DEVELOPMENT OF A SIMULATION TOOL FOR ESTIMATING THE RECONFIGURATION AND RESTORATION TIMES OF A DISTRIBUTED INTELLIGENCE OPTICAL MESH NETWORK

By

WAI YEU CHAN

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1999

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the degree of
MASTERS OF SCIENCE
December, 2001

DEVELOPMENT OF A SIMULATION TOOL FOR

ESTIMATING THE RECONFIGURATION AND

RESTORATION TIMES OF A DISTRIBUTED

INTELLIGENCE OPTICAL MESH

NETWORK

Thesis Approved:

George Scheets

Thesis Adviser

Rao Yarlagadda

James C West

Timothy J. Pettibone

Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to my advisor, Dr. George Scheets for his moral support and guidance beyond academic supervision. I am very thankful for his willingness and flexibility in setting meetings times for consultation and also his patience in explaining subject matter, answering all questions and clearing all doubts that I have. Furthermore, I would like to extend my appreciation to the committee members, Dr. Jong-Moon Chung, Dr. Yarlagadda and Dr. James West.

Not forgetting, I would like to thank Mark Wendel, at that time Senior Network Engineer in the Network Architecture Group of Williams Network for providing me with necessary information and answering any questions pertaining to the network.

Last but not least, I would like to dedicate this thesis to my parents, brother, sisters and friends, who always stood by me and providing me with their unconditional support and encouragements to succeed and most importantly for believing in me.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Rapid growth in the fiber facilities in the telecommunications network has brought the need for higher bandwidth requirements. By the mid 1990's, concerns with service efficiency and equipment interoperability on an end-to-end basis as well as the need for bandwidth requirements higher than the DS3 level brought service providers to deploy the solutions enabled by Synchronous Optical Network or better known as SONET. SONET defines interface standards at the physical and data-link layer of the OSI seven-layer model. The standard defines a hierarchy of interface rates that allow data streams at different rates to be multiplexed. SONET establishes Optical Carrier (OC) levels from 51.84 Mbps (about the same as a DS3 line) to 10 Gbps, with 40 Gbps rates in trials. SONET comprises both an optical interface and specifications for the rate and format of optical signal transmission [1] and can be used to support both narrowband and broadband services.

With the significantly higher configuration flexibility and bandwidth availability, SONET offers several advantages over the older telecommunications system. These advantages include:

- Reduction in equipment requirements and an increase in network reliability

- Provision of overhead and payload bytes – the overhead bytes permit management of the payload bytes on an individual basis and facilitate centralized fault sectionalization.
- Definition of a synchronous multiplexing format for carrying lower level digital signals (such as DS–1, DS–3) and a synchronous structure that greatly simplifies the interface to digital switches, digital cross-connect switches, and add-drop multiplexers
- Availability of a set of generic standards that enable products from different vendors to be connected
- Definition of a flexible architecture capable of accommodating future applications, with a variety of transmission rates [3]

In brief, SONET defines OC levels and electrically equivalent Synchronous Transport Signals (STSs) for the fiber-optic–based transmission hierarchy.

There are three different types of SONET Transport Network Architectures, point-to-point, rings, and mesh networks.

Point-to-point: Point-to-point networks consist of pairs of nodes that are connected by two (ideally) physically diverse transmission paths. In case of a failure of the working path, the working traffic of the single point to point is switched to the protection path. Many autonomous point-to-point topologies must be placed to protect a network using point-to-point protection methods [10].

Ring: A ring network is defined as a collection of nodes forming a closed loop, where each node is connected via a full duplex communications facility [1]. In case of a failure, the demands of any affected traffic pairs within the ring can be rerouted in the opposite direction. Thus, the spare protection capacity of the ring is shared by all of the working traffic on the ring. Note that, theoretically, the point-to-point topology is a two-node ring [10].

Mesh: A Mesh network is a topology in which devices are connected with many redundant interconnections between network nodes. To be considered a mesh, at least two nodes on the network should have a degree of connectivity (i.e. the number of links coincident on that node) that exceeds two. In case of a failure, the affected paths can reroute over a variety of diverse paths, in contrast to a single alternate path for the point-to-point and ring architectures. Generally, in the mesh architectures, as network connectivity increases, required protection capacity decreases because of the sharing of spare capacity among many potential pairs of nodes [10].

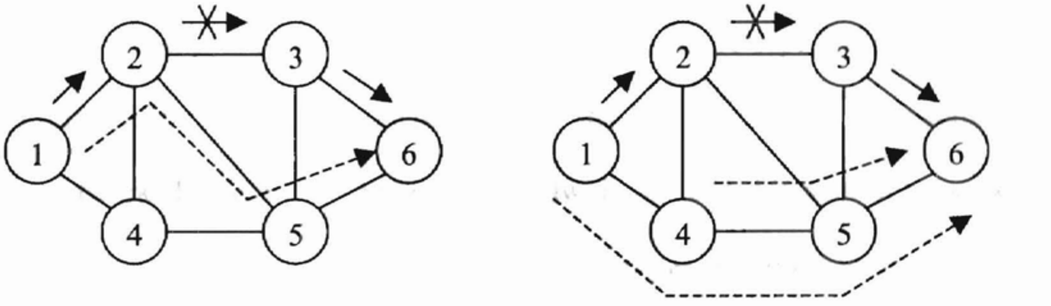
Each type of the SONET Transport Network Architecture has its own restoration mechanism.

Point-to-point: Automatic Protection Switching (APS) systems are common on point-to-point links. There are two types of APS architectures defined in the SONET standards, 1:N APS and 1+1 APS [1]. 1:N APS allows one of the N (permissible values for N are from 1 to 14) working channels to be bridged to a single protection channel [1]. 1+1 protection switching is a form of 1:1 APS with the head end permanently bridged and the signal simultaneously transmitted over two circuits. The decision as to which circuit to use is made by the tail end [1].

Rings: SONET Self-Healing Rings (SHR) provide redundant bandwidth so that disrupted services can be restored automatically following network failures [1]. Two different types of rings exist. Unidirectional rings move working traffic around the ring in one direction, while bi-directional rings move working traffic in either direction. SONET rings provide fast restoration, less than 50 ms on fiber failures.

Mesh: Mesh restoration uses switches or Digital Cross Connect Systems (DCSs) to reroute traffic around a failure point. Mesh networks typically require less excess bandwidth for restoration purposes than do rings, as the extra bandwidth on each link can be shared and typically provides full or partial restoration capacity for many possible failed links [1].

There are two types of restoration techniques possible, line restoration and path restoration. Line restoration uses the line layer information to trigger the restoration process and restores connections between the nodes immediately adjacent to a cable failure by rerouting all affected working circuits over the same restoration route regardless of the sources and destinations of the affected working circuits [1]. Path restoration on the other hand, restores the affected STS paths on an end-to-end basis [1]. To better explain the difference of the above line and path restoration, let's look at an example [1]. In this example, the link between Nodes 2 and 3 carries two STS-1 circuits



STS-1 paths	Original Paths	Restored Paths
(1,6)	1-2-3-6	1-2-5-3-6
(4,6)	4-2-3-6	4-2-5-3-6

(a) Line Restoration

STS-1 paths	Original Paths	Restored Paths
(1,6)	1-2-3-6	1-4-5-6
(4,6)	4-2-3-6	4-5-6

(b) Path Restoration

Figure 1: Line restoration vs. Path restoration [1]

STS-1 (1,6) and STS-1 (4,6). If that link fails and the line restoration method is used [see Figure 1(a)], route 2-3 is replaced by route 2-5-3, and all channels use the new route when they pass from Node 2 to Node 3. On the other hand, if the path restoration method is used [see Figure 1(b)], each circuit [i.e. STS-1 (1,6) and STS-1 (4,6)] affected by the link failure selects a new route for restoration. For example, STS-1 circuit (4,6) may select new route 4-5-6 and STS-1 (1,6) may select route 1-4-5-6 or 1-2-5-6.

In a comparison between APS, Rings and Meshes, APS systems are more appropriate in areas where point to point demand is extremely high, and rings are appropriate in geographically limited areas requiring fast standards-based multiple site restoration capabilities [1]. Mesh on the other hand is appropriate for relatively large areas. Compared to the SHR, the DCS self-healing mesh network needs less protection capacity at the expense of longer restoration time. The spare capacity savings for the DCS self-healing network are due primarily to sharing of spare capacities across the entire network. Therefore less redundant bandwidth is needed in a mesh network. Ring disadvantages, including bottlenecks where traffic is passed from one ring to another, and excessive bandwidth requirements for restoration compared to other topologies, have also contributed to the rise of meshes over rings.

Significant recent traffic growth has outstripped SONET's ability to provide connectivity in a cost effective manner. Today, SONET alone does not provide sufficient capacity on a fiber strand. Advances in WDM and other optical technologies have resulted in a much more complex environment whereas before, a carrier's SONET digital system would swap STS-N time slots, and today proprietary switches might exchange STS-N time slots, wavelengths, or all traffic on an entire fiber (multiple wavelengths).

When a failure occurs, network providers have at least two different techniques that can be used to restore the affected services. One option would be to reconfigure the entire network making full use of multiple layer restoration efforts occurring in a simultaneous or near simultaneous manner. However, considering there are no standardized means of communication between each of the different layers of the network, conflicts may occur because each layer is independently executing its restoration efforts, with no coordination occurring between the layers. Such restoration efforts could be very complicated and today they are not well understood.

A second option would be to avoid all potential conflicts between the layers, and execute restoration efforts systematically, starting with the layer most immediate to the failure, for example a router in the event of a router card failure, an ATM switch in the event of an ATM trunk failure, or- the main focus of this study- the optical layer in the event of a fiber cut. This technique will slow down restoration efforts, and will require the implementation of back-off timers. For instance, when a failure due to a fiber break occurs, the optical layer will immediately begin to restore failed circuits. Other higher-level devices such as ATM Switches and IP Routers will remain idle, giving the optical layer an opportunity to repair the failure, until their back-off timers expire.

Due to the small knowledge base currently associated with multi-layer restoration and the resulting lack of understanding towards preventing layers from working at cross-purposes with each other, and due to the lack of standardized protocols that would allow improved communications between the layers in order to better coordinate multi-layer restoration, the sponsor of this project, Williams Network, has elected to implement restoration effects systematically. This study focuses on the distributed restoration process of a mesh network in the event of a fiber failure. Of key concern is an accurate estimate of the back-off time required for the higher layer devices. Too long a timer will result in time being wasted in restoring the affected circuits in the event the optical layer is unable to affect restoration. Too short a timer will result in restoration efforts kicking-in before the optical layer is finished, and possibly interfering with the restoration efforts occurring at the optical layer. Hence a key focus of this study is to develop software useful for estimating the proper back-off time.

CHAPTER II

LITERATURE REVIEW

The initial stage of the project was spent on a literature review of related work by other researchers. The purpose of this review was to search for articles that addressed multi-layer network restoration, and also to find articles that discuss the restoration times for the multi-layer restoration efforts or restoration at the optical layer.

Few papers were found addressing multi-layer restoration. Joseph Kroculick & Cynthia Hood [4] addressed issues such as multi-layer resilience, multi-layer escalation schemes, how these affect network survivability, and suggested some production rules to indicate which layer should handle a specific failure. They also recommend single layer survivability strategies and allocate these policies to each layer within a multi-layer network. Escalation strategies that apply to a single domain and ATM-VP/SDH 1:N subnetworks are also discussed in great detail. However, this paper provides no hard analysis and does not provide restoration and reconfiguration efforts to a SONET network. Kroculick and Hood [5] published another article similar to [4] in another conference.

Dirceu Cavendish of C&C Research Laboratories [6] provides a tutorial on SONET & WDM. He does include a small discussion on the interaction between SONET & WDM protection, as well as a table on time responses of various APS mechanisms, which lists 'typical' detection and restoration times of different types of equipment. However, this article does not address SONET mesh network restoration and the table does not include the restoration times of an optical mesh network.

The topics of discussion in Yinghua Ye and Sudhir Dixit's article [7] include an overview of existing optical protection/restoration schemes and of the envisioned IP-centric Dense Wavelength Division Multiplexed (DWDM) based optical data network architecture, and a mesh based hybrid optical protection scheme that utilizes multi-fiber physical links along with hierarchical OXC structure and provides a joint protection/restoration scheme that is coordinated at both the IP and WDM layers. Again there are no specific discussions on the amount of time an optical network requires to restore its links due to failure.

In the final paper, Hawker, I., Johnson, D. and Chng, R [8] provides a discussion of ATM versus SONET restoration. The article does include some vague claims regarding which layers will react first to a fault and also some simulated restoration times, though there is little explanation as to the details regarding how their restoration times were determined. The simulated restoration times are for SONET networks in general, and do not clarify if a ring or mesh network is being restored. The article also does not discuss possible interaction between optical layers other higher level layers.

These were the only papers found in the IEEE database that had any reference to multi-layer restoration or restoration time-to-restore. None of the above papers addressed the main issue of interest to this study, which is the time to restore a distributed control optical mesh network.

CHAPTER III

PROBLEM STATEMENT

The test network that was used in this study is the Spring 2001 Williams Communications Group Optical Network, controlled by Ciena CoreDirector switches located in17 Point-of-Presence (POP) cities. This optical network is shown in Appendix A. Fiber optic cable provides the bandwidth for the 27 links between the switches.

A problem arises when a fiber optic link between the city pairs breaks. When this happens, some of the traffic from certain end-to-end to destinations will be affected. For example, (refer to Appendix A and Appendix F, which contains Traffic Matrix entries, and Appendix G, which contains a list of minimum hop routes) all traffic between Anaheim and Albany will be affected if either the Anaheim – Dallas, Dallas – Chicago, or Chicago - Albany link fails.

When such a failure happens, the entire optical network will be affected. Traffic originally routed over the failed link will not be sent to its designated destination. Therefore, restoration efforts will begin starting with the directly affected switches. Because head ends are responsible for reconfiguring the circuits that are affected by the failed link, the Ciena CoreDirector proprietary algorithm designates head ends to search for a new shortest route to reroute the traffic. This new route will then be used to restore the affected traffic. The above restoration efforts take some time to process and complete before the network is fully restored. As mentioned previously, higher level devices will implement a back-off time prior to initiating restoration efforts, so that the optical layer has an opportunity to repair the failure.

Therefore, the problem to be solved in this study is to develop a program that will provide an estimate of how much time an optical mesh network takes to restore itself in the event of a broken link. This time will then be set as the back-off time to the higher-level devices. The following chapter discusses the solution to this problem.

CHAPTER IV

SOLUTIONS TO PROBLEM

This chapter provides a step-by-step explanation of what was done in terms of determining the restoration times of the optical mesh network when a failure due to a broken link occurs.

Specifications of the test network were provided by Williams Network personnel [11]. As mentioned previously and shown in Appendix A, a map of the Williams Core Director network was developed. This map allows better visualization of the optical network and also gives a better understanding of how the traffic will be rerouted when there is a break in one of the fiber links.

After completion of the network map, the distance of the fiber links was estimated using Yahoo! Map, which provides road distances between cities. These distances are accurate enough for program testing and can be replaced with actual fiber distances as required. The table on Appendix B provides a listing of these distances, which are required for calculating the propagation time between the switches using the formula

Prop Time = d / 0.6c

Where, c = speed of light in free space = $3\text{x}10^8$ m/s = 186411.36 miles/sec

d = distance of link in miles

The .6 accounts for the slower light speed in fiber and for time lost in circuit switched, time division multiplexed, time-slot interchange devices. For instance, the distance between Dallas and Anaheim is about 1424.6 miles. Therefore the propagation time between Dallas and Anaheim would be approximately 12.7 milliseconds.

Also, we have to take into consideration the Switch Processing Time (SPT) of each SONET switch, which is the time it takes a switch to process a 'link-is-down' message. In this study, the switches are considered to be of similar specifications, and therefore the Switch Processing Time for each switch will be identical and will be a parameter set by the user.

Another delay would be the Switch Reconfiguring Time (SRT), which is the time it takes a SONET switch to process a reconfiguration request. This is another user chosen parameter that will be identical for all switches.

CHAPTER V

RESTORATION ALGORITHM

Today there exists no standards based protocol for the restoration of optical mesh networks. The restoration algorithm used in this study is based on a Fall 2000 series of EMail exchanges between Dr. George Scheets at OSU and Mark Wendel, at the time a Senior Network Engineer in the Network Architecture Group of Williams Network. The algorithm approximates, without violating non-disclosure agreements, to matching the proprietary signaling algorithm used by Ciena CoreDirector SONET switches [11].

In the event of a line cut, the two SONET switches adjacent to the cut will note the loss of the Optical Connection. The propagation times from the cut locations to the Point of Presence (POP) where the adjacent SONET switches are located are most probably not the same, but are being ignored for Version 1.0 of this algorithm as it is assumed that this specific time has negligible impact on the overall restoration time. The switch processing time may also differ slightly at the two endpoints depending upon whether or not the optical signals pass through the same POP configuration of DWDM Section Terminating Equipment or Mutltiplexer Line Terminating Equipment just prior to hitting a SONET Switch. This value, too, is treated as being identical.

Hence for Version 1.0 we assume that both the terminating SONET switches start the restoration process at this point, at the same time, and therefore begin the restoration clock and set its time as t = 0 seconds. At this instance, traffic routed over the broken link is unavailable and the SONET switches immediately adjacent to the break are aware that the connection has been lost, have processed this information, and are poised to alert other hardware in the network.

Subsequent SONET frames passed on to local POP hardware would now indicate a loss of signal. As there are 8,000 SONET frames a second, the time delay between the adjacent-to-the-break SONET switch noting loss of signal and signaling this information to other local hardware would be a random value uniformly distributed between [1, 1/8000] second. Back-off timers on local higher level ATM switches or IP routers would start at this time.

The two adjacent-to-the-break SONET switches then initiate restoration efforts. Link-is-down messages are flooded to all the core switch nodes of the network. These messages are received at different times depending on the distance from these nodes to the adjacent-to-the-break SONET switches. When a core switch receives a link-is-down message, it will check its list of STS-N circuits it, as a head end, is responsible for reconfiguring to determine whether or not any of the circuits are affected by the break, and determine an alternate end-to-end route using the minimum hop rule and path restoration for all the affected circuits. These processes are assumed to take SPT seconds. Head ends with circuits affected by the break will then send reconfigure requests to all nodes involved with rerouting of the affected path. This simulation assumes that each affected head end circuit will require a set of reconfiguration messages be transmitted to appropriate intermediate nodes. For example, if four circuits between Albany and Boston are affected by a break and must be rerouted via Albany - New York - Boston, and

Boston is the head end, Boston must send four reconfiguration requests to New York and four to Albany. Boston will not piggyback these requests together and send one request to New York for reconfiguration of four circuits, and one to Albany.

In the real world, each SONET Switch will be processing reconfiguration requests received from other switches while simultaneously juggling, if head end for some affected circuits, the generation of reconfiguration requests for other switches to execute. These set of processes, occuring on the network in parallel (i.e. each switch is working on its own set of reconfiguration requests), must be simulated in series due to the single processor nature of most PC's. C++ code was written to accomplish this by...

```
Looping on each head end in order of earliest 'awareness' of the failed link

    Looping on each affected circuit

        Calculating delay times to all SONET Switches affected by the rerouting
        of this circuit and logging that time in an Event Array associated with each
        switch. These Event Arrays contain the times that reconfiguration
        requests are received by each switch. An entry is also made into the head
        end Event Array.

        Incrementing the head end 'time' by SRT seconds prior to examining the
        next circuit.

    End circuit loop.

End head end loop.
```

At this point, the program then examines each switches' Event Array times to insure sufficient time has elapsed (SRT seconds) to allow a reconfiguration to be completed. Requests may have been received by a switch faster than they can be executed, requiring that the requests be queued. Essentially the array values are adjusted such that the times between Event Array elements are a minimum of SRT seconds apart.

Appendix C shows the flowchart of the program and Appendix D contains the actual program code. The program was written to be as generic as possible, and will provide the estimated restoration times when there is a fiber failure for an arbitrary user defined network and an arbitrary user defined line failure. Input files required to run this program are network.dat, paths.dat and connections.dat.

The Network.dat file (Appendix E) shows the distance between the directly connected switch locations. The first two columns of the file are the directly connected switch locations and the third column is the distance between the locations.

When running the program, users will be asked to enter endpoints of a broken link. Users will have to refer to optical network map (Appendix A) and type in the exact switch name. For example, to break the Dallas and Anaheim link on the test network, the user will have to enter DLLSTX ANHMCA. The order of the switches does not matter but it will have to match, exactly- including case, the names listed in the Network.dat file. Following that, the user will be asked for their choice of Switch Processing Time (SPT) and Switch Reconfiguration Time (SRT).

The program obtains the minimum hop routes from the paths.dat file (Appendix G), which can be generated by a program called NEAFE [12]. The reconfiguration estimation time program then simulates a line break by removing the chosen connection from the network. The program will then consult the connections.dat file (Appendix F) in order to determine the circuits that were routed over the affected link before the break. The program will identify every city on the second column of the connection pairs as the head end of the circuit and every city on the third column the destination. The first column of the file is the number of circuits for each optical path. Head ends will be responsible for the restoration of all the circuits that are affected by the break. The program will determine a list of all the affected circuits. Head ends will then search for an alternate route using the minimum hop rule in order to reroute the circuits that it's responsible for.

Following this, head ends will then send reconfigure requests messages to nodes involved with establishing an alternate route. As mentioned previously, these requests are logged into an Event Array associated with each switch.

CHAPTER VI

RESULTS

The final test of the program was to simulate a failure of the most heavily used fiber of the test network, and compare the resulting estimate of the time to restore the system given a distributed control restoration algorithm, with estimates obtained earlier simulating the system restoration time given a centralized control restoration algorithm [9]. The simulation was based on the optical network of Appendix A. Three simulations were run with 100, 300, and 500 full duplex circuits distributed over the system with circuit endpoints chosen in proportion to the human population surrounding the POPs. The Anaheim to Dallas connection was found to be the most heavily used link given these constraints. Refer to Appendix H, I and J for a sample portion of the results of the simulation of 500 full duplex circuits. A Switch Processing Time of 100 milliseconds and Switch Reconfiguration Time of 50 milliseconds chosen for this test case. Figure 2 shows a plot of the three different reconfiguration times. For a network of 100 circuits, it takes 1.57 seconds to reconfigure the 33 affected circuits. When this same link is failed in the 300 and 500 circuits network cases, 92 circuits fail requiring 4.17 seconds to restore, and 154 circuits fail requiring 6.83 seconds to restore, respectively. Results from this simulation differ slightly from the results in [9] which estimated the reconfiguration time of the same fiber failure on the same network, assuming centralized control of the

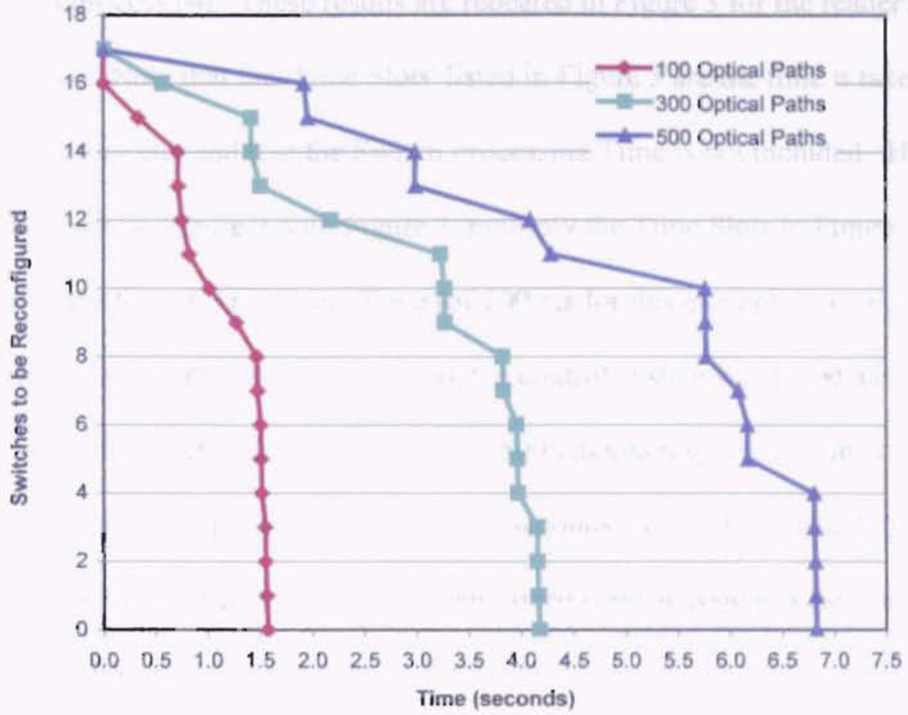


Figure 2: 100, 300 and 500 Circuits Reconfiguration Times of a Distributed Control System

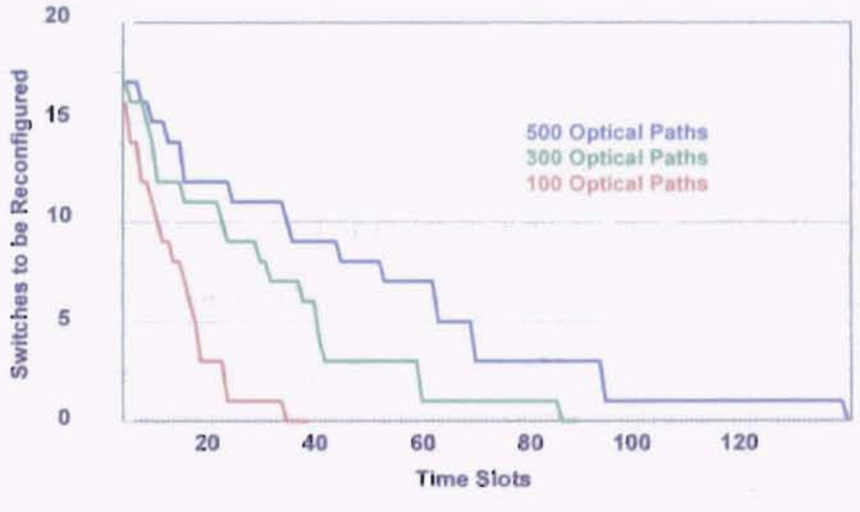


Figure 3: 100, 300 and 500 Circuits Reconfiguration Times of a Centralized Control System [9]

restoration process [9]. These results are repeated in Figure 3 for the reader's convenience. Note that the 'Time Slots' listed in Figure 3 are the time it takes a switch to reconfigure a circuit and that the Switch Processing Time is not included. Hence to properly compare Figure 2 with Figure 3, multiply the Time Slots in Figure 3 by 50 ms, and add in the Switch Processing Time (of 100 ms for this example) to the result. While the estimated restoration time of a distributed control system was noted as 1.57, 4.17, and 6.83 seconds for a network of 100, 300, and 500 circuits respectively, the centralized control system will require 1.6, 4.15, and 6.8 seconds. These latter results from [9] are considered to be "best case" in that the centralized control system is assumed to be perfect and allow no "idle time" at any of the SONET switches due to gaps in the times between executed reconfiguration requests, gaps that may occur in reality with a real-world centralized control system and are even more likely to occur in reality with a distributed control system.

Two key results are noted from the above experiment. The first is that the distributed control system outperformed the "best case" result in the 100 circuit network. Investigation revealed that the Minimum Hop routing algorithms used in this study and in [9] were, in some cases, yielding different identical-hop-count restoration routes. For example, on Anaheim to Albany circuits, two different reroute paths were noted. Anaheim-Phoenix-Houston-Dallas-Chicago-Albany on the code written here, and Anaheim-Salt Lake City-Denver-Kansas City-Chicago-Albany used in the centralized control experiment. This resulted in switches having different numbers of reconfiguration requests, and completing their required reroutes in slightly different times.

The second key result to note is that, in this experiment, there is negligible difference in the restoration times between the centralized system and the distributed control system, a result contrary to original expectations. Investigation revealed this was a result of the Switch Reconfiguration Times being considerably greater than the average switch-to-switch propagation delays. With the choice in this study of an SRT of 50 milliseconds, queues build up at the switches, with reconfiguration requests arriving faster than they are processed. As a result, negligible slack time occurs between reconfigurations at each switch.

To verify that this claim is correct, a simulation of the same 500 circuit network using a SPT of 10 milliseconds and a SRT of 1 milliseconds was run. The distributed control system required 168.7 milliseconds to reconfigure all 154 circuits. The centralized control system on the other hand took 144 milliseconds to reconfigure. In this case, the SRT is smaller than the average switch-to-switch propagation delay. Reconfiguration requests would, in many cases, be processed prior to the arrival of the next request, leaving switches with significant idle time.

The results above are based on what is believed to be the worst-case scenario. In our test network, Dallas - Anaheim is the most heavily loaded link. A fiber failure here will necessitate more reroute requests than a break anywhere else in the system. Therefore, a logical assumption would be that it would take longer to restore this connection than any other fiber failure in the network. However, this assumption may not be completely true for all cases. Take for example, a break occurring between Boston and New York City up in the northeast corner of the network, fiber that also carries a large number of customer circuits. Due to the less centralized location, it will take longer for the 'link-is-down' message to be flooded across the entire network. As a result switches on the far side of the network on the west coast will not initiate restoration efforts in as timely a manner as if the break were more centralized, and as a result this fiber, carrying fewer customer circuits than the Anaheim-Dallas connection, may actually take longer to restore. This is one scenario that follow-on studies, using the code developed here, will investigate in more detail.

CHAPTER VII

CONCLUSION

Based on the rapid growth in the telecommunications industry, optical mesh network architectures are being deployed on a lot of large-scale networks today. In this study, we briefly discussed the interactions among the different network layers that may occur while restoring failed communications. However, today these layers act independently of one another and the possible interactions that can occur among them are little understood. Given a fiber cut, the most viable solution available now for service providers is to provide back-off timers at Layer 2 and Layer 3 to avoid multi-layer thrashing and race conditions, giving time for restoration efforts at Layer 1 to restore the affected communications.

This study developed code that can estimate the restoration times of an Optical Mesh network when a failure such as a broken link occurs, providing a solid foundation upon which to build improved versions. There are several modifications that can be made to improve the code. One shortcoming is that the code does not track the affect of small signaling bandwidth, which may result in reconfiguration requests getting slowed and/or delayed in queues at intermediate switches.

The minimum hop rule used in this code only takes into account the number of hops but not the distance of the hops as well. Some of the circuits have more than one route with the same number of hops. When this happens, the code will choose a route at random. Future studies should develop code that could calculate the minimum distance routes.

This study also does not track possible failures of reconfiguration requests, which might necessitate retransmissions, further delaying network restoration. This is another area that future work should take that into consideration.

REFERENCES

[1] Tsong-Ho Wu and Noriaki Yoshikai, "SONET/SDH Transport and Network Integrity," *ATM Transport and Network Integrity*, Academic Press, 1997.

[2] Walter J. Goralski, "Customer and Carrier Advantages," *SONET, A guide to Synchronous Optical Network,* McGraw Hill, 1997.

[3] "An introduction to SONET, " Synchronous Optical Network (SONET), http://www.iec.org/online/tutorials/sonet/topic01.html

[4] Kroculick, J., Hood, C., "Applying the Policy Concept to Avoid Logical Race Condition in Multilayer Network Restoration", 1999 Military Communications Conference, November 1999. MILCOM '99 IEEE.

[5] Kroculick, J., Hood, C., "Defining Provably Correct Escalation Policies for Multilayer Network Restoration", Global Telecommunications Conference, December 1999. GLOBECOM '99

[6] Cavendish, D., "Evolution of Optical Transport Technologies: From SONET/SDH to WDM, IEEE Communications, June 2000.

[7] Ye, Y., Dixit, S., "On Joint Protection/Restoration in IP-Centric DWDM-Based Optical Transport Networks", IEEE Communications, June 2000.

[8] Hawker, I., Johnson, D., Chng, R "Distributed restoration in telecommunications networks", Fifth IEEE Conference on Telecommunications, 1995.

[9] NFOEC paper, Mark C. Wendel, Dr. George Scheets, Dr. Jong-Moon Chung, "MESH Optical QOS and Multi-Layer restoration, National Fiber Optic Engineering Conference, Baltimore, MD, July 2001

[10] Curtis A. Siller, Jr., Mansoor Shafi, "Survivability and Robust Architechture", *SONET/SDH A sourcebook of Synchronous Networking*, IEEE Press, 1996.

[11] Mark Wendel, Senior Network Engineer, Williams Network, Personal Communications, Fall 2000.

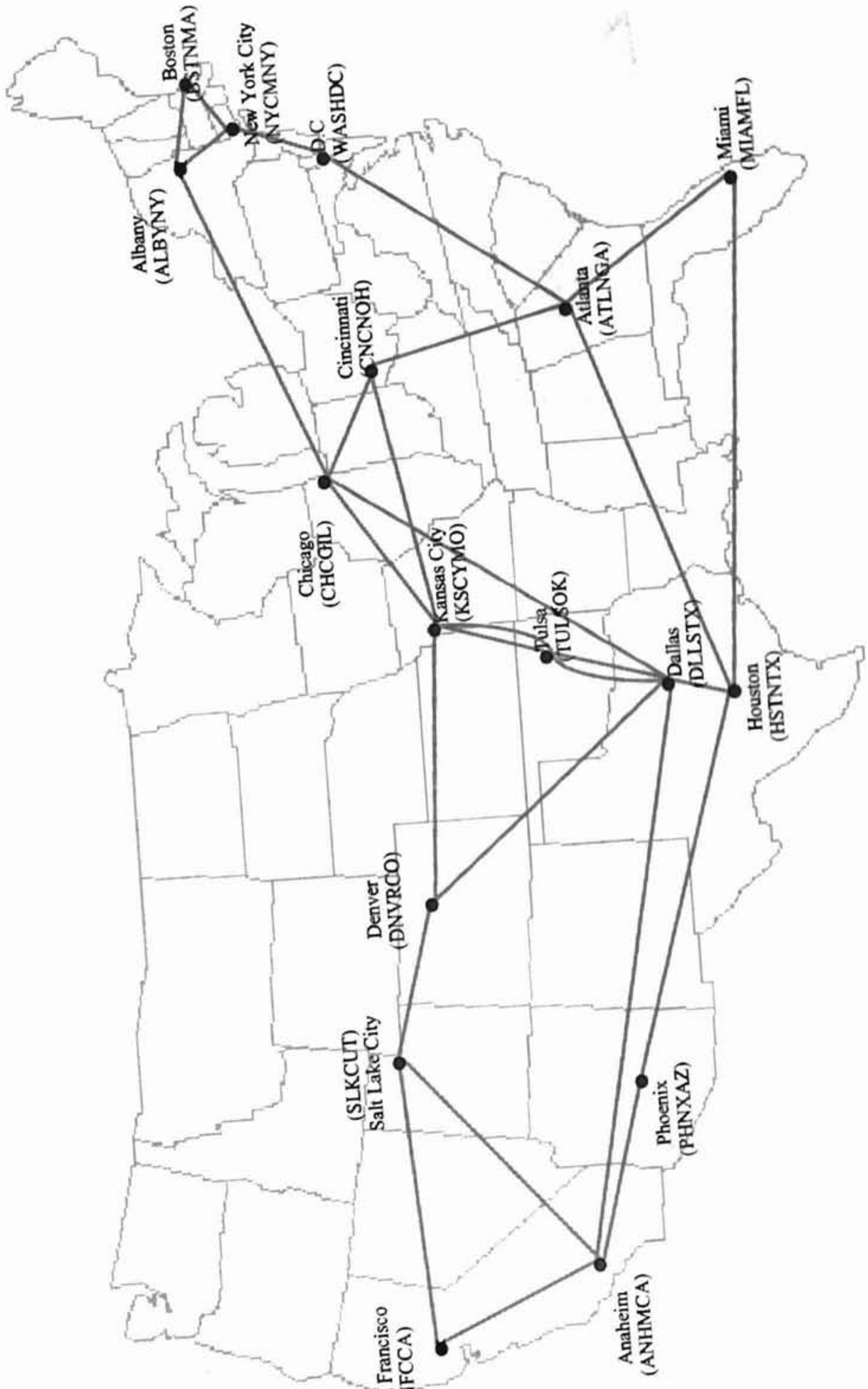
[12] NEAFE (Network Equipment Analysis Front End), code written by Kim Ferris, Eva Degreef, Eva Jahan, & Ryan McQuillen for Senior Design II, Fall 2000 - Spring 2001, Oklahoma State University.

[13] Regis J. Bates, "SONET and SDH", *Broadband Telecommunications Handbook*, McGraw Hill Telecommunications, 2000.

APPENDIXES

APPENDIX A

Optical Network



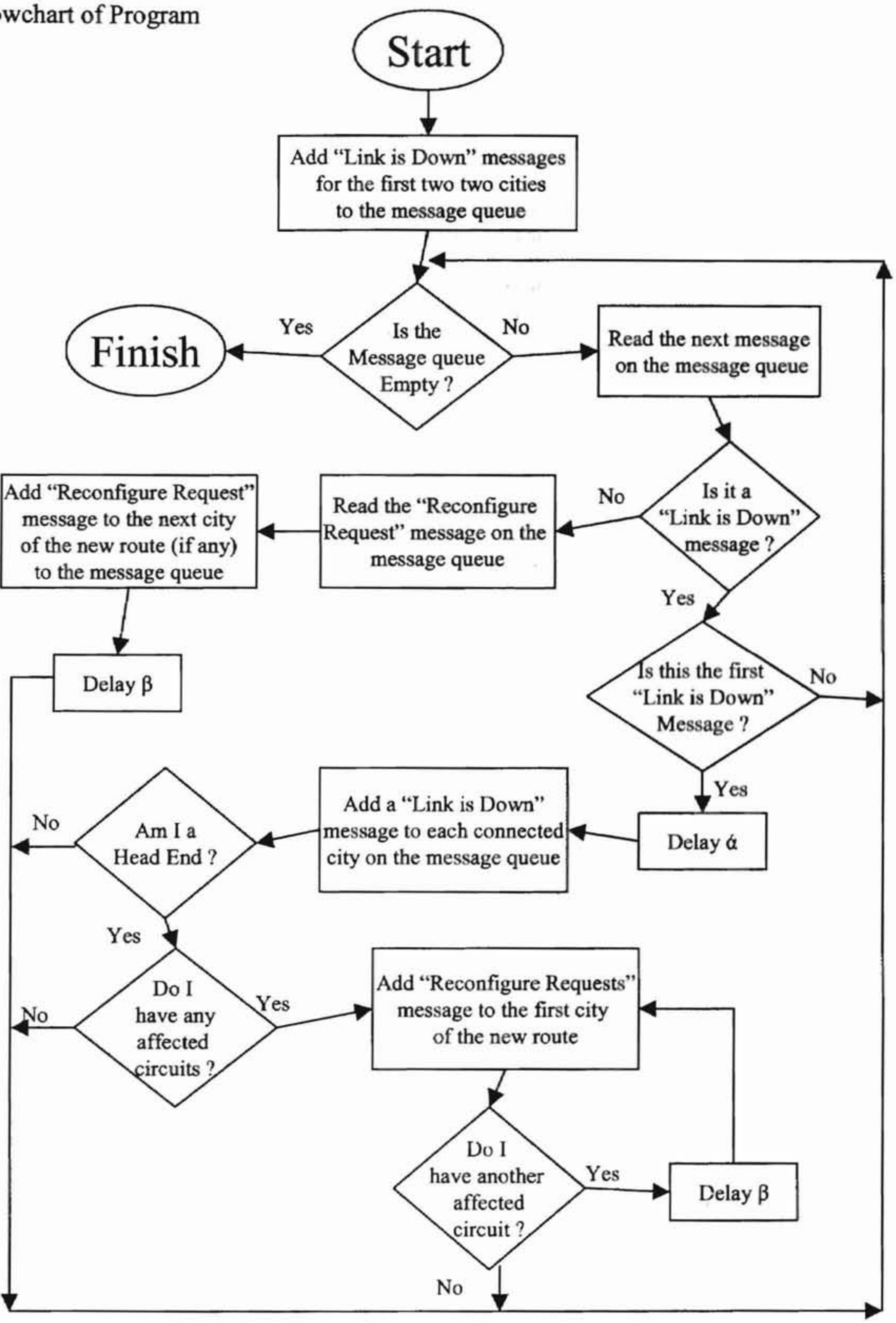
APPENDIX B

Distance between SONET switches

■	SNFCCA	ANHMCA	PHNXAZ	SLKCUT	DNVRCO	DLLSTX	HSTNTX	TULSOK	KSCYMO	CHCGIL	CNCNOK	ATLNTA	MIAMFL	WASHDC	NYCMNY	BSTNMA	ALBYNY
SNFCCA	■	406.6		736													
ANHMCA	406.6	■	360.2	682.9		1425											
PHNXAZ		360.2	■				1179										
SLKCUT	736	682.9		■	532.9												
DNVRCO				532.9	■	879			601								
DLLSTX		1425			879	■	239.2	310.4	552.6	966.9							
HSTNTX			1179			239.2	■					792.8	1219				
TULSOK						310.4		■	246.7								
KSCYMO					601	552.6		246.7	■	529.9	594.4						
CHCGIL						966.9			529.9	■	296.1						817.9
CNCNOK									594.4	296.1	■	466.9					
ATLNTA							792.8				466.9	■	694.4	637.4			
MIAMFL							1219					694.4	■				
WASHDC												637.4		■	225.1		
NYCMNY														225.1	■	214.7	155.1
BSTNMA															214.7	■	166.2
ALBYNY										817.9					155.1	166.2	■

APPENDIX C

Flowchart of Program



Start
Add "Link is Down" messages for the first two two cities to the message queue
Is the Message queue Empty ?
Yes
Finish
No
Read the next message on the message queue
Is it a "Link is Down" message ?
No
Read the "Reconfigure Request" message on the message queue
Add "Reconfigure Request" message to the next city of the new route (if any) to the message queue
Delay β
Yes
Is this the first "Link is Down" Message ?
No
Yes
Delay α
Add a "Link is Down" message to each connected city on the message queue
Am I a Head End ?
No
Yes
Do I have any affected circuits ?
No
Yes
Add "Reconfigure Requests" message to the first city of the new route
Do I have another affected circuit ?
Yes
Delay β
No

APPENDIX D

C++ Program

```
////////////////////////////////////////////////////////////////////////
//
// Network simulation
//
// This program simulates the time to reconfigure all the circuits
// of a SONET based network after a single link has been broken.
//
// The input comes from three files - network.dat, connections.dat,
// and paths.dat. (see file descriptions in the loadData function).
// Also, the user is prompted for the endpoints of the broken link
// and the processing and reconfiguration delays.
//
// The output consists of three files - results1.txt, results2.txt,
// and results3.txt.
// * results1.txt contains a list of messages received by all cities
//   sorted by the time the messages are received.
// * results2.txt contains a list of the times that each circuit takes
//   to be reconfigured followed by 3 lists for each city: 1) a list
//   of circuits originating from that city that were reconfigured as
//   a result of the broken link (with the new circuit route), 2) a
//   list of reconfigure request messages received by that city, sorted
//   by the time received, and 3) a list of reconfigure request messages
//   received by that city, sorted by the time the switch is finished
//   processing the message.
// * results3.txt contains a list of all the linkdown messages sorted
//   by the time the messages are received.
//
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// Header files
//
#include <istream>
#include <ostream>
#include <fstream>
#include <iostream>
#include <windows.h>
#include <stdlib.h>
#include <list>
#include <queue>
#include <vector>
#include <map>
#include <string>
using namespace std;

////////////////////////////////////////////////////////////////////////
// Simulation Parameters
```

```
//
// Initial Delay 0.000125
// Processing Delay 0.1
// Linkdown delay
// Reconfiguration delay 0.05
//
double initial_delay = 0.000125;
double processing_delay, linkdown_delay, reconfig_delay;

//////////////////////////////////////////////////////////////////////
// Type definitions based on STL containers
//
typedef vector<string> stringVector;
typedef multimap<string, stringVector, less<string> > routingTable;
typedef map<string, string, less<string> > stringMap;
typedef map<string, bool, less<string> > booleanMap;
typedef map<string, double, less<string> > doubleMap;
typedef map<string, doubleMap, less<string> > graph;
typedef map<string, routingTable::iterator> iteratorMap;

//////////////////////////////////////////////////////////////////////
// Message types enumeration
//
// * Linkdown message are forwarded between all connected switches.
// * Reconfigure Request messages are forwarded along the connection routes
// * Reconfigure Complete messages indicate when a simulated connection has
//   been re-routed. They are only used in the simulation and not part of
//   the actual reconfiguration protocol
//
enum MessageType { LinkDown, ReconfigureRequest, StartReconfigure,
   ReconfigureComplete, ReconfigureNext };

//////////////////////////////////////////////////////////////////////
// Message structure definition
//
struct Message {
    MessageType type;
    string city1;      // First end of the broken link
    string city2;      // Second end of the broken link
    string source;      // Sender of this message
    string destination; // Receiver of this message
           string headend;     // Headend of the connection
           //                  (for reconfiguration requests)
           string tail;        // Tail of the connection
           //                  (for reconfiguration requests)
    double time;        // Time the message will be recieved
           //                  at the destination
           double time2;       // Time the destination city will be
           //                  finished processing this message

    // Constructor for LinkDown messages
    Message(MessageType tp,
                string c1,
                              string c2,
                              string s,
                string d,
                        double t) :
      type(tp), city1(c1), city2(c2), source(s), destination(d),
                headend(d), tail(""), time(t) { };

   // Constructor for Reconfiguration messages
        Message(MessageType tp,
             string c1,
                        string c2,
                        string s,
                        string d,
                        string h,
                        string tl,
                        double t) :
      type(tp), city1(c1), city2(c2), source(s), destination(d),
                headend(h), tail(tl), time(t) { };
};

// Type definition for message queue sorted by message time
typedef priority_queue<Message, vector<Message>, greater<Message> > messageQueue;
typedef map<string, messageQueue> resultsQueue;

// Compare message times (for sorting)
bool operator < (const Message& lhs, const Message& rhs)
  { return lhs.time < rhs.time; }

bool operator > (const Message& lhs, const Message& rhs)
  { return lhs.time > rhs.time; }

///////////////////////////////////////////////////////////////////////////
// Compute the time required to send a message over the given distance
// time = distance / 0.6 c
//
double computeTime(double distance)
{
  return distance / 111846;
}

///////////////////////////////////////////////////////////////////////////
// Handle a "LinkDown" message
//
void linkDown(messageQueue& messages, graph& network, Message& msg)
{
  string city1 = msg.city1;
  string city2 = msg.city2;
  string source = msg.source;
  string destination = msg.destination;
  double time = msg.time;

  // Forward linkdown message to each connected city
  // except for the source of the message
  doubleMap& cities = network[destination];
  for (doubleMap::iterator start = cities.begin();
      start != cities.end();
      ++start)
  {
    string neighbor = (*start).first;
    double distance = (*start).second;

    if( neighbor != msg.source )
    {
            double newTime = time + computeTime(distance) + linkdown_delay;
      Message newMsg(LinkDown, city1, city2, destination, neighbor, newTime);
      messages.push(newMsg);
          }
  }

  // Send a Reconfigure Next message
  Message nextMsg(ReconfigureNext, city1, city2, "", "",
            destination, "", time + linkdown_delay);
  messages.push(nextMsg);
}

////////////////////////////////////////////////////////////////////////
// Find the minimum hop routes from the start city
// Using a breadth-first graph traversal
//
void minHops(graph& network, string& start, stringMap& tree)
{
  list<string> cities;
  cities.push_back(start);
  tree[start] = start;
  while( !cities.empty() )
  {
    // Get the next city in the list
    string nextcity = cities.front();
    cities.pop_front();

    // Visit each connected city
    doubleMap& distances = network[nextcity];
    for (doubleMap::iterator it = distances.begin();
            it != distances.end();
       ++it)
    {
      string c = (*it).first;

      // Check if the city was already visited
      if( tree.count(c) == 0 )
      {
        // Add the next city to the tree
        tree[c] = nextcity;
        cities.push_back(c);
      }
    }
  }
}

////////////////////////////////////////////////////////////////////////
// Simulate reconfiguring a connection
// This is called after receiving "Reconfigure Next" messages
//
void reconfigure(messageQueue& messages, graph& network, doubleMap& schedule,
            routingTable& connections, routingTable& newConnections,
                                        iteratorMap& nextRoute, Message& msg )
{
  string target;
  string city1 = msg.city1;
  string city2 = msg.city2;
  string currentcity = msg.headend;
  double time = msg.time;
  bool found = false;
  stringMap tree;

  // Find the next route to be reconfigured
  while( !found && nextRoute.count(currentcity) > 0
       && nextRoute[currentcity] != connections.end()
              && (*(nextRoute[currentcity])).first == currentcity )
  {
    stringVector myRoute = (*(nextRoute[currentcity])).second;

    // Check if the route is affected by the broken link
    for( int i = 0; i < myRoute.size() -1; i++)
    {
      string city = myRoute[i];
      if(city == city1 || city == city2)
      {
               string nextCity = myRoute[ i+1 ];
        if(nextCity == city2 || nextCity == city1)
        {
          found = true;
          target = myRoute[myRoute.size()-1];
                                break;
        }
      }
    }

           // Skip to the next connection
    if( !found )
             nextRoute[currentcity]++;
          }

  // Reconfigure the affected connection
  if( found )
  {
    // Find the next shortest path
    minHops(network, currentcity, tree);

    // get the new route from the tree (from tail to head)
    stringVector newRoute;
            for( string prevCity = target:
               prevCity != currentcity;
               prevCity = tree[prevCity] )
                    {
                      newRoute.push_back(prevCity);
                    }
            newRoute.push_back(currentcity);

            // Reverse the route (head to tail)
            reverse(newRoute.begin(), newRoute.end());
          newConnections.insert(
            routingTable::value_type(currentcity,newRoute));

          double newTime = max(time, schedule[currentcity]);

          //Send a Start Reconfigure message
     Message startMsg(StartReconfigure, city1, city2, currentcity, currentcity,
              currentcity, target, newTime);
          startMsg.time2 = newTime + reconfig_delay;
     messages.push(startMsg);


          // Update schedule
          schedule[currentcity] = newTime + reconfig_delay;

     // Send a reconfigure request message to the first city on the route
          string src = newRoute[0];
          string dest = newRoute[1];

          // Send a Reconfigure Next message
     Message nextMsg(ReconfigureNext, city1, city2, src, dest,
              currentcity, target, newTime + reconfig_delay);
     messages.push(nextMsg);

          double distance = network[src][dest];
          newTime += computeTime(distance);

          Message newMsg(ReconfigureRequest, city1, city2, src, dest,
                 currentcity, target, newTime);
          messages.push(newMsg);

     // Update the route iterator for the next connection
          nextRoute[currentcity]++;
  }
}

//////////////////////////////////////////////////////////////////////////
// Handle a reconfigure request message
//
void reconfigureRequest(messageQueue& messages, graph& network,
           routingTable& newConnections, doubleMap& schedule,
                                    Message& msg)
{
  string city1 = msg.city1;
  string city2 = msg.city2;
  string destination = msg.destination;
  string headend = msg.headend;
  string target = msg.tail;

  // The time for the next message is the greater of the time this
  // message was receieved and the time the destination switch is
  // finished processing its last reconfigure request message
  double time = max(msg.time,schedule[destination]);

  // Update schedule
  schedule[destination] = time + reconfig_delay;
  msg.time2 = time + reconfig_delay;

  // Send a reconfigure complete message if complete
  if( destination == target )
  {
    Message rmsg(ReconfigureComplete, city1, city2,
             "", "", headend, target, msg.time2);
          messages.push(rmsg);
          return;
  }

  // Find the new route for the connection
  routingTable::iterator route_it = newConnections.find(headend);
  while( route_it != newConnections.end()
                  && ((*route_it).first == headend)
                  && ((*route_it).second.back() != target) )
    route_it++;
  if( route_it == newConnections.end() ||
     (*route_it).second.back() != target )
  {
    cout << "Error - unable to find new route" << endl;
          while(1);
  }

  // Find the next city on the route
  stringVector& route = (*route_it).second;
  stringVector::iterator it = find( route.begin(),
                        route.end(), destination );
  if( it == route.end() )
  {
    cout << "Error - unable to find next city in route" << endl;
          while(1);
  }
  string nextCity = *(++it);

  // Send a reconfigure request message to the next city on the route
  double distance = network[destination][nextCity];
  double newTime = time + computeTime(distance);
  Message newmsg(ReconfigureRequest, city1, city2, destination,
           nextCity, headend, target,newTime);
  messages.push(newmsg);

}

////////////////////////////////////////////////////////////////////////////
// Load connection paths from file
// paths.dat - minumum hop routes for connections before broken link
// Only the connections listed in connections.dat are actually used
//
bool loadPaths(routingTable& connections)
{
  // Open input files
  ifstream pathData("paths.dat");
  if( !pathData )
  {
    cout << "Error opening path data file" << endl;
    return false;
  }

  // Begin parsing input file
  bool start = false;
  char buffer[256];
  stringVector path;
  while( pathData.getline(buffer,256,'\n') )
  {

   if( start )
        {
                string city(buffer);

                // The path is terminated by "Number of hops"
                if( city.find( "Number") != -1 )
                {
                  start = false;

                  // Add the path to the table
                  connections.insert(routingTable::value_type(path[0],path));

                  // Add the reverse path to the table
              reverse(path.begin(), path.end());
                  connections.insert(routingTable::value_type(path[0],path));
                  path.clear();
                }
                else
                {
                  // Add the next city to the path
                  path.push_back(city);
                }
        }

        // Start a new path after a line of '-'
   if( buffer[0] == '-' )
        {
          start = true;
        }

  }

  return true;
}


///////////////////////////////////////////////////////////////////////
// Read the network properties from the data files
// network.dat - distance of links
// connections.dat - connection routes
//
bool loadData(graph& network, routingTable& connections,
        string& brokenLink1, string& brokenLink2, ofstream& output)
{
```

```
// Open input files
ifstream networkData("network.dat");
if( !networkData )
{
  cout << "Error opening network data file" << endl;
  return false;
}

ifstream connectionData("connections.dat");
if( !connectionData )
{
  cout << "Error opening connection data file" << endl;
  return false;
}
cout << "Loading data files" << endl;

routingTable paths;
if( !loadPaths(paths) )
  return false;

output << "Network data" << endl;

string city1, city2;
double distance;

output << "Broken link between " << brokenLink1
   << " and " << brokenLink2 << endl;

// Read network data one line at a time
// Example: Anaheim Dallas 100.55
while( networkData >> city1 >> city2 >> distance )
{
  output << city1 << " to " << city2 << " = " << distance << endl;
  network[city1][city2] = distance;
  network[city2][city1] = distance;
}
output << endl;

output << "Connection data" << endl;

// Read connection data one line at a time
// Example 2 Dallas Anaheim
int count;
string head, tail;
while( connectionData >> count >> head >> tail )
{
        stringVector path;

  // Search for the path in the paths table
  routingTable::iterator route_it = paths.find(head);
        while( route_it != paths.end()
                && ((*route_it).first == head)
                        && ((*route_it).second.back() != tail) )
            route_it++;
```

```
          // If no path was found, calculate a minimum hop path
          if( route_it == paths.end() ||
            (*route_it).second.back() != tail )
          {
             cout << "Error - unable to find path : " << head
                      << " to " << tail << " in paths file. Using min hop path." << endl;

             // Get minimum hop tree from head
      stringMap tree;
      minHops(network,head,tree);

                   // Get the minimum hop route from the tree
      // This route will be reversed (tail to head)
      stringVector route;
      for( string prevCity = tail;
                prevCity != head;
                      prevCity = tree[prevCity] )
       route.push_back(prevCity);

      route.push_back(head);

      // Reverse the route (head to tail)
      reverse(route.begin(), route.end());

                   path = route;
          }
    else
          {
             // Use the path from the file
             path = (*route_it).second;
          }

          // Display the path
          string display;
          for( int i = 0; i < path.size(); i++)
                   display += path[i] + " ";
          output << display << endl;

    // Add the route to the cities connections
          for( int j = 0; j < count; j++)
                   connections.insert(routingTable::value_type(head,path));
  }
  output << endl;
  cout << connections.size() << " connections" << endl;

  // Remove the broken links from the network
  network[brokenLink1].erase(brokenLink2);
  network[brokenLink2].erase(brokenLink1);

  return true;
}

/////////////////////////////////////////////////////////////////////////
// Save the results to files
//
void saveResults(resultsQueue& results, messageQueue& results2,
          routingTable& newConnections)
{

 // Open output files
 ofstream output("results2.txt");
 if( !output)
 {
          cout << "Error opening output file results2.txt" << endl;
          return;
 }

 ofstream output2("results3.txt");
 if( !output2)
 {
          cout << "Error opening output file results3.txt" << endl;
          return;
 }

 cout << "Saving results" << endl;

 // Iterate through each city in the results queue
 for( resultsQueue::iterator it = results.begin();
    it != results.end();
    it++ )
 {
   string city = (*it).first;
   messageQueue& mqueue = (*it).second;

          output << city << endl << endl;

          // Show reconfigured connections for this city
          bool showHeader = true;
          for (routingTable::iterator route_it = newConnections.find(city);
               route_it != newConnections.end() &&
                             ((*route_it).first == city);
               ++route_it)
          {
             if( showHeader )
                    {
                      output << "Reconfigured Connections:" << endl;
                             showHeader = false;
                    }
             stringVector route = (*route_it).second;
                    for(stringVector::iterator it = route.begin();
                       it != route.end();
                              it++)
                              {
                                output << *it << " ";
                              }
                    output << endl;
          }

   // Iterate through the message queue for this city
          messageQueue reconfigQueue;  // Separate reconfig request messages
          messageQueue reconfig2Queue; // Separate reconfig request messages
          //                      by finishing times
```

```
while( !mqueue.empty() )
{
  Message msg = mqueue.top();
  mqueue.pop();
  switch( msg.type )
  {
    case ReconfigureRequest:
                          case StartReconfigure:
           // Save in the reconfig queue
                          {
                          reconfigQueue.push(msg);

                          // Change the message time to the message finishing time
                          Message msg2 = msg;
                          msg2.time = msg.time2;
                          reconfig2Queue.push(msg2);
                          }
    break;

    case ReconfigureComplete:
    // Display message
    output << msg.time << " : " << msg.headend
       << " to " << msg.tail << " Reconfigure Complete " << endl;
    break;
  }
}

       // Display the reconfig request messages
output << endl << "Reconfig Request Messages:" << endl;
while( !reconfigQueue.empty() )
{
  Message msg = reconfigQueue.top();
  reconfigQueue.pop();

  // Display message
  output << msg.time << " : " << msg.source
       << " to " << msg.destination << " Reconfigure Request for "
                            << msg.headend << " to " << msg.tail << endl;
       }

       // Display the reconfig request finished messages
output << endl << "Reconfig Request Messages (After Processing):"
            << endl;
while( !reconfig2Queue.empty() )
{
  Message msg = reconfig2Queue.top();
  reconfig2Queue.pop();

  // Display message
  output << msg.time << " : " << msg.source
       << " to " << msg.destination << " Reconfigure Request for "
                            << msg.headend << " to " << msg.tail << endl;
       }

output << "--------------------------------------------------------"
```

```
              << endl;
  }

  // Display the LinkDown messages
  output2 << "Link Down Messages" << endl << endl;
  while( !results2.empty() )
  {
    Message msg = results2.top();
    results2.pop();

    output2 << msg.time << " : " << msg.source
         << " to " << msg.destination << " Link Down" << endl;
  }
}

/////////////////////////////////////////////////////////////////////
// Initialize the connection iterator map
//
void initialize(routingTable& connections, iteratorMap& nextRoute)
{
          // Set each iterator to the first route for
          // the city in the connections table
          string city;
          for( routingTable::iterator route_it = connections.begin();
              route_it != connections.end();
                     route_it++ )
          {
            string nextCity = (*route_it).first;
            if( city != nextCity )
            {
                     city = nextCity;
                     nextRoute[city] = route_it;
            }
          }
}

/////////////////////////////////////////////////////////////////////
// Main processing loop for network simulation
//
void main()
{
  // Keep track of cities which have received a linkdown message
  booleanMap alreadyProcessed;

  // Create the message queue
  messageQueue messages;

  // Create the results queues
  resultsQueue results;
  messageQueue results2;

  // Create the network
  graph network;

  // Create the connections
  routingTable connections;
  routingTable newConnections;

  // Keep track of message processing times for each city
  doubleMap schedule;

  // Prompt user for broken link
  string brokenLink1, brokenLink2;
  cout << "Enter endpoints of broken link (i.e. Enter DLLSTX ANHMCA to break Dallas Anaheim link) :
";
  cin >> brokenLink1 >> brokenLink2;

  // Prompt user for Processing delay

  double delay;
  cout << "Enter Switch Processing Time (i.e. Enter 100 for 100 milliseconds) : ";
  cin >> delay;

  if( delay )

    processing_delay = delay / 1000;

  // Prompt user for Reconfigure delay

  delay = 0;
  cout << "Enter Switch Reconfiguration Time (i.e. Enter 50 for 50 milliseconds) : ";
  cin >> delay;

  if( delay )

          reconfig_delay = delay / 1000;

  linkdown_delay = processing_delay + initial_delay;

  // Open output file
  ofstream output("results1.txt");
  if( !output)
  {
          cout << "Error opening output file results1.txt" << endl;
          while(1);
  }



  output << "Processing delay = " << processing_delay << endl;

  output << "Reconfig delay = " << reconfig_delay << endl;

  // Initialize network
  if( !loadData(network, connections, brokenLink1, brokenLink2, output) )
  {
    cout << "Error loading data" << endl;
    while(1);
  }

  // Keep track of next route to be reconfigured
  iteratorMap nextRoute;
```

```
initialize(connections, nextRoute);

cout << "Starting simulation" << endl;

// Signal broken link
Message msg1(LinkDown, brokenLink1, brokenLink2,
        brokenLink1, brokenLink1, 0);
Message msg2(LinkDown, brokenLink2, brokenLink1,
        brokenLink2, brokenLink2, 0);
messages.push(msg1);
messages.push(msg2);

// Message Loop
while( !messages.empty() )
{
  Message msg = messages.top();
  messages.pop();
  switch( msg.type )
  {
    case ReconfigureRequest:
    // Display message
    output << msg.time << " : " << msg.source
       << " to " << msg.destination << " Reconfigure Request for "
                           << msg.headend << " to " << msg.tail;
           reconfigureRequest(messages, network, newConnections,
                              schedule, msg);
                  output << " : " << msg.time2 << endl;

                  // Copy the message to the results queue
           results[msg.destination].push(msg);
    break;

    case ReconfigureComplete:
    // Display message
    output << msg.time << " : " << msg.headend
       << " to " << msg.tail << " Reconfigure Complete " << endl;

                  // Copy the message to the results queue
           results[msg.destination].push(msg);
    break;

                  case StartReconfigure:
    // Display message
    output << msg.time << " : " << " Start Reconfiguration for "
                            << msg.headend << " to " << msg.tail << endl;

                  // Copy the message to the results queue
           results[msg.destination].push(msg);
    break;

    case LinkDown:
    if( !alreadyProcessed[msg.destination] )
    {
                    // Display message
                           output << msg.time << " : " << msg.source
       << " notifies " << msg.destination << " Link Is Down" << endl;
```

```
      linkDown(messages, network, msg);
      alreadyProcessed[msg.destination] = true;

            // Copy the message to the results queue
            results2.push(msg);
    }
    break;

                case ReconfigureNext:
                // Reconfigure the next connection
          reconfigure(messages, network, schedule, connections, newConnections,
                        nextRoute, msg);
  }
}

saveResults(results, results2, newConnections);
cout << "Done" << endl;

// Hold results window open
while(1);
                                                  }
```

APPENDIX E

Network.dat file

```
ALBYNY BSTNMA 166.2
ALBYNY NYCMNY 155.1
ALBYNY CHCGIL 817.9
ANHMCA DLLSTX 1424.9
ANHMCA SLKCUT 682.9
ANHMCA PHNXAZ 360.2
ANHMCA SNFCCA 406.6
ATLNGA WASHDC 637.4
ATLNGA CNCNOH 466.9
ATLNGA MIAMFL 694.4
BSTNMA NYCMNY 214.7
CHCGIL CNCNOH 296.1
CHCGIL KSCYMO 529.9
CHCGIL DLLSTX 966.9
CNCNOH KSCYMO 594.4
DNVRCO KSCYMO 601
DLLSTX TULSOK 310.4
DLLSTX DNVRCO 879
DLLSTX HSTNTX 239.2
DLLSTX KSCYMO 552.6
DNVRCO SLKCUT 532.9
HSTNTX ATLNGA 792.8
HSTNTX MIAMFL 1218.8
HSTNTX PHNXAZ 1179.4
SLKCUT SNFCCA 736
TULSOK KSCYMO 246.7
WASHDC NYCMNY 225.1
```

APPENDIX F

Connections.dat file for 500 circuits

```
2       ANHMCA     ALBYNY
11      ANHMCA     ATLNGA
9       ANHMCA     BSTNMA
22      ANHMCA     CHCGIL
4       ANHMCA     CNCNOH
9       ANHMCA     DLLSTX
5       ANHMCA     DNVRCO
11      ANHMCA     HSTNTX
5       ANHMCA     KSCYMO
6       ANHMCA     MIAMFL
8       ANHMCA     PHNXAZ
3       ANHMCA     SLKCUT
6       ANHMCA     SNFCCA
2       ANHMCA     TULSOK
13      ANHMCA     WASHDC
1       ATLNGA     ALBYNY
2       ATLNGA     CNCNOH
2       ATLNGA     DNVRCO
2       ATLNGA     KSCYMO
2       ATLNGA     MIAMFL
3       ATLNGA     PHNXAZ
1       ATLNGA     SLKCUT
1       ATLNGA     TULSOK
1       BSTNMA     ALBYNY
4       BSTNMA     ATLNGA
2       BSTNMA     CNCNOH
3       BSTNMA     DLLSTX
2       BSTNMA     DNVRCO
4       BSTNMA     HSTNTX
2       BSTNMA     KSCYMO
2       BSTNMA     MIAMFL
3       BSTNMA     PHNXAZ
1       BSTNMA     SLKCUT
1       BSTNMA     TULSOK
8       CHCGIL     BSTNMA
2       CHCGIL     ALBYNY
9       CHCGIL     ATLNGA
4       CHCGIL     CNCNOH
8       CHCGIL     DLLSTX
5       CHCGIL     DNVRCO
9       CHCGIL     HSTNTX
4       CHCGIL     KSCYMO
5       CHCGIL     MIAMFL
7       CHCGIL     PHNXAZ
3       CHCGIL     SLKCUT
6       CHCGIL     SNFCCA
2       CHCGIL     TULSOK
11      CHCGIL     WASHDC
```

```
1       CNCNOH     KSCYMO
1       CNCNOH     SLKCUT
1       DLLSTX     ALBYNY
4       DLLSTX     ATLNGA
2       DLLSTX     CNCNOH
2       DLLSTX     DNVRCO
4       DLLSTX     HSTNTX
2       DLLSTX     KSCYMO
2       DLLSTX     MIAMFL
3       DLLSTX     PHNXAZ
1       DLLSTX     SLKCUT
1       DLLSTX     TULSOK
1       DNVRCO     ALBYNY
1       DNVRCO     CNCNOH
1       DNVRCO     KSCYMO
1       DNVRCO     SLKCUT
1       HSTNTX     ALBYNY
5       HSTNTX     ATLNGA
2       HSTNTX     CNCNOH
2       HSTNTX     DNVRCO
2       HSTNTX     KSCYMO
3       HSTNTX     MIAMFL
4       HSTNTX     PHNXAZ
2       HSTNTX     SLKCUT
1       HSTNTX     TULSOK
1       KSCYMO     SLKCUT
1       MIAMFL     ALBYNY
1       MIAMFL     CNCNOH
1       MIAMFL     DNVRCO
1       MIAMFL     KSCYMO
2       MIAMFL     PHNXAZ
1       MIAMFL     SLKCUT
1       MIAMFL     TULSOK
2       NYCMNY     ALBYNY
24      NYCMNY     ANHMCA
10      NYCMNY     ATLNGA
8       NYCMNY     BSTNMA
21      NYCMNY     CHCGIL
4       NYCMNY     CNCNOH
8       NYCMNY     DLLSTX
5       NYCMNY     DNVRCO
10      NYCMNY     HSTNTX
4       NYCMNY     KSCYMO
6       NYCMNY     MIAMFL
8       NYCMNY     PHNXAZ
3       NYCMNY     SLKCUT
6       NYCMNY     SNFCCA
2       NYCMNY     TULSOK
```

12	NYCMNY	WASHDC
1	PHNXAZ	ALBYNY
1	PHNXAZ	CNCNOH
2	PHNXAZ	DNVRCO
2	PHNXAZ	KSCYMO
1	PHNXAZ	SLKCUT
1	PHNXAZ	TULSOK
1	SNFCCA	ALBYNY
3	SNFCCA	ATLNGA
2	SNFCCA	BSTNMA
1	SNFCCA	CNCNOH
2	SNFCCA	DLLSTX
1	SNFCCA	DNVRCO
3	SNFCCA	HSTNTX
1	SNFCCA	KSCYMO
2	SNFCCA	MIAMFL
2	SNFCCA	PHNXAZ
1	SNFCCA	SLKCUT
1	SNFCCA	TULSOK
3	SNFCCA	WASHDC
1	WASHDC	ALBYNY
5	WASHDC	ATLNGA
5	WASHDC	BSTNMA
2	WASHDC	CNCNOH
5	WASHDC	DLLSTX
3	WASHDC	DNVRCO
6	WASHDC	HSTNTX
2	WASHDC	KSCYMO
3	WASHDC	MIAMFL
4	WASHDC	PHNXAZ
2	WASHDC	SLKCUT
1	WASHDC	TULSOK

APPENDIX G

Sample portion of Paths.dat file

```
************************************************************
Link:  ALBYNY (node 0) <--> BSTNMA (node 1)

Minimum Hop Path:
---------------------------------
ALBYNY
BSTNMA
Number of Hops = 1
************************************************************

************************************************************
Link:  ALBYNY (node 0) <--> NYCMNY (node 2)

Minimum Hop Path:
---------------------------------
ALBYNY
NYCMNY
Number of Hops = 1
************************************************************

************************************************************
Link:  ALBYNY (node 0) <--> CHCGIL (node 3)

Minimum Hop Path:
---------------------------------
ALBYNY
CHCGIL
Number of Hops = 1
************************************************************

************************************************************
Link:  ALBYNY (node 0) <--> WASHDC (node 4)

Minimum Hop Path:
---------------------------------
ALBYNY
NYCMNY
WASHDC
Number of Hops = 2
************************************************************

************************************************************
Link:  ALBYNY (node 0) <--> ATLNGA (node 5)

Minimum Hop Path:
---------------------------------
ALBYNY
NYCMNY
```

```
WASHDC
ATLNGA
Number of Hops = 3
*************************************************************

*************************************************************
Link:  ALBYNY (node 0) <--> CNCNOH (node 6)

Minimum Hop Path:
---------------------------------
ALBYNY
CHCGIL
CNCNOH
Number of Hops = 2
*************************************************************

*************************************************************
Link:  ALBYNY (node 0) <--> KSCYMO (node 7)

Minimum Hop Path:
---------------------------------
ALBYNY
CHCGIL
KSCYMO
Number of Hops = 2
*************************************************************

*************************************************************
Link:  ALBYNY (node 0) <--> DLLSTX (node 8)

Minimum Hop Path:
---------------------------------
ALBYNY
CHCGIL
DLLSTX
Number of Hops = 2
*************************************************************

*************************************************************
Link:  ALBYNY (node 0) <--> HSTNTX (node 9)

Minimum Hop Path:
---------------------------------
ALBYNY
CHCGIL
DLLSTX
HSTNTX
Number of Hops = 3
*************************************************************

*************************************************************
Link:  ALBYNY (node 0) <--> MIAMFL (node 10)

Minimum Hop Path:
---------------------------------
ALBYNY
NYCMNY
```

APPENDIX H

Sample Portion of Results 1

```
Processing delay = 0.1
Reconfig delay = 0.05
Network data
Broken link between DLLSTX and ANHMCA
ALBYNY to BSTNMA = 166.2
ALBYNY to NYCMNY = 155.1
ALBYNY to CHCGIL = 817.9
ANHMCA to DLLSTX = 1424.9
ANHMCA to SLKCUT = 682.9
ANHMCA to PHNXAZ = 360.2
ANHMCA to SNFCCA = 406.6
ATLNGA to WASHDC = 637.4
ATLNGA to CNCNOH = 466.9
ATLNGA to MIAMFL = 694.4
BSTNMA to NYCMNY = 214.7
CHCGIL to CNCNOH = 296.1
CHCGIL to KSCYMO = 529.9
CHCGIL to DLLSTX = 966.9
CNCNOH to KSCYMO = 594.4
DNVRCO to KSCYMO = 601
DLLSTX to TULSOK = 310.4
DLLSTX to DNVRCO = 879
DLLSTX to HSTNTX = 239.2
DLLSTX to KSCYMO = 552.6
DNVRCO to SLKCUT = 532.9
HSTNTX to ATLNGA = 792.8
HSTNTX to MIAMFL = 1218.8
HSTNTX to PHNXAZ = 1179.4
SLKCUT to SNFCCA = 736
TULSOK to KSCYMO = 246.7
WASHDC to NYCMNY = 225.1

Connection data
ANHMCA DLLSTX CHCGIL ALBYNY
ANHMCA DLLSTX HSTNTX ATLNGA
ANHMCA DLLSTX CHCGIL ALBYNY BSTNMA
ANHMCA DLLSTX CHCGIL
ANHMCA DLLSTX CHCGIL CNCNOH
ANHMCA SNFCCA
ANHMCA DLLSTX TULSOK
ANHMCA DLLSTX HSTNTX ATLNGA WASHDC
ATLNGA WASHDC NYCMNY ALBYNY
ATLNGA CNCNOH
ATLNGA CNCNOH KSCYMO DNVRCO
ATLNGA CNCNOH KSCYMO
ATLNGA CNCNOH KSCYMO DNVRCO SLKCUT
ATLNGA CNCNOH KSCYMO TULSOK
BSTNMA ALBYNY
BSTNMA NYCMNY WASHDC ATLNGA
BSTNMA ALBYNY CHCGIL CNCNOH
BSTNMA ALBYNY CHCGIL DLLSTX
BSTNMA ALBYNY CHCGIL KSCYMO DNVRCO
BSTNMA ALBYNY CHCGIL DLLSTX HSTNTX
BSTNMA ALBYNY CHCGIL KSCYMO
CHCGIL ALBYNY
CHCGIL CNCNOH ATLNGA
CHCGIL CNCNOH
CHCGIL DLLSTX
CHCGIL KSCYMO DNVRCO
CHCGIL DLLSTX HSTNTX
CHCGIL KSCYMO
HSTNTX PHNXAZ
HSTNTX DLLSTX DNVRCO SLKCUT
HSTNTX DLLSTX TULSOK
KSCYMO DNVRCO SLKCUT
MIAMFL ATLNGA WASHDC NYCMNY ALBYNY
MIAMFL ATLNGA CNCNOH
MIAMFL HSTNTX DLLSTX DNVRCO
WASHDC ATLNGA CNCNOH KSCYMO DNVRCO SLKCUT
WASHDC ATLNGA CNCNOH KSCYMO TULSOK

0 : DLLSTX to DLLSTX Link Down
0 : ANHMCA to ANHMCA Link Down
0.100125 :  Start Reconfiguration for ANHMCA to ALBYNY
0.102264 : DLLSTX to HSTNTX Link Down
0.1029 : DLLSTX to TULSOK Link Down
0.103345 : ANHMCA to PHNXAZ Link Down
0.103345 : ANHMCA to PHNXAZ Reconfigure Request for ANHMCA to ALBYNY : 0.153345
0.10376 : ANHMCA to SNFCCA Link Down
0.105066 : DLLSTX to KSCYMO Link Down
0.106231 : ANHMCA to SLKCUT Link Down
0.107984 : DLLSTX to DNVRCO Link Down
0.10877 : DLLSTX to CHCGIL Link Down
0.11389 : PHNXAZ to HSTNTX Reconfigure Request for ANHMCA to ALBYNY : 0.16389
0.116029 : HSTNTX to DLLSTX Reconfigure Request for ANHMCA to ALBYNY : 0.166029
0.124674 : DLLSTX to CHCGIL Reconfigure Request for ANHMCA to ALBYNY : 0.174674
0.131987 : CHCGIL to ALBYNY Reconfigure Request for ANHMCA to ALBYNY : 0.181987
0.150125 :  Start Reconfiguration for ANHMCA to ALBYNY
0.153345 : ANHMCA to PHNXAZ Reconfigure Request for ANHMCA to ALBYNY : 0.203345
0.16389 : PHNXAZ to HSTNTX Reconfigure Request for ANHMCA to ALBYNY : 0.21389
0.166029 : HSTNTX to DLLSTX Reconfigure Request for ANHMCA to ALBYNY : 0.216029
0.174674 : DLLSTX to CHCGIL Reconfigure Request for ANHMCA to ALBYNY : 0.224674
0.181987 : ANHMCA to ALBYNY Reconfigure Complete
0.181987 : CHCGIL to ALBYNY Reconfigure Request for ANHMCA to ALBYNY : 0.231987
0.200125 :  Start Reconfiguration for ANHMCA to ATLNGA
0.203345 : ANHMCA to PHNXAZ Reconfigure Request for ANHMCA to ATLNGA : 0.253345
0.203885 :  Start Reconfiguration for SNFCCA to ALBYNY
0.209477 : HSTNTX to ATLNGA Link Down
0.210466 : SNFCCA to SLKCUT Reconfigure Request for SNFCCA to ALBYNY : 0.260466
0.210505 : KSCYMO to CNCNOH Link Down
0.213286 : HSTNTX to MIAMFL Link Down
0.21389 : PHNXAZ to HSTNTX Reconfigure Request for ANHMCA to ATLNGA : 0.26389
0.21523 : SLKCUT to DNVRCO Reconfigure Request for SNFCCA to ALBYNY : 0.26523
0.216208 : CHCGIL to ALBYNY Link Down
```

```
0.220979 : HSTNTX to ATLNGA Reconfigure Request for ANHMCA to ATLNGA : 0.270979
0.223089 : DNVRCO to DLLSTX Reconfigure Request for SNFCCA to ALBYNY : 0.273089
0.224674 :  Start Reconfiguration for CHCGIL to SNFCCA
0.231734 : DLLSTX to CHCGIL Reconfigure Request for SNFCCA to ALBYNY : 0.324674
0.231987 : ANHMCA to ALBYNY Reconfigure Complete
0.233319 : CHCGIL to DLLSTX Reconfigure Request for CHCGIL to SNFCCA : 0.323089
0.250125 :  Start Reconfiguration for ANHMCA to ATLNGA
0.253345 : ANHMCA to PHNXAZ Reconfigure Request for ANHMCA to ATLNGA : 0.303345
0.253885 :  Start Reconfiguration for SNFCCA to ATLNGA
0.257521 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to ATLNGA : 0.350125
0.26389 : PHNXAZ to HSTNTX Reconfigure Request for ANHMCA to ATLNGA : 0.31389
0.270979 : ANHMCA to ATLNGA Reconfigure Complete
0.270979 : HSTNTX to ATLNGA Reconfigure Request for ANHMCA to ATLNGA : 0.320979
0.280948 : DLLSTX to DNVRCO Reconfigure Request for CHCGIL to SNFCCA : 0.330948
0.281987 : CHCGIL to ALBYNY Reconfigure Request for SNFCCA to ALBYNY : 0.331987
0.285713 : DNVRCO to SLKCUT Reconfigure Request for CHCGIL to SNFCCA : 0.335713
0.292294 : SLKCUT to SNFCCA Reconfigure Request for CHCGIL to SNFCCA : 0.353885
0.303345 : ANHMCA to PHNXAZ Reconfigure Request for SNFCCA to ATLNGA : 0.353345
0.31389 : PHNXAZ to HSTNTX Reconfigure Request for SNFCCA to ATLNGA : 0.36389
0.315301 : ATLNGA to WASHDC Link Down
0.317719 : ALBYNY to NYCMNY Link Down
0.317819 : ALBYNY to BSTNMA Link Down
0.320979 : HSTNTX to ATLNGA Reconfigure Request for SNFCCA to ATLNGA : 0.370979
0.320979 : ANHMCA to ATLNGA Reconfigure Complete
0.324674 :  Start Reconfiguration for CHCGIL to SNFCCA
0.331987 : SNFCCA to ALBYNY Reconfigure Complete
0.333319 : CHCGIL to DLLSTX Reconfigure Request for CHCGIL to SNFCCA : 0.383319
0.341178 : DLLSTX to DNVRCO Reconfigure Request for CHCGIL to SNFCCA : 0.391178
0.345942 : DNVRCO to SLKCUT Reconfigure Request for CHCGIL to SNFCCA : 0.395942
0.350125 :  Start Reconfiguration for ANHMCA to ATLNGA
0.352523 : SLKCUT to SNFCCA Reconfigure Request for CHCGIL to SNFCCA : 0.453885
0.353345 : ANHMCA to PHNXAZ Reconfigure Request for ANHMCA to ATLNGA : 0.403345
0.353885 :  Start Reconfiguration for SNFCCA to ATLNGA
0.353885 : CHCGIL to SNFCCA Reconfigure Complete
0.357521 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to ATLNGA : 0.450125
0.36389 : PHNXAZ to HSTNTX Reconfigure Request for ANHMCA to ATLNGA : 0.41389
0.370979 : SNFCCA to ATLNGA Reconfigure Complete
0.370979 : HSTNTX to ATLNGA Reconfigure Request for ANHMCA to ATLNGA : 0.420979
0.374674 :  Start Reconfiguration for CHCGIL to SNFCCA
0.383319 : CHCGIL to DLLSTX Reconfigure Request for CHCGIL to SNFCCA : 0.433319
0.391178 : DLLSTX to DNVRCO Reconfigure Request for CHCGIL to SNFCCA : 0.441178
0.395942 : DNVRCO to SLKCUT Reconfigure Request for CHCGIL to SNFCCA : 0.445942
0.402523 : SLKCUT to SNFCCA Reconfigure Request for CHCGIL to SNFCCA : 0.503885
0.403345 : ANHMCA to PHNXAZ Reconfigure Request for SNFCCA to ATLNGA : 0.453345
0.41389 : PHNXAZ to HSTNTX Reconfigure Request for SNFCCA to ATLNGA : 0.46389
0.417844 :  Start Reconfiguration for NYCMNY to ANHMCA
0.419857 : NYCMNY to WASHDC Reconfigure Request for NYCMNY to ANHMCA : 0.469857
0.420979 : ANHMCA to ATLNGA Reconfigure Complete
0.420979 : HSTNTX to ATLNGA Reconfigure Request for SNFCCA to ATLNGA : 0.470979
0.424674 :  Start Reconfiguration for CHCGIL to SNFCCA
0.425556 : WASHDC to ATLNGA Reconfigure Request for NYCMNY to ANHMCA : 0.520979
0.433319 : CHCGIL to DLLSTX Reconfigure Request for CHCGIL to SNFCCA : 0.483319
0.441178 : DLLSTX to DNVRCO Reconfigure Request for CHCGIL to SNFCCA : 0.491178
0.445942 : DNVRCO to SLKCUT Reconfigure Request for CHCGIL to SNFCCA : 0.495942
0.450125 :  Start Reconfiguration for ANHMCA to ATLNGA
0.452523 : SLKCUT to SNFCCA Reconfigure Request for CHCGIL to SNFCCA : 0.603885
```

APPENDIX I

Sample Portion of Results 2

```
ALBYNY

Reconfig Request Messages:
0.131987 : CHCGIL to ALBYNY Reconfigure Request for ANHMCA to ALBYNY
0.181987 : CHCGIL to ALBYNY Reconfigure Request for ANHMCA to ALBYNY
0.281987 : CHCGIL to ALBYNY Reconfigure Request for SNFCCA to ALBYNY
0.739047 : CHCGIL to ALBYNY Reconfigure Request for SNFCCA to BSTNMA
0.789047 : CHCGIL to ALBYNY Reconfigure Request for SNFCCA to BSTNMA
1.61923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
1.66923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
1.71923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
1.76923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
1.81923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
1.86923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA

Reconfig Request Messages (After Processing):
0.181987 : CHCGIL to ALBYNY Reconfigure Request for ANHMCA to ALBYNY
0.231987 : CHCGIL to ALBYNY Reconfigure Request for ANHMCA to ALBYNY
0.331987 : CHCGIL to ALBYNY Reconfigure Request for SNFCCA to ALBYNY
0.789047 : CHCGIL to ALBYNY Reconfigure Request for SNFCCA to BSTNMA
0.839047 : CHCGIL to ALBYNY Reconfigure Request for SNFCCA to BSTNMA
1.66923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
1.71923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
1.76923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
1.81923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
1.86923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
1.91923 : NYCMNY to ALBYNY Reconfigure Request for NYCMNY to SNFCCA
-------------------------------------------------------
ANHMCA

Reconfigured Connections:
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL ALBYNY
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL ALBYNY
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA
ANHMCA PHNXAZ HSTNTX ATLNGA WASHDC NYCMNY BSTNMA
ANHMCA PHNXAZ HSTNTX ATLNGA WASHDC NYCMNY BSTNMA
ANHMCA PHNXAZ HSTNTX ATLNGA WASHDC NYCMNY BSTNMA
ANHMCA PHNXAZ HSTNTX ATLNGA WASHDC NYCMNY BSTNMA
ANHMCA PHNXAZ HSTNTX ATLNGA WASHDC NYCMNY BSTNMA
ANHMCA PHNXAZ HSTNTX ATLNGA WASHDC NYCMNY BSTNMA
ANHMCA PHNXAZ HSTNTX ATLNGA WASHDC NYCMNY BSTNMA
ANHMCA PHNXAZ HSTNTX ATLNGA WASHDC NYCMNY BSTNMA
ANHMCA PHNXAZ HSTNTX ATLNGA WASHDC NYCMNY BSTNMA
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX DLLSTX CHCGIL
ANHMCA PHNXAZ HSTNTX ATLNGA CNCNOH
ANHMCA PHNXAZ HSTNTX ATLNGA CNCNOH
ANHMCA PHNXAZ HSTNTX ATLNGA CNCNOH
ANHMCA PHNXAZ HSTNTX ATLNGA CNCNOH
ANHMCA PHNXAZ HSTNTX DLLSTX
ANHMCA PHNXAZ HSTNTX DLLSTX
ANHMCA PHNXAZ HSTNTX DLLSTX
ANHMCA PHNXAZ HSTNTX DLLSTX
ANHMCA PHNXAZ HSTNTX DLLSTX
ANHMCA PHNXAZ HSTNTX DLLSTX
ANHMCA PHNXAZ HSTNTX DLLSTX
ANHMCA PHNXAZ HSTNTX DLLSTX
ANHMCA PHNXAZ HSTNTX DLLSTX
ANHMCA SLKCUT DNVRCO
ANHMCA SLKCUT DNVRCO
ANHMCA SLKCUT DNVRCO
ANHMCA SLKCUT DNVRCO
ANHMCA SLKCUT DNVRCO
ANHMCA PHNXAZ HSTNTX
ANHMCA PHNXAZ HSTNTX
ANHMCA PHNXAZ HSTNTX
ANHMCA PHNXAZ HSTNTX
ANHMCA PHNXAZ HSTNTX
ANHMCA PHNXAZ HSTNTX
ANHMCA PHNXAZ HSTNTX
ANHMCA PHNXAZ HSTNTX
ANHMCA PHNXAZ HSTNTX
ANHMCA PHNXAZ HSTNTX
ANHMCA PHNXAZ HSTNTX
ANHMCA SLKCUT DNVRCO KSCYMO
ANHMCA SLKCUT DNVRCO KSCYMO
ANHMCA SLKCUT DNVRCO KSCYMO
ANHMCA SLKCUT DNVRCO KSCYMO
ANHMCA SLKCUT DNVRCO KSCYMO
ANHMCA PHNXAZ HSTNTX MIAMFL
ANHMCA PHNXAZ HSTNTX MIAMFL
ANHMCA PHNXAZ HSTNTX MIAMFL
ANHMCA PHNXAZ HSTNTX MIAMFL
ANHMCA PHNXAZ HSTNTX MIAMFL
ANHMCA PHNXAZ HSTNTX MIAMFL
ANHMCA PHNXAZ HSTNTX DLLSTX TULSOK
```

```
Reconfig Request Messages:
0.100125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ALBYNY
0.150125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ALBYNY
0.200125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.250125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.257521 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to ATLNGA
0.350125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.357521 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to ATLNGA
0.450125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.500125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.507521 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to ATLNGA
0.556566 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
0.600125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.700125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.706566 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
0.800125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.806566 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
0.900125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.906566 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
0.957521 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to HSTNTX
1.00013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
1.00657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.00752 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to HSTNTX
1.05752 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to HSTNTX
1.10657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.15752 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to MIAMFL
1.20013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
1.20657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.20752 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to MIAMFL
1.30657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.30752 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to WASHDC
1.35752 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to WASHDC
1.40752 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to WASHDC
1.45657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.50013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
1.55657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.60657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.75657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.80013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
1.85657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.00013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.00657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
```

```
2.10657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.15013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.15657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.20657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.30013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.30657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.35657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.45012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.45657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.50657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.60012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.60657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.70657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.75012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.75657 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.90012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.95012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.00012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.05012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.10012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.15012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.20012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.25012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.30012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.35012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.40012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.45012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.50012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.55012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.60012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.65012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.70012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.75012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.80012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.85012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.90012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.95012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
4.00012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
4.05012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CNCNOH
4.10012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CNCNOH
4.15012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CNCNOH
4.20012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CNCNOH
4.25012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DLLSTX
4.30012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DLLSTX
4.35012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DLLSTX
4.40012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DLLSTX
4.45012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DLLSTX
4.50012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DLLSTX
4.55012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DLLSTX
4.60012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DLLSTX
4.65012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DLLSTX
4.70012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DNVRCO
4.75012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DNVRCO
4.80012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DNVRCO
4.85012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DNVRCO
4.90012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to DNVRCO
```

```
4.95012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.00012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.05012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.10012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.15012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.20012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.25012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.30012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.35012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.40012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.45012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to HSTNTX
5.50012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to KSCYMO
5.55012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to KSCYMO
5.60012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to KSCYMO
5.65012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to KSCYMO
5.70012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to KSCYMO
5.75012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to MIAMFL
5.80012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to MIAMFL
5.85012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to MIAMFL
5.90012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to MIAMFL
5.95012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to MIAMFL
6.00012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to MIAMFL
6.05012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to TULSOK
6.10012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to TULSOK
6.15012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.20012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.25012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.30012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.35012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.40012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.45012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.50012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.55012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.60012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.65012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.70012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
6.75012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to WASHDC
```

Reconfig Request Messages (After Processing):

```
0.150125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ALBYNY
0.200125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ALBYNY
0.250125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.300125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.350125 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to ATLNGA
0.400125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.450125 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to ATLNGA
0.500125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.550125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.600125 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to ATLNGA
0.650125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.700125 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
0.750125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.800125 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
0.850125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
0.900125 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
0.950125 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
```

```
1.00013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.05013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
1.10013 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to HSTNTX
1.15013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.20013 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to HSTNTX
1.25013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to ATLNGA
1.30013 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to HSTNTX
1.35013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.40013 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to MIAMFL
1.45013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.50013 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to MIAMFL
1.55013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
1.60013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.65013 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to WASHDC
1.70013 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to WASHDC
1.75013 : SNFCCA to ANHMCA Reconfigure Request for SNFCCA to WASHDC
1.80013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.85013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
1.90013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
1.95013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.00013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.05013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.10013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.15013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.20013 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.25013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.30013 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.35012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.40012 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.45012 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.50012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.55012 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.60012 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.65012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.70012 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.75012 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.80012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
2.85012 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.90012 : PHNXAZ to ANHMCA Reconfigure Request for NYCMNY to ANHMCA
2.95012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to BSTNMA
3.00012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
3.05012 : ANHMCA to ANHMCA Reconfigure Request for ANHMCA to CHCGIL
```

APPENDIX J

Results 3

```
0 : DLLSTX notifies DLLSTX Link Is Down
0 : ANHMCA notifies ANHMCA Link Is Down
0.102264 : DLLSTX notifies HSTNTX Link Is Down
0.1029 : DLLSTX notifies TULSOK Link Is Down
0.103345 : ANHMCA notifies PHNXAZ Link Is Down
0.10376 : ANHMCA notifies SNFCCA Link Is Down
0.105066 : DLLSTX notifies KSCYMO Link Is Down
0.106231 : ANHMCA notifies SLKCUT Link Is Down
0.107984 : DLLSTX notifies DNVRCO Link Is Down
0.10877 : DLLSTX notifies CHCGIL Link Is Down
0.209477 : HSTNTX notifies ATLNGA Link Is Down
0.210505 : KSCYMO notifies CNCNOH Link Is Down
0.213286 : HSTNTX notifies MIAMFL Link Is Down
0.216208 : CHCGIL notifies ALBYNY Link Is Down
0.315301 : ATLNGA notifies WASHDC Link Is Down
0.317719 : ALBYNY notifies NYCMNY Link Is Down
0.317819 : ALBYNY notifies BSTNMA Link Is Down
```

VITA

Wai Yeu Chan

Candidate for the Degree of

Master of Science

Thesis: DEVELOPMENT OF A SIMULATION TOOL FOR ESTIMATING THE RECONFIGURATION AND RESTORATION TIMES OF A DISTRIBUTED INTELLIGENCE OPTICAL MESH NETWORK

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Penang, Malaysia, on December 10, 1977, the son of Mr. and Mrs. Chan Ah Swee. The youngest child of 5 brothers and sisters. Personal hobbies include, cooking, traveling, snow skiing, watching television, playing and watching tennis games, and pretty much any kind of outdoor activities. Very outgoing, friendly and enjoy meeting new people.

Education: Graduated from St. Xavier's Institution, Penang, Malaysia in 1995. Received Associate Degree in Engineering from Rima College, Penang, Malaysia, Summer 1996. Graduated from Oklahoma State University with a Bachelor of Science in Electrical Engineering, May 1999. Completed the requirements for the Master of Science degree at Oklahoma State University in December 2001.

Professional: Interned at Williams Communications Group as a Sales Engineer, May 2000 to August 2001. Research Assistant at the Department of Electrical and Computer Engineering, Oklahoma State University, August 2000 to August 2001.