Connection rerouting for network reconfiguration

Nitin Jose
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Recommended Citation
Jose, Nitin, "Connection rerouting for network reconfiguration" (2003). Retrospective Theses and Dissertations. 19444.
https://lib.dr.iastate.edu/rtd/19444

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Connection rerouting for network reconfiguration

by

Nitin Jose

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Arun K. Somani, Major Professor
Ahmed E. Kamal
Soma Chaudhuri

Iowa State University
Ames, Iowa

2003

Copyright © Nitin Jose, 2003. All rights reserved.
Graduate College
Iowa State University

This is to certify that the master’s thesis of

Nitin Jose

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy
TABLE OF CONTENTS

LIST OF TABLES ................................................................. v
LIST OF FIGURES ................................................................. vi
ABSTRACT ................................................................. vii

CHAPTER 1 INTRODUCTION ........................................ 1
1.1 Introduction ....................................................... 1
1.1.1 Outline ....................................................... 2
1.2 Motivation for Rerouting of Connections ...................... 3

CHAPTER 2 CONNECTION REROUTING .............................. 4
2.1 Objective .......................................................... 4
2.2 Issues in Connection rerouting .................................. 5
2.3 Previous work ...................................................... 6
2.4 Rerouting/Reconfiguration Methods ............................ 7
2.5 Mathematical Description of the Problem ...................... 8

CHAPTER 3 PROCEDURE FOR REROUTING OF CONNECTIONS 11
3.1 Resolving Cyclic Dependencies ............................... 11
LIST OF TABLES

Table 4.1  Results for NSFNET topology ........................................ 29
Table 4.2  Results for NJLATA topology ........................................ 30
Table 4.3  Results for MESH 3x3 topology ...................................... 30
Table 4.4  Results for MESH 4x4 topology ...................................... 30
LIST OF FIGURES

Figure 3.1  Rerouting in the case of a link failure  . . . . . . . . . . . . . . . . . . . . . . . 12
Figure 3.2  Dependency graph for the rerouting for failure of link (7, 8) . . . 13
Figure 3.3  Alternate routing for the same example. . . . . . . . . . . . . . . . . . . . . . 14
Figure 3.4  Rerouting in the case of failure of links (1, 2) or (1, 4) . . . . . . . . 15
Figure 3.5  Example to illustrate minimal decyclization. . . . . . . . . . . . . . . . . 19
Figure 3.6  Example to illustrate determination of rerouting or . . . . . . . . . . . . . 22
Figure 4.1  Dependent connections vs Request rate  . . . . . . . . . . . . . . . . . . . . . . 31
Figure 4.2  Mean Path Length vs Request rate  . . . . . . . . . . . . . . . . . . . . . . . . 32
Figure 4.3  Mean Disruption Time vs Request rate  . . . . . . . . . . . . . . . . . . . . . . 33
ABSTRACT

Networks employing wavelength division multiplexing carry huge volumes of traffic, hence maintaining a high level of service availability is important. This makes fault tolerance an important issue. In general the failed connections need to be rerouted. After a link or node failure, alternate paths for restoration of the failed connections may exist only if connection rerouting is performed. In addition to link or node failure scenarios, connection rerouting or network reconfiguration may also prove helpful for accommodating new requests which may otherwise be blocked. Since connections on backbone optical networks, have longer call durations, it is crucial to accommodate new blocked requests through alternate means, instead of blocking them till existing connections terminate and make a path available. These situations make network reconfiguration important.

With the maturing of optical communication technology and increase in network traffic, most connections plying on today's backbone networks are high data rate connections. It is therefore critical that the connections have minimal disruption times during network reconfiguration. In this thesis, we propose a methodology for minimizing the disruption time and the number of disruptions in the network during connection rerouting.
CHAPTER 1 INTRODUCTION

1.1 Introduction

The enormous bandwidth required in backbone networks, and the nature of connections on them make fault tolerance an important concern. The connections on the backbone networks service large institutions or ISP's and hence are monitorily very important. As in most networks minimizing call blocking and survivability are core goals. Further they would require that the network connections have very low disruption time. A failure in back-bone networks can have serious repercussions, causing huge financial losses and interrupting many services or industries that are dependent on the network. In the event of a link or node failure, the failed connections would have to be rerouted. In general since a physical link in a WDM (Wavelength division multiplexing) optical network, i.e., an optical cable, comprises of many optical fibers supporting sometimes hundreds of wavelengths and many TDM (Time Division Multiplexing) time-slots per WDM wavelength channel, the number of failed connections in the case of an optical cable cut could be large. Rerouting all these connections on the existing network may not be possible. This will result in dropping a large number of connections. Fault toler-
 ance by allocating bandwidth for backup paths will result in very low network resource utilization; and dropping all failed connections is not advisable. In light of this alternate means like rerouting connections in an effort to accommodate the failed connections becomes an attractive proposition. A lightpath (LP), which is an all-optical wavelength division multiplexed (WDM) channel, is taken as the minimum unit of reconfiguration. The rerouting approach involves finding new lightpaths (LP) in the faulty network (LP design), and then realizing these new LPs by selectively removing the old LPs (LP realization) [9].

1.1.1 Outline

The remainder of this thesis is organized as follows: In Section 1.2 we introduce the motivations and applications for rerouting of connections. In Sections 2.1 and 2.2 we discuss the objective of our work and the issues involved in connection rerouting and the various ways of doing it respectively. In Section 2.3 we discuss previous related research. In Section 2.4 we look into the types of rerouting/reconfiguration methods. In Section 2.5 we develop a mathematical representation of the problem. In Section 3.1 we explain methods for resolving cyclic dependencies in a resource dependency graph for determining the order of rerouting. In Section 3.2 we present the algorithms involved. In Section 4.1 the metrics we use for comparing rerouting methods are introduced. In Section 4.2 we show the results of our simulation based experimentation. In Section 4.3 we present our conclusions and describe possible future work.
1.2 Motivation for Rerouting of Connections

In static routing, the path of the connection between a given source-destination pair is predetermined and fixed. In contrast in a network with dynamic routing, the order of the routed requests determines the paths of the connections. If a call cannot be rerouted/established on a given network, it may be accommodated if some of the existing connections are rerouted. We consider such scenario where connection rerouting is helpful.

A common scenario would be in the event of a link or node failure, where the failed connections can be rerouted if alternate paths exist. Otherwise rerouting existing connections might provide a path for the failed connections. The former technique of restoration is of two kinds, link based and path based. They do not involve disruptions of connections unaffected by the failures. The latter reconfiguration approach is discussed in this paper. It has been shown in [3], that the latter reconfiguration approach dealt with in this paper is superior to the former rerouting approach in terms of wavelength usage efficiency.

Another scenario where rerouting might help is in easing congestion. If the network is partially congested due to non-uniform traffic distribution, then rerouting the connections in the congested parts of the network, may help ease congestion. Thus we could have lower blocking probability and better utilization of the network bandwidth. Further if there exists no path for routing a new request of high priority, it can be accommodated by rerouting such that a low priority connection is disrupted to accept the new request.
CHAPTER 2 CONNECTION REROUTING

2.1 Objective

Our objective is to find a suitable method to perform connection rerouting. We assume that we are given an initial network topology $G$ with a set of existing connections $C$ and also a target network topology $G'$ with its set of connections $C'$ of which some maybe new connections and some of the old connections may have been rerouted or dropped. Several researchers [4, 5, 6] have addressed the issues related to been published dealing with obtaining a suitable target network topology. In this paper we do not delve into this aspect and instead assume that we already have a target network topology at hand. In general since the transition from an old topology to a new topology would lead to considerable overheads and traffic disruptions, network designers should consider minimizing the reconfiguration cost in addition to optimizing network performance in the process of obtaining the target topology for dynamic reconfiguration. In [4, 5], minimizing the connections involved in rerouting have been considered while obtaining the target topologies. A more complete mathematical description of the problem addressed is given in Section 2.5. This transition from the initial network topology to the target
network topology is called the reconfiguration phase [10] (or LP realization). This thesis discusses the reconfiguration phase. In choosing a suitable method for this phase, there are several issues that need to be considered. These are delved into in Section 2.2. The motivation for rerouting/reconfiguration is as explained in Section 1.2.

2.2 Issues in Connection rerouting

The ability to reconfigure the logical connectivity is a promising feature, but managing the lightwave network during the reconfigurations phase is a complex issue. The penalty paid as a result of connection rerouting, is the disruption and reestablishment of existing connections, and in certain cases some connections will have to be stalled for some time. Also in addition to this penalty, the rerouting/re-establishment of the connections is not a trivial task and demands that adequate infrastructure be in place for calculating the new paths and co-ordinating the re-establishment of connections in a manner that results in minimal disruption time. The time required for tearing down of a connection and its re-establishment contribute to the disruption time. We assume in this work that the time for tearing down a connection is constant and independent of the connection length. The connection tearing down time would be very small compared to the re-establishment time and hence may be neglected. Re-establishment of a connection would require the resources (like switches, wavelength converters etc) in the new path, be made available and then configured for the connection.

In addition to connection disruption time, the other issues involved are of resource
dependencies and synchronization. In rerouting requests, the resources needed by a
connection request $r_i$, for its new path, might be occupied by another connection $r_j$,
which in turn need to be rerouted first to its new path. Here $r_i$ can only be rerouted after
$r_j$ is rerouted, i.e., "$r_i$ depends on $r_j$". This dependence between rerouted requests can
be represented as a graph, which we refer to as the "resource dependency graph". Thus
the right order of rerouting need to be determined. Further a request $r_i$ might depend
on request $r_j$, which might depend on say $r_k$, which in turn might again depend on $r_i$.
So $r_i, r_j, r_k, r_i$ would form a dependency cycle. Such dependency cycles in a resource
dependency graph of the requests would need to be resolved in order to determine the
order of rerouting in the network.

2.3 Previous work

Our work discusses the reconfiguration phase[10] which is defined as the transition
between the current logical topology and a target graph topology. In [10], the target
connection diagram is reached by a series of steps referred to as Branch-exchange opera-
tions (BE) by them. BEs remove any two arcs, $(i, j)$ and $(k, l)$, from a graph replacing
them with new arcs $(i, l)$ and $(k, j)$. A sequence of connection diagrams is constructed
such that each successive connection diagram differs by a single BE. This scheme is not
only very computation intensive, but also has very large reconfiguration time. The same
connection may undergo multiple disruptions, before it reaches its target route. In [11],
a scheme is proposed in which the disrupted connections are never stalled, by a basic
operation of circuit migration they describe as MTV_WR (Move-to-Vacant Wavelength-Retuning) method. But this scheme is complex and involves multiple moves for the same rerouted connection. The reconfiguration method proposed in [9] is an example of serial rerouting (serial rerouting is explained in Section 2.4). In their scheme, the failed connections are added to a set $P$ that includes all the connections to be rerouted. They are rerouted one after another and in the process of rerouting a connection, all the connections on which it is dependent are disrupted and added to the set $P$. This scheme has the disadvantage of having very large disruption times for the connections. Another similar approach towards serial rerouting is given in [8], which involves the construction of an auxiliary graph (which is an undirected bipartite graph) based on the conflict relationship of the new and old topologies.

2.4 Rerouting/Reconfiguration Methods

The connection rerouting could be serial, parallel or could be a combination of the two, which we refer to as hybrid. If all the connections that need to be rerouted are disrupted simultaneously and then re-established on their new routes, we refer to the process as parallel rerouting. On the other hand if both tearing down connections and re-establishing them is done sequentially, then we refer to it as serial rerouting.

Parallel rerouting is more complex, especially in a large network, since it would involve synchronization in reconfiguring the network resources across the network like switches, wavelength converters etc. It involves a lot of simultaneous disruptions. But
parallel rerouting has the advantage that resource dependency issues involved with serial rerouting does not arise.

Serial rerouting on the other hand will be easier to implement when compared to parallel rerouting, and in such a scheme only the connections dependent on the request being rerouted need to be disrupted. Only a very small number of connections will be down at any given time, thus providing more alternate multiple hop paths on the logical topology for compensating failed connections. But the serial rerouting process takes a longer time, since the process is not parallelized even when the dependencies do not exist. The reconfiguration method proposed in [9] (described in Section 2.3 ), is an example of serial rerouting. This scheme has the disadvantage of having very large disruption times for the connections as mentioned earlier.

In this thesis work we present a scheme for obtaining the order to reroute connections. The resultant order obtained can be used for serial or hybrid rerouting. In the hybrid rerouting/reconfiguration scheme, rerouting for the connections which does not have resource dependencies can be performed in parallel and for the connections with resource dependencies, the dependencies are resolved if possible by stalling certain connections, and else they are routed serially.

2.5 Mathematical Description of the Problem

The problem we are trying to address here is to find the order in which the connection rerouting need to be performed. This would involve constructing the resource dependency
graph of the requests. Then the number of requests disrupted in order to break the dependency cycles, needs to be minimal. Also to be taken into consideration is that the disruption time of the connections should be minimal.

Let $G$ be the representation of the physical topology of the network as a graph. If $C$ is the set of $k$ connections in $G$, then $C = \{c_i \mid i \in N, 1 \leq i \leq k\}$, where $c_i$ is the $i$th connection in the network. Let $R(C) = \{r(c_i) \mid c_i \in C\}$ be the set of routes in $G$, where $r(c_i)$ is the route of connection $c_i$ on the network $G$. Let the network after reconfiguration be $G'$, with a collection $C'$ of $l$ requests, where $C' \subseteq C$ and $l \leq k$, since after the reconfiguration some requests might have been dropped. Let $R'(C') = \{r'(c_i) \mid c_i \in C'\}$ be the set of routes in $G'$, where $r'(c_i)$ is the route of connection $c_i$ on the network $G'$.

The set of dropped connections is $C_{\text{dropped}} = C - C'$, where $|C_{\text{dropped}}| = l - k$. The routes for the set of undisrupted connections is given by, $R_{\text{UC}} = R(C) \cap R'(C')$ and the routes for the set of disrupted or dropped connections in $G$ is given by $R_{\text{DC}} = R(C) - R_{\text{UC}}$, of which the disrupted connections will be rerouted to the set of routes $R'_{\text{DC}}$ given by $R'_{\text{DC}} = R'(C') - R_{\text{UC}}$. Thus the connection $c_i$ needs to be moved from route $r(c_i)$ on $G$, to route $r'(c_i)$ on $G'$ where $c_i, \exists r(c_i) \in R_{\text{DC}}, c_i \notin C_{\text{dropped}}$ ($\exists$ is short for “such that”).

If $r(c_i) = \{l_k \mid l_k \in G \text{ and } l_k \text{ is a resource that forms the route for } c_i\}$, where $l_k$ is a resource in $G$, which forms the route $r(c_i)$, then $r(c_i)$ depends on route $r(c_j)$, if $\exists l_k \in r'(c_i) \cap r(c_j)$. We denote this relationship as $c_i \rightarrow c_j$. Thus the resource dependency graph can be constructed, which will be a digraph consisting of nodes $\mathcal{n}(c_i)$ pertaining
to each connection \( c_i \ni r(c_i) \in R_{DC} \), and a link from vertex \( n(c_i) \) to \( n(c_j) \) will exist if \( c_i \rightarrow c_j \). The timing model chosen for rerouting could be a constant (i.e., a fixed time is assumed for the re-establishment of a connection), or it can be assumed that the time required for re-establishment is a function of the path length of the new route of the connection. If we assume that the re-establishment time \( w(c_i) \) is linearly dependent on the new path length of a connection \( c_i \), it can be represented as \( w(c_i) = \gamma |r'(c_i)| \), where \( \gamma \) is a constant. Corresponding to every link \( c_i \rightarrow c_j \), we can associate \( w(c_i) \) as the weight of that link.

We need to perform the following steps:

- Construct the weighted digraph \( G_N \) obtained with nodes \( n(c_i) \), corresponding to all \( c_i \ni r(c_i) \in R_{DC} \).

- This graph \( G_N \) needs to be minimally decyclized.

- An order has to be determined for connection disruption and re-establishment.

The connection rerouting could be serial, parallel or the hybrid scheme (explained in Section 2.4). In this thesis we consider the serial and hybrid schemes.
CHAPTER 3  PROCEDURE FOR REROUTING OF CONNECTIONS

3.1 Resolving Cyclic Dependencies

When a link fails in a network, the existing connections might need to be rerouted to accommodate the disrupted connections. This can be done serially, parallelly or by the hybrid scheme (explained in Section 2.4).

Consider the 8 nodes, 11 links network, with the connections a, b, c, & d as shown in the Fig. 3.1.

Let the routes for these connections be $r(a)$, $r(b)$, $r(c)$ and $r(d)$ respectively. Nodes are numbered from 1 to 8. Let the links be referred to as $(i, j)$, $(i, j)$ being the link connecting nodes $i$ and $j$. If as shown in Fig. 3.1(B), the link $(7, 8)$ fails, then note that the failed connection cannot be rerouted. Connection rerouting by network reconfiguration can salvage the failed connection. The re-routed connections $a$, $b$, $c$ and $d$ on their new routes $r'(a)$, $r'(b)$, $r(c)$ and $r'(d)$ respectively are as shown in Fig. 3.1(B). It can be seen that $r'(a) \cap r(b) \neq \emptyset$, and hence $a$ depends on $b$, represented as $a \rightarrow b$. Similarly we have, $b \rightarrow c$, $b \rightarrow d$ and $d \rightarrow b$. The last two dependencies between $b$ and $d$ form
a cycle. This dependency cycle needs to be resolved.

This cyclic dependency can be resolved if free resources (alternate paths) exist, which can be inserted temporarily into the dependency cycle as shown in the Fig. 3.2(A). If a free path/resource that can be inserted into the dependency cycle does not exist, then a connection not in the dependency graph (i.e., a connection $c_i \not\in R_{UC}$ that is not being rerouted) can be disrupted and moved to some free resource or path temporarily to make available alternate paths/resources in-order to break the dependency cycle as shown in the Fig. 3.2(B). Any of the above means for breaking the dependency cycle is possible only if there exists suitable free resources (alternate paths), unlike in the example network shown in Fig. 3.1. But this might result in increasing the disruption time of the connections, in addition to disrupting connections from $R_{UC}$, i.e., the ones that are not being rerouted.

Figure 3.1 Rerouting in the case of a link failure
Figure 3.2 Dependency graph for the rerouting for failure of link (7, 8)

Another method to break dependency cycles, is to keep one of the connections in the cycle disrupted for a period of time until the resources required for its new path is available. Say for example, \( a \rightarrow b, b \rightarrow c, c \rightarrow d \) and \( d \rightarrow a \), then connections \( a \), \( b \), \( c \) and \( d \) form a dependency cycle. If connection \( a \) is disrupted and stalled in order to break the cycle, then connection \( a \) cannot be re-established until \( d \), \( c \) and \( b \) are re-established. Thus if a connection has to be disrupted to break a dependency cycle, then it has to remain disrupted and cannot be re-established until all the other connections in the dependency cycle have been rerouted. Thus if the cycle is large or the sum of the weights \( w(c_i) \) of the connections \( c_i \) in the cycle is large, then the down time of the connection which is disrupted is going to be large.
Figure 3.3  Alternate routing for the same example.

If the connection that is being disrupted has a high priority or if the cycle servicing time is high, then the former two schemes might prove more attractive. In general however, the last scheme is more practical and easier to implement. Also large or complex cycles occur very rarely during practical connection rerouting, as will be observed from our simulation experiment results as presented in Section 4.2.

In the example network, the cyclic dependency that arose due to failure of the link (7, 8) could have been avoided if the initial routing was as shown in the Fig. 3.3(A). The rerouting in that case is shown in 3.3(B) and can be seen that it does not have any dependency cycles.

Note that the cyclic dependency shown initially would arise only in the case of failure of the link (7, 8). The resources used in the routing in Fig 3.1 is 6 links whereas in Fig 3.3, the connections require 7 links. In the case of failure of link (1, 2) or link (1, 4),
the re-routing of the connections are as shown in the Fig. 3.4(B). The rerouting in this case also gives rise to cyclic dependency. So if we assume uniform failure rates for the links, the routing in 3.1 has a lesser probability of leading to a cyclic dependency than the routing in Fig 3.4, since the former will have a cyclic dependency only if link (7, 8) fails, whereas the latter will have a cyclic dependency if either link (1, 2) or link (1, 4) fails. The cyclic dependencies cannot be avoided completely, but can only be minimized and hence resolving them becomes important.

3.2 Algorithm for Determining Order of Connection Rerouting

The resource dependency graph $G_N$ defined in Section 2.5 is to be constructed first. The resource dependency graph $G_N$ is the weighted digraph with nodes $n(c_i)$, corresponding to all $c_i, \exists r(c_i) \in R_{DC}$. A directed link from $n(c_i)$ to $n(c_j)$ exists in $G_N$,
if for connections $c_i$ and $c_j$, $c_i \rightarrow c_j$. $G_N$ might have multiple roots or might consist of multiple trees. Note that if network reconfiguration is being performed because of a link or node failure, then all the failed connections will have a corresponding root in the graph $G_N$.

To perform minimal decyclization on $G_N$, we need to determine all elementary cycles\(^1\). Johnson's algorithm\([12]\) (given in the Appendix) can be used to find all elementary cycles in a strongly connected component. A strongly connected component of a directed graph is a maximal subset of vertices containing a directed path from each vertex to all others in the subset. The vertices of any directed graph can be partitioned into a set of disjoint strongly connected components. The following transformations and operations need to be performed for minimal decyclization:

1. Form the depth-first forest (DFF) $G_F$ of $G_N$. The result may be composed of several depth-first trees. We find the finishing times $f(c_i)$ of each node/vertex $n(c_i)$. The finishing time of a vertex is the time when all its out-going edges have been traversed.

2. Form $G_R$ from $G_F$, by reversing the directions of all edges in $G_F$.

3. Form the depth-first forest of $G_R$, by visiting the nodes in order of their decreasing finishing times $f(c_i)$ which was determined in step 1.

\(^1\)A cycle is elementary if all vertices except the first are visited exactly once.
4. The resultant DFF obtained from $G_R$, will consist of depth first trees, and each tree will be a collection of nodes that forms a *strongly connected component*\(^2\) (SCC).

5. An acyclic component graph is now formed where each node represents a strongly connected component.

6. Adjacency matrix for each strongly connected component (SCC) is constructed from the link structure in graph $G_N$. On the adjacency matrix of each strongly connected component, Johnson's algorithm\(^{12}\) (given in the Appendix) is run on each SCC to obtain the list of all elementary cycles/circuits in the graph $G_N$.

- Each elementary cycle is expressed as a Boolean sum of its edges. The boolean product of expressions pertaining to all elementary cycles is obtained and represented in the sum of products (SOP) form.

- The term that consists of the smallest number of edges is called the minimum-feedback arc set.

The process of finding all strongly connected components has an order of complexity of $O(E)$ for a graph with $V$ nodes or vertices and $E$ edges. The order of complexity of Johnson's algorithm used to find all elementary cycles in a SCC is $O((n + e)(c + 1))$,\(^{2}\)

\(^2\)The *strongly connected components* of a graph are the equivalence classes/collection of vertices, where every pair of vertices are mutually reachable.
for an SCC with \( n \) nodes, \( e \) edges and \( c \) elementary cycles. Note that the maximum number of edges \( E \) in a graph is \( \Theta(V^2) \), but the maximum number of elementary cycles in a graph will be exponential with \( V \). Also the task of finding the minimum term in the SOP of the boolean product of the edges in all the cycles, is of exponential time with the number of cycles.

Minimal decyclization as mentioned above determines minimum number of edges to be removed to break all cycles. Our goal however is to remove a node, corresponding to the connection that is temporarily disconnected (not moved) to break the cycle. Therefore the term in the SOP expression we would be interested in, would be the one with minimum number of nodes, since those nodes will correspond to the connection requests that will have to be stalled to break dependency cycles. If there are multiple such terms, the term with the maximum sum of link weights is chosen so that the connections that are disrupted are the ones which have a large re-establishment time. The motivation behind this is that the stalled connection in a cycle has to remain stalled till all the connections in the cycle have been re-established, and by stalling the connection with a larger re-establishment time, the stall time for the stalled one is reduced.

The process explained above is demonstrated using an example. Consider the resource dependency digraph with four nodes and six edges shown in Fig 3.5. There is only one strongly connected component (SCC) in it, consisting of all the nodes in the graph. Using Johnson's algorithm on it will determine all elementary cycles. There are
weight $w(e_i)$ of every edge $e_i$ is 1

$(x, y)$ where $x$ is the disconnect time
and $y$ is the reconnect time

Figure 3.5 Example to illustrate minimal decyclization.

three elementary cycles in the graph. They are

1. $(e_1, e_2, e_3)$,

2. $(e_2, e_3, e_6)$ and

3. $(e_2, e_5, e_4, e_6)$.

The weights of the edges $w(e_i), 1 \leq i \leq 6,$ is all a constant, say one. Expressing each of
the cycles as a boolean sum of its edges and taking its boolean product will result in:

$$(e_1 + e_2 + e_3) \ast (e_2 + e_3 + e_6) \ast (e_2 + e_5 + e_4 + e_6)$$

The term with least number of nodes in the SOP (sum of products) form of the above
expression would be $e_2$. So $e_2$ is the edge that is to be broken for minimal decyclization.
In cases where a node is common to more than one edge of the minimum feedback arc set, it can be chosen for disconnection, thus removing two arcs with a single disruption. Else either the source or destination node of the edge from the minimum feedback arc set can be randomly chosen for disconnection. Let the set of links to be removed, for breaking the cycles be $L_{\text{decyc}}$.

After this the order of disconnections and reconnections needs to be obtained. The connections which can be rerouted first are the ones which do not have any dependencies. In the dependency graph they would correspond to the nodes that do not have any outgoing links, i.e., the nodes with out-degree zero referred to as leaves. The nodes ready for rerouting after that would be the ones with direct links to the leaves. So it can be observed that a traversal of the graph from leaves to the root would give the order of rerouting. This process can be made convenient by reversing all the link directions. It should be noted that a connection corresponding to a node cannot be reconnected unless all its children nodes have been disconnected. The disconnect time and reconnect time will be different only for the nodes corresponding to the connections stalled for decyclization.

With the above given rationale we have the following algorithm to obtain the order of rerouting:

- Reverse the direction of all the links in the graph $G_N$ to obtain $G_R$ as before.
- Mark all the edges $c_i \rightarrow c_j \in L_{\text{decyc}}$ as removed. Associate a tuple $(x, y)$, with each node where $x$ denotes the disconnect time and $y$ denotes the reconnect time
of the connection. Initialize all $x$'s and $y$'s to zero.

- Start a DF traversal from each of the nodes with in-degree zero (the \textit{removed} edges should not contribute to the degree of the nodes). During the DF traversal only links that are not \textit{removed} is used.

- During each traversal the tuple $(x, y)$ corresponding to each node should be updated as below:
  
  - A node should have its disconnect time greater than each of its parent nodes, by at least the weight of the non-disrupted link connected between them.
  
  - The reconnect times of each node encountered during the traversal should be greater than its parent nodes, by at least the weight of the disrupted link connected between them.

- The updated values of disconnect and reconnect times obtained after all DF traversals from nodes with in-degree zero is completed, gives the order for disconnection and re-establishment.

In general, the time required for disconnection is negligible compared to the re-establishment time which is either a constant or proportional to the path length, depending on the timing model considered.

Consider the dependency graph in Fig. 3.6(A), which has 5 nodes (each corresponding to a rerouted connection on the network topology graph) and 10 links (dependencies).
Figure 3.6 Example to illustrate determination of rerouting or decyclization edge.
Let $[i, j]$ denote the link from node $i$ to node $j$. Its a graph with a single SCC. All elementary cycles in the graph are: $(0, 3, 2)$, $(0, 3, 2, 1)$, $(0, 4, 1)$, $(0, 4, 1, 3, 2)$, $(0, 4, 2)$, $(0, 4, 2, 1)$, $(0, 4, 3, 2)$, $(0, 4, 3, 2, 1)$ and $(1, 3, 2)$. The minimal decyclization set consists of the links $[0, 4]$, $[3, 2]$ which are shown as dotted lines in Fig. 3.6(B). The link directions are reversed. DF traversals from all nodes with in-degree zero (node 3) are performed. The steps involved as described above are shown in detail in Fig. 3.6(C)-(J).

The time required for network reconfiguration is 5 time units for the hybrid rerouting scheme, and the sum total of downtime of all connections is 11 time units which will also be the total rerouting time for a serial rerouting scheme.

For the example in Fig. 3.5, edge $e_2$ is removed and a depth first traversal on the graph with the link directions reversed, as described in the algorithm above gives the order in which the requests should disconnect and reconnect. For this example, the tuples corresponding to each node is as shown in Fig. 3.5. The time required for network reconfiguration is 5 time units for the hybrid rerouting scheme, and the sum total of downtime of all connections is 8 time units. For the serial rerouting scheme, the order of reconnections can be obtained from the graph as for the hybrid. But if multiple nodes have the same reconnect times then they will have to be done sequentially. Disconnections are done as and when there is a dependency. For parallel rerouting, no order is involved as all rerouted connections are disrupted and reconnected simultaneously.
CHAPTER 4 RESULTS AND CONCLUSION

4.1 Penalty

The different schemes for rerouting can be compared by evaluating the penalty of each scheme. Different measures for penalty can be used. Some of them are as follows:

- The penalty incurred as a result of connection rerouting or network reconfiguration can be measured as a function of the time required to make the network fully functional again. We define TFR (Time for rerouting) as the time needed for all connections to be restored, after a failure had occurred or in general the reconfiguration process was initiated.

- Another measure would be the number of connections stalled during decyclization, i.e., $|L_{\text{decyc}}|$.

- Probably the most appropriate metric would be to measure penalty as a function of the priority of a connection and the downtime in number of connection-time units (i.e., the sum total of downtime of all connections if all have the same priority). We call this the total disruption time (TDT). Thus if for every disrupted connection
c_i, \in R_{DC}, we can associate a corresponding pair (d_i, r_i), where d_i is the disconnect time and r_i is the reconnect time and all connections have the same priority, then the penalty would be given by:

\[ TDT = \sum_i (r_i + w(c_i) - d_i) \forall i \in N \ni c_i \in R_{DC}, c_i \notin C_{dropped} \]  \hspace{1cm} (4.1)

\[ = \sum_{i \ni c_i \in L_{deacy}} (r_i - d_i) + \sum_{i \ni c_i \in R_{DC}, c_i \notin C_{dropped}} w(c_i) \]  \hspace{1cm} (4.2)

Expression (4.2) can be obtained from expression (4.1) since we have r_i = d_i for c_i \in R_{DC}, c_i \notin L_{deacy}.

- Dividing this by the number of disrupted connections, i.e., the number of connections c_i such that c_i \in R_{DC}, c_i \notin C_{dropped}, provides the mean disruption time per disrupted connection (MDT).

The TFR for serial rerouting would be given by:

\[ TFR_{serial} = \sum_{i \ni c_i \in R_{DC}, c_i \notin C_{dropped}} w(c_i) \]  \hspace{1cm} (4.3)

On the other hand TFR for hybrid rerouting would be determined by the dependencies among the rerouted connections. Lower dependencies would imply that more connections can be reouted parallelly and hence a lower TFR. But the upper bound for TFR_{hybrid} would be given by TFR_{serial} as in equation (4.3), which would occur when the dependencies are very high and rerouting connections in parallel is not possible. The
lower bound of $TFR_{serial}$ would be $TFR_{parallel}$, when there are no dependencies at all. Parallel rerouting would take a fixed time which would be much less than serial or even hybrid.

Using the number of connections stalled during decyclization as a metric would help in comparing schemes only if cycles exist, since otherwise $|L_{decyc}|$ would be zero. Also this metric would yield the same value in the case of hybrid or serial rerouting when our scheme is used. The existence of cycles is not an issue in parallel rerouting and hence this metric is not used for it.

Note that for the rerouting order we have presented, both hybrid and serial rerouting will yield the same total disruption time (TDT) and mean disruption time per disrupted connection (MDT). It is given by equations (4.1) and (4.2). For parallel rerouting $TDT$ would be the same as for the other schemes in the absence of cycles and would be proportional to the average number of connections rerouted. $MDT$ would be proportional to the mean path length (MPL) of the new paths of the rerouted connections if the connection rerouting time for a single rerouted connection is proportional to its new path length.

4.2 Results

For the penalty incurred in terms of rerouting time and operational complexity due to network reconfiguration, the advantage achieved in general will depend on the traffic in the network and the application for which network reconfiguration is used. The
methodology for connection rerouting as mentioned in this paper was simulated and studied for the case of \textit{L+1 fault recovery} \cite{16} in optical networks.

\textit{L+1} fault recovery is a path-based fault tolerance strategy. The \textit{L+1} fault recovery scheme is as follows: for a network with \( L \) links, \( L \) subgraphs with one link missing from the original network configuration are considered. A connection request is serviced only when it can be routed on all of the \( L \) subgraphs. In the event of a link failure, the connections are rerouted to the paths used by them on the corresponding subgraph with the failed link missing.

The simulation and study was carried out on the following networks: NSFNET (14 nodes, 23 links), NJLATA (11 nodes, 22 links), MESH-3x3 (9 nodes, 18 links), MESH-4x4 (16 nodes, 32 links) all with 16 wavelengths and no wavelength conversion. The rerouted path lengths were taken as the edge weights in the resource dependency graph, \textit{i.e.}, we assumed all connections and requests of the network to have the same priority. The size of the resource dependency graph and the penalty incurred for the NSFNET over 500,000 requests with an arrival rate of 300 requests/unit time and mean call duration of 1 unit time is as shown below:

\begin{verbatim}
Arrival rate : 300.00
nodes = 14
links = 22
Requests = 500000

Fault occurance: RANDOM
Faulty component: LINK

MTNF (Mean Time to Next Failure) = 0.010000
MRRT (Mean Repair Rate) = 10.000000

Topology : nsfnet-16
\end{verbatim}
<table>
<thead>
<tr>
<th>Failed Link</th>
<th>TotalReq</th>
<th>ReroutedReq</th>
<th>DependentReq</th>
<th>MeanPathLength</th>
<th>TimeForRerouting</th>
<th>TotalDisruptionTime</th>
<th>MeanDisruptionTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL</td>
<td>TR</td>
<td>RQ</td>
<td>DR</td>
<td>MPL</td>
<td>TFR</td>
<td>TDT</td>
<td>MDT</td>
</tr>
<tr>
<td>[20]</td>
<td>157</td>
<td>154</td>
<td>92</td>
<td>5</td>
<td>1.83766</td>
<td>24</td>
<td>303</td>
</tr>
<tr>
<td>[13]</td>
<td>154</td>
<td>149</td>
<td>107</td>
<td>1</td>
<td>1.97987</td>
<td>34</td>
<td>302</td>
</tr>
<tr>
<td>[13]</td>
<td>153</td>
<td>151</td>
<td>101</td>
<td>1</td>
<td>2.05298</td>
<td>34</td>
<td>314</td>
</tr>
<tr>
<td>[1]</td>
<td>172</td>
<td>170</td>
<td>111</td>
<td>3</td>
<td>1.74706</td>
<td>16</td>
<td>320</td>
</tr>
<tr>
<td>[18]</td>
<td>164</td>
<td>163</td>
<td>101</td>
<td>3</td>
<td>1.93252</td>
<td>20</td>
<td>332</td>
</tr>
<tr>
<td>[8]</td>
<td>169</td>
<td>162</td>
<td>115</td>
<td>6</td>
<td>1.92593</td>
<td>28</td>
<td>364</td>
</tr>
<tr>
<td>[3]</td>
<td>147</td>
<td>144</td>
<td>85</td>
<td>2</td>
<td>1.90972</td>
<td>29</td>
<td>291</td>
</tr>
<tr>
<td>[18]</td>
<td>151</td>
<td>148</td>
<td>100</td>
<td>1</td>
<td>1.98649</td>
<td>61</td>
<td>300</td>
</tr>
<tr>
<td>[18]</td>
<td>153</td>
<td>151</td>
<td>99</td>
<td>4</td>
<td>1.88079</td>
<td>25</td>
<td>306</td>
</tr>
<tr>
<td>[10]</td>
<td>164</td>
<td>161</td>
<td>102</td>
<td>0</td>
<td>1.83230</td>
<td>23</td>
<td>295</td>
</tr>
<tr>
<td>[17]</td>
<td>165</td>
<td>163</td>
<td>100</td>
<td>3</td>
<td>1.76074</td>
<td>17</td>
<td>299</td>
</tr>
<tr>
<td>[19]</td>
<td>152</td>
<td>151</td>
<td>105</td>
<td>2</td>
<td>1.88079</td>
<td>35</td>
<td>292</td>
</tr>
<tr>
<td>[18]</td>
<td>166</td>
<td>164</td>
<td>116</td>
<td>5</td>
<td>1.88415</td>
<td>34</td>
<td>341</td>
</tr>
<tr>
<td>[13]</td>
<td>146</td>
<td>142</td>
<td>106</td>
<td>3</td>
<td>2.06338</td>
<td>46</td>
<td>351</td>
</tr>
<tr>
<td>[20]</td>
<td>173</td>
<td>169</td>
<td>111</td>
<td>4</td>
<td>1.79882</td>
<td>24</td>
<td>342</td>
</tr>
<tr>
<td>[6]</td>
<td>156</td>
<td>152</td>
<td>111</td>
<td>4</td>
<td>1.89474</td>
<td>31</td>
<td>319</td>
</tr>
<tr>
<td>[4]</td>
<td>167</td>
<td>165</td>
<td>107</td>
<td>5</td>
<td>1.89091</td>
<td>33</td>
<td>362</td>
</tr>
<tr>
<td>[19]</td>
<td>156</td>
<td>153</td>
<td>96</td>
<td>2</td>
<td>1.88235</td>
<td>34</td>
<td>297</td>
</tr>
<tr>
<td>[17]</td>
<td>154</td>
<td>152</td>
<td>99</td>
<td>5</td>
<td>1.87500</td>
<td>18</td>
<td>313</td>
</tr>
</tbody>
</table>

The TFR given is for the hybrid rerouting scheme and we denote the TFR for serial rerouting as $TFR_{serial}$. Then $TFR_{serial}$ can be computed from the above table by $TFR_{serial} = RQ \times MPL$, since $TFR_{serial}$ is the sum of the weights of the links in the dependency graph if we assume that the time for rerouting a connection is proportional to its path length after rerouting. It can be observed that MDT given (the same for serial and hybrid schemes) equals the MPL when NDE (number of decyclization edges) equals zero. $MDT_{parallel}$ would be equal to the MPL given. The results above can be verified to satisfy the relation $TDT = RQ \times MDT$ for the hybrid and serial scheme and
Table 4.1 Results for NSFNET topology

<table>
<thead>
<tr>
<th>Req rate</th>
<th>TR</th>
<th>RQ</th>
<th>DR</th>
<th>NDE</th>
<th>MPL</th>
<th>TFR</th>
<th>TDT</th>
<th>MDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>51.8831</td>
<td>49.8831</td>
<td>14.9351</td>
<td>0.4026</td>
<td>2.1650</td>
<td>10.8182</td>
<td>110.3247</td>
<td>2.2111</td>
</tr>
<tr>
<td>100</td>
<td>92.4808</td>
<td>90.0962</td>
<td>48.6731</td>
<td>1.7115</td>
<td>2.1616</td>
<td>18.8077</td>
<td>204.8269</td>
<td>2.2744</td>
</tr>
<tr>
<td>200</td>
<td>134.1538</td>
<td>131.5769</td>
<td>84.2692</td>
<td>2.4231</td>
<td>2.0053</td>
<td>24.1154</td>
<td>281.4615</td>
<td>2.1422</td>
</tr>
<tr>
<td>300</td>
<td>159.1429</td>
<td>156.1905</td>
<td>104.1429</td>
<td>3.1429</td>
<td>1.8916</td>
<td>29.6190</td>
<td>317.7619</td>
<td>2.0365</td>
</tr>
<tr>
<td>400</td>
<td>176.1875</td>
<td>173.0000</td>
<td>117.3750</td>
<td>3.5000</td>
<td>1.8441</td>
<td>28.0000</td>
<td>340.8125</td>
<td>1.9720</td>
</tr>
<tr>
<td>500</td>
<td>188.2222</td>
<td>184.7222</td>
<td>116.6111</td>
<td>3.8333</td>
<td>1.7272</td>
<td>24.3889</td>
<td>343.3333</td>
<td>1.8620</td>
</tr>
<tr>
<td>600</td>
<td>202.1818</td>
<td>198.3636</td>
<td>128.0909</td>
<td>5.3636</td>
<td>1.6729</td>
<td>24.4545</td>
<td>361.0000</td>
<td>1.8201</td>
</tr>
</tbody>
</table>

$TDT_{parallel} = RQ \times MPL.$

The results of the simulation for the different network topologies are shown in Table 4.1, Table 4.2, Table 4.3 and Table 4.4. The abbreviations used in the tables are as follows:

- TR (Total number of Requests),
- RQ (No. of rerouted requests),
- DR (No. of dependencies between connections during rerouting),
- NDE (No. of connections stalled for breaking resource dependency cycles),
- MPL (Mean path length of the connections),
- TFR (Time for rerouting as explained earlier),
- TDT (total disruption time) and
- MDT (Mean disruption time).
Table 4.2  Results for NJLATA topology

<table>
<thead>
<tr>
<th>Req rate</th>
<th>TR</th>
<th>RQ</th>
<th>DR</th>
<th>NDE</th>
<th>MPL</th>
<th>TFR</th>
<th>TDT</th>
<th>MDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>51.4865</td>
<td>49.3649</td>
<td>12.0811</td>
<td>0.2973</td>
<td>1.7864</td>
<td>8.5541</td>
<td>89.5405</td>
<td>1.8113</td>
</tr>
<tr>
<td>100</td>
<td>98.1667</td>
<td>95.7333</td>
<td>40.1833</td>
<td>1.0500</td>
<td>1.7414</td>
<td>13.2833</td>
<td>171.9000</td>
<td>1.7965</td>
</tr>
<tr>
<td>200</td>
<td>150.9091</td>
<td>147.9545</td>
<td>79.1364</td>
<td>2.5909</td>
<td>1.6110</td>
<td>17.6364</td>
<td>252.7273</td>
<td>1.7075</td>
</tr>
<tr>
<td>300</td>
<td>182.0769</td>
<td>178.4615</td>
<td>102.3846</td>
<td>3.7692</td>
<td>1.5536</td>
<td>20.3846</td>
<td>294.5385</td>
<td>1.6505</td>
</tr>
<tr>
<td>400</td>
<td>208.8333</td>
<td>204.8333</td>
<td>119.4167</td>
<td>3.9167</td>
<td>1.4642</td>
<td>19.8333</td>
<td>316.2500</td>
<td>1.5445</td>
</tr>
<tr>
<td>500</td>
<td>234.4286</td>
<td>229.5714</td>
<td>135.7143</td>
<td>5.0000</td>
<td>1.4079</td>
<td>19.1429</td>
<td>349.2143</td>
<td>1.5222</td>
</tr>
<tr>
<td>600</td>
<td>247.5000</td>
<td>243.0000</td>
<td>146.0000</td>
<td>3.3333</td>
<td>1.3896</td>
<td>25.3333</td>
<td>354.1667</td>
<td>1.4582</td>
</tr>
</tbody>
</table>

Table 4.3  Results for MESH 3x3 topology

<table>
<thead>
<tr>
<th>Req rate</th>
<th>TR</th>
<th>RQ</th>
<th>DR</th>
<th>NDE</th>
<th>MPL</th>
<th>TFR</th>
<th>TDT</th>
<th>MDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>50.3854</td>
<td>48.1875</td>
<td>9.1354</td>
<td>0.1458</td>
<td>1.5394</td>
<td>5.6042</td>
<td>74.6979</td>
<td>1.5494</td>
</tr>
<tr>
<td>100</td>
<td>100.75</td>
<td>97.6042</td>
<td>36.5</td>
<td>0.8333</td>
<td>1.5326</td>
<td>9.9375</td>
<td>152.6041</td>
<td>1.5643</td>
</tr>
<tr>
<td>200</td>
<td>170</td>
<td>166.5</td>
<td>98.6818</td>
<td>1.6818</td>
<td>1.4926</td>
<td>16.2727</td>
<td>255.2727</td>
<td>1.5328</td>
</tr>
<tr>
<td>300</td>
<td>200.1666</td>
<td>196.8333</td>
<td>130.6111</td>
<td>3.5</td>
<td>1.4644</td>
<td>20.4444</td>
<td>305.2222</td>
<td>1.5508</td>
</tr>
<tr>
<td>400</td>
<td>216.3888</td>
<td>212.5555</td>
<td>146.5</td>
<td>3.6111</td>
<td>1.4217</td>
<td>20.7222</td>
<td>316.3333</td>
<td>1.4880</td>
</tr>
<tr>
<td>500</td>
<td>226.9</td>
<td>222.1</td>
<td>153.5</td>
<td>3</td>
<td>1.3921</td>
<td>22.2</td>
<td>326.6</td>
<td>1.4705</td>
</tr>
<tr>
<td>600</td>
<td>230.4285</td>
<td>224.7142</td>
<td>153.7143</td>
<td>4.1428</td>
<td>1.4016</td>
<td>24.4285</td>
<td>329.7143</td>
<td>1.4674</td>
</tr>
</tbody>
</table>

Table 4.4  Results for MESH 4x4 topology

<table>
<thead>
<tr>
<th>Req rate</th>
<th>TR</th>
<th>RQ</th>
<th>DR</th>
<th>NDE</th>
<th>MPL</th>
<th>TFR</th>
<th>TDT</th>
<th>MDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>50.2235</td>
<td>48.5647</td>
<td>9.8235</td>
<td>0.0941</td>
<td>2.1448</td>
<td>9.3882</td>
<td>104.6471</td>
<td>2.1525</td>
</tr>
<tr>
<td>100</td>
<td>101.8864</td>
<td>100.2955</td>
<td>43.0682</td>
<td>0.4091</td>
<td>2.1529</td>
<td>16.0909</td>
<td>218.5909</td>
<td>2.1786</td>
</tr>
<tr>
<td>200</td>
<td>174.8500</td>
<td>172.7500</td>
<td>111.1500</td>
<td>2.0000</td>
<td>2.0219</td>
<td>26.4500</td>
<td>363.3000</td>
<td>2.1044</td>
</tr>
<tr>
<td>300</td>
<td>221.7500</td>
<td>219.3333</td>
<td>145.9167</td>
<td>1.7500</td>
<td>1.8555</td>
<td>27.8333</td>
<td>418.0000</td>
<td>1.9067</td>
</tr>
<tr>
<td>400</td>
<td>257.7273</td>
<td>255.5455</td>
<td>177.3636</td>
<td>3.8182</td>
<td>1.7501</td>
<td>27.6364</td>
<td>470.9091</td>
<td>1.8438</td>
</tr>
<tr>
<td>500</td>
<td>276.4286</td>
<td>273.7143</td>
<td>187.7143</td>
<td>2.8571</td>
<td>1.6786</td>
<td>29.7143</td>
<td>472.8571</td>
<td>1.7284</td>
</tr>
<tr>
<td>600</td>
<td>290.6667</td>
<td>287.6667</td>
<td>185.0000</td>
<td>2.6667</td>
<td>1.6248</td>
<td>20.3333</td>
<td>480.3333</td>
<td>1.6698</td>
</tr>
</tbody>
</table>
These tables show the various penalty metrics defined above for varying request arrival rates. The average number of dependencies (DR) is plotted in Fig. 4.1 for varying request rates. In general the dependencies increase with the connections present in the network. We can expect the dependency cycles and hence the connections stalled for decyclization, to increase with the number of dependencies. This trend can be observed from the columns corresponding to DR (dependencies during rerouting) and NDE (number of decyclization edges). The stalled connections (NDE) can be seen to be increasing with the request rate, indicating that the number and complexity of dependency cycles increase with the number of connections present. Fig. 4.1 shows that the number of dependencies are higher for the mesh networks which can attributed to
their high connectivity; but the cycles formed can be decyclized with fewer stalls for the mesh networks, than for NJLATA or NSF as can be seen from the tables.

As expected, the results show that with the number of links and nodes, the mean disruption time (MDT) increases. This is because MDT is dependent on the mean path length of the requests since the model we adopted assumes that the time for rerouting a connection is proportional to the path of the new length being established. The MDT for Mesh-4x4 network is lower than the MDT for NSFNET although the former has more number of links since the mean path length for it is less than that of NSFNET as can be seen in Fig 4.2. For the networks with larger diameter, we can expect greater mean path length for uniformly distributed source-destination traffic. The rerouted
path lengths were used as edge weights in the resource dependency graph, i.e., the time taken for connection re-establishment was taken to be proportional to the established path length. This explains the greater average reconfiguration penalty for networks with greater network diameter. Also since the mean path length is decreasing with increasing arrival rates, so is the variation in network reconfiguration penalty in terms of MDT as shown in Fig. 4.3.

4.3 Conclusion and Future Work

We have presented in this thesis a methodology and the algorithms involved for network reconfiguration or connection rerouting in order to minimize the number of
connections stalled and the mean disruption time of the connections. The scheme we have presented is a combination of serial and parallel reconfiguration. But the same scheme can be used to even find the order for serial reconfiguration. Note that the number of dependency cycles and the connections stalled for resolving them (NDE), decreases after peaking at an intermediate arrival rate. Another observation that can be made is that the number of decyclization edges and the number of cycles is very low. Since the time complexity of Johnson’s algorithm is $O((n+e)(c+1))$ as mentioned earlier, the low number of cycles present implies that the time for the computation required for connection rerouting would be less and makes this methodology practically viable.
Johnson's Algorithm

Johnson's algorithm[12] is used to find all elementary cycles or circuits in a graph. A directed graph $G = (V, E)$ consists of a nonempty and finite set of vertices $V$ and a set $E$ of ordered pairs of distinct vertices called edges. $G$ has $n$ vertices and $e$ edges. Johnson's algorithm accepts a graph $G$ represented by an adjacency structure $A_g$ composed of an adjacency list $A_g(v)$ for each $v \in V$. The list $A_g(v)$ contains $u$ if and only if edge $(v, u) \in E$. The algorithm assumes that vertices are represented by integers from 1 to $n$.

The circuit or elementary-path finding algorithm is as follows:

begin
integer_list_array $A_k(n), B(n)$;
logical_array blocked(n);
integer $s$;

logical procedure CIRCUIT (integer $v$)
begin
    logical $f$;
    procedure UNBLOCK (integer $u$)
    begin
        blocked(u) <= false;
        for (w in B(u)) do
            delete w from B(u);
            if blocked(w) then
                UNBLOCK(w);
        end if;
    end for;
    end UNBLOCK;
    $f$ <= false;
    stack $v$;
    blocked(v) <= true;
end CIRCUIT;
for ( w in Ak(v) ) do
  if w=s then
    output circuit composed of stack followed by s;
    f <= true;
  else if not blocked(w) then
    if CIRCUIT(w) then
      f <= true;
    end if;
  end if;
end for;
if f then
  UNBLOCK(v);
else
  for ( w in Ak(v) ) do
    if v not in B(w) then
      put v on B(w);
    end if;
  end for;
end if;
unstack v;
CIRCUIT <= f;
end CIRCUIT;

empty stack;
s <= 1;

while s < n do
  Ak <= adjacency structure of strong component K with least vertex in
  subgraph of G induced by {s, s+1, .... ,n};
  if Ak not empty then
    s <= least vertex in Vk;
    for ( i in Vk ) do
      blocked(i) <= false;
      B(i) <= NULL;
    end for;
    dummy <= CIRCUIT(s);
    s <= s + 1;
  else
    S <= n;
  end if;
end while;
end;
BIBLIOGRAPHY


ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Arun K. Somani for his constant guidance and support throughout this research and the writing of this thesis. I am indebted to him for his timely suggestions and the motivating and enlightening discussions we had. I owe much to the present and past members of the DCNL Group for the helpful discussions I had with them.

I wish to express my deepest sense of gratitude to my family members, who have been a constant source of encouragement and support when I needed them the most, and whose blessings and prayers made me achieve whatever I have. I would also like to extend my gratitude to my friends Mr. Kumar L. Parthasarathy, Mr. L. N. Subramanian, Mr. N. Sriram, Mr. Sampath Dechu, Ms. Shantha E. Daniel, Mr. V. Bharath, Mr. V. Nataraj, Mr. V. Rajesh and Ms. Vineela Manne, here at Iowa State University for the help they extended, whenever the need arose.

Finally my thanks goes to all who have contributed directly or indirectly in successful completion of this work.