Link failure protection and restoration in WDM optical networks

by

Fangcheng Tang

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Lu Ruan, Major Professor
Johnny S. Wong
Ahmed E. Kamal

Iowa State University

Ames, Iowa

2005

Graduate College
Iowa State University

This is to certify that the Master's thesis of

Fangcheng Tang

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In a wavelength-division-multiplexing (WDM) optical network, the failure of fiber links may cause the failure of multiple optical channels, thereby leading to large data loss. Therefore the survivable WDM optical networks where the affected traffic under link failure can be restored, have been a matter of much concern. On the other hand, network operators want options that are more than just survivable, but more flexible and more efficient in the use of capacity. In this thesis, we propose our cost-effective approaches to survive link failures in WDM optical networks.

Dynamic establishment of restorable connections in WDM networks is an important problem that has received much study. Existing algorithms use either path-based method or link-based method to protect a dynamic connection; the former suffers slow restoration speed while the latter requires complicated online backup path computation. We propose a new dynamic restorable connection establishment algorithm using p-cycle protection. For a given connection request, our algorithm first computes a working path and then computes a set of p-cycles to protect the links on the working path so that the connection can survive any single link failure. The key advantage of the proposed algorithm over the link-based method is that it enables faster failure restoration while requires much simpler online computation for connection establishment.

Tree-based schemes offer several advantages such as scalability, failure impact restriction and distributed processing. We present a new tree-based link protection scheme to improve the hierarchical protection tree (p-tree) scheme [31] for single link failure in mesh networks, which achieves 100% restorability in an arbitrary 2-connected network. To minimize the total spare capacity for single link failure protection, an integer linear programming (ILP) formulation is provided. We also develop a fast double-link failure restoration scheme by message signaling to

take advantage of the scalable and distributed processing capability of tree structure.

CHAPTER 1. Introduction

1.1 The Architecture of WDM Optical Network

Wavelength-division multiplexing (WDM) [1] optical networks with ultra-high capacity are believed to be the backbone transport networks for the next generation Internet. Wavelength-division multiplexing divides the tremendous bandwidth of a fiber into many non-overlapping wavelengths (WDM *channels*). Each channel can be operated asynchronously and in parallel at the speed of a few Gbps. The architecture of WDM network consists of wavelength cross-connects(OXCs) interconnected by fiber links. Wavelength cross-connects are classified into wavelength-selective cross-connects and wavelength-interchanging cross-connects. An optical signal passing through a wavelength-selective cross-connect is routed from an input fiber to an output fiber without wavelength conversion. An optical signal passing through a wavelength-interchanging cross-connect may be converted from the input wavelength to a different output wavelength. A *lightpath* is an all-optical connection which may span multiple fiber links to provide a circuit-switched interconnection between two nodes. In WDM networks, a connection request is satisfied by establishing a *lightpath* between source node and the destination node of the connection. Without wavelength converters, a lightpath would occupy the same wavelength on all fiber links it traverses; This property is called *wavelength continuity constraint.* Clearly, networks with wavelength converters have a lower blocking probability compared to those without because they only require some wavelength(which can be different) to be free on each link of the path, whereas networks without converters require the same wavelength to be free on all the links of a path in order to satisfy a request. In this work, wavelength continuity constraint is not considered, so that a lightpath may use different wavelengths on its multiple fiber links.

1.2 Survivable WDM Optical Networks

In an optical network, the failure of a network component such as a fiber link can lead to the failure of all the lightpaths traversing it. Since lightpath carries tremendous traffic, such a failure will lead to severe disruption to networks. If the traffic can be restored under the failure of a network component, the network is called *survivable.* The approaches to the survivable WDM optical networks have been discussed in [21]-[24]. In general, there exist two basic approaches to survive the link failure in optical networks: *protection* and *restoration.* The protection scheme finds a *backup path* for each *working path* and reserves spare capacity on the *backup path* before any failure occurs. Under normal condition, traffic is carried on the working path; when a link failure occurs on the working path, traffic is switched to the backup path. The restoration scheme finds a backup path dynamically and reroutes the affected traffic to the backup path after failure occurs. Therefore, the restoration scheme is more efficient in utilizing capacity while it is slower than protection scheme since the restoration scheme dynamically discovers a backup path then reroute the traffic to that path.

(a) Path-based **(b) Link-based**

Figure 1.1 Protection/Restoration Schemes

Lightpath protection/restoration schemes can be classified into two categories: *path-based* and *link-based.* In path-based protection/restoration, the source and destination nodes of the connections traversing the failed link reroute the traffic to their backup paths on an end-to-end basis. In link-based protection/restoration, all the connections that traverse the failed link are rerouted around the link, while the source and destination nodes of the connections traversing

the failed link are oblivious of the link failure. Fig 1.1 illustrates path-based and link-based schemes, where the working path is $(1, 2, 3, 6)$. If there is a failure on the link $(2, 3)$, the traffic on the working path will be rerouted to an end-to-end backup path $(1, 4, 5, 6)$ in the path-based scheme(Fig 1.1(a)), while in the link-based scheme(Fig 1.1(b)), the traffic on the failed link $(2, 3)$ is rerouted to $(2, 4, 5, 3)$, therefore the backup path $(1, 2, 4, 5, 3, 6)$ is used.

Wavelength channels on the backup path can be either *dedicated* or *shared.* In the dedicated method, the backup wavelength reserved on the links of the backup path are not shared with other backup paths. In contrast, in the shared method, the backup wavelength reserved on the links of the backup path may be shared with other backup paths. As a result, backup channels are multiplexed among different failure scenarios (which are not expected to occur simultaneously), and therefore shared method is more capacity efficient compared with dedicated method.

1.3 Issues and Motivation

1.3.1 Dynamic Establishment of Restorable Connection

Although efficient in spare capacity utilization, path-based method suffers slow restoration due to two reasons. First, after the upstream node adjacent to a failed link detects the failure, it needs to send a failure notification signal all the way back to the source node, which is responsible for switching the traffic to the backup lightpath. Second, due to backup sharing, backup lightpath can't be setup at the connection establishment time. Thus, once the source node is notified of a failure, it must use a signaling protocol to setup the backup lightpath before it can switch the traffic over to the backup lightpath. The first problem of path-based method can be overcome by using link-based method, where failure recovery is done by the upstream node of the failed link using the backup lightpath associated with that link. This leads to faster restoration than path-based method since the need to notify the source node about the failure is eliminated. However, the second problem of path-based method also exists in link-based method. In addition, link-based method is complicated to implement: for each link on the working path we need to compute a backup path and reserve spare capacity on that path. To achieve efficient capacity utilization, backup paths must explore both inter-connection sharing

and intra-connection sharing to reuse the backup capacity reserved for the existing connections as much as possible. (Inter-connection sharing states that as long as two working paths are link-disjoint, their backup paths can share backup capacity; Intra-connection sharing states that the backup paths for different links on the same working path can share backup capacity.) This requires the network nodes to maintain either complete or aggregated information about the existing connections and leads to complicated backup path computation algorithm [10][11].

The trend in backbone transport networks is to dynamically provision connections, i.e., lightpaths are established on demand as connection requests arrive at the network. Both path-based and link-based lightpath protection methods have been studied in dynamic establishment of restorable connections. (A connection is restorable if it can survive a single link failure.) With path-based method, given a connection request, the problem is to compute a working lightpath and a backup lightpath so that the total wavelength consumption of the connection is minimized by exploiting backup capacity sharing. This problem has received extensive study [2][3][4][5][6][7][8][9]. With link-based method, the problem is the same as the path-based case except that a set of backup lightpaths needs to be computed for a given connection request and the problem has been studied in [10][11].

The existing algorithms use either path-based method or link-based method to protect a dynamic connection, the former suffers slow restoration speed while the latter requires complicated online backup path computation. Therefore we propose a new algorithm for dynamic restorable connection establishment using *p-Cycle* protection. The *p-Cycle* concept was introduced by Grover [12]. p-Cycle can be characterized as pre-configured protection cycles in a mesh network. With the hybrid cycle/mesh approach, p-Cycle achieves the benefits of both alternatives: the efficiency of mesh restoration and the restoration speed of ring networks [12][13]. The key advantage of our proposed algorithm over the link-based method is that it enables faster failure restoration while requires much simpler online computation for connection establishment.

1.3.2 Tree-based Protection and Restoration Scheme

Tree-based protection schemes have been studied in recent years [26]-[31]. Tree-based protection schemes have several advantages. Network trees can be constructed or updated by distributed algorithms. The nodes do not need to know the complete network topology. Only the local information about their neighbors are sufficient to construct the network trees. When each node in the network has determined its parent node in the network, the network tree is built automatically. In addition, with the property of fast updating, network trees provide the potential to handle multiple link failures.

Single-link failures are common failure scenarios. While the recovery from the failure of link might take a few hours or a few days, it is certainly conceivable that a second link fails in this duration, thus causing two links to be down at the same time. Double-link failure recovery have received much attention recently. A framework for multiple-link failure recovery is proposed in [37] where three different options are given as follows.

1. Pre-computed recovery (before first failure)

   In the case of pre-computed recovery, the alternative paths of secondary failures are calculated already before a primary failure occurs.

2. Re-computed recovery (after first failure)

   In this case the secondary alternative paths for working and backup paths affected by a primary failure are calculated after the primary failure.

3. Re-restoration (after secondary failures)

   Corresponding to single failure restoration, the alternative route computation for secondary failures could be done on-demand after the occurrence of a secondary failure. Notice that the concept of *first-failure protection and second-failure restoration*(1FP-2FR) [32] has been proposed where the primary recovery mechanism is a protection scheme. The idea of 1FP-2FR is the following: The backup path for each network link is determined before the first link failure. When a first link failure occurs, the pre-determined backup path for the failed link will be used for traffic restoration. When a second failure occurs, a backup path for that

failed link is computed dynamically, then the affected traffic is rerouted to that path.

Pre-computed recovery schemes have been studied in [33], [34], [35], [36] and [38]. [35],[36] and [38] propose p-Cycle design model to achieve dual-failure restorability, [34] gives a capacity optimized formulation using backup multiplexing for double-link failures and [33] proposes heuristic algorithm that pre-computes backup paths for links. Re-computed recovery are also studied in [36], where p-Cycles and spare capacity can be reconfigured dynamically after the first failure, creating a new set of p-Cycles that are optimized to withstand possible second failures. However, few researches focus on the re-restoration schemes.

Since network trees provide the capability to handle multiple link failures, we propose a protection tree scheme for first-failure protection and second-failure restoration.

1.4 Outline of The Thesis

The rest of the thesis is organized as follows. In chapter 2, we propose a new dynamic restorable connection establishment algorithm using p-cycle protection. For a given connection request, our algorithm first computes a working path and then computes a set of p-cycles to protect the links on the working path so that the connection can survive any single link failure. The key advantage of the proposed algorithm over the link-based method is that it enables faster failure restoration while requires much simpler online computation for connection establishment. The algorithm consists of three components: offline computation of primary p-cycles, online computation of the working path, and online computation of p-cycles for working path protection. We propose two algorithms for the first component called SLA and Grow and two algorithms for the second component called PS and PNS. We will show the performance of the different combination of our proposed algorithms in term of the total capacity required.

In chapter 3, we present a new tree-based link protection scheme for single link failure in mesh networks, which achieves 100% restorability in an arbitrary 2-connected network. To minimize the total spare capacity for single link failure protection, an integer linear programming(ILP) formulation is provided. We also propose a fast double-link failure restoration scheme by message signaling to take advantage of the scalable and distributed processing capability of tree structure.

The performance of our schemes are shown by simulation.

Finally we give a summary of our work in chapter 4.

CHAPTER 2. Dynamic Establishment of Restorable Connections using p-Cycle Protection

In this chapter, we propose a new dynamic restorable connection establishment algorithm using p-cycle protection. This chapter is organized as follows. In Section 2.1, we introduce the p-cycle concept. In Section 2.2, we describe the new dynamic restorable connection establishment algorithm using p-cycle protection. We study the performance of the algorithm in Section 2.3. Finally, we summarize our work in Section 2.4.

2.1 The p-Cycle Concept

p-Cycle is a novel method for protecting working capacity (i.e., capacity used by the working paths of the connections) in WDM networks that can achieve ring-like recovery speed while maintaining the capacity efficiency of a mesh-restorable network [12]. A p-cycle is a pre-configured cycle formed out of the spare capacity in the network. A p-cycle of length k occupies k units of spare capacity in total, with one unit on each link on the cycle. (One unit of capacity is equal to one wavelength.) Like a self-healing ring, a p-cycle provides one restoration path for every on-cycle link; Unlike a self-healing ring, a p-cycle also provides two restoration paths for every *straddling link*–a link whose two end nodes are on the cycle but itself is not on the cycle. Fig 2.1 illustrates the concept of p-cycle. In the figure, $a-b-c-d-f-a$ is a p-cycle. For the on-cycle link $a-b$, the p-cycle provides one restoration path $a-f-d-c-b$. For the straddling link $f-b$, the p-cycle provides two restoration paths: $f-a-b$ and $f-d-c-b$. Thus, a p-cycle can protect one unit of working capacity on every on-cycle link and protect two units of working capacity on every straddling link. This allows a p-cycle to protect more working capacity than a self-healing ring of the same size while maintaining the high restoration speed of a self-healing

ring because only the two end nodes of a failed link need to reconfigure their cross-connects to switch the traffic onto the restoration paths. The efficiency of a p-cycle is defined as $\frac{2s+k}{k}$ where s is the number of straddling links of the p-cycle and k is the number of on-cycle links. The denominator is the total spare capacity consumed by the p-cycle and the numerator is the total working capacity that the p-cycle can protect. For the p-cycle shown in Fig. 2.1, the efficiency is $\frac{2*3+5}{5} = 2.2$ since it has 3 straddling links and 5 on-cycle links. Clearly, the larger is the efficiency, the more working capacity can a unit of spare capacity on the p-cycle protect.

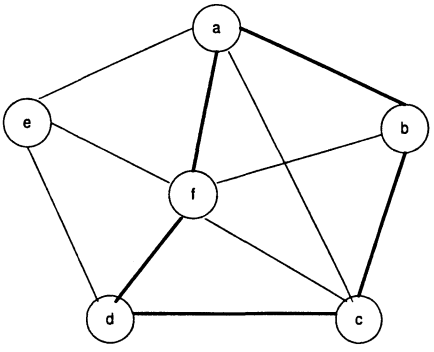


Figure 2.1 A p-cycle example

Studies on p-cycle have been focused on static traffic, where the connections to be established are given and the problem is to find an optimal set of p-cycles to protect them. Two versions of the optimization problem have been studied: non-joint version and joint version. In the non-joint version, working capacity on every link is assumed given (i.e., routing of the working paths for the connections are known) and the goal is to find a set of p-cycles to protect the working capacity so that the total spare capacity used by the p-cycles is minimized [12][14][15][16]. In the joint version, the routing of the working paths and the p-cycles are computed jointly so that the total capacity (i.e., working capacity plus spare capacity) is minimized [17][18][19]. To our best knowledge, no study has been done on applying p-cycles to protect dynamic connections. In this chapter, we give the first algorithm for dynamically establishing restorable connections using p-cycle protection and describe its advantage over link-based protection method.

2.2 Dynamic Restorable Connection Establishment using p-Cycle Protection

Given a connection request, our algorithm first computes a working path for the connection and then for each link on the working path computes a p-cycle to protect it. This enables the connection to survive any single link failure because when a link on the working path fails the p-cycle associated with the failed link can provide a restoration path for it.

The p-cycles to be created for protecting a working path are chosen from a set of primary p-cycles precomputed offline. Thus, our algorithm consists of three components: 1) offline computation of primary p-cycles, 2) online computation of working path, and 3) online computation of p-cycles for protecting the working path. We describe each of these components in this section. We assume the network nodes are capable of wavelength conversion so that links on the working path of a connection as well as links on a p-cycle can use any available wavelength on the link.

2.2.1 Offline Computation of Primary p-Cycles

Before network operation, we precompute one primary p-cycle for each link in the network as follows. First, we compute a set S of p-cycles so that every link l in the network can be protected by at least one p-cycle in S, i.e., there is at least one p-cycle in S for which l is either an on-cycle link or a straddling link. Next, for each link l in the network, we obtain the p-cycle with the highest efficiency among all p-cycles in S that can protect l and designate it as the primary p-cycle of l.

Various algorithms can be used to generate S. In this chapter, we adopt two existing algorithms: Straddling Link Algorithm (SLA) [20] and Grow [15].

2.2.1.1 Straddling Link Algorithm (SLA)

Straddling Link Algorithm (SLA) generates a set of p-cycles as follows. For each link l, the algorithm first finds the shortest path P_1 between the two end nodes of l that does not use l. It then tries to find the shortest path P_2 between the two end nodes of l that is node-disjoint with

P_1 and does not use l. If P_2 can be found, then it is combined with P_1 to form a p-cycle. This p-cycle can protect l since l is a straddling link. If P_2 cannot be found, then P_1 is combined with l to form a p-cycle. This p-cycle can protect l since l is on-cycle. Let m be the number of links in the network, then SLA generates exactly m p-cycles and every link is protected by at least one of these p-cycles.

2.2.1.2 Grow

The Grow algorithm we use here is a variation of the original Grow algorithm in that our algorithm generates exactly m p-cycles for a given network while the original Grow algorithm generates m^2n p-cycles in the worst case. (m and n are the number of links and the number of nodes in the network respectively.) Our Grow algorithm first uses SLA to generate a set of m p-cycles, it then expands each of these p-cycles to create larger p-cycles as follows. Let c be a p-cycle generated by SLA, we pick one link l on c and try to find the shortest path between the end nodes of l that is node-disjoint from c. If such a path cannot be found, we pick another link on c and try again. If such a path can be found, we remove l from c and add the path to c to get a larger p-cycle. (Note that this converts l from an on-cycle link to a straddling link.) We then pick a link on the new cycle and start over the process until no path can be found to replace any link on the cycle to expand the cycle. Among all the intermediate p-cycles obtained in the process, we keep the one with the best efficiency. Thus, a total of m p-cycles are generated by Grow. Note that whenever we replace an on-cycle link by a path, we get a larger p-cycle with possibly higher efficiency due to the increase of the straddling links. Thus, the p-cycles generated by Grow are guaranteed to have better or same efficiency as the p-cycles generated by SLA.

2.2.2 Online Working Path Computation

During network operation, connection requests arrive at the network dynamically. In this section, we present an online algorithm for computing the working path for a given connection request.

Our algorithm first removes all links with no free wavelengths from the network graph. We then assign a cost to every link in the resulting graph and use Dijkstra's algorithm to compute the least cost path between the source node and the destination node. If such a path can be found, then it is designated as the working path of the connection. Otherwise, the connection request is denied.

The cost of a link is a function of the *protectable working capacity* of the link, which is defined as the number of working capacity units on the link that can be protected by the existing p-cycles in the network. We use $P(l)$ to denote the protectable working capacity of a link l. Before the first connection request arrives at the network, no p-cycle is created in the network and therefore $P(l)$ is 0 for all l. As connection requests arrive at the network, p-cycles will be created to protect the working paths of the connections and the $P(l)$ values will be updated according to the algorithm to be described in the next section.

We propose two different link cost functions for working path computation: Protectable Working Capacity NonSensitive and Protectable Working Capacity Sensitive.

2.2.2.1 Protectable Working Capacity NonSensitive (PNS)

The cost of link l is computed as the following:

$$cost(l) = \begin{cases} \text{K} & \text{if } P(l) = 0 \\ 1 & \text{if } P(l) > 0 \end{cases}$$

where K is a large constant.

The motivation is to avoid using links that cannot be protected by any existing p-cycles whenever possible, which is achieved by assigning a high cost to the links whose $P(l)$ value is 0. For all other links, the cost is set to 1 to indicate that it is desirable to use these links on the working path since these links are already protected by some existing p-cycles. This link cost function is not sensitive to the protectable working capacity on the links because all links with $P(l)$ value greater than 0 get the same cost.

2.2.2.2 Protectable Working Capacity Sensitive (PS)

The cost of link l is computed as the following:

$$cost(l) = \begin{cases} \text{K} & \text{if } P(l) = 0 \\ \frac{1}{P(l)} & \text{if } P(l) > 0 \end{cases}$$

where K is a large constant.

Same as PNS, we assign a high cost to those links whose $P(l)$ value is 0 since we want to avoid using these links. Unlike PNS, the cost of a link with $P(l)$ value greater than 0 is sensitive to the protectable working capacity on the link in that the link cost is set to the reciprocal of its $P(l)$ value. The cost function favors the use of links with large $P(l)$ value, which can delay the time when a link's $P(l)$ value drops to 0 because a link with larger $P(l)$ value is used before using a link with smaller $P(l)$ value whenever possible. It's desirable to prevent a link's $P(l)$ value from dropping to 0 because when a working path uses a link whose $P(l)$ value is 0, we must create a new p-cycle to protect the link, which incurs additional spare capacity consumption.

2.2.3 Online Computation of p-Cycles for Working Path Protection

After the working path of a connection is computed, we need to compute a set of p-cycles to protect the working path. For each link l on the working path, our algorithm computes a p-cycle to protect it according to the following two cases.

- Case 1: $P(l) = 0$

  In this case, no existing p-cycle can protect l, so we need to create a new p-cycle–the primary p-cycle of l–to protect it. (Note that the primary p-cycles of all links are pre-computed offline.) Let C be the primary p-cycle of l. If any link on C does not have free wavelengths, then the p-cycle cannot be created to protect l and the connection request must be denied. Otherwise, we create the p-cycle by configuring the OXCs in the on-cycle nodes to interconnect one available wavelength in each on-cycle link. Each node on the p-cycle also records the ports used by the p-cycle so that when a link protected by this p-cycle fails in the future, the end nodes of the failed link know how to switch the traffic

to the restoration path provided by the p-cycle. We also update the protectable working capacity of the links protected by the newly created p-cycle as follows. For every link e on the p-cycle we increase $P(e)$ by 1 if $e \neq l$; for every link e that is a straddling link of the p-cycle we increase $P(e)$ by 2 if $e \neq l$ and we increase $P(e)$ by 1 if $e = l$. The update is based on the fact that once a p-cycle is created, it can protect one unit of working capacity on every on-cycle link and protect two units of working capacity on every straddling link. Since link l will consume one unit of protectable working capapcity provided by the newly created p-cycle, $P(l)$ is not increased if l is an on-cycle link and $P(l)$ is increased by 1 if l is a straddling link.

- Case 2: $P(l) > 0$

  In this case, at least one existing p-cycle can protect l, so we randomly choose one of those p-cycles to protect l. Since the working path of the current connection consumes one unit of protectable working capacity on l, we need decrease $P(l)$ by 1.

2.2.4 Comparison with Link-Based Protection Method

The advantage of the proposed p-cycle protection method over the conventional link-based protection method in dynamic restorable connection establishment is twofold. First, the p-cycle based method is much simpler to implement than the link-based method. In link-based method, for each link on the working path a backup path needs to be computed. To achieve a reasonable spare capacity consumption on the backup paths, backup capacity sharing needs to be exploited, which leads to complicated backup path computation algorithms. On the other hand, p-cycle based method is very simple: for each link l on the working path we simply check the value of $P(l)$. If it is 0, we create a new p-cycle to protect it and update the protectable working capacity of those links protected by the new p-cycle; otherwise we simply decrease $P(l)$ by 1. There is no need to compute an individual backup path for each link on the working path like the link-based method. In addition, the p-cycles to be created during connection establishment are precomputed offline, therefore the online computation required for dynamic restorable connection establishment is minimal. Second, p-cycle based method supports much

faster failure recovery than link-based method. In link-based method, although the backup paths are computed at the time of connection establishment, we cannot configure the OXCs to setup the backup paths at the connection establishment time due to the sharing of wavelengths among the backup paths. Thus, upon a link failure, a signaling protocol must be used to first setup the backup path for the failed link before the upstream node of the failed link can switch the traffic over to the backup path. In contrast, p-cycles are fully configured when they are created. Thus, upon a link failure, the two end nodes of the failed link just need to reconfigure their cross-connects to "break into" the p-cycles protecting the working capacities on the link so that traffic can be switched to the restoration paths provided by the p-cycles.

2.3 Numerical Results

We used two test networks to evaluate the performance of our p-cycle based dynamic restorable connection establishment algorithm. The two test networks are the 28-node 45-link USA long haul network and the 43-node 71-link France network taken from [15].

Since the performance of the online connection establishment algorithm depends on the primary p-cycles generated offline, we first compare the two offline primary p-cycle generation algorithms (SLA and Grow) for the number of unique primary p-cycles generated, the average primary p-cycle length and the average primary p-cycle efficiency. Table 2.1 shows the comparison of SLA and Grow for the two test networks. Note that in all cases, the number of unique primary p-cycles is smaller than the number of links in the network. This is because some links can have the same primary p-cycle. As shown in the table, the primary p-cycles generated by Grow are larger and have higher efficiency than those generated by SLA.

We implemented the online algorithm for computing p-cycles to protect a working path presented in Section 2.2.3 and three different online working path computation algorithms, namely SP, PNS, and PS. SP is the algorithm that simply uses the shortest path as the working path for a connection. PS and PNS are the algorithms given in Section 2.2.2 that use protectable working capacity sensitive and protectable working capacity nonsensitive link cost functions respectively. The two different offline primary p-cycle generation algorithms and the three

Table 2.1 Characteristics of the Primary p-Cycles

	USA		France	
	SLA	Grow	SLA	Grow
Number of unique p-cycles	36	29	59	50
Average p-cycle length	6.14	13.24	6.49	13.20
Average p-cycle efficiency	1.32	1.80	1.34	1.86

different online working path computation algorithms give us a total of six different algorithms for dynamic connection establishment: SLA+SP, SLA+PNS, SLA+PS, Grow+SP, Grow+PNS, Grow+PS. We evaluated the performance of these six algorithms under two demand sets for each test network. The first demand set is uniform, which contains one demand for every pair of nodes. The second demand set is random, which contains a large number of randomly generated demands. For the USA network the uniform demand set contains 378 demands and the random demand set contains 1000 demands. For the France network the uniform demand set contains 903 demands and the random demand set contains 3000 demands.

Table 2.2 through Table 2.5 give the performance results of the six algorithms for USA network and uniform demand, USA network and random demand, France network and uniform demand, France network and random demand respectively. Each table shows the number of p-cycles created, total working capacity used by the working paths, total spare capacity used by the p-cycles, the total capacity used by all demands (total working capacity plus total spare capacity), redundancy (the ratio of total spare capacity to total working capacity), average working path length, and average ratio of working path length to shortest working path length (i.e., length of the working path when SP is used).

As shown in the four tables, with a fixed working path computation algorithm, significantly less p-cycles are used to protect the demands when Grow is used than when SLA is used. This is expected because Grow generates primary p-cycles with higher efficiency and longer length than SLA does, thus a p-cycle generated by Grow can protect much more working capacity than

Table 2.2 USA network with 378 Uniform Demands

	SLA			Grow		
	SP	PNS	PS	SP	PNS	PS
# p-cycles	283	154	155	111	61	60
Working Cap.	1273	1486	1622	1273	1368	1503
Spare Cap.	2303	1292	1311	2255	1239	1220
Total Cap.	3576	2778	2933	3528	2607	2723
Redundancy	1.81	0.87	0.81	1.77	0.91	0.81
Ave working path length	3.37	3.93	4.29	3.37	3.62	3.98
Ave ratio to SP length	1.00	1.17	1.27	1.00	1.07	1.18

Table 2.3 USA network with 1000 Random Demands

	SLA			Grow		
	SP	PNS	PS	SP	PNS	PS
# p-cycles	682	388	391	256	155	153
Working Cap.	3224	3850	4162	3224	3610	3822
Spare Cap.	5596	3329	3367	5204	3157	3119
Total Cap.	8820	7179	7529	8428	6767	6941
Redundancy	1.74	0.86	0.81	1.61	0.87	0.82
Ave working path length	3.22	3.85	4.16	3.22	3.61	3.88
Ave ratio to SP length	1.00	1.19	1.29	1.00	1.12	1.19

a p-cycle generated by SLA can. With a fixed offline primary p-cycle generation algorithm, PS and PNS require much less p-cycles to protect the demands than SP does. This can be explained as follows. SP always uses the shortest path as the working path, which may contain some links with no protectable working capacity and therefore new p-cycles must be created to protect those links. On the other hand, both PNS and PS try to avoid routing the working path over a link with no protectable working capacity, thus they can reduce the number of p-cycles created for protecting the working path.

In terms of the total capacity used, Grow has better performance than SLA under all combinations of test network, demand set, and working path computation algorithm.

We now compare the performance of the three working path computation algorithms. Since Grow leads to less total capacity consumption than SLA, we focus on the performance comparison of SP, PNS, and PS when Grow is used to generate the primary p-cycles. As shows in Table 2.2 through Table 2.5, the performance of PNS and PS are much better than SP in terms of

Table 2.4 France Network with 903 Uniform demands

	SLA			GROW		
	SP	PNS	PS	SP	PNS	PS
# p-cycles	711	532	551	296	202	220
Working Cap.	3458	4713	4948	3458	3845	4270
Spare Cap.	5679	4151	4287	5506	3661	3886
Total Cap.	9137	8864	9235	8964	7506	8156
Redundancy	1.64	0.88	0.87	1.59	0.95	0.91
Ave working path length	3.83	5.22	5.48	3.83	4.26	4.73
Ave ratio to SP length	1.00	1.36	1.43	1.00	1.11	1.23

Table 2.5 France Network with 3000 Random demands

	SLA			GROW		
	SP	PNS	PS	SP	PNS	PS
# p-cycles	2275	1680	1730	954	659	701
Working Cap.	11068	15090	15915	11068	12297	13234
Spare Cap.	18138	13092	13478	17671	11826	12313
Total Cap.	29206	28182	29393	28739	24123	25547
Redundancy	1.64	0.87	0.85	1.60	0.96	0.93
Ave working path length	3.69	5.03	5.31	3.69	4.10	4.41
Ave ratio to SP length	1.00	1.36	1.44	1.00	1.11	1.20

the total capacity used. For USA network with uniform demand, PNS and PS require 26.11% and 22.82% less total capacity than SP respectively. For USA network with random demand, the capacity saving by PNS and PS are 19.71% and 17.64% respectively. The capacity saving by PNS and PS in France network with uniform demand are 16.27% and 9.01% respectively, and the capacity saving by PNS and PS in France network with random demand are 16.06% and 11.11% respectively. It's clear that PNS and PS result in more total working capacity consumption than SP does because SP always uses the shortest path as the working path. However, PNS and PS lead to significantly less total spare capacity consumption than SP does so that the total capacity consumption is less. This demonstrates that by avoiding using links with no protectable working capacity when routing a working path, we can effectively reuse the existing p-cycles to protect the working path and the resulted spare capacity saving is greater than the increase in working capacity consumption caused by not using the shortest path as the working path. In addition, the increase of the working path length is minor when PS and PNS are used.

As shown in the tables, the average increase of the working path length over the shortest working path length is around 10% and 20% for PNS and PS respectively.

A comparison of PNS and PS can give us some interesting insight about them. In all cases, the redundancy of PS is smaller than that of PNS but PS results in more total capacity consumption than PNS. The lower redundancy of PS means that a unit of spare capacity can protect more units of working capacity, thus PS is more effective in reusing the existing p-cycles for working path protection. However, the working path found by PS tends be longer than that found by PNS because in PNS all links with at least one protectable working capacity unit are used with equal probability while in PS links with higher protectable working capacity are preferred to links with lower protectable working capacity. The higher total capacity consumption by PS indicates that its higher working capacity consumption caused by using longer working paths overweighs its benefit gained in reusing existing p-cycles effectively.

In summary, Grow is better than SLA because it generates primary p-cycles with higher efficiency and PNS is better than PS and SP because it allows efficient reuse of existing p-cycles for working path protection while keeping the working path not much longer than the shortest path. Thus, Grow combined with PNS gives the best performance in terms of total capacity consumption.

2.4 Summary

In this chapter, we propose a new algorithm for dynamic establishment of restorable connections using p-cycle protection. The algorithm is much simpler to implement than the conventional link-based algorithm because there is no need to compute an individual backup path for every link on the working path. In addition, the algorithm achieves much faster restoration speed because the p-cycles used for working path protection are fully configured when they are created while in link-based method the backup paths can't be configured until after a link failure occurs. Our algorithm consists of three components: offline computation of primary p-cycles, online computation of working path, and online computation of p-cycles for working path protection. The aim of the first component is to generate a good p-cycle for protecting each network

link in advance so that the online computation for connection establishment is minimal. Two algorithms (SLA and Grow) are proposed for this component. For the second component, we propose two algorithms (PNS and PS) designed to find a working path that can take advantage of the existing p-cycles for its protection. We conducted simulations to study the performance of the different combinations of the proposed algorithms. The results show that Grow combined with PNS gives the best performance in terms of total capacity consumption.

CHAPTER 3. A Protection Tree Scheme for First-Failure Protection and Second-Failure Restoration

In this chapter we propose a new tree-based link protection scheme for single link failure in mesh networks, which achieves 100% restorability in an arbitrary 2-connected network. We also propose a distributed algorithm for fast double-link failure restoration using protection tree to take advantage of the scalable and distributed processing capability of tree structure. This chapter is organized as follows. In Section 3.1, we introduce the concept of hierarchical protection tree (p-tree). In Section 3.2 we propose our new p-tree scheme based on the previous work. Then we give an integer linear programming(ILP) formulation in Section 3.3 to minimize the total spare capacity for single link failure protection. In Section 3.4 we propose a fast double-link failure restoration scheme. We study the performance of our p-tree schemes in Section 3.5. Finally, we summarize our work in Section 3.6.

3.1 The Hierarchical P-Tree Scheme

A link protection scheme based on the concept of hierarchical protection tree (p-tree) is proposed in [31]. This scheme forms a hierarchical spanning tree in the network that which is a special type of spanning tree for link protection in which the links in the higher layers provide more protection capacity than the lower layers of the tree.

For each node u in the network other than the root, there is exactly one parent in the hierarchical tree which is called *primary parent* of node u. Other than *primary parent*, the neighbor nodes except its children are called *backup parent* of node u. Fig 3.1 shows a hierarchical p-tree in an arbitrary network, where thick lines denote tree links and thin lines denote non-tree links. In this example, node D is the *primary parent* of node G, and the neighbor nodes B, C

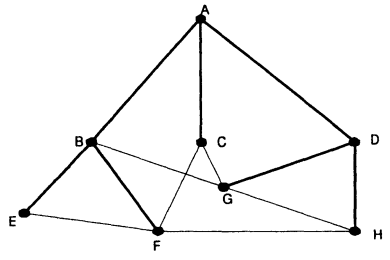


Figure 3.1 A hierarchical p-tree

and H are the *backup parents* of node G. The protection scheme used for this hierarchical p-tree is described as follows:

1. If a link not on the p-tree fails, the two nodes adjacent to the failure will switch the traffic to their primary parent so that the traffic is restored through a path consisting of only tree links. For example, if link $B - G$ in Fig 3.1 fails , the traffic on it will be restored through $G - D - A - B$.

2. If a link on the p-tree fails, one of the nodes adjacent to the failure will be disconnected from p-tree. This node will switch the traffic to one of its *backup parents* so that the traffic is restored through a path consisting of one or more tree links plus exactly one non-tree link. For example, if link $D - G$ in Fig 3.1 fails, G will switch the traffic to its backup parent C so that the path $G - C - A - D$ is used to restore the traffic.

The hierarchical p-tree scheme allows spare capacity sharing. For example, in Fig 3.1, since D is the *primary parent* of node G, link $D - G$ is on the backup path of link $B - G$,$C - G$ and $H - G$. Therefore, the spare capacity reserved on link $D - G$ is shared by the backup paths for $B - G$,$C - G$ and $H - G$. To protect against any single link failure, a spare capacity at least equal to the maximum of working capacities on link $B - G$,$C - G$ and $H - G$ should be reserved on link $D - G$. In [31], a distributed heuristic algorithm is proposed to solve the following problem: Given a connected simple network and the working and spare capacity on links, find a hierarchical p-tree to maximize the restorability of network. With the distributed

algorithm, the p-tree can be maintained dynamically by message exchanges among the network nodes in the case of network node/link addition and removal.

However, a problem with the hierarchical p-tree in [31] is that it may not find a backup path for a link even if the network is 2-connected. In particular, when a node has no *backup parent*, the scheme can not find a backup path for any on-tree link adjacent to this node. For example, node D in Fig 3.1 has no *backup parent*, when link $A-D$ fails, the scheme can not find a backup path for it. In the next section, we propose a new p-tree scheme to solve this problem.

3.2 A New P-Tree Protection Scheme

In this section, we present two theorems and propose a new p-tree scheme which can survive any single link failure in an arbitrary 2-connected network based on the following theorem.

Definition: Let $T(V, E_T)$ denote a spanning tree of a 2-connected network $G(V, E)$ where V is the set of nodes , E is the set of links in the network G and $E_T \subseteq E$. A link $e \in E_T$ is called *tree link*, a link $e \in E - E_T$ is called *non-tree link*. For any *tree link* $e = (u, v)$, a path between u and v which is link disjoint with e and contains exactly one *non-tree link* is called a *back path*. For any *non-tree link* $e = (u, v)$, a path between u and v which is link disjoint with e, and contains only *tree links* is called a *tree path*.

In Fig 3.1, $A - C - G - D$ is a *back path* of *tree link* (A, D) because it contains exactly one *non-tree link* (C, G). $F - B - A - D - H$ is a *tree path* of *non-tree link* (F, H) because all the links on the path are *tree links*.

Theorem 1: Given a connected network and a spanning tree of the network, for any *non-tree link*, there exists exactly one *tree path*.

Proof: Let $e = (u, v)$ be a *non-tree link*. Since the spanning tree contains all nodes, u and v are on the spanning tree. Therefore, there is a *tree path* for e. There must be exactly one *tree path* for e because a spanning tree contains no cycles. ■

Theorem 2: Given a 2-connected network and a spanning tree of the network, for any *tree link*, there exists a *back path*.

Proof: Let $e = (u, v)$ be a *tree link*. The removal of link e will break the spanning tree into

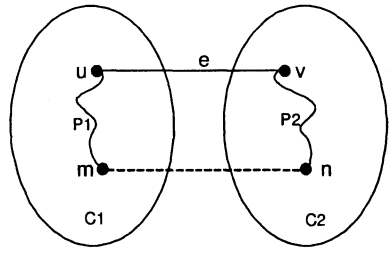


Figure 3.2 Proof of Theorem 2

two components, c_1 and c_2, as shown in Fig 3.2. Since the network is 2-connected, by Menger's theorem [25], there must be a path between u and v that does not contain e. Thus, there must exist a *non-tree link* between c_1 and c_2, say (m, n). Since node $u, m \in c_1$, there exists a path $\mathcal{P}_1$ between u and $m \in c_1$. Similarly there exists a path $\mathcal{P}_2$ between v and $n \in c_2$. Hence, the path $\mathcal{P}_1 \bigcup \{(m,n)\} \bigcup \mathcal{P}_2$ is a *back path* of the *tree link* e. ■

As discussed in the section 3.1, the hierarchical p-tree scheme proposed in [31] can not find a *backup path* for some *tree link* such as link (A, D) in Fig 3.1. However, based on Theorem 2, we know there exists a *back path* $D - G - C - A$ for (A, D). Therefore, the failure on (A, D) can be restored by its *back path* $D - G - C - A$.

Based on Theorem 1 and Theorem 2, we propose the following p-tree link protection scheme:

1. If a *non-tree link* fails, as original hierarchical p-tree scheme, the traffic on the failed link is re-routed to its *tree path.*

2. If a *tree link* fails, one of its *back paths* is selected to re-route the traffic on the failed link. The ILP formulation given in the section 3.3 determines which one of the *back paths* is selected with the objective of minimizing the total spare capacity required for surviving any single link failure.

The above p-tree scheme can restore traffic upon any single link failure in a 2-connected network, because Theorem 1 and Theorem 2 guarantee that there exists a backup path for any link in the network.

3.3 ILP Formulation

Consider a network $G(V, E)$ with a set of nodes V and a set of links E. In this chapter, a link $e = (i, j) \in E$ denotes the bidirectional link between node i and node j, where $i < j$. Each link $e = (i, j) \in E$ is associated with two arcs, $(i \rightarrow j)$ and $(j \rightarrow i)$.

In this section, we give an ILP formulation for the following problem:
Given a 2-connected network $G(V, E)$ and the working capacity on link $e \in E$, find a p-tree $T(V, E_T)$ where $E_T \subseteq E$ and determine the backup path for every link $e \in E$ following our p-tree scheme with the objective of minimizing the total spare capacity required to survive any single link failure.

We are given the following inputs.

- G: Topology of the network.
- V: nodes in the network (numbered 1 through N).
- E: set of the bidirectional links in G. $(i, j) \in E$ denotes a link between node i and node j where $i < j$.
- $\overline{E}$: set of the arcs in G. $(i \rightarrow j) \in \overline{E}$ denotes an arc from node i to node j.
- w_{ij}: working capacity on link (i, j).
- $c_{i,j}$:cost of a unit of capacity on link (i, j), unit cost is assumed here, i.e., $c_{i,j} = 1$.

The following variables are determined by the ILP.

- $T_{i,j}$: take on the value of 1 if link (i, j) is on the p-tree, otherwise 0.
- $F_{i,j}^{m,n}$: take on the value of 1 if the restoration route from m to n goes through arc $(i \rightarrow j)$, otherwise 0.
- $\phi_{i,j}^{m,n}$: take on the value of 1 if link (m, n) is protected by link (i, j).
- $\delta_{i,j}^{m,n}$: take on the value of 1 if link (m, n) is protected by a *tree link* (i, j).
- $s_{i,j}$: spare capacity reserved on link (i, j).

Our objective is to minimize the total spare capacity reserved:

Minimize $\sum_{(i,j)\in E} c_{i,j}s_{i,j}$

Subject to the following constraints:

$$\sum_{(i,j)\in E} T_{i,j} = N - 1 \tag{3.1}$$

$$\sum_{i,s.t.(i\to j)\in\overline{E}} F_{i,j}^{m,n} - \sum_{k,s.t.(j\to k)\in\overline{E}} F_{j,k}^{m,n} = \begin{cases} -1 & m = j \\ 1 & n = j \\ 0 & otherwise \end{cases}$$

$$\forall(m,n) \in E, \forall j = 1..N \tag{3.2}$$

$$F_{m,n}^{m,n} = 0, \forall(m,n) \in E \tag{3.3}$$

$$\phi_{i,j}^{m,n} = F_{i,j}^{m,n} + F_{j,i}^{m,n}, \forall(m,n) \in E, \forall(i,j) \in E \tag{3.4}$$

$$\sum_{(i,j)\in E} \phi_{i,j}^{m,n} - \sum_{(i,j)\in E} \delta_{i,j}^{m,n} = T_{mn}, \forall(m,n) \in E \tag{3.5}$$

$$\delta_{i,j}^{m,n} \le \phi_{i,j}^{m,n}, \forall(m,n) \in E, \forall(i,j) \in E \tag{3.6}$$

$$\delta_{i,j}^{m,n} \le T_{i,j}, \forall(m,n) \in E, \forall(i,j) \in E \tag{3.7}$$

$$\phi_{i,j}^{m,n} + T_{i,j} - 1 \le \delta_{i,j}^{m,n}, \forall(m,n) \in E, \forall(i,j) \in E \tag{3.8}$$

$$w_{m,n} \times \phi_{i,j}^{m,n} \le s_{i,j}, \forall(m,n) \in E, \forall(i,j) \in E \tag{3.9}$$

Constraints (3.1) and (3.5) ensure a p-tree is set up based on the following theorem.

Theorem 3: Let $T(V_T, E_T)$ be a subgraph of a connected graph $G(V, E)$. If $|E_T| = |V| - 1$ and $\forall(u,v) \in E - E_T$, there exists a path $\mathcal{P}$ between u and v s.t. $l \in E_T$ for all $l \in \mathcal{P}$, then $T(V_T, E_T)$ is a spanning tree of $G(V, E)$.

Proof:

1. First, we prove T is connected by contradiction. Assume T is not connected, then T consists of at two components. Let T_1 and T_2 be two components in T. Since G is connected, there must exist $(u,v) \in E - E_T$ with u in T_1 and v in T_2. With the condition

we have, there is a path in T that connects u and v. This means that T_1 and T_2 are connected in T , which is a contradiction.

2. Next, we prove $|V_T| = |V|$ by contradiction. Assume $|V_T| \neq |V|$, then there exists a node $u \in V - V_T$. Since G is connected, u is connected with some node $v \in V_T$, such that $(u, v) \in E - E_T$. With the condition we have, there exists a path in E_T that connects u and v, this means that $u \in V_T$ which is a contradiction.

3. We have proved that $T(V_T, E_T)$ is connected and $|V_T| = |V|$. With the condition $|E_T| = |V| - 1$, T must be acyclic. By definition, a connected acyclic subnetwork of G containing all the nodes in G is a spanning tree of network G. ■

The left side of Equation (3.5) denotes the number of *non-tree links* that protect link (m, n). For a *non-tree link* (m, n), the right side of (3.5) equals 0, which ensures that a *non-tree link* is protected by a backup path consisting of *tree links* only. Thus, for any *non-tree link* there is a *tree path*. Constraint (3.1) ensures that the number of *tree links* is $|V| - 1$. According to theorem 3, constraint (3.1) and (3.5) ensure that a spanning tree of network G is found.

Using standard network flow formulation, constraint (3.2) ensures that for all $(m, n) \in E$, there is a backup path $\mathcal{R}$ from m to n. Constraint (3.3) ensures that $\mathcal{R}$ does not use link (m, n).

Constraint (3.4) ensures that link (m, n) is protected by link (i, j), if and only if the backup path from m to n goes through either $(i \rightarrow j)$ or $(j \rightarrow i)$.

Constraint (3.5) implies that a *back path* is selected as the backup path of a *tree link* and a *tree path* is selected as the backup path of a *non-tree link*.

Constraint (3.6)-(3.8) ensure that $\delta_{i,j}^{m,n} = 1$, if and only if link (m, n) is protected by link (i, j) and link (i, j) is a *tree link*.

Constraint (3.9) ensures that enough spare capacity is reserved on each link to protect again any single link failure.

3.4 Double-Link Failure Restoration

When a link failure occurs, it might take a few hours to a few days to repair the failed link. It is certainly conceivable that a second link failure might occur in this duration, leading to double-link failure in the network. Double-link failure has been studied in [33]-[38].

In this section, we expand our p-tree scheme to deal with double-link failure, where the concept of *first-failure protection and second-failure restoration*(1FP-2FR) [32] is implemented. The idea of 1FP-2FR is the following: When a p-tree is constructed, the backup path for each network link is determined. When a first link failure occurs, the pre-determined backup path for the failed link will be used for traffic restoration. When a second failure occurs, a backup path for that failed link will be searched dynamically using the algorithm given in this section.

We assume that a p-tree is pre-computed by the ILP given in the Section 3.3 and the root of the p-tree is selected randomly. Each node has a tree ID in the dotted decimal notation described in [31]. For example, the root node has a tree ID of 1. The tree IDs for its children are 1.1, 1.2, 1.3 etc. A node with tree ID 1.2.3.2 is a child of the node with tree ID 1.2.3.

3.4.1 Double-Link Failure Restoration Model

Suppose link e and f fail successively, and f fails before e is repaired. Once the failure on link e is detected, its pre-computed backup path $p(e)$ is used to re-route the traffic on e. At the same time, all nodes in the network are informed of the failure of e through messages $MSG\_FAILURE1$(as shown in Fig 3.3) sent by the master node of the failed link e. (The master node of a failed link is one of the two end nodes of a failed link that has the smaller tree ID in lexicographic order.) $MSG\_FAILURE1$ includes tree ID of master node and the other adjacent node of the failed link. In addition, a group of tree IDs of the nodes on the backup path $p(e)$ of e are encapsulated in $MSG\_FAILURE1$, so that all nodes in the network are informed of the backup path of the first failed link e. The REQ field in $MSG\_FAILURE2$(as shown in Fig 3.3) denotes whether the secondary backup path search is requested or not. When the second failure on link f occurs, the master node of f will set the value of REQ based on the condition whether backup path $p(f)$ of f uses e and backup path $p(e)$ of e uses f. There are

four possible cases as described in [33].

Figure 3.3 Message Format

1. $p(f)$ does not use e and $p(e)$ does not use f. In this case $p(e)$ will continue to be used to re-route the traffic on e, and $p(f)$ will be used to re-route the traffic on f.

2. $p(f)$ does not use e, but $p(e)$ uses f. Since $p(f)$ is not affected by the two failures, the working traffic on f and the re-routed traffic on f due to the failure on e are both switched from f to $p(f)$. Thus, the working traffic originally routed on e is now routed on $p(e) - \{f\} \bigcup p(f)$.

3. $p(f)$ uses e, but $p(e)$ does not use f. In this case, $p(e)$ works properly under the two failures. Similar to case 2, the working traffic on e and the re-routed traffic on e due to the failure on f are both switched from e to $p(e)$. Thus, the working traffic originally routed on f is now routed on $p(f) - \{e\} \bigcup p(e)$.

4. $p(f)$ uses e and $p(e)$ uses f. In this case, both of $p(f)$ and $p(e)$ are down. Real-time search for a secondary backup path $p'(f)$ of f that does not use e is needed. The working traffic on e will be re-routed on $p(e) - \{f\} \bigcup p'(f)$, and the working traffic on f is switched from f to $p'(f)$.

Note that this scheme requires that the master node of second failed link f knows the first failure on e and the backup path $p(e)$ of e. On receiving $MSG\_FAILURE1$, it finds $p(f)$ uses e and $p(e)$ uses f(case 4), it broadcasts $MSG\_FAILURE2$ to the network with $REQ = true$, requesting for the secondary backup path as in the following approaches. Otherwise (case 1,2 and 3) it broadcasts $MSG\_FAILURE2$ to the network with $REQ = false$ and switches the traffic on

f to the pre-computed backup path $p(f)$(case 1 and 2) or the backup path $p(f) - \{e\} \bigcup p(e)$(case 3). In the next section, we give a distributed algorithm for finding the secondary backup path.

3.4.2 Algorithm for Finding the Secondary Backup Path

Note that the secondary backup path $p'(f)$ for link f must be link disjoint with e and f so that it can work properly in the case that both e and f are down. There are three scenarios in finding such a $p'(f)$ with the condition $e \in p(f)$ and $f \in p(e)$. These scenarios are depicted in the Figure 3.4- 3.6, where $T(x)$ denotes a tree rooted at node x, thick line denotes *tree link* and thin line denotes *non-tree link*. m_e and n_e denote the master node and the other node incident to link e respectively, m_f and n_f denote the master node and the other node incident to link f respectively.

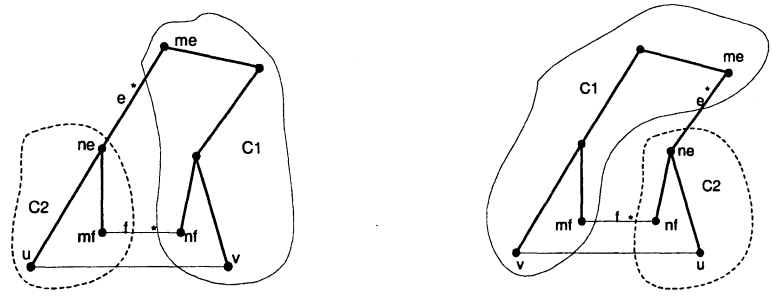


Figure 3.4 Scenario 1 in secondary backup path search

1. **Scenario 1**: e is a *tree link* and f is *non-tree link*.

   As shown in Fig 3.4, the removal of link e divides the p-tree into two components c_1 and c_2. There are two cases: in the first case, m_f is in a different component as m_e; in the second case, m_f is in the same component as m_e. In both cases, if there is a *non-tree link* other than link f between c_1 and c_2, say link (u, v), then there exists a path between m_f and n_f that goes through some *tree links* and the *non-tree link* (u, v), and the path can be used as the secondary backup path $p'(f)$ of link f.

2. **Scenario 2**: e is a *non-tree link* and f is a *tree link*.

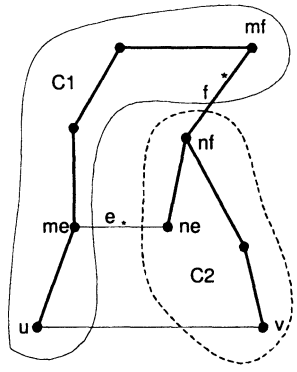
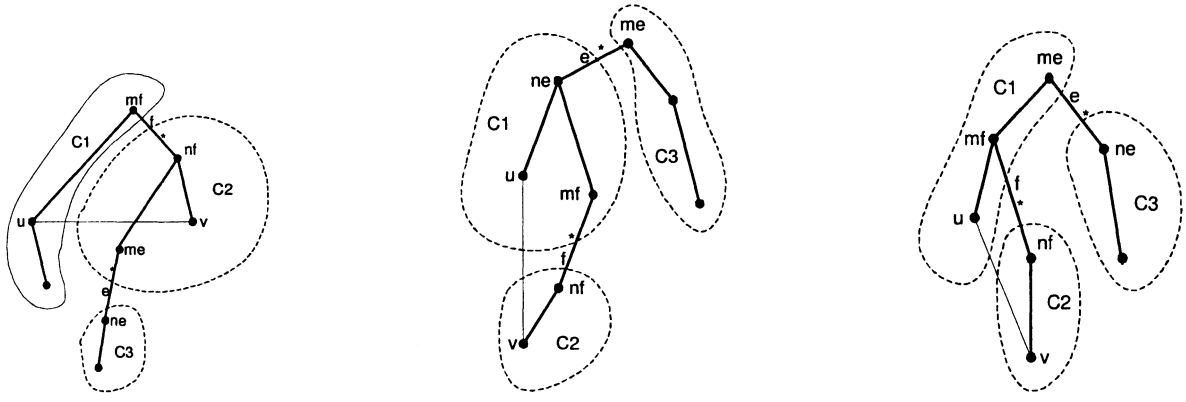


Figure 3.5 Scenario 2 in secondary backup path search



Scenario 3.1 **Scenario** 3.2 **Scenario** 3.3

Figure 3.6 Scenario 3 in secondary backup path search

As shown in Fig 3.5, the removal of link f divides the p-tree into two components c_1 and c_2. If there is a *non-tree link* other than link e between c_1 and c_2, say link (u, v), then there exists a path between m_f and n_f that go through some *tree links* and the *non-tree link* (u, v), and the path can be used as the secondary backup path $p'(f)$ of link f.

3. **Scenario 3**: both e and f are *tree links*.

As shown in Fig 3.6, the removal of e and f divides the p-tree into three components c_1, c_2 and c_3, where m_f belongs to c_1 and n_f belongs to c_2. There are three cases. In the first case, m_f is neither in the same component as m_e nor in the same component as n_e. In

the second case, m_f is in the same component as n_e. In the third case, m_f is in the same component as m_e. In all three cases, if there is a *non-tree link* between c_1 and c_2, say link (u, v), then there exists a path between m_f and n_f that go through some *tree links* and the *non-tree link* (u, v), and the path can be used as the secondary backup path $p'(f)$ of link f.

We now describe how the idea outlined above can be implemented.

Firstly, we define two functions **IS_ON_TREE** and **IS_TREE_LINK**.

The function **IS_ON_TREE**(x, y) checks whether node y is on the sub-tree $T(x)$ which is rooted at node x. If y is on $T(x)$, the tree ID of x must be the prefix of that of y. We say that a string x is a *prefix* of a string y, denoted as $x \sqsubset y$, if $y = xw$ for some string w. For example, the node with tree ID 1.1.2.3.2 is on a sub-tree rooted at the node with tree ID 1.1.2 because 1.1.2 is a *prefix* of 1.1.2.3.2.

IS_ON_TREE(x, y) {

1 if $x \sqsubset y$

2 return *true*

3 else

4 return *false*

5 }

The function **IS_TREE_LINK**(x, y) checks whether link (x, y) is a tree link or not, assuming tree ID of x is smaller than tree ID of y in the lexicographic order. If the node y is on $T(x)$ and y's tree ID has one more field than that of x, then (x, y) is a *tree link*. Function **LENGTH** returns the length of x's fields. For example. Suppose x = 1.1.2 and y = 1.1.2.3, then $LENGTH(x) = 3$, $LENGTH(y) = 4$. Since y is on $T(x)$ and y's tree ID has one more field than x's tree ID, we know (x, y) is a *tree link*.

IS_TREE_LINK(x, y) {

1 if $x \sqsubset y$ AND $LENGTH(y) - LENGTH(x) == 1$

2 return *true*

3 else

4 return *false*

5 }

Suppose link $e = (m_e, n_e)$ and link $f = (m_f, n_f)$ fail successively, where m_e, m_f are the master nodes of link e and f respectively. On receiving a message $MSG\_FAILURE2$ with $REQ = true$, each node u runs the **MESSAGE_PROCESS** algorithm. If u has a desirable neighbor node v, it will create a message MSG_ACK (shown in refmsg-ack) and send it to master node m_f of link f, when m_f receives MSG_ACK, it will run algorithm **RECOVERY_PATH** to determine a secondary backup path $p'(f)$ based on the information in MSG_ACK. To simplify the work of m_f to find the secondary backup path, the following *consistency constraint* of the message MSG_ACK is required: the node u must be in the same component of p-tree with m_f after link f fails.

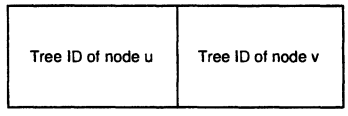


Figure 3.7 The message format of MSG_ACK

The detail of algorithm **MESSAGE_PROCESS** for the three scenarios is given below. The following notations are used in the algorithm description. T denotes the p-tree, $T(u)$ be the subtree rooted at node u. $V_T(u)$ denotes the set of nodes in $T(u)$, $E_T(u)$ denotes the set of links in $T(u)$.

1. **Scenario 1**: e is a *tree link* and f is a *non-tree link*.

   The removal of link e will divide the p-tree into two components $T(n_e)$ and $T - T(n_e)$. Using algorithm **IS_ON_TREE**, node u knows whether u is in $V_T(n_e)$. If so, it checks whether it has a neighbor node v such that $v \notin V_T(n_e)$. If such a node v is found, u will create package MSG_ACK and send it to m_f. To satisfy the *consistency constraint* of package MSG_ACK, u needs to verify whether it is in the same component of p-tree with

m_f after the failure on f. (If $m_f \in V_T(n_e)$, then u is in the same component of p-tree with m_f, because $u \in V_T(n_e)$.) In scenario 1.1, u is in the same component of p-tree with m_f, so the *consistency constraint* of message *MSG_ACK* is satisfied. In scenario 1.2, u is not in the same component of p-tree with m_f. Therefore, the tree ID of u and v need to be switched when they are encapsulated into the message *MSG_ACK* to satisfy the *consistency constraint*. The function **SWITCHID**(u,v) is used to switch the tree IDs of u and v into *MSG_ACK*.

2. **Scenario 2**: e is a *non-tree link* and f is a *tree link*.

   The removal of link f will divide p-tree into two components $T(n_f)$ and $T - T(n_f)$. If $u \notin V_T(n_f)$ and u has a neighbor node v such that $v \in V_T(n_f)$, node u will create message *MSG_ACK* and send it to m_f. Note that in this scenario, $u \notin V_T(n_f)$ ensures u is in the same component of p-tree with m_f after the failure on f, therefore the *consistency constraint* of *MSG_ACK* is satisfied.

3. **Scenario 3**: both e and f are *tree links*.

   The removal of link e and f will divide p-tree into three components. In scenario 3.1, the three connected components of p-tree are $T(n_e)$, $T(n_f) - T(n_e)$ and $T - T(n_f)$. If node $u \notin V_T(n_f)$ and u has a neighbor node v such that $v \in V_T(n_f) - V_T(n_e)$, u will create message *MSG_ACK* and send it to m_f. In scenario 3.2, the three connected components of p-tree will be $T(n_f)$, $T(n_e) - T(n_f)$ and $T - T(n_e)$. If $u \in V_T(n_e) - V_T(n_f)$ and u has a neighbor node v such that $v \in V_T(n_f)$, u will create message *MSG_ACK* and send it to m_f. Scenario 3.3 covers the third cases, where the three components of p-tree are $T(n_e)$, $T(n_f)$ and $T - T(n_e) - T(n_f)$. If $u \notin V_T(n_e)$ and $u \notin V_T(n_f)$and u has a neighbor v such that $v \in V_T(n_f)$, u will create message *MSG_ACK* and send it to m_f. Note that in all three cases, u is in the same component of p-tree with m_f after the failure on f, therefore the *consistency constraint* of *MSG_ACK* is always satisfied.

The pseudo algorithm of **MESSAGE_PROCESS** is given as follows.

MESSAGE_PROCESS {

//Scenario 1

1 if $IS\_TREE\_LINK(m_e, n_e)$ AND NOT $IS\_TREE\_LINK(m_f, n_f)$

2 {

3 if $IS\_ON\_TREE(n_e, u)$ AND u has a neighbor node v
s.t. NOT $IS\_ON\_TREE(n_e, v)$

4 {

//Scenario 1.2

5 if NOT $IS\_ON\_TREE(n_e, m_f)$

6 SWITCHID(u, v)

7 Send MSG_ACK to m_f

8 }

9 }

//Scenario 2

10 if NOT $IS\_TREE\_LINK(m_e, n_e)$ AND $IS\_TREE\_LINK(m_f, n_f)$

11 {

12 if NOT $IS\_ON\_TREE(n_f, u)$ AND u has a
neighbor node v s.t. $IS\_ON\_TREE(n_f, v)$

13 Send MSG_ACK to m_f

14 }

//Scenario 3

15 if $IS\_TREE\_LINK(m_e, n_e)$ AND $IS\_TREE\_LINK(m_f, n_f)$

16 {

//Scenario 3.1

17 if $IS\_ON\_TREE(n_f, m_e)$ {

18 if NOT $IS\_ON\_TREE(n_f, u)$ AND u has a
neighbor node v s.t. $IS\_ON\_TREE(n_f, v)$ AND
NOT $IS\_ON\_TREE(n_e, v)$

18 Send MSG_ACK to m_f.

20 }

//Scenario 3.2

21 else if $IS\_ON\_TREE(n_e, m_f)$ {

22 if $IS\_ON\_TREE(n_e, u)$ AND NOT $IS\_ON\_TREE(n_f, u)$ And u has a neighbor v s.t. $IS\_ON\_TREE(n_f, v)$

23 Send MSG_ACK to m_f.

24 }

//Scenario 3.3

25 else {

26 if NOT $IS\_ON\_TREE(n_e, u)$ AND NOT $IS\_ON\_TREE(n_f, u)$ AND u has a neighbor node v s.t. $IS\_ON\_TREE(n_f, v)$

27 Send MSG_ACK to m_f.

28 }

29 }

30 }

When the master node m_f of link f receives the message MSG_ACK, it computes a secondary backup path $p'(f)$ of f using the **RECOVERY_PATH** procedure, where $\mathcal{P}$ is composed of the *tree path* between m_f and u, the link (u, v) and the *tree path* between v and n_f. The **RECOVERY_PATH** procedure uses the **P_TREE_PATH**(x, y) procedure to find a path between x and y in the p-tree. The pseudo code for the **RECOVERY_PATH** procedure and **P_TREE_PATH** procedure are given below.

RECOVERY_PATH {

1 $\mathcal{P} = \emptyset$

2 $\mathcal{P} = \mathcal{P} \bigcup P\_TREE\_PATH(m_f, u)$

3 $\mathcal{P} = \mathcal{P} \bigcup \{(u, v)\}$

4 $\mathcal{P} = \mathcal{P} \bigcup P\_TREE\_PATH(v, n_f)$

5 return $\mathcal{P}$

6 }

P_TREE_PATH(x, y) {

1 $\mathcal{P} = \emptyset$

2 if $IS\_ON\_TREE(x, y)$

3 go to LOOP2

4 LOOP1:

5 $\mathcal{P} = \mathcal{P} \bigcup \{(x, parent(x))\}$

6 if NOT $IS\_ON\_TREE(parent(x), y)$

7 {

8 $x = parent(x)$

9 go back to LOOP1

10 }

11 $x = parent(x)$

12 if (x == y)

13 return $\mathcal{P}$

14 LOOP2:

15 $\mathcal{P} = \mathcal{P} \bigcup \{(y, parent(y))\}$

16 if $x! = parent(y)$

17 {

18 $y = parent(y)$

19 go back to LOOP2

20 }

21 return $\mathcal{P}$

22 }

3.5 Numerical Results

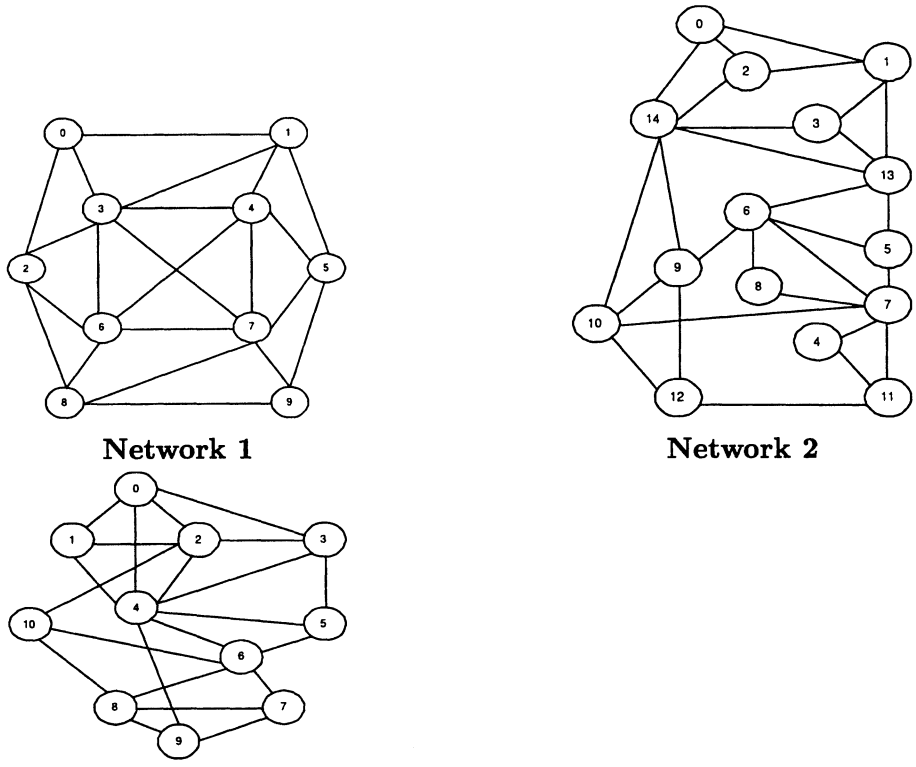


Figure 3.8 Topology of test network

Three test networks(shown in Fig 3.8) are used to evaluate the performance of our p-tree scheme. Net1 is an artificial 10-node 22-link network taken from [39]. Net2 is the 15-node 28-link Bellcore New Jersey LATA network, which is a widely used metropolitan area model. Net3 is a modified NJ LATA network with 11 nodes and 22 links taken from [34]. A uniform demand matrix with 2 demand units between every node pair is used for all three test networks.

Table 3.1 shows the performance results of our p-tree scheme for single link failure, which is obtained by solving the ILP given in Section 3.3. Table 3.1 gives the total working capacity, the total spare capacity, and the redundancy(i.e. the ratio of the total spare capacity to the total

working capacity). Due to the sharing of spare capacity on links shared by multiple backup paths, the p-tree scheme leads to less than 100% redundancy.

Table 3.2 shows the performance results of our p-tree scheme for double-link failure. #Total pairs is the number of all possible $< e, f >$ pairs assuming link e and f fail successively. #Survivable pairs is the number of $< e, f >$ pairs where $e \notin p(f)$ or $f \notin p(e)$ so that there is no need to find a secondary backup path for link f. R_1 is the ratio of the number of restorable pairs when a secondary backup path for link f is searched if necessary to the number of total pairs under the condition that the spare capacity allocated in the network is optimized in the single link failure(i.e. the spare capacity allocation is provided by the ILP). R_2 is the ratio of the number restorable pairs when a secondary backup path for link f is searched if necessary to the number of total pairs under the condition that the spare capacity on each link is sufficiently large. In NET1, NET2 and NET3, #Survivable pairs is 400, 572 and 386, that is, for 87%, 76% and 84% of all possible double link failures, double-link failure is restored without secondary backup path search so that the restoration of double-link failure is as fast as Method II of double-link failure recovery approaches proposed in [33] where two backup paths are pre-computed for each link. With the spare capacity planned for single link failure, our p-tree scheme with the secondary backup path search algorithm can restore 72%, 67% and 71% of all the double link failures in NET1, NET2 and NET3 respectively. Assuming sufficient spare capacity is available in the network, the percentage of restorable double-link failures reaches 99%, 99% and 98% in NET1, NET2 and NET3 respectively. There are two reasons that the percentage is less than 100%. First, a secondary backup path for link f may not exist. Second, our algorithm may not be able to find a secondary backup path for link f even though such a path exists because we require that the secondary backup path can only use one *non-tree link*. As shown in table 3.2, a majority of double link failures can be restored with our algorithm even though the spare capacity is planned for single link failure. In addition, our algorithm can achieve almost 100% restorability for double link failure when spare capacity in the network is sufficiently large. Thus, the proposed scheme is effective for the restoration of double link failure.

Table 3.1 Result of the ILP Solution

Network	Work Cap.	Spare Cap.	Redundancy
NET1	142	96	0.68
NET2	456	424	0.93
NJ-LATA	190	178	0.94

Table 3.2 Result of Double-link Failure Restoration

Network	#Total pairs	#Survivable pairs	R1	R2
NET1	462	400	0.72	0.99
NET2	756	572	0.67	0.99
NJ-LATA	462	386	0.71	0.98

3.6 Summary

In this chapter, we propose a new p-tree scheme that provides 100% restorability for single-link failure in a 2-connected network. To minimize the spare capacity required, we give an ILP formulation to determine the optimal p-tree and spare capacity allocation for a network with working capacity given on each link. We also develop a distributed fast restoration algorithm for dealing with double-link failure, which searches for a secondary backup path for the second link failure in real-time when necessary. The numerical results show that a majority of the double-link failures can be restored with our algorithm even though the spare capacity is planned for single-link failure. In addition, our algorithm can achieve almost 100% restorability for double link failure when spare capacity in the network is sufficiently large.

CHAPTER 4. Summary

We propose a new algorithm for dynamic establishment of restorable connections using p-cycle protection. Each link on the working path is protected by the pre-computed p-cycles which leads to the easier implementation compared with the conventional link-based algorithm because backup path computation is not required in our algorithm. In addition, our algorithm achieves much faster restoration speed since p-cycles are fully configured when they are created. Three components are designed for our algorithm: offline computation of primary p-cycles, online computation of working path, and online computation of p-cycles for working path protection. Two algorithms (SLA and Grow) are proposed for the first component which generates good candidate p-cycles in advance. We also propose two algorithm (PNS and PS) to the second component which intends to find the working path to take advantage of the existing p-cycles for its protection. The third component assigns the existing p-cycles or creates new p-cycles to protect all links on the working path. Simulations of the different combination of our proposed algorithms are conducted, which shows that Grow combined with PNS gives the best performance in terms of total capacity consumption.

We also propose a p-tree link protection scheme which provides 100% restorability for single-link failure in a 2-connected network. In this part, we give an ILP formulation to determine the optimal p-tree and the spare capacity allocation on the p-tree such that the total spare capacity requirement is minimized. By local message signalling, p-tree can be easily maintained with the changes of network components(links or nodes). With this property of p-tree, we propose a distributed algorithm to handle double link failure. Numerical results show that our scheme achieves fast restoration speed since the secondary backup path for the second failed link is not required in a majority of double link failures. In addition, our algorithm achieves high

restorability for double-link failures when the spare capacity is planned for single link failure, and it achieves almost 100% restorability for double-link failures when spare capacity is sufficient large.

REFERENCES

[1] B.Mukherjee, Optical Communication Networks, *New York: McGraw-Hill,* July 1997.

[2] M. Kodialam, T. V. Lakshman, Dynamic Routing of Bandwidth Guaranteed Tunnels with Restoration, *Proc. of IEEE INFOCOM'00,* pages 902-911, 2000.

[3] S. Sengupta and R. Ramamurthy, Capacity efficient distributed routing of mesh-restored lightpaths in optical networks, *Proc. IEEE GLOBECOM,* pages 2129-2133, 2001.

[4] G. Mohan, C. S. R. Murthy, and A. K. Somani, Efficient algorithms for routing dependable connections in WDM optical networks, *IEEE/ACM Transactions on Networking,* Vol. 9, No. 10, pages 553-566, October 2001.

[5] C. Xin, Y. Ye, S. Dixit, and C. Qiao. A joint working and protection path selection approach in WDM optical networks. *Proc. of IEEE GLOBECOM'01,* pages 2165-2168, 2001.

[6] H. Luo and L. Ruan, Load Balancing Heuristics for Dynamic Establishment of Restorable Lightpaths, *Proc. of ICCCN'02,* pages 472-477, October 2002.

[7] G. Li, D. Wang, C. Kalmanek, and R. Doverspike. Efficient distributed path selection for shared restoration connections, *Proc. of IEEE INFOCOM'02,* pages 140-149, June 2002.

[8] C. Qiao and D. Xu. Distributed Partial Information Management (DPIM) Schemes for Survivable Networks - Part I. *Proc. of IEEE INFOCOM'02,* pages 23-27, 2002.

[9] Pin-Han Ho, J. Tapolcai, and H. T. Mouftah. Diverse routing for shared protection in survivable optical networks. *Proc. of IEEE GLOBECOM'03,* pages 2519-2523, December 2003.

[10] X. Su and C.-F. Su. An online distributed protection algorithm in WDM networks. *Proc. of IEEE ICC'01*, pages 1571-1575, 2001.

[11] M. Kodialam and T. V. Lakshman. Dynamic routing of locally restorable bandwidth guaranteed tunnels using aggregated link usage information. *Proc. of IEEE INFOCOM'01,* pages 376-385, 2001.

[12] W.D. Grover, D. Stamatelakis, Cycle-oriented distributed preconfiguration: ring-like speed with mesh-like capacity for self-planning network restoration, *Proc. of IEEE ICC 1998,* pages 537-543, June 1998.

[13] W.D. Grover, D. Stamatelakis, Bridging the ring-mesh dichotomy with p-cycles,*Proc. of DRCN Workshop 2000,* 2000.

[14] D.A. Schupke, C.G. Gruber, and A. Autenrieth, Optimal configuration of p-cycles in WDM networks, *Proc. of IEEE ICC 2002,* pages 2761-2765, April 2002.

[15] J. Doucette, D. He, W. D. Grover, O. Yang, Algorithmic Approaches for Efficient Enumeration of Candidate p-Cycles and Capacitated p-Cycle Network Design, *Proceedings of the Fourth International Workshop on the Design of Reliable Communication Networks (DRCN 2003),* pages 212-220, October 2003.

[16] D.A. Schupke, An ILP for Optimal p-Cycle Selection without Cycle Enumeration, *Proc. of the Eighth Working Conference on Optical Network Design and Modelling (ONDM),* February 2004.

[17] W. D. Grover, J. E. Doucette, Advances in Optical Network Design with p-Cycles: Joint optimization and pre-selection of candidate p-cycles, *Proc. of the IEEE-LEOS Summer Topical Meeting on All Optical Networking,* pages WA2-49-WA2-50, July 2002.

[18] C.G. Gruber, Resilient Networks With Non-Simple p-Cycles, *Proc. of International Conference on Telecommunications (ICT 2003),* February 2003.

[19] C. Mauz, p-cycle Protection in Wavelength Routed Networks, *Proc. of the Seventh Working Conference on Optical Network Design and Modelling (ONDM'03)*, February 2003.

[20] H. Zhang, O. Yang, Finding protection cycles in DWDM networks, *Proc. of IEEE ICC*, pages 2756-2760, April/May 2002.

[21] T. H. Wu "Emerging technologies for fiber network survivability", *IEEE Communications Magazine*, Vol. 33, No. 2, pages 58-74, February 1995.

[22] S. Ramamurthy, Biswanath Mukherjee "Survivable WDM Mesh Networks, Part I-Protection", *IEEE Infocom '99*, New York, March 1999.

[23] S. Ramamurthy, Biswanath Mukherjee "Survivable WDM Mesh Networks, Part II-Restoration", *ICC '99*, Vancouver, CA, June 1999.

[24] S. Ramamurthy, Laxman Sahasrabuddhe "Survivable WDM Mesh networks", *Journal of Lightwave Technology*, Vol. 21, No. 4, April 2003.

[25] J. A. Bondy and U. S. R. Murty "Graph Theory with Applications", *American Elsevier Publishing*, 1976.

[26] S. S. Lumetta, M. Medard "Towards a deeper understanding of link restoration algorithms for mesh networks", *Proc. of IEEE INFOCOM 2001*, Anchorage, AL, USA, Vol.1, pages 367-375, April 2001.

[27] D. Zappala, A. Fabbri, V.lo "An evaluation of shared multicast tree with multiple cores", *Telecommunication Systems*, vol.19, no.3/4, pages 461-479, 2002.

[28] Y. Zhang, O. Yang "A distributed tree algorithm for WDM network protection/restoration", *Proc. of IEEE International Conference on High Speed Networks and Multimedia Communications HSNMC'02*, Jeju Island, Korea, pages 289-294, July 2002.

[29] M. Medard, et al. "Redundant tress for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs", *IEEE/ACM Transactions on networking*, Vol. 7, No. 5, October 1999.

[30] Y. Zhang, O. Yang "Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs", *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pages 641-651, October 1999.

[31] Shahram Shah-Heydari, Oliver Yang "Hierarchical Protection Tree Scheme for Failure Recovery in Mesh Networks", *Photonic Network Communications*, 7:2, pages 145-159, March 2004.

[32] W. D. Grover "Mesh-based Survivable Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking", *Prentice Hall, Upper Saddle River, New Jersey, 2003*, chapter 8, pages 529.

[33] H. Choi, S. Subramaniam and H.A. Choi "On Double-Link Failure Recovery in WDM Optical Networks", *IEEE INFOCOM 2002*, 2002.

[34] W. He, M. Sridharan and A. K. Somani "Capacity optimization for surviving double-link failures in mesh-restorable optical networks", *Proc. of SPIE - Volume 4874* , OptiComm 2002: Optical Networking and Communications, pages 13-24, July 2002.

[35] D. A. Schupke "The tradeoff between the number of deployed p-cycles and the survivability to dual fiber duct failures", *Proc. of ICC 2003*, pages 1428-1432, 2003.

[36] D. A. Schupke, W. D. Grover and M. Clouqueur "Strategies for Enhanced Dual Failure Restorability with Static or Reconfigurable p-Cycle Networks", *International Conference on Communication ICC 2004* , Paris, France, July 2004.

[37] D. A. Schupke, A. Autenrieth, and T. Fischer "Survivability of multiple fiber duct failures", *Proc. of DRCN Workshop 2001*, 2001.

[38] D. A. Schupke "Multiple failure survivability in WDM networks with p-cycles ", *Circuits and Systems, ISCAS '03*, Volume: 3, pages III-866 - III-869 vol.3, 25-28 May 2003.

[39] Iraschko, R.R.; MacGregor, M.H.; Grover, W.D. "Optimal capacity placement for path restoration in STM or ATM mesh-survivable networks ", *IEEE/ACM Transactions on Networking*, Volume: 6 , Issue: 3, pages 325 - 336, June 1998.

ACKNOWLEDGMENTS

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, I am indebted to my major professor in my graduate career: Dr. Lu Ruan. I am grateful to her for her guidance, patience, and support throughout this research. I learned many things (research, writing, cooperation skills, etc.) from Dr.Ruan, whose vision and impeccable insights have been and will continue to be an inspiration to me.

I would also like to thank my committee member, Dr. Ahmed Kamal, for his valuable comments and suggestions such as the improvement of ILP formulation. I appreciate Dr. Jonny Wong for his help and serving as my POS members.

I would like to thank other members in my research group, Chang Liu and Zhi Liu. Without them, this work would not have been completed as smoothly and successfully as it has been.

Finally, I would also like to thank my family for the support they provided me through my entire life and in particular, I appreciate the love and support provided by my girlfriend, Ivy, in the past years.