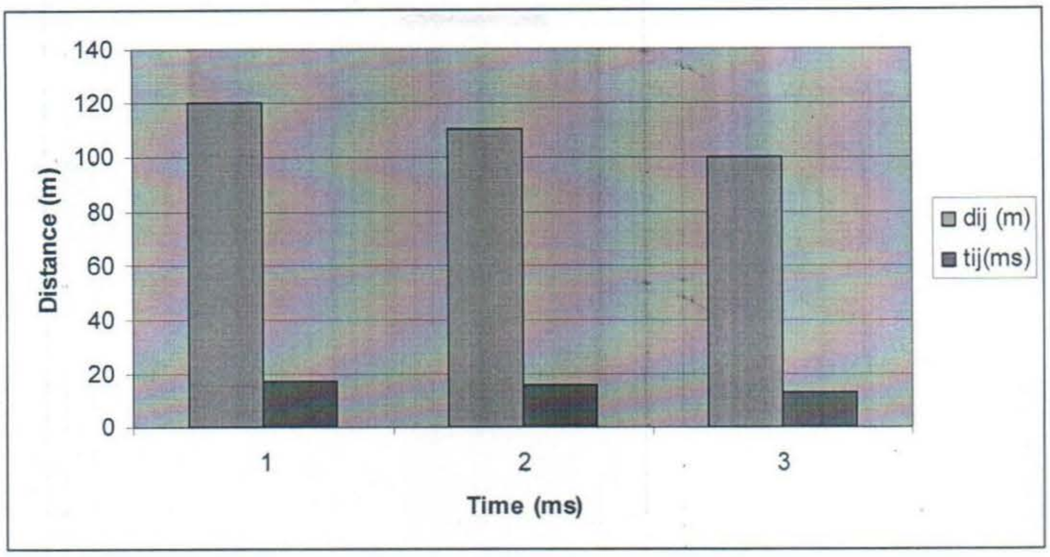


Figure 9: Graph showing Time (ms) against Distance (m)

4.2.1 Modified Dijkstra's algorithm

Here it is assumed that the direct distance between any two node (*dij*) in the network of *n* nodes is given, and all the distances are non-negative. The algorithm proceeds by assigning to all nodes a label which is either temporary or permanent. A temporary label represent an upper bound on the shortest distance from node 1 to that node; while a permanent label is the actual shortest distance from node 1 to that node.

Initially, the source node 1 is given a permanent label of zero. All other nodes (2, 3, . . ., *n*) are assigned temporary label equal to the direct distance from node 1 to the node in question. Any node which cannot be reached directly from node 1 is assigned a temporary label of ∞, while all the other nodes receive temporary labels equal to *dij*. The algorithm then makes these tentative node labels, one at a time, permanent labels. As soon as the sink node receives a permanent label, the shortest distance from the source node to the sink node is immediately known.

Iterative steeps of the algorithm

Pre-step: Initialize by assigning a permanent label of zero to the source node. All other node labels are temporary and are equal to the direct distance from the source node to that node. Select the minimum of these temporary labels and declare it permanent. In case of ties, choose as the new temporary label for that node (if the old temporary label is still minimal, then it will remain unchanged during this step).

Step 2: Select the minimum of all temporary labels, and declare it permanent. In case of ties, select any one of them (but exactly one), and declare it permanent. If this happen to be the sink node then terminate. Otherwise return to step 1.

To find the sequence of node in the shortest path from node 1 to node n, a label indicating the node from which each permanently labeled node was labeled should be available. Then by retracing the path backwards from the sink node to the source node, the minimal path may be constructed. An alternative method is to determine which nodes have permanent labels that differ by exactly the length of the connecting arc. Again by retracing the path backwards from n to 1, the shortest path may be found.

In this research, our network model in the University of Calabar environment was as follows;

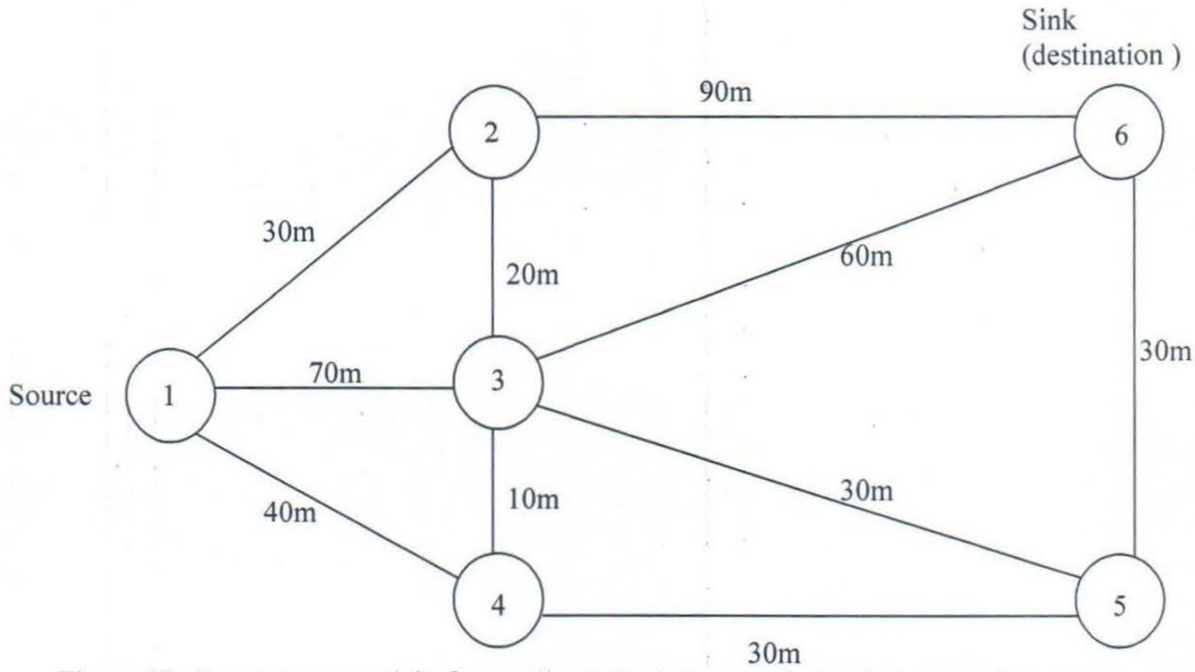


Figure 10: A prototype model of an undirected wireless packet switch network system in the University of Calabar.

Consider an undirected network shown in figure 10 where numbers along the arcs (i,j) represent distances between node i and j. Assume that the distance from i to j is the same as from j to i (i.e., all arcs are two way streets). The problem is to determine the shortest distance and the length of the shortest path from node 1 to node 6.

Solution

Initially node 1 is labeled permanently as zero, and all other nodes are given temporary labels equal to their direct distance from node 1. Thus, the node labels at step 1, denoted by L(1), are:

$L(1) = [\underset{\bullet}{0}, 30, 70, 40, \infty, \infty]$

(An asterisk indicates a permanent label).

At step 2, the smallest of the temporary labels is made permanent. Thus node 2 gets a permanent label equal to 30, and it is the shortest distance from node 1 to node 2. To understand the logic behind this step, consider any other path from node 1 to node 2 through an intermediate node j = 3, 4, 5, 6. The shortest distance from node 1 to node j will be atleast equal to 30 and $dj2$ is non-negative since all the distances are assumed to be non-negative. Hence any other path from node 1 to node 2 cannot have distance less than 30, and the shortest distance from node 1 to node 2 is 30. Thus at step 2 the node labels are;

$L(2) = [\underset{\bullet}{0}, \underset{\bullet}{30}, 70, 40, \infty, \infty]$

For each of the remaining nodes $j(j$ = 3, 4, 5, 6), compute a number which is the sum of the permanent label of node 2 and the direct distance from node 2 to node j. Compare this number with the temporary label of node j, and the smaller of the two values becomes the new tentative label for node *j*. For example, the new temporary label for node 3 is given by minimum of (30 + 20, 70) = 50

Similarly, for node 4, 5 and 6, the new temporary labels are 40, ∞ and 120 respective. Once again the minimum of the new temporary labels is made permanent. Thus, at step 3, node 4 gets a permanent label as shown below:

$L(3) = [\underset{\bullet}{0},\ \underset{\bullet}{30},\ 50,\ \underset{\bullet}{40},\ \infty,\ 120]$

Now using the permanent label of node 4, the new temporary labels of nodes 3, 5 and 6 are computed as 50, 70 and 120, respectively. Node 3 gets a permanent label and the node labels at step 4 are;

$L(4) = [\underset{\bullet}{0},\ \underset{\bullet}{30},\ \underset{\bullet}{50},\ \underset{\bullet}{40},\ 70,\ 120]$

It should be emphasized here that at each step, only the node which has been recently labeled permanent is used for further calculations. Thus, at step 5 the permanent label of node 3 is used to update the temporary labels of nodes 5 and 6 (if possible). Node 5 gets a permanent label and the node labels at step 5 are;

$L(5) = [\underset{\bullet}{0},\ \underset{\bullet}{30},\ \underset{\bullet}{50},\ \underset{\bullet}{40},\ \underset{\bullet}{70},\ 110]$

Using the permanent label of node 5, the temporary label of node 6 is changed to 100 and is made permanent. The algorithm now converges and terminates, and the shortest distance from node 1 to node 6 is 100. As a matter of fact, we have the shortest distance from node 1 to every other node in the network as shown below:

$L(6) = [\underset{\bullet}{0},\ \underset{\bullet}{30},\ \underset{\bullet}{50},\ \underset{\bullet}{40},\ \underset{\bullet}{70},\ \underset{\bullet}{100}]$

To determine the sequence of nodes in the shortest path from node 1 to node 6, we walk backwards from node 6. Node *j* (*j* = 1, 2, 3, 4, 5) proceeds node 6 if the difference between the permanent labels of node 6 and *j* equals the length of the arc from *j* to 6. This gives node 5 as its immediate predecessor. Similarly node 4 precedes node 5, and the immediate predecessor of node 4 is node 1. Thus the shortest path from node 1 to node 6 is;

$1 \rightarrow 4 \rightarrow 5 \rightarrow 6$

To compute the shortest path between every pair of nodes in the network, then we have to repeat the modified Dijkstra's algorithm four times taking node 2, 3, 4 and 5 as the source node.

From the undirected network diagram of the model of a wireless packet switch network system in the University of Calabar, the system was analyzed in six(6) stages using the following keynotes;

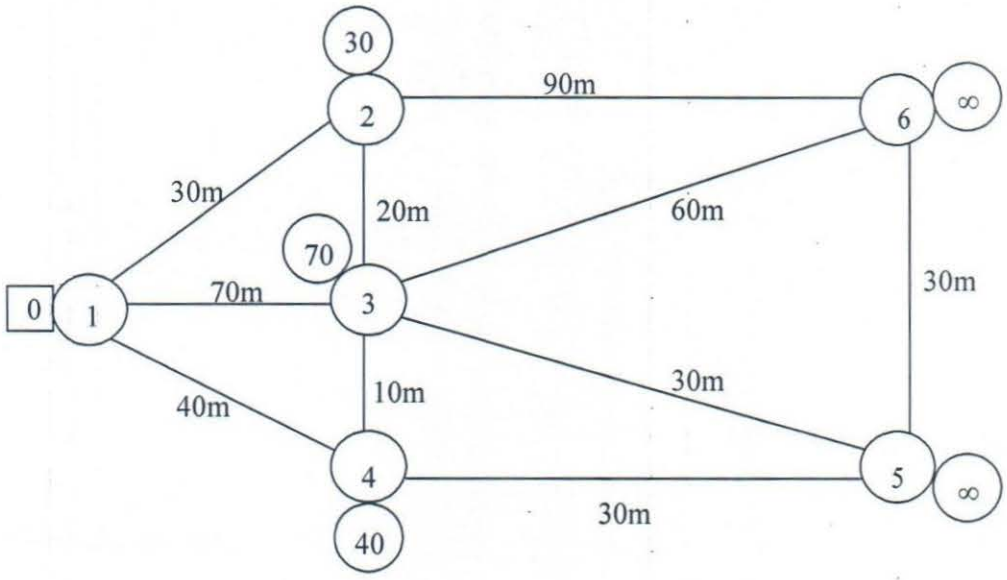
Where □ = permanent label of a node

O = temporary label of a node

At Stage 1:

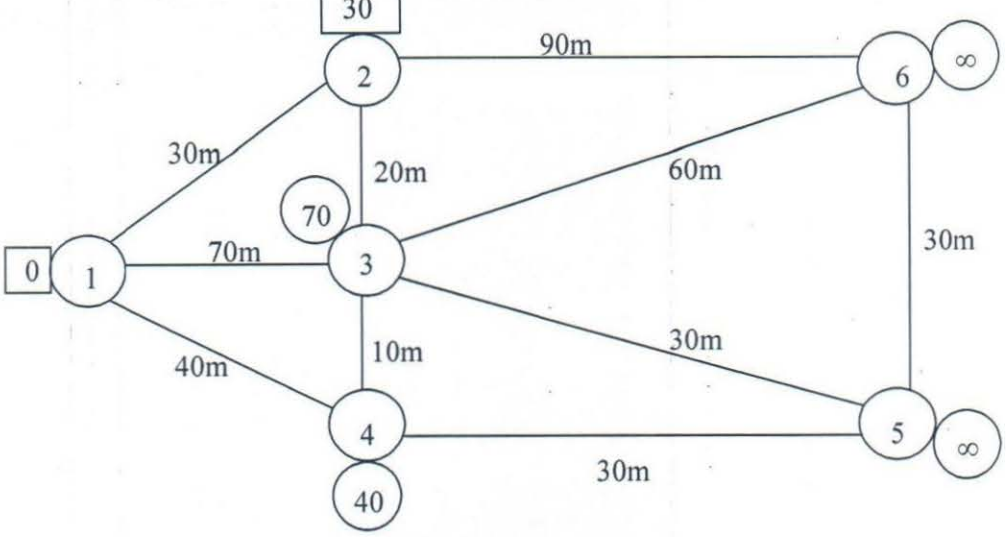
$L(1) = \{\underset{\bullet}{0},\ 30,\ 70,\ 40,\ \infty,\ \infty\}$

An asterisk's indicates a permanent label (*)



at Stage 2:

$$L\,(2) \;=\left\{\underset{\bullet}{0},\ \underset{\bullet}{30},\ 70,\ 40,\ \infty,\ \infty\right\}$$



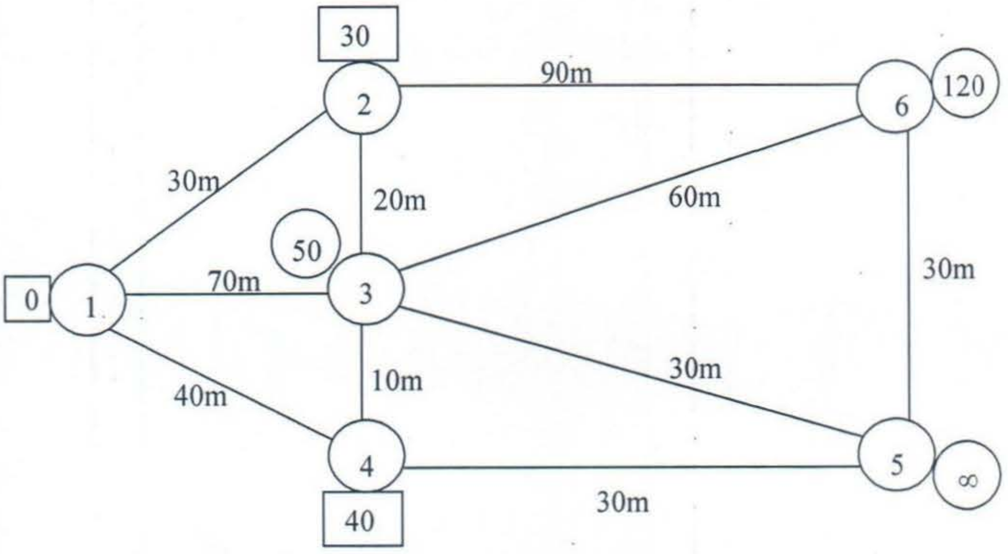
At Stage 3:

$$L\,(3) \;=\left\{\underset{\bullet}{0},\ \underset{\bullet}{30},\ 50,\ \underset{\bullet}{40},\ \infty,\ 120\right\}$$

route 1→2→3 = min {0 + 30 + 20} = 50

route 1→3 = min {0 + 70} = 70

Thus new temporary label of node 3 = 50



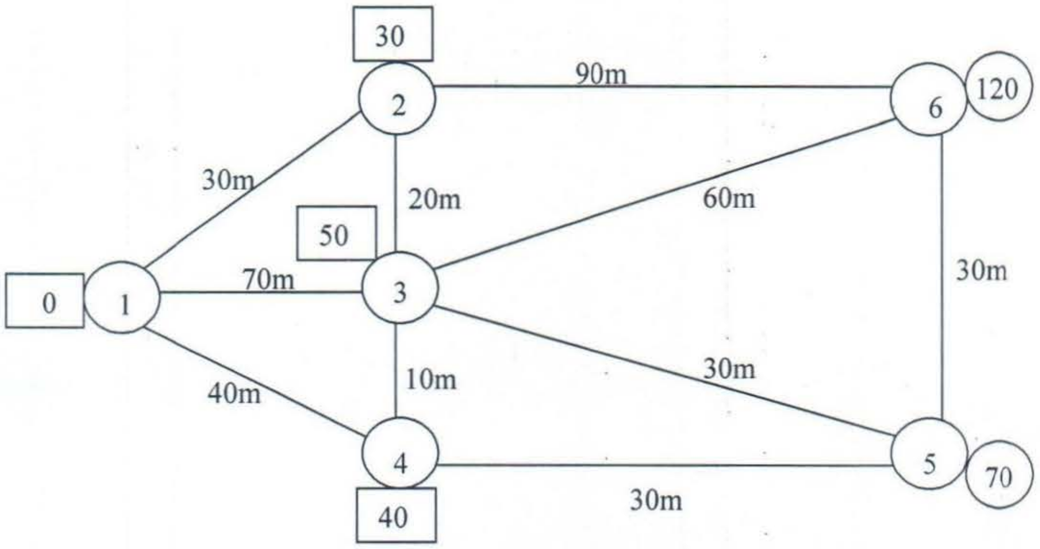
at Stage 4:

$L(4) = \{\underset{\bullet}{0},\ \underset{\bullet}{30},\ \underset{\bullet}{50},\ \underset{\bullet}{40},\ 70,\ 120\}$

route $1 \rightarrow 3 \rightarrow 5 = \min\{0 + 50 + 30\} = 80$

route $1 \rightarrow 4 \rightarrow 5 = \min\{0 + 40 + 30\} = 70$

Thus, the temporary label for node 5 = 70



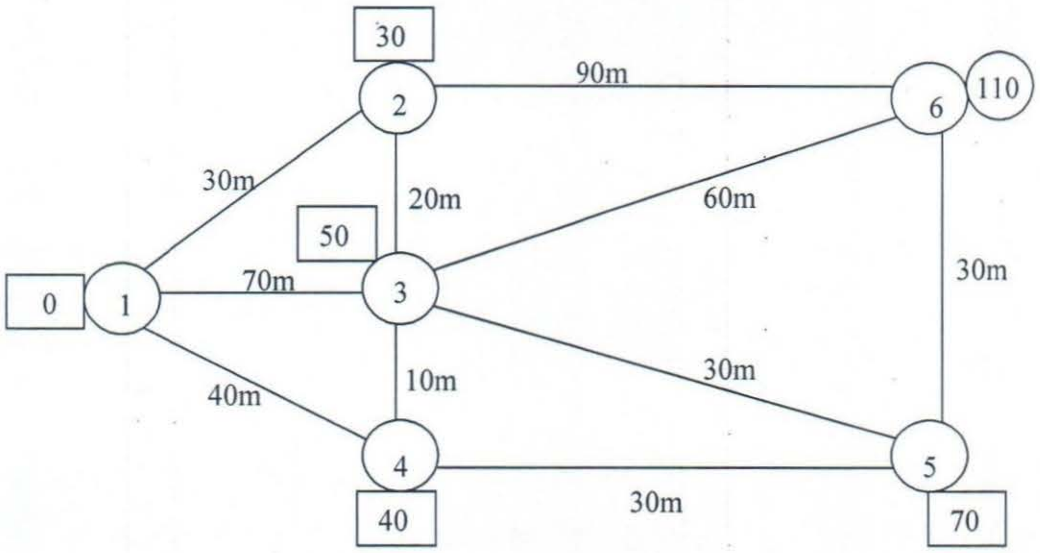
at Stage 5:

$L(5) = \{\underset{\bullet}{0},\ \underset{\bullet}{30},\ \underset{\bullet}{50},\ \underset{\bullet}{40},\ \underset{\bullet}{70},\ 110\}$

route $1 \rightarrow 2 \rightarrow 6 = \min\{0 + 30 + 90\} = 120$

route $1 \rightarrow 3 \rightarrow 6 = \min\{0 + 50 + 60\} = 110$

Thus, the new temporary label for node 6 = 110



at Stage 6:

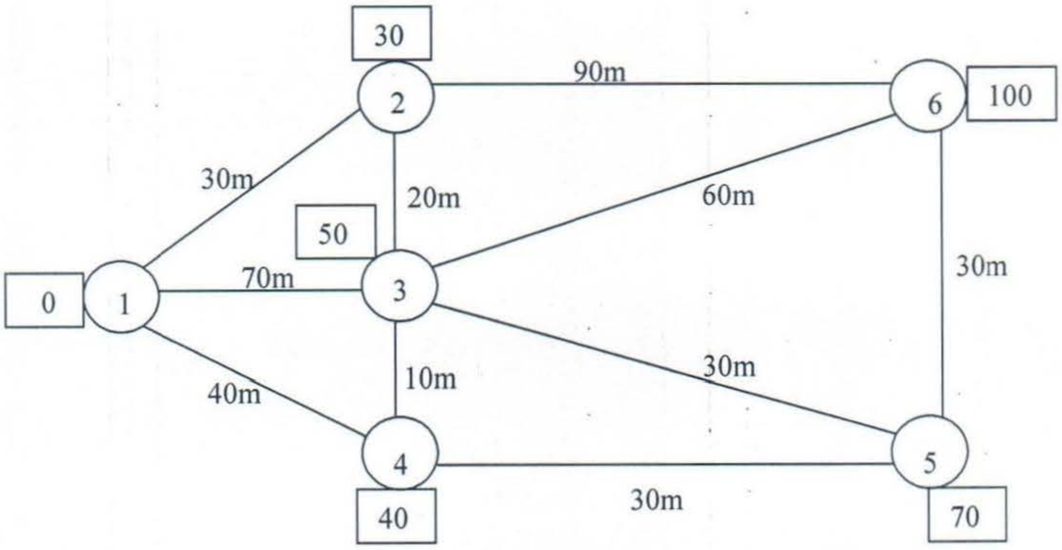
$L(6) = \{\underset{\bullet}{0}, \underset{\bullet}{30}, \underset{\bullet}{50}, \underset{\bullet}{40}, \underset{\bullet}{70}, \underset{\bullet}{100}\}$

route $1 \rightarrow 3 \rightarrow 6 = \min\{0 + 50 + 60\} = 110$

route $1 \rightarrow 4 \rightarrow 5 \rightarrow 6 = \min\{0 + 40 + 30 + 30\} = 100$

Thus, the new temporary label for node 6 = 100

Hence node 6 takes a permanent label of 100 and the algorithm terminates here.



Important areas (domains) where the modified Dijkstra's algorithm can be applied

Many more problems than you might at first think can be cast as shortest path problems, making modified Dijkstra's algorithm a powerful and general tool. For example;

i) The modified Dijkstra's algorithm is applied to automatically find directions between physical locations, such as driving directions on websites like Mapquest or Google Maps.

j) In a networking or telecommunication applications, modified Dijkstra's algorithm has been used for solving the min-delay path problem (which is the shortest path problem). For example in data network routing, the goal is to find the path for data packets to go through a switching network with minimal delay.

k) It is also used for solving a variety of shortest path problems arising in plant and facility layout, robotics, transportation, and Very Large Scale Integration (VLSI) design.

l) It can also be used for solving the following;

   a. Electricity flow

   b. Fluid flow in pipes

   c. Scheduling

4.3 Discussion of findings

This section of the research presents discussion of the findings emanating from this study as well as their interpretations. However, it was found that Route 1 shows a distance of 120m with a speed of 7.06m/s and time range of 0.017s to the destination and back through the Internet Control Message Protocol (ICMP Echo). Route 2 shows a distance of 110m with a speed of 6.88m/s and time range of 0.016s to the destination and back through the Internet Control Message Protocol (ICMP Echo). Finally, route 3 shows a distance of 100m with a speed of 7.69m/s and time range of 0.013s to the destination and back through the Internet Control Message Protocol (ICMP Echo). This indicates that, distance is directly proportional to time, and inversely proportional to the speed taken for data to move to and fro in the packet switch.

The following conclusion were drawn, based on the results.

1. The shortest path from the simulation of the modified Dijkstra's algorithm in the wireless packet switch network was Node 1 → Node 4 → Node 5 → Node 6 and the packet uses ICMP ECHO to traversed back to the source node. In computer networks the routing is based on the shortest path problem which minimized the overall cost of setting up computer networks and increase system efficiency.
2. The main drawback of the traditional Dijkstra's algorithm was the consumption of huge memory which was as a result of large infinite values in the network.
3. The open shortest path first (OSPF) mechanism protocol suit introduced for routers provided effective congestion control in TCP/IP network through high link utilization, regulate queues, bounded delay and delay variation, minimal packet losses, adequate and effective differentiation among different drop procedure's traffic, fast system response and robustness to varying system dynamics.
4. The open shortest path first (OSPF) provide quality of services and high link utilization, with minimal losses, and bounded queue fluctuations and delays.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

This chapter deals with the summary of the entire research work which is presented under the following sub-headings.

5.1 Summary

5.2 Conclusion

5.3 Recommendations/ Suggestions for future work

5.1 Summary

The study was designed to analyse the shortest pathway and time determination in a wireless packet switch network system in the University of Calabar environment. Six nodes (nodes1→node2 ..., node6) were prepared to guide the study and were tested with Open Shortest Path First (OSPF) using modified Dijkstra algorithm. The combination of both Open Shortest Path First (OSPF) and modified Dijsktra algorithm provides useful problem solving method in networking approach by coding of the algorithm and the ability to manage and control the network which was becoming more difficult because of the increase demand of the internet for time/delay-sensitive applications with differing quality of service requirements (e.g voice over IP, video streaming, peer-to-peer, interactive games, etc.). Some additional OSPF technique was used to evaluate the routers to complement the endpoint congestion control methods and the recently led concept of active queue management.

This was in line with Bast *et al.* (2007) postulation of the possible criteria of minimized travel time, total path length or estimated travel cost are similar in GPS devices; as these usually have a limited amount of memory and CPU power (Delling, 2008). Due to these backdrop, several devices was used differently to solve various kinds of wireless connection in order to query a web service, which computes the desired path using more sophisticated algorithms than those available on the potable device, because, endpoint demand and interest was fastest path to reach a destination. The reemerging problems, which are; network topology dynamics and reconfigurability, user terminal and service mobility; large network size and user populations; diversity of applications (QoS) requirements were put in place. modified Dijkstra algorithm was use to determine the minimal length from the starting point of vertex. Open Shortest Path First (OSPF) allowed the entry in one area and represented in other areas of backbone which was associated with a summary address and a mask. Mask was applied to the address to match the summary address with routing-entry used.

5.2 Conclusion

The simulated results and analysis of this research study shows that, the modified Dijkstra's algorithm have a better performance than the traditional Dijkstra's algorithm that have huge consumption of memory due to huge infinite values. The choice of modified Dijkstra's algorithm (MDA) and open shortest path first (OSPF) was to ensure total eradication of the routing problem in the wireless packet switch network in the University of Calabar environment. MDA handles the SP cost while OSPF handles (1) QoS (2) high link utilization with minimal losses (3) bounded queue fluctuations and delay. The data structure handles the minimal memory core utilization and runtime utilization of the algorithm. The modified Dijkstra's algorithm uses the linked list priority queue and minimal weight cost on the system. This development greatly increase the efficiency of the system. The shortest path was found to be:

Node 1 → Node 4 → Node 5 → Node 6 and back using the ICMP ECHO.

Two experiments were conducted in the network: tracet and ping test. Three alternate routes were chosen with the help of the tracet experiment. The ping test was conducted on these three routes to determine the time taken by a packet to traverse from source node to the destination node in the network. The results obtained from the ping test was as follows:

Route 1: Node 1 → Node 2 → Node 6, $d_{ij} = 120$

Route 2: Node 1 → Node 4 → Node 3 → Node 6, $d_{ij} = 110$

Route 3: Node 1 → Node 4 → Node 5 → Node 6, $d_{ij} = 100$

From the network simulated results it was found that;

Route 1, having a *dij* of 120m took 0.017s for the packet to traverse from the source node to the destination node and back.

Route 2, having a *dij* of 110m took 0.016s for the packet to traverse from the source node to the destination node and back.

Route 3, having a *dij* of 100m took 0.013s for the packet to traverse from the source node to the destination node and back.

Thus, from the analysis of the results it was found that the shortest path in the network took the shortest time for the packet of 30MB to traverse from the source node to the destination node thus justify the research aim and objective.

5.3 Recommendations / Suggestions for future work

Even if the speed and versatility showed by the modified shortest paths algorithm proposed in this research work should be sufficient for most practical applications, there is still much room for research in the field. For some applications, it would be desirable to have very fast query times and no additional overhead when changing the cost functions. Examples of this are route planners which use different cost functions depending on the vehicle type that is querying the path computing service. Although this seems an impossible challenge, it is still an interesting subject of research, maybe assuming some restrictions on the cost functions to simplify the problem. If the cost functions are similar, the Open Shortest Path First (OSPF) algorithm discussed in research has showed promising result. Another interesting direction for future research is multicriteria optimization. Routing applications in general networks (not necessarily road networks) often have to deal with several objective functions that the user would like to minimize; e.g., traveling time and number of connections for airport routing, or travelling time and motorway fees for road networks. How to formalize this problem is unclear.

Some approaches rely on finding all the Pareto optima, and let the user choose among them. However, computing all the Pareto optima is a difficult task, and could greatly benefit from speedup techniques. The researcher believes that the techniques presented in this research could be used as a building brick for efficient algorithms in the multi-objective case. At the moment of finalizing this research, the researcher is aware that an extension of the Open Shortest Path First (OSPF) algorithm shows very good preliminary results. The greatest drawbacks of OSPF are its long preprocessing time and the capability of dealing with static scenarios only. It would be interesting to hybridize OSPF with the techniques proposed in this work, so as to be able to perform efficient multicriteria optimization on dynamic networks.

There is of course room for improvement in the algorithm itself. It would be desirable to be able to find a first feasible solution as soon as possible. The starting point provided to the solver determines the chances of finding such a solution. The researcher believe that employing constraint programming techniques to round to the nearest integer as many fractional integer variables as possible, while still maintaining constraint feasibility, could greatly help. The researcher also plan to test different solvers to evaluate performance as a stand-alone heuristic, and to reduce the number of parameters of the algorithm; preliminary tests show that a different combination of solvers through the main phase of the algorithm yields significantly better solution quality with minimal CPU time.

Cornu'ejols, G., Liberti, L. & Nannicini, G. Improved strategies for branching on general disjunctions. Technical Report 2071, Optimization Online, 2008. Available from World Wide Web: http://www.optimization-online.com.

Creswell, J. (2003). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Thousand Oaks, California: Sage Publications

David M. (2012). Mount "Design and Analysis of Computer Algorithms" Department of Computer Science.

David, A., Bader, S., Kamesh, M. & Milena, M. (2007). Approximating Betweenness Centrality. College of Computing, Georgia Institute of Technology. Bader.kintali,kamesh,mihali@cc.gatecch.edu

Delling, D. & Nannicini, G. (2008). Bidirectional Core-Based Routing in Dynamic Time-Dependent Road Networks. In S.-H. Hong, H. Nagamochi, and T. Fukunaga, editors, *Proceedings of the 19th International Symposiumon Algorithms and Computation (ISAAC 08)*, volume 5369 of *Lecture Notes in Computer Science*, pp. 813–824.

Delling, D. & Nannicini, G. (2008). Core routing on dynamic time-dependent road networks. Technical Report 2156, Optimization Online, 2008. Available From WorldWideWeb: http://www.optimization-online.com.

Delling, D. (2008). Time-Dependent SHARC-Routing. In *Proceedings of the 16th Annual European Symposiumon Algorithms (ESA'08)*, volume 5193 of *Lecture Notes in Computer Science*, pp. 332–343.

Delling, D. (2009). *Engineering and Augmenting Route Planning Algorithms*. PhD thesis, Fakult¨at f ¨ur Informatik, Universit¨at Fridericiana zu Karlsruhe (TH), Germany.

De Vaus, D. A. (2006). Research design in social research. London: SAGE, 2001; Troochim, William, M. K. *Research Methods Knowledge Base*

Dijkstra, E. W. (2010). "A note on two problems in connexion with graphs". *Numerische Mathematik **1**: 269–271.*

Fortz, B. & Thorup, M. (2002). "Optimizing OSPF/IS-IS weights in a changing world," *IEEE Journal on Selected Areas in Communications, 20* (5), 756–767.

Frana, Phil (August 2010). "An Interview with Edsger W. Dijkstra". Communications of the ACM 53 (8): 41–47.

GeeksforGeeks "Dynamic Programming" A computer science portal for geeks http://www.geeksforgeeks.org/dynamicprogramming-set-23-bellman-ford-algorithm/

Goldberg, A. & Harrelson, C. (2005). Computing the shortest path: A meets graph theory. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*, pp. 156–165, Philadelphia, SIAM.

Goldberg, A. V. Kaplan, H. & Werneck, R. F. (2008). Shortest Path Algorithms with Preprocessing. In C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors, *Shortest Paths: Ninth DIMACS Implementation Challenge*, DIMACS Book. American Mathematical Society.

Goldberg, A. V., Kaplan, H. & Werneck, R. F. (2008). Shortest Path Algorithms with Preprocessing. In C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors, *Shortest Paths: Ninth DIMACS Implementation Challenge*, DIMACS Book. American Mathematical Society.

Hansen, P., Mladenovi´c, N. & Uroˇsevi´c. D. (2006). Variable neighbourhood search and local branching. *Computers and Operations Research*, 33(10):3034–3045.

Hassan, M., & Sirisena, H. (2001). Optimal control of queues in computer networks. *IEEE International Conference on Communications.*

Hopfield, J. J. (1982). Neutral networks and physical systems with emergent collective computational abilities. *Procedures for Natural Academic Science, 79*, 2558-2558

Hopfield, J. J. & Tank, D. W. (1985). Neural computations of decision in optimization problems, *Biol. Cybern.*, 52, 141-152.

Hougardy, S. (2010). The Floyd-Warshall, ―Algorithm on Graphs with Negative Cycles‖, University of Bonn.

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Algorithm

https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

https://en.wikipedia.org/wiki/Network_congestion#Congestion_control (2015).

ICCRG, (2006). Internet Congestion Control Research Group. http://oakham.cs.ucl.ac.uk/mailman/listinfo/iccrg

Ikeda, T., Tsu, M., Imai, H., Nishimura, S., Shimoura, H., Hashimoto, T., Tenmoku, K. & Mitoh, K. (2004). A fast algorithm for finding better routes by ai search techniques. In *Proceedings for the IEEE Vehicle Navigation and Information Systems Conference*, pp. 291–296.

Internetworking Technology Handbook (2002). Internet Protocols (IP), Cisco Sytems, Inc.

Irny, S.I. and Rose, A.A. (2005) "Designing a Strategic Information Systems Planning Methodology for Malaysian Institutes of Higher Learning (isp- ipta). *Issues in Information System,* 5, 1.

Jacobson, V. (1988). Congestion avoidance and control. *Proceedings of ACM SIGCOMM 1998,* 314-329.

Kairanbay, M. & Hajar, M. J. (2013). A review and evaluations of shortest path algorithms. *International Journal of Scientific & Technology Research, 2*, 6.

Katabi, D., Handley, M., & Rohrs, C. (2002). Congestion control for high bandwidth-delay product networks. *Proceedings of ACM SIGCOMM 2002.*

Kenneth, H. R. (2003). Discrete Mathematics and its applications, 5th Edition. Addison Wesley

Kerner, B. S. (2004). *The Physics of Traffic*. Springer, Berlin.

Keshav, S. (2001). Congestion Control in Computer Networks. *Ph.D. thesis*, University of California Berkeley

Kiran, Y. & Ranjit B., (2010). An Approach to Find Kth Shortest Path using Fuzzy Logic. *International Journal of Computational Cognition, 8*(1)

Lavor, C., Liberti, L. & Maculan. N. (2006). Computational experience with the molecular distance geometry problem. In J. Pint´er, editor, *Global Optimization: Scientific and Engineering Case Studies.*

Li, T., Qi, L. & Ruan, D. (2008). An Efficient Algorithm for the Single-Source Shortest Path Problem in Graph Theoryl, Proc. of 3rd *International Conference on Intelligent System and Knowledge Engineering,* 1: 152-157.

Liberti, L., Nannicini, G. & Mladenovi´c. N. (2008). A good recipe for solving MINLPs. In V. Maniezzo, T. Stuetze, and S. Voss, editors, *MATHEURISTICS: Hybridizing metaheuristics and mathematical programming*, Operations Research/Computer Science Interface Series.

Martinez, J. C., Flich, J., Robles, A., Lopez, P. &Duato, J. (2004). "Supporting adaptive routing in IBA switches", Systems Architect 49 pp. 441-449.

Mehlhorn, Kurt; Sanders, Peter *(2008). Algorithms and Data Structures: The Basic Toolbox. Springer.*

Nannicini, G., Baptiste, P., Barbier, G., Krob, D. & Liberti, L. (2008). Fast paths in large-scale dynamic road networks. *Computational Optimization and Applications.*

Nannicini, G., Delling, D., Liberti, L. & Schultes, D. (2008). Bidirectional: A research for time-dependent fast paths. InMcGeoch (104), pages 334–346.

Nannicini, G., Delling, D., Liberti, L. & Schultes. D. (2008). Bidirectional: A research on time-dependent road networks. Technical Report 2154, Optimization Online, 2008. Available from World Wide Web: http://www.optimization-online.com.

Networks with Time-Dependent, Stochastic Arc Costs. IEEE International Conference on Systems, Man, and Cybernetics. Humans, Information and Technology 2, 1716-1721.

Newman, M. E. J. (2010). *Networks: An Introduction.* Oxford, UK: Oxford University Press.

Pióro, M. & Medhi, D. (2004). *Routing, flow, and capacity design in communication and computer networks*, Mor-gan Kaufmann, CA, San Diego.

Pitsillides, A., & Sekercioglu, A. (2000). Congestion Control. In Pedrycz, W. & Vasilakos, A. V. (Eds.), *Computational Intelligence in Telecommunications Networks* (pp. 109-158). Boca Raton, FL: CRC Press, ISBN: 0-8493-1075-X.

Puzis, R., Yagil, D., Elovici, Y. & Braha, D. (2009). Collaborative attack on Internet users' anonymity, Internet Research 19(1)

Pyrga, E., Schulz, F., Wagner, D. & Zaroliagis, C. (2007). Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12: 2-4.

Ramakrishnan, K., Floyd, S., & Black, D. (2001). The addition of explicit congestion notification (ECN) to IP. *Request for Comments RFC 3168,* Internet Engineering Task Force.

Rétvári, G. & Cinkler, T. (2004). "Practical OSPF traffic engi-neering," *IEEE Communications Letters, 8*(11), 689–691.

Rowe, S. & Schuh, M. (2005). Computer Networking, Pearson, Prentice Hall.

Sanders, P. & Schultes, D. (2005). Highway hierarchies hasten exact shortest path queries. In G. Stolting Brodal and S. Leonardi, editors, *13th Annual European Symposium on Algorithms (ESA 2005)*, volume 3669 of *Lecture Notes in Computer Science*, pp. 568–579.

Sanders, P. & Schultes, D. (2006). Engineering highway hierarchies. In *ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 804–816.

Schultes, D. (2005). Fast and exact shortest path queries using highway hierarchies. *Master Thesis, Informatik, Universit¨at des Saarlandes*.

Silla, F. & Duato, J. (2000). "High –performance routing in networks of workstations with irregular Topology," *IEEE Transfer Parallel Distribution Systems, 11*(7): 699-719.

Skiena, S. & Revilla, A. (2014). Programming Challenges, The Programming Contest Training Manuall pp. 248 – 250.

Soltani, A. R., Tawfik, H. & Goulermas, J. Y (2002). "Path planning in construction sites: Performance evaluation of the dijkstra, a* and GA search algorithms," *Advanced Engineering Informatics, 16,* (4), 291–303, 2002.

Stevens, W. (1997). TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. *Request for Comments RFC 2001,* Internet Engineering Task Force.

Vaibhavi, P. & Chitra, B. (2014). A survey paper of Bellman-Ford algorithm and Dijkstra algorithm for finding shortest path in GIS application ME in Information Technology, Kalollnstitute of Technology & Research Center, Gujarat, India. *International Journal of P2P Network Trends and Technology, 5*

Waller, S. T. & Ziliaskopoulos, A.K. (2002).On the Online Shortest Path Problem with Limited Arc Cost Dependencies. Networks, 40(4), 216-227.

Wang, Z. (2001). "Internet QoS: Architectures and mechanisms for quality of service," Academic Press, CA, San Diego.

Wei, W., Hai, Z., Hui, L., Jun, Z., Peng, L., Zheng, L., Naiming, G., Jian, Z., Bo, L., Shuang, Y., Hong, L. & Kunzhan, Y. (2010). Research on LFS Algorithm in Software Network. Information Science and Engineering Northeastern University, Shenyang, China. *Journal of Software Engineering & Applications, 3*: 185-189.

Xi, C., Qi, F. & Wei, L. (2006). A New Shortest Path Algorithm based on Heuristic Strategy,‖ Proc. of the 6th *World Congress on Intelligent Control and Automation, 1*: 2531–2536.

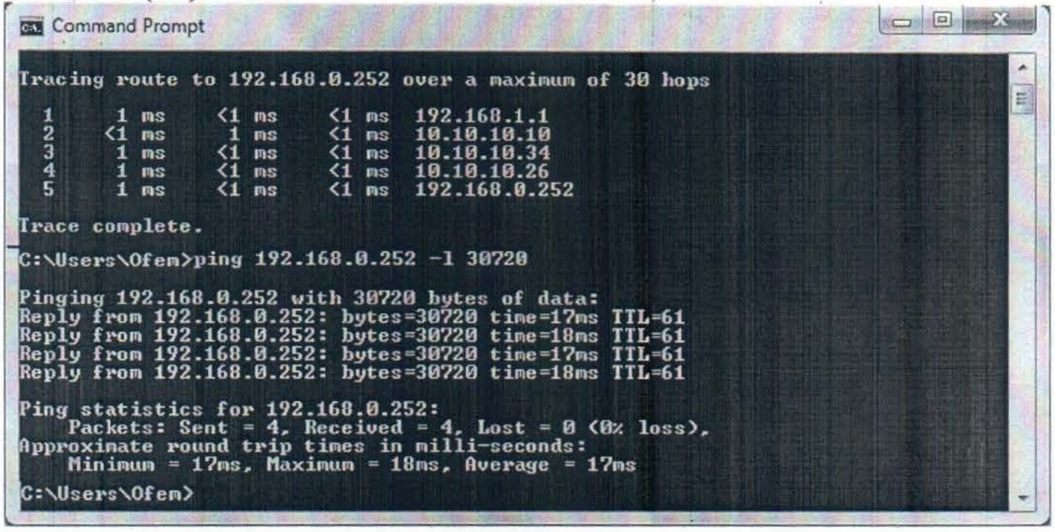
Zhang, X., Zhao, H., Zhang, W. B. & Li, C. (2006). "Research on CFR algorithm for Internet," *Journal on Communications, 27*(9)

APPENDIX 1

Ping test output result

After setting up the network, the default modified Dijkstra's algorithm populated the routing tables shown in the different routes (R1-R3)

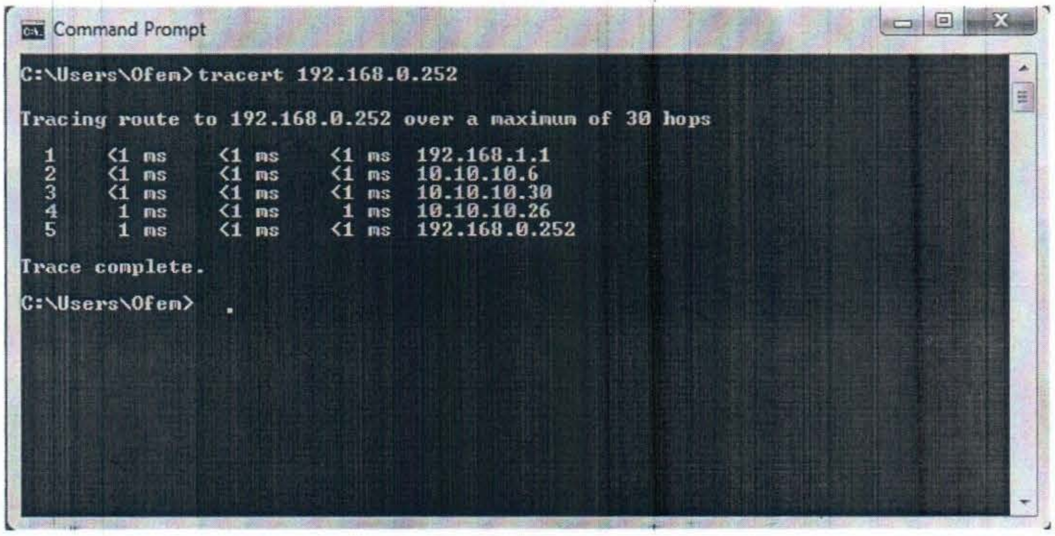
Route one (R1)



Command Prompt

```
Tracing route to 192.168.0.252 over a maximum of 30 hops

  1     1 ms    <1 ms    <1 ms  192.168.1.1
  2    <1 ms     1 ms    <1 ms  10.10.10.10
  3     1 ms    <1 ms    <1 ms  10.10.10.34
  4     1 ms    <1 ms    <1 ms  10.10.10.26
  5     1 ms    <1 ms    <1 ms  192.168.0.252

Trace complete.

C:\Users\Ofem>ping 192.168.0.252 -l 30720

Pinging 192.168.0.252 with 30720 bytes of data:
Reply from 192.168.0.252: bytes=30720 time=17ms TTL=61
Reply from 192.168.0.252: bytes=30720 time=18ms TTL=61
Reply from 192.168.0.252: bytes=30720 time=17ms TTL=61
Reply from 192.168.0.252: bytes=30720 time=18ms TTL=61

Ping statistics for 192.168.0.252:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 17ms, Maximum = 18ms, Average = 17ms

C:\Users\Ofem>
```

Packet path: Source-node1 –node 3-node6-Destination

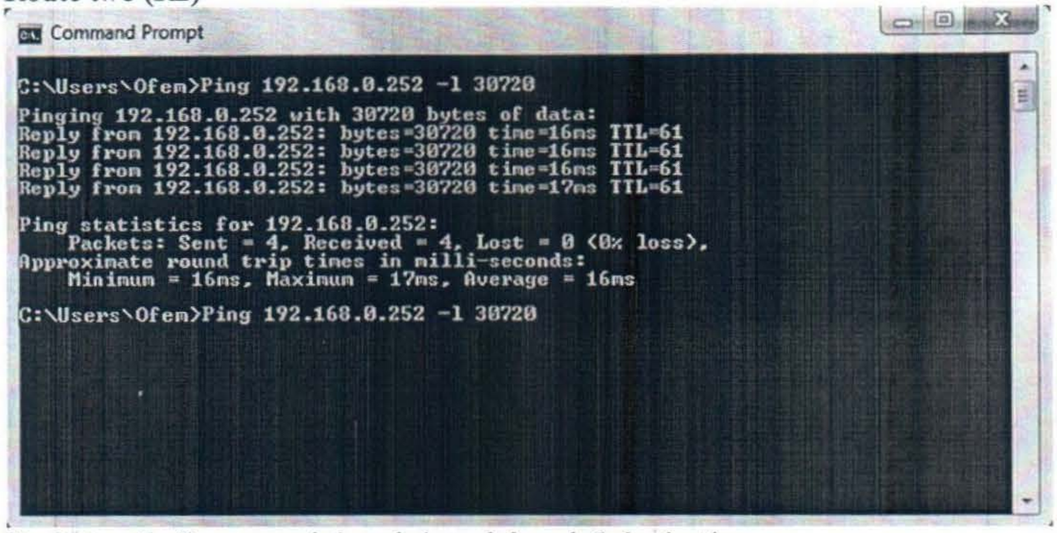


Command Prompt

```
C:\Users\Ofem>tracert 192.168.0.252

Tracing route to 192.168.0.252 over a maximum of 30 hops

  1    <1 ms    <1 ms    <1 ms  192.168.1.1
  2    <1 ms    <1 ms    <1 ms  10.10.10.6
  3    <1 ms    <1 ms    <1 ms  10.10.10.30
  4     1 ms    <1 ms     1 ms  10.10.10.26
  5     1 ms    <1 ms    <1 ms  192.168.0.252

Trace complete.

C:\Users\Ofem>
```

Load test with a load of 30MB

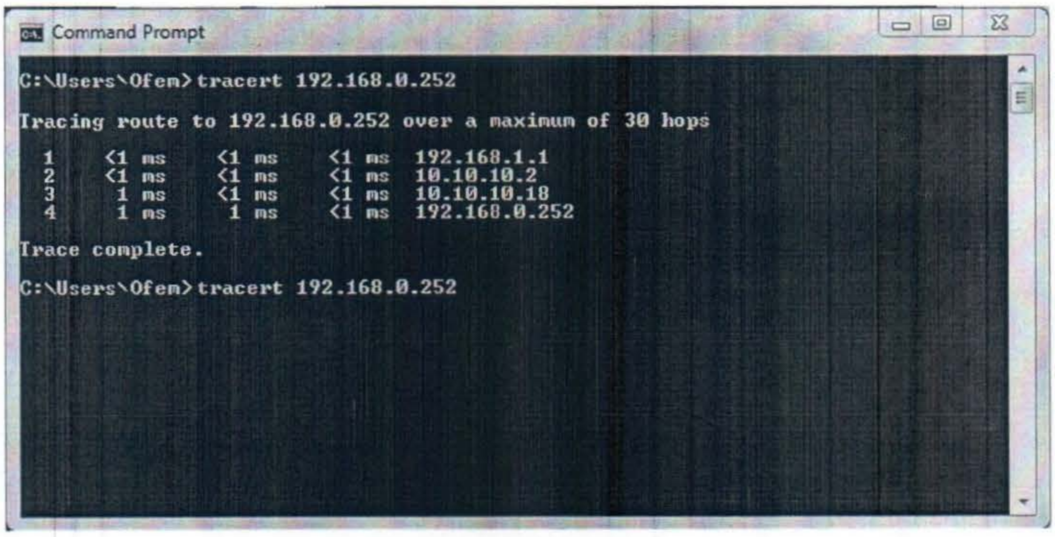
Route two (R2)



Command Prompt

```
C:\Users\Ofem>Ping 192.168.0.252 -l 30720

Pinging 192.168.0.252 with 30720 bytes of data:
Reply from 192.168.0.252: bytes=30720 time=16ms TTL=61
Reply from 192.168.0.252: bytes=30720 time=16ms TTL=61
Reply from 192.168.0.252: bytes=30720 time=16ms TTL=61
Reply from 192.168.0.252: bytes=30720 time=17ms TTL=61

Ping statistics for 192.168.0.252:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 16ms, Maximum = 17ms, Average = 16ms

C:\Users\Ofem>Ping 192.168.0.252 -l 30720
```

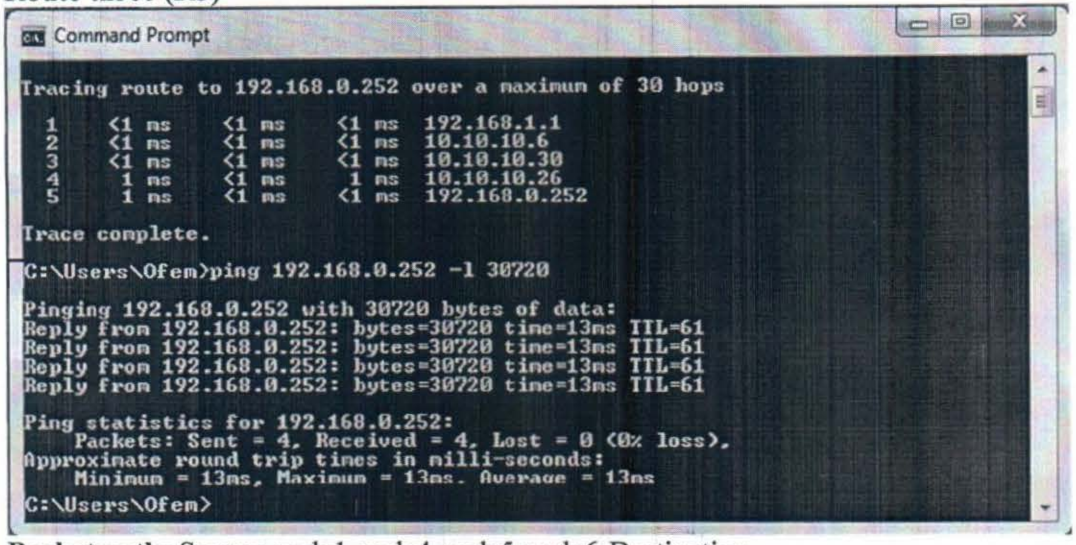
Packet path: Source-node1-node4- node3-node6-destination



Command Prompt

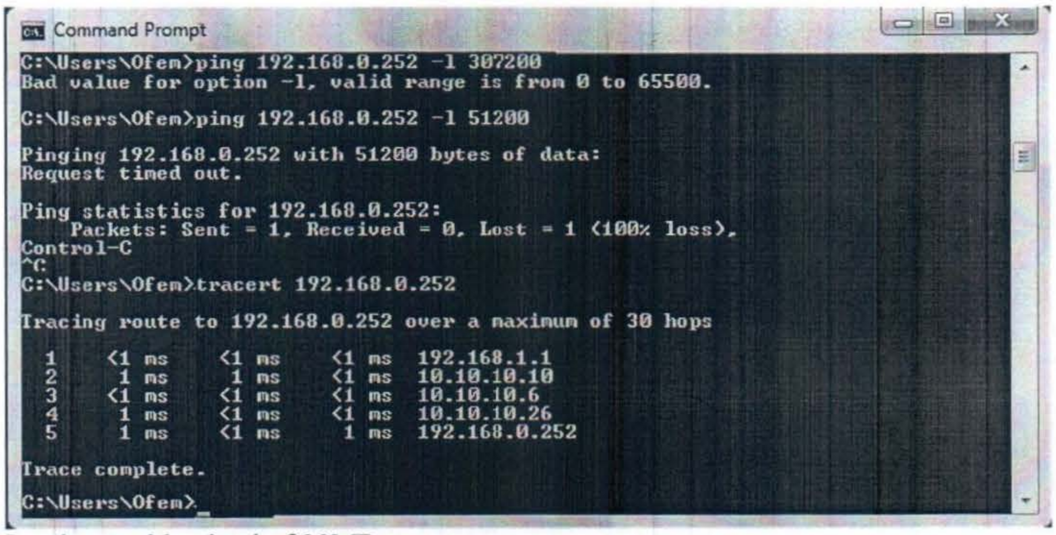
```
C:\Users\Ofem>tracert 192.168.0.252

Tracing route to 192.168.0.252 over a maximum of 30 hops

  1    <1 ms    <1 ms    <1 ms  192.168.1.1
  2    <1 ms    <1 ms    <1 ms  10.10.10.2
  3     1 ms    <1 ms    <1 ms  10.10.10.18
  4     1 ms     1 ms    <1 ms  192.168.0.252

Trace complete.

C:\Users\Ofem>tracert 192.168.0.252
```

Route three (R3)



```
Tracing route to 192.168.0.252 over a maximum of 30 hops

  1    <1 ms    <1 ms    <1 ms  192.168.1.1
  2    <1 ms    <1 ms    <1 ms  10.10.10.6
  3    <1 ms    <1 ms    <1 ms  10.10.10.30
  4     1 ms    <1 ms     1 ms  10.10.10.26
  5     1 ms    <1 ms    <1 ms  192.168.0.252

Trace complete.

C:\Users\Ofem>ping 192.168.0.252 -l 30720

Pinging 192.168.0.252 with 30720 bytes of data:
Reply from 192.168.0.252: bytes=30720 time=13ms TTL=61
Reply from 192.168.0.252: bytes=30720 time=13ms TTL=61
Reply from 192.168.0.252: bytes=30720 time=13ms TTL=61
Reply from 192.168.0.252: bytes=30720 time=13ms TTL=61

Ping statistics for 192.168.0.252:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 13ms, Maximum = 13ms, Average = 13ms

C:\Users\Ofem>
```

Packet path: Source-node1-node4-node5-node6-Destination



```
C:\Users\Ofem>ping 192.168.0.252 -l 307200
Bad value for option -l, valid range is from 0 to 65500.

C:\Users\Ofem>ping 192.168.0.252 -l 51200

Pinging 192.168.0.252 with 51200 bytes of data:
Request timed out.

Ping statistics for 192.168.0.252:
    Packets: Sent = 1, Received = 0, Lost = 1 (100% loss),
Control-C
^C
C:\Users\Ofem>tracert 192.168.0.252

Tracing route to 192.168.0.252 over a maximum of 30 hops

  1    <1 ms    <1 ms    <1 ms  192.168.1.1
  2     1 ms     1 ms    <1 ms  10.10.10.10
  3    <1 ms    <1 ms    <1 ms  10.10.10.6
  4     1 ms    <1 ms    <1 ms  10.10.10.26
  5     1 ms    <1 ms     1 ms  192.168.0.252

Trace complete.

C:\Users\Ofem>_
```

Load test with a load of 30MB

APPENDIX II

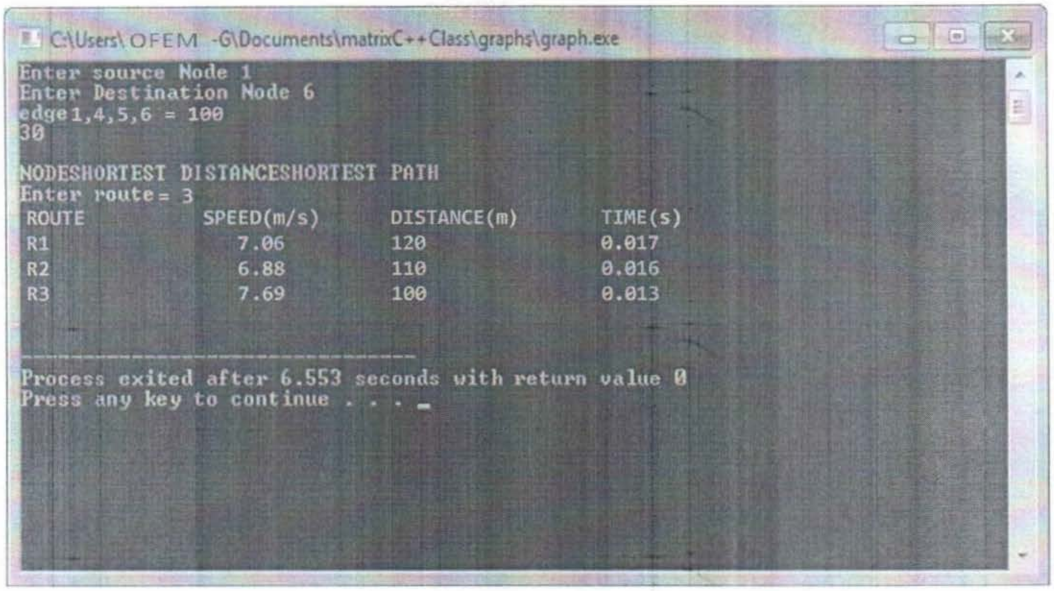
```
// C++ class to represent a Graph
#include <iostream>
#include <iomanip>
using namespace std;

class Graph
{
	int * Amatrix;

	public:
		int n;

		int size(int m)
		{
			return (m*(m-1)/2);
		}

		int row(int m)
		{
			return ((m-1)*(2*n-m)/2);
		}

		int column(int v1, int v2)
		{
			return (v1 - v2 - 1);
		}

		Graph(int n)
		{
			this->n = n;
			Amatrix = new int[size(n)];

			// fill matrix with infinities. in this case, infinity
is taken to be 10000000
			int sz = size(n);
			for(int i = 0; i<sz; i++)
				Amatrix[i] = 10000000;
		}

		Graph() {
		}

		int index(int v1, int v2)
		{
			if( valid(v1,v2))
			{
				if(v1<v2)
					return (row(v1) + column(v1,v2));
				else if(v1==v2) return 0;
				return (row(v2) + column(v2,v1));
			}
			return -1;
		}

		bool valid(int v1, int v2)
		{
			if((v1>0 && v1<=n) && (v2>0 && v2<=n))
```

```
                        return true;
                return false;
            }

            bool input(int v1, int v2, int weight)
            {
                if( valid(v1,v2) )
                {
                        if(v1<v2)
                        {
                                Amatrix[index(v1,v2)] = weight;
                                return true;
                        }
                        else
                        {
                                Amatrix[index(v2,v1)] = weight;
                        }
                }

                return false;
            }

        int edge(int v1, int v2)
        {
            if(valid(v1,v2))
            {
                if(v1<v2) return Amatrix[index(v1,v2)];
                return Amatrix[index(v2,v1)];
            }
            return -1;          // return a negative number if the
vertices are not in range
        }

};


// Dijkstra

class Dijkstra
{
        public:
            Graph graph;
            int n;
            int *L;     // Array containing Labaels
            int E; // Evaluation Node
            bool *visited;      // Array containing flag values that
determines wheither a Node Label is temporary or permanent
            int source, destination;  // source and destination nodes
            int *path;          // Array containing the paths from source
node to destinaton nodes

            Dijkstra(Graph g)
            {
                graph = g;
            }

            void shortest_path(int *l, int *pat, int j, int i)
            {
                if(j<1 || i<0)
                {
                        cout<<pat[0]<<" -";
                        return;
                }
                else
                {
                        bool end = false;
                        while(pat[j] == 0 && j>0)
                        {
                                i = --j -1;
                         }

                         while( j>-1 && !end)
                         {
                                if(((L[pat[j]-1] - L[pat[i] -1]) ==
graph.edge(pat[j], pat[i])))
                                {
                                        shortest_path(L,pat,i,i-1);
                                        cout<<pat[j]<<" - ";
                                        end = true;
                                 }
                                 --i;
                         }

                }
        }

        void initialize(int sourceNode)
        {
                L = new int[6];
                visited = new bool[6];
                path = new int[6];

                L[sourceNode - 1] = 0;          // set the label of
source node to zero
                visited[sourceNode - 1] = true; // mark source nodes as
temporary
                E =sourceNode;      // set the next evaluation node E to
source
                path[0]= sourceNode;

                for(int i=1; i<=6; i++)
                {
                        if( i != sourceNode)
                        {
                                L[i-1] = graph.edge(sourceNode,i);
                                visited[i-1] = false;
                        }
                }
        }

        int shortest_dist(int sourceNode, int destinationNode)
        {
                // step I --------- initialisation
                initialize(sourceNode);
                int min = 10000000;
                int k =1;
```

```
                    // Step II --------------- Assign temp Loabel to all
unvisited Nodes and select the minimum
                    while( !visited[destinationNode - 1])  // this loop
runs until destinationNode is marked as visited
                    {
                        // Step II --- begins here
                        min = 10000000;              // reset minimum
distance to infinity

                        for(int j = 1; j<=6; j++)
                        {
                            if( !visited[j - 1] )
                            {
                                int sum = L[E -1] + graph.edge(E,j);
                                if(L[j-1] > sum) L[j - 1] = sum;
                            }
                        }

                        // Step III ---- select minimum of all the
temporary labels
                        for(int i = 1; i<=6; i++)
                        {
                            if( !visited[i-1])
                            {
                                if(L[i-1] <=min)
                                {
                                    min = L[i - 1];
                                    E = i;
                                }
                            }
                        }
                        visited[E-1] = true;
                        path[k++] = E;
                    }
                    return min;
            }


            void route(int r)
            {
                    double R;
                    double speed;
                    speed = R = 0;
                    double time =0;
                    string rout ="";
                    string bandwidth = "";


        cout<<"ROUTE"<<setw(20)<<"BANDWIDTH(mbps)"<<setw(15)<<"SPEED(m/s)"<<s
etw(15)<<"DISTANCE"<<setw(15)<<"TIME(m/s)"<<endl;

                    while( r>0)
                    {
                        switch(5-r)
                        {
                            case 1:
                                R = graph.edge(1,2) +
graph.edge(2,6);
                                bandwidth = "0.512";
```

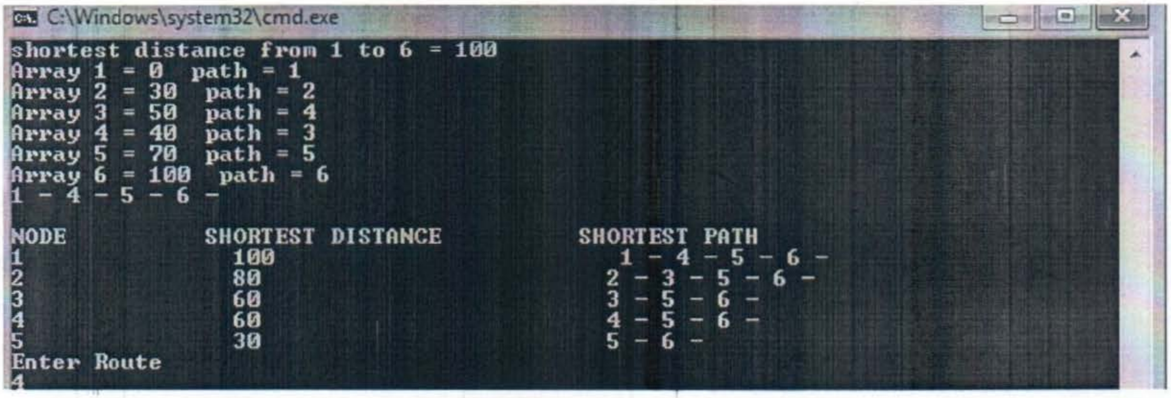
```
                                        time = 0.065*R;
                                        speed = 0.512*R/time;
                                        rout = "R1";
                                        break;
                                case 2:
                                        R = graph.edge(1,4) +
graph.edge(4,5) + graph.edge(5,6);
                                        bandwidth = "0.256";
                                        time = 0.065*R;
                                        speed = 0.256*R/time;
                                        rout = "R2";
                                        break;
                                case 3:
                                        R = graph.edge(1,2) +
graph.edge(2,3) + graph.edge(3,5) + graph.edge(5,6);
                                        bandwidth = "2";
                                        time = 0.065*R;
                                        speed = 2*R/time;
                                        rout = "R3";
                                        break;
                                case 4:
                                        R = graph.edge(1,4) +
graph.edge(4,3) + graph.edge(3,6);
                                        bandwidth = "0.064";
                                        time = 0.065*R;
                                        speed = 0.064*R/time;
                                        rout = "R4";
                                        break;
                                default:
                                        R = 0;
                        }
                        cout<<rout<<setw(20-
rout.capacity())<<bandwidth<<setw(22 -
bandwidth.capacity())<<speed<<setw(15)<<(int)R<<setw(14)<<time<<endl;
                        r--;
                }
        }
};


// main function
int main()
{
        Graph graph(6);
        Dijkstra dijkstra(graph);
        int n, s, d;
        cout<<"Enter source Node ";
        cin>>s;
        cout<<"Enter Destination Node ";
        cin>>d;

        graph.input(1,2,30);
        graph.input(1,3,70);
        graph.input(1,4,40);
        graph.input(2,3,20);
        graph.input(2,6,90);
        graph.input(3,4,10);
        graph.input(3,5,30);
        graph.input(3,6,60);
        graph.input(4,5,30);
        graph.input(5,6,30);

         cout<<"edge 2,6 = "<<graph.edge(2,6)<<endl;

        cout<<dijkstra.shortest_dist(s,d);

        cout<<"\n"<<"\n"<<"NODE"<<setw(10)<<"SHORTEST
DISTANCE"<<setw(10)<<"SHORTEST PATH"<<endl;


        cout<<"Enter route"<<endl;
        int i;
        cin>>i;
        dijkstra.route(i);
}
```

OUTPUT



C:\Users\ OFEM -G\Documents\matrixC++Class\graphs\graph.exe

```
Enter source Node 1
Enter Destination Node 6
edge1,4,5,6 = 100
30

NODESHORTEST DISTANCESHORTEST PATH
Enter route= 3
 ROUTE              SPEED(m/s)       DISTANCE(m)        TIME(s)
 R1                    7.06          120                0.017
 R2                    6.88          110                0.016
 R3                    7.69          100                0.013


--------------------------------
Process exited after 6.553 seconds with return value 0
Press any key to continue . . .
```



C:\Windows\system32\cmd.exe

```
shortest distance from 1 to 6 = 100
Array 1 = 0  path = 1
Array 2 = 30  path = 2
Array 3 = 50  path = 4
Array 4 = 40  path = 3
Array 5 = 70  path = 5
Array 6 = 100  path = 6
1 - 4 - 5 - 6 -

NODE           SHORTEST DISTANCE         SHORTEST PATH
1                100                        1 - 4 - 5 - 6 -
2                80                       2 - 3 - 5 - 6 -
3                60                       3 - 5 - 6 -
4                60                       4 - 5 - 6 -
5                30                       5 - 6 -
Enter Route
4
```

APPENDIX III

IP Plan of the different nodes with their source and destination interface

Routes	Network Address	Interfaces-Src	IP-Src	Interface-Dest	IP-Dest
Node1-2	10.10.10.0	ether1	10.10.10.1/30	ether1	10.10.10.2/30
Node1-3	10.10.10.4	ether2	10.10.10.5/30	ether1	10.10.10.6/30
Node1-4	10.10.10.8	ether3	10.10.10.9/30	ether1	10.10.10.10/30
Node2-3	10.10.10.12	ether2	10.10.10.13/30	ether2	10.10.10.14/30
Node2-6	10.10.10.16	ether3	10.10.10.17/30	ether1	10.10.10.18/30
Node3-4	10.10.10.20	ether3	10.10.10.21/30	ether2	10.10.10.22/30
Node3-5	10.10.10.24	ether4	10.10.10.25/30	ether1	10.10.10.26/30
Node3-6	10.10.10.28	ether5	10.10.10.29/30	ether2	10.10.10.30/30
Node4-5	10.10.10.32	ether3	10.10.10.33/30	ether2	10.10.10.34/30
Nod-e5-6	10.10.10.36	ether3	10.10.10.37/30	ether3	10.10.10.38/30
Src-Ntw	192.168.1.0	Node5-ether4	192.168.1.0/24		
Dest-Ntw	192.168.0.0	Node6-ether4	192.168.0.1/24		

4.3 Network Configuration

Node A

```
[admin@Node1] > /ip add

[admin@Node1] /ip address> add address=10.10.10.1/30 interface=ether1

[admin@Node1] /ip address> add address=10.10.10.5/30 interface=ether2

[admin@Node1] /ip address> add address=10.10.10.9/30 interface=ether3

[admin@Node1] /ip address> add address=192.168.1.1/24 interface=ether5

[admin@Node1] /ip address> print

Flags: X - disabled, I - invalid, D - dynamic

#  ADDRESS          NETWORK        INTERFACE

0  10.10.10.1/30    10.10.10.0     ether1

1  10.10.10.5/30    10.10.10.4     ether2

2  10.10.10.9/30    10.10.10.8     ether3

3  192.168.1.1/24   192.168.1.0    ether5

[admin@Node1] /ip address> /int ether

[admin@Node1] /interface ethernet> set ether1 bandwidth=4M/4M

[admin@Node1] /interface ethernet> set ether2 bandwidth=256k/256k

[admin@Node1] /interface ethernet> set ether3 bandwidth=2M/2M

[admin@Node1] /interface ethernet>

[admin@Node1] /interface ethernet> print

Flags: X - disabled, R - running, S - slave

#    NAME          MTU MAC-ADDRESS         ARP        MASTER-PORT

SWITCH

0 R  ether1     1500 D4:CA:6D:03:8F:EF enabled   none        switch1

1 R  ether2     1500 D4:CA:6D:03:8F:F0 enabled   none        switch1
```

```
2 R  ether3      1500 D4:CA:6D:03:8F:F1 enabled    none          switch1
3    ether4      1500 D4:CA:6D:03:8F:F2 enabled    none          switch1
4 R  ether5      1500 D4:CA:6D:03:8F:F3 enabled    none          switch1
[admin@Node1] /interface ethernet> /rout ospf
[admin@Node1] /routing ospf> area
[admin@Node1] /routing ospf area> add name=Src-LAN area-id=1.1.1.1
[admin@Node1] /routing ospf area> ..
[admin@Node1] /routing ospf> net
[admin@Node1] /routing ospf network> /ip add print
Flags: X - disabled, I - invalid, D - dynamic
#  ADDRESS          NETWORK       INTERFACE
0  10.10.10.1/30    10.10.10.0    ether1
1  10.10.10.5/30    10.10.10.4    ether2
2  10.10.10.9/30    10.10.10.8    ether3
3  192.168.1.1/24   192.168.1.0   ether5
[admin@Node1] /routing ospf network> add network=10.10.10.0/30 area=backbone
[admin@Node1] /routing ospf network> add network=10.10.10.4/30 area=backbone
[admin@Node1] /routing ospf network> add network=10.10.10.8/30 area=backbone
[admin@Node1] /routing ospf network> add network=192.168.1.0/24 area=Src-LAN
[admin@Node1] /routing ospf network>
 [admin@Node1] /routing ospf network> print
Flags: X - disabled, I - invalid
#  NETWORK          AREA
0  10.10.10.0/30    backbone
1  10.10.10.4/30    backbone
```

```
2  10.10.10.8/30     backbone

3  192.168.1.0/24    Src-LAN

[admin@Node1] /routing ospf network> ..

[admin@Node1] /routing ospf> ls

[admin@Node1] /routing ospf lsa> print


AREA      TYPE          ID           ORIGINATOR    SEQUENCE-NUMBER   AGE
backbone  router        10.10.10.1   10.10.10.1    0x80000007        19
backbone  router        10.10.10.17  10.10.10.17   0x80000012        74
backbone  router        10.10.10.29  10.10.10.29   0x80000011        52
backbone  router        10.10.10.33  10.10.10.33   0x80000012        44
backbone  router        10.10.10.37  10.10.10.37   0x8000000A        1490
backbone  router        192.168.0.1  192.168.0.1   0x80000009        1489
backbone  network       10.10.10.2   10.10.10.17   0x80000001        74
backbone  network       10.10.10.6   10.10.10.29   0x80000001        52
backbone  network       10.10.10.10  10.10.10.33   0x80000001        44
backbone  network       10.10.10.14  10.10.10.29   0x80000006        1392
backbone  network       10.10.10.22  10.10.10.33   0x80000006        1392
backbone  network       10.10.10.26  192.168.0.1   0x80000005        1628
backbone  network       10.10.10.30  10.10.10.37   0x80000006        1394
backbone  network       10.10.10.34  10.10.10.37   0x80000006        1388
backbone  network       10.10.10.38  192.168.0.1   0x80000005        1489
backbone  summary-n... 192.168.1.0  10.10.10.1    0x80000001        18
Src-LAN   router        10.10.10.1   10.10.10.1    0x80000001        19
Src-LAN   summary-n... 10.10.10.0   10.10.10.1    0x80000001        19
Src-LAN   summary-n... 10.10.10.4   10.10.10.1    0x80000001        19
Src-LAN   summary-n... 10.10.10.8   10.10.10.1    0x80000001        19
```

```
Src-LAN    summary-n... 10.10.10.12   10.10.10.1       0x80000001            19

Src-LAN    summary-n... 10.10.10.16   10.10.10.1       0x80000001            19

Src-LAN    summary-n... 10.10.10.20   10.10.10.1       0x80000001            19

Src-LAN    summary-n... 10.10.10.24   10.10.10.1       0x80000001            19

Src-LAN    summary-n... 10.10.10.28   10.10.10.1       0x80000001            19

Src-LAN    summary-n... 10.10.10.32   10.10.10.1       0x80000001            19

[admin@Node1] /routing ospf lsa>

[admin@Node1] /routing ospf lsa>

[admin@Node1] /routing ospf lsa> ..

[admin@Node1] /routing ospf> rout

[admin@Node1] /routing ospf route> print


# DST-ADDRESS       STATE        COST      GATEWAY         INTERFACE

0 10.10.10.0/30     intra-area    10       0.0.0.0       ether1

1 10.10.10.4/30     intra-area    10       0.0.0.0       ether2

2 10.10.10.8/30     intra-area    10       0.0.0.0       ether3

3 10.10.10.12/30    intra-area    20       10.10.10.2    ether1

                                            10.10.10.6    ether2

4 10.10.10.16/30    intra-area    30       10.10.10.6    ether2

5 10.10.10.20/30    intra-area    20       10.10.10.6    ether2

                                            10.10.10.10    ether3

6 10.10.10.24/30    intra-area    20       10.10.10.6    ether2

7 10.10.10.28/30    intra-area    20       10.10.10.6    ether2

8 10.10.10.32/30    intra-area    20       10.10.10.10   ether3

9 10.10.10.36/30    intra-area    30       10.10.10.6    ether2

                                           10.10.10.10   ether3

10 192.168.1.0/24   intra-area    10       0.0.0.0       ether5

[admin@Node1] /routing ospf route>
```

Node 2

```
[admin@MikroTik] > /system
[admin@MikroTik] /system> id
[admin@MikroTik] /system identity> set name=Node2
[admin@Node2] /system identity> /ip add
[admin@Node2] /ip address> add address=10.10.10.2/30 interface=ether1
[admin@Node2] /ip address> add address=10.10.10.13/30 interface=ether2
[admin@Node2] /ip address> add address=10.10.10.17/30 interface=ether3
[admin@Node2] /ip address>
[admin@Node2] /ip address> /int ether
[admin@Node2] /interface ethernet> set ether1 bandwidth=4M/4M
[admin@Node2] /interface ethernet> set ether2 bandwidth=8M/8M
[admin@Node2] /interface ethernet> set ether3 bandwidth=64k/64k
[admin@Node2] /interface ethernet>
[admin@Node2] /interface ethernet> /rout ospf
[admin@Node2] /routing ospf>
[admin@Node2] /routing ospf> /ip add pri
Flags: X - disabled, I - invalid, D - dynamic
 #  ADDRESS            NETWORK       INTERFACE
 0  10.10.10.2/30      10.10.10.0    ether1
 1  10.10.10.13/30     10.10.10.12   ether2
 2  10.10.10.17/30     10.10.10.16   ether3
[admin@Node2] /routing ospf> net
[admin@Node2] /routing ospf network>
add  comment  disable  edit  enable  export  find  print  remove  set
[admin@Node2] /routing ospf network> add network=10.10.10.0/30 area=backbone
[admin@Node2] /routing ospf network> /rout ospf lsa print
```

```
AREA        TYPE     ID           ORIGINATOR    SEQUENCE-NUMBER     AGE
backbone    router   10.10.10.1   10.10.10.1    0x8000000F          655
backbone    router   10.10.10.2   10.10.10.2    0x80000002          6
backbone    router   10.10.10.17  10.10.10.17   0x80000015          947
backbone    router   10.10.10.29  10.10.10.29   0x80000017          153
backbone    router   10.10.10.33  10.10.10.33   0x80000013          707
backbone    router   10.10.10.37  10.10.10.37   0x8000000C          351
backbone    router   192.168.0.1  192.168.0.1   0x80000012          126
backbone    network  10.10.10.1   10.10.10.1    0x80000001          9
backbone    network  10.10.10.6   10.10.10.29   0x80000002          715
backbone    network  10.10.10.10  10.10.10.33   0x80000002          707
backbone    network  10.10.10.22  10.10.10.33   0x80000008          252
backbone    network  10.10.10.26  192.168.0.1   0x80000007          490
backbone    network  10.10.10.30  10.10.10.37   0x80000008          255
backbone    network  10.10.10.34  10.10.10.37   0x80000008          249
backbone    network  10.10.10.38  192.168.0.1   0x80000007          351
[admin@Node2] /routing ospf network> add network=10.10.10.12/30 area=back
[admin@Node2] /routing ospf network> /rout ospf lsa print

AREA      TYPE     ID           ORIGINATOR   SEQUENCE-NUMBER   AGE
backbone  router   10.10.10.1   10.10.10.1   0x80000010        87
backbone  router   10.10.10.2   10.10.10.2   0x80000003        3
backbone  router   10.10.10.17  10.10.10.17  0x80000015        1025
backbone  router   10.10.10.29  10.10.10.29  0x80000017        231
backbone  router   10.10.10.33  10.10.10.33  0x80000013        785
backbone  router   10.10.10.37  10.10.10.37  0x8000000C        429
backbone  router   192.168.0.1  192.168.0.1  0x80000012        204
backbone  network  10.10.10.1   10.10.10.1   0x80000001        87
backbone  network   10.10.10.6   10.10.10.29   0x80000002        793

backbone  network   10.10.10.10  10.10.10.33   0x80000002        785

backbone  network   10.10.10.22  10.10.10.33   0x80000008        330

backbone  network   10.10.10.26  192.168.0.1   0x80000007        568

backbone  network   10.10.10.30  10.10.10.37   0x80000008        333

backbone  network   10.10.10.34  10.10.10.37   0x80000008        327

backbone  network   10.10.10.38  192.168.0.1   0x80000007        429

[admin@Node2] /routing ospf network> /ip add print

Flags: X - disabled, I - invalid, D - dynamic


 #  ADDRESS          NETWORK         INTERFACE

 0  10.10.10.2/30    10.10.10.0      ether1

 1  10.10.10.13/30   10.10.10.12     ether2

 2  10.10.10.17/30   10.10.10.16     ether3

[admin@Node2] /routing ospf network> add network=10.10.10.16/30 area=backbone

[admin@Node2] /routing ospf network> /rout ospf lsa print


AREA      TYPE      ID           ORIGINATOR    SEQUENCE-NUMBER   AGE

backbone  router    10.10.10.1   10.10.10.1    0x80000010        139

backbone  router    10.10.10.2   10.10.10.2    0x80000005        7

backbone  router    10.10.10.17  10.10.10.17   0x80000015        1077

backbone  router    10.10.10.29  10.10.10.29   0x80000018        45

backbone  router    10.10.10.33  10.10.10.33   0x80000013        837

backbone  router    10.10.10.37  10.10.10.37   0x8000000C        481

backbone  router    192.168.0.1  192.168.0.1   0x80000012        256

backbone  network   10.10.10.1   10.10.10.1    0x80000001        139

backbone  network   10.10.10.6   10.10.10.29   0x80000002        845

backbone  network   10.10.10.10  10.10.10.33   0x80000002        837
backbone   network    10.10.10.14   10.10.10.29    0x80000001          45
backbone   network    10.10.10.22   10.10.10.33    0x80000008          382
backbone   network    10.10.10.26   192.168.0.1    0x80000007          620
backbone   network    10.10.10.30   10.10.10.37    0x80000008          385
backbone   network    10.10.10.34   10.10.10.37    0x80000008          379
backbone   network    10.10.10.38   192.168.0.1    0x80000007          481
[admin@Node2] /routing ospf network> ..
[admin@Node2] /routing ospf> lsa
[admin@Node2] /routing ospf lsa>
find  print
[admin@Node2] /routing ospf lsa> ..
[admin@Node2] /routing ospf>
area              instance  nbma-neighbor  route       export
area-border-router  interface  neighbor      sham-link
as-border-router   lsa       network       virtual-link
[admin@Node2] /routing ospf> instance print
Flags: X - disabled, * - default
 0  * name="default" router-id=0.0.0.0 distribute-default=never
      redistribute-connected=no redistribute-static=no redistribute-rip=no
      redistribute-bgp=no redistribute-other-ospf=no metric-default=1
      metric-connected=20 metric-static=20 metric-rip=20 metric-bgp=auto
      metric-other-ospf=auto in-filter=ospf-in out-filter=ospf-out
[admin@Node2] /routing ospf> neighbor print
 0 instance=default router-id=192.168.0.1 address=10.10.10.18 interface=ether3
    priority=1 dr-address=10.10.10.18 backup-dr-address=10.10.10.17 state="Full"
    state-changes=5 ls-retransmits=0 ls-requests=0 db-summaries=0 adjacency=59s
 1 instance=default router-id=10.10.10.29 address=10.10.10.14 interface=ether2
    priority=1 dr-address=10.10.10.14 backup-dr-address=10.10.10.13 state="Full"
```

```
  state-changes=5 ls-retransmits=0 ls-requests=0 db-summaries=0
  adjacency=1m46s
2 instance=default router-id=10.10.10.1 address=10.10.10.1 interface=ether1
  priority=1 dr-address=10.10.10.1 backup-dr-address=10.10.10.2 state="Full"
  state-changes=5 ls-retransmits=0 ls-requests=0 db-summaries=0
  adjacency=3m19s
[admin@Node2] /routing ospf> route
[admin@Node2] /routing ospf route> print

# DST-ADDRESS      STATE        COST      GATEWAY       INTERFACE
0 10.10.10.0/30    intra-area   10        0.0.0.0       ether1
1 10.10.10.4/30    intra-area   20        10.10.10.1    ether1
                                          10.10.10.14   ether2
2 10.10.10.8/30    intra-area   20        10.10.10.1    ether1
3 10.10.10.12/30   intra-area   10        0.0.0.0       ether2
4 10.10.10.16/30   intra-area   10        0.0.0.0       ether3
5 10.10.10.20/30   intra-area   20        10.10.10.14   ether2
6 10.10.10.24/30   intra-area   20        10.10.10.14   ether2
                                          10.10.10.18   ether3
7 10.10.10.28/30   intra-area   20        10.10.10.14   ether2
8 10.10.10.32/30   intra-area   30        10.10.10.1    ether1
                                          10.10.10.14   ether2
                                          10.10.10.18   ether3
9 10.10.10.36/30   intra-area   20        10.10.10.18   ether3
[admin@Node2] /routing ospf route>
```

Node 4

```
[admin@MikroTik] /system identity> set name=Node4

[admin@Node4] /system identity> /ip add

[admin@Node4] /ip address>

[admin@Node4] /ip address> add address=10.10.10.10/30 interface=ether1

[admin@Node4] /ip address> add address=10.10.10.22/30 interface=ether2

[admin@Node4] /ip address> add address=10.10.10.33/30 interface=ether3

[admin@Node4] /ip address>

[admin@Node4] /ip address> print

Flags: X - disabled, I - invalid, D - dynamic


 #  ADDRESS          NETWORK        INTERFACE

 0  10.10.10.10/30   10.10.10.8     ether1

 1  10.10.10.22/30   10.10.10.20    ether2

 2  10.10.10.33/30   10.10.10.32    ether3


[admin@Node4] /ip address> /int ether

[admin@Node4] /interface ethernet> set ether1 bandwidth=2M/2M

[admin@Node4] /interface ethernet> set ether2 bandwidth=10M/10M

[admin@Node4] /interface ethernet> set ether3 bandwidth=4M/4M

[admin@Node4] /interface ethernet> print

Flags: X - disabled, R - running, S - slave

 #   NAME      MTU MAC-ADDRESS          ARP      MASTER-PORT    SWITCH

 0   ether1   1500 4C:5E:0C:D4:98:79    enabled  none           switch1

 1   ether2   1500 4C:5E:0C:D4:98:7A    enabled  none           switch1

 2   ether3   1500 4C:5E:0C:D4:98:7B    enabled  none           switch1

 3   ether4   1500 4C:5E:0C:D4:98:7C    enabled  none           switch1

 4 R ether5   1500 4C:5E:0C:D4:98:7D    enabled  none           switch1
```

```
[admin@Node4] /interface ethernet>

[admin@Node4] /interface ethernet>

[admin@Node4] /interface ethernet> /rout ospf

[admin@Node4] /routing ospf>

[admin@Node4] /routing ospf>

[admin@Node4] /routing ospf>

[admin@Node4] /routing ospf> /ip add print

Flags: X - disabled, I - invalid, D - dynamic

#  ADDRESS          NETWORK        INTERFACE

0  10.10.10.10/30   10.10.10.8     ether1

1  10.10.10.22/30   10.10.10.20    ether2

2  10.10.10.33/30   10.10.10.32    ether3

[admin@Node4] /routing ospf>

[admin@Node4] /routing ospf> net

[admin@Node4] /routing ospf network> add

comment  copy-from  disabled  network  area

[admin@Node4] /routing ospf network> add network=10.10.10.8/30 area=backbone

[admin@Node4] /routing ospf network> add network=10.10.10.20/30 area=backbone

[admin@Node4] /routing ospf network> add network=10.10.10.32/30 area=backbone

[admin@Node4] /routing ospf network>
```

Node 5

```
[admin@MikroTik] > /system id

[admin@MikroTik] /system identity>

[admin@MikroTik] /system identity> set name=Node5

[admin@Node5] /system identity>

[admin@Node5] /system identity> /ip add

[admin@Node5] /ip address>

[admin@Node5] /ip address> add address=10.10.10.30/30 interface=ether1

[admin@Node5] /ip address> add address=10.10.10.34/30 interface=ether2

[admin@Node5] /ip address> add address=10.10.10.37/30 interface=ether3

[admin@Node5] /ip address>

[admin@Node5] /ip address>

[admin@Node5] /ip address> /int ether

[admin@Node5] /interface ethernet>

[admin@Node5] /interface ethernet> set ether1 bandwidth=4M/4M

[admin@Node5] /interface ethernet> set ether2 bandwidth=4M/4M

[admin@Node5] /interface ethernet> set ether3 bandwidth=4M/4M

[admin@Node5] /interface ethernet>

[admin@Node5] /interface ethernet>

[admin@Node5] /interface ethernet> print

Flags: X - disabled, R - running, S - slave

#  NAME      MTU MAC-ADDRESS         ARP      MASTER-PORT     SWITCH

0  ether1    1500 D4:CA:6D:03:21:A3  enabled  none            switch1

1  ether2    1500 D4:CA:6D:03:21:A4  enabled  none            switch1

2  ether3    1500 D4:CA:6D:03:21:A5  enabled  none            switch1

3  ether4    1500 D4:CA:6D:03:21:A6  enabled  none            switch1

4 R ether5   1500 D4:CA:6D:03:21:A7  enabled  none            switch1
```

5a. Print out the IP addresses configured on the router

```
[admin@Node6] /routing ospf network> /ip add print
Flags: X - disabled, I - invalid, D - dynamic
#  ADDRESS          NETWORK        INTERFACE
0  10.10.10.18/30   10.10.10.16    ether1
1  10.10.10.26/30   10.10.10.24    ether2
2  10.10.10.38/30   10.10.10.36    ether3
3  192.168.0.1/24   192.168.0.0    ether4
```

5b. Use the network addresses on the printscreen to add networks to the OSPF network tab

```
[admin@Node6] /routing ospf network> add network=10.10.10.16/30 area=backbone
[admin@Node6] /routing ospf network> add network=10.10.10.24/30 area=backbone
[admin@Node6] /routing ospf network> add network=10.10.10.36/30 area=backbone
```

5c. Include the Destination LAN network to the OSPF network but first, the network must be in a separate area

```
[admin@Node6] /routing ospf network> ..
[admin@Node6] /routing ospf>
[admin@Node6] /routing ospf> area
[admin@Node6] /routing ospf area> add name=Dst-LAN area-id=2.2.2.2
[admin@Node6] /routing ospf area> ..
```

5d. Destination LAN is now added to the OSPF network

```
[admin@Node6] /routing ospf> network
[admin@Node6] /routing ospf network> add network=192.168.0.0/24 area=Dst-LAN
[admin@Node6] /routing ospf network> print
Flags: X - disabled, I - invalid
#  NETWORK          AREA
0  10.10.10.16/30   backbone
```

```
1  10.10.10.24/30   backbone

2  10.10.10.36/30   backbone

3  192.168.0.0/24   Dst-LAN

[admin@Node6] /routing ospf network>
```

4.4 Presentation of Result

```
[admin@Node5] /routing ospf route> print

# DST-ADDRESS      STATE        COST      GATEWAY        INTERFACE

0 10.10.10.0/30    intra-area   30        10.10.10.33    ether2

                                          10.10.10.38    ether3

1 10.10.10.4/30    intra-area   30        10.10.10.33    ether2

2 10.10.10.8/30    intra-area   20        10.10.10.33    ether2

3 10.10.10.12/30   intra-area   30        10.10.10.33    ether2

                                          10.10.10.38    ether3

4 10.10.10.16/30   intra-area   20        10.10.10.38    ether3

5 10.10.10.20/30   intra-area   20        10.10.10.33    ether2

6 10.10.10.24/30   intra-area   10        0.0.0.0        ether1

7 10.10.10.28/30   intra-area   30        10.10.10.33    ether2

8 10.10.10.32/30   intra-area   10        0.0.0.0        ether2

9 10.10.10.36/30   intra-area   10        0.0.0.0        ether3

[admin@Node5] /routing ospf route> ..

[admin@Node5] /routing ospf> lsa

[admin@Node5] /routing ospf lsa> print
```

```
AREA      TYPE     ID           ORIGINATOR    SEQUENCE-NUMBER

AGE
                                             0x8000000E   118
backbone  router   10.10.10.17  10.10.10.17
                                             0x8000001E   1160
backbone  router   10.10.10.29  10.10.10.29
backbone  router   10.10.10.33  10.10.10.33  0x8000000B   81
backbone  router   10.10.10.37  10.10.10.37  0x8000000E   58
backbone  router   192.168.0.1  192.168.0.1  0x80000012   59
backbone  router   192.168.1.1  192.168.1.1  0x8000000A   49
backbone  network  10.10.10.1   192.168.1.1  0x80000005   162
backbone  network  10.10.10.5   192.168.1.1  0x80000004   630
backbone  network  10.10.10.9   192.168.1.1  0x80000004   1714
backbone  network  10.10.10.14  10.10.10.29  0x80000004   567
backbone  network  10.10.10.18  192.168.0.1  0x80000005   119
backbone  network  10.10.10.22  10.10.10.33  0x80000004   518
backbone  network  10.10.10.33  10.10.10.33  0x80000001   81
backbone  network  10.10.10.38  192.168.0.1  0x80000001   59
```

4.5 Test for all routes connectivity

```
[admin@Node5] > ping 10.10.10.1
HOST                     SIZE TTL TIME  STATUS
10.10.10.1                 56  63 10ms
10.10.10.1                 56  63 0ms
sent=2 received=2 packet-loss=0% min-rtt=0ms avg-rtt=5ms max-rtt=10ms
```

```
[admin@Node5] > ping 10.10.10.10
HOST                          SIZE TTL TIME  STATUS
10.10.10.10                    56  64 0ms
10.10.10.10                    56  64 0ms
10.10.10.10                    56  64 0ms
10.10.10.10                    56  64 0ms
10.10.10.10                    56  64 0ms
sent=5 received=5 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms
[admin@Node5] > ping 10.10.10.13
HOST                          SIZE TTL TIME  STATUS
10.10.10.13                    56  62 5ms
10.10.10.13                    56  62 0ms
sent=2 received=2 packet-loss=0% min-rtt=0ms avg-rtt=2ms max-rtt=5ms

[admin@Node5] > ping 10.10.10.14
HOST                          SIZE TTL TIME  STATUS
10.10.10.14                    56  62 1ms
10.10.10.14                    56  62 0ms
sent=2 received=2 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=1ms

[admin@Node5] > ping 10.10.10.17
HOST                          SIZE TTL TIME  STATUS
10.10.10.17                    56  63 0ms
10.10.10.17                    56  63 0ms
10.10.10.17                    56  63 0ms
10.10.10.17                    56  63 0ms
sent=4 received=4 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms
```

```
[admin@Node5] > ping 10.10.10.18
HOST                          SIZE TTL TIME  STATUS
10.10.10.18                     56  64 0ms
10.10.10.18                     56  64 0ms
10.10.10.18                     56  64 0ms
10.10.10.18                     56  64 0ms
sent=4 received=4 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms

[admin@Node5] > ping 10.10.10.21
HOST                          SIZE TTL TIME  STATUS
10.10.10.21                     56  63 0ms
10.10.10.21                     56  63 0ms
10.10.10.21                     56  63 0ms
10.10.10.21                     56  63 0ms
sent=4 received=4 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms

[admin@Node5] > ping 10.10.10.22
HOST                          SIZE TTL TIME  STATUS
10.10.10.22                     56  64 0ms
10.10.10.22                     56  64 0ms
10.10.10.22                     56  64 0ms
sent=3 received=3 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms

[admin@Node5] > ping 10.10.10.25
HOST                          SIZE TTL TIME  STATUS
10.10.10.25                     56  64 5ms
10.10.10.25                     56  64 5ms
10.10.10.25                     56  64 6ms
sent=3 received=3 packet-loss=0% min-rtt=5ms avg-rtt=5ms max-rtt=6ms
```

```
[admin@Node5] > ping 10.10.10.26
HOST                        SIZE TTL TIME  STATUS
10.10.10.26                   56  64 6ms
10.10.10.26                   56  64 7ms
sent=2 received=2 packet-loss=0% min-rtt=6ms avg-rtt=6ms max-rtt=7ms

[admin@Node5] > ping 10.10.10.29
HOST                        SIZE TTL TIME  STATUS
10.10.10.29                   56  64 0ms
10.10.10.29                   56  64 0ms
sent=2 received=2 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms

[admin@Node5] > ping 10.10.10.30
HOST                        SIZE TTL TIME  STATUS
10.10.10.30                   56  64 0ms
10.10.10.30                   56  64 0ms
sent=2 received=2 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms

[admin@Node5] > ping 10.10.10.33
HOST                        SIZE TTL TIME  STATUS
10.10.10.33                   56  64 0ms
10.10.10.33                   56  64 0ms
sent=2 received=2 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms

[admin@Node5] > ping 10.10.10.34
HOST                        SIZE TTL TIME  STATUS
10.10.10.34                   56  64 5ms
10.10.10.34                   56  64 5ms
sent=2 received=2 packet-loss=0% min-rtt=5ms avg-rtt=5ms max-rtt=5ms

[admin@Node5] > ping 10.10.10.37
HOST                        SIZE TTL TIME  STATUS
10.10.10.37                   56  64 5ms
10.10.10.37                   56  64 5ms
sent=2 received=2 packet-loss=0% min-rtt=5ms avg-rtt=5ms max-rtt=5ms
```

```
[admin@Node5] > ping 10.10.10.38
HOST                           SIZE TTL TIME  STATUS
10.10.10.38                      56  64 0ms
10.10.10.38                      56  64 0ms
10.10.10.38                      56  64 0ms
sent=3 received=3 packet-loss=0% min-rtt=0ms avg-rtt=0ms max-rtt=0ms
```