2010

Survivability schemes for optical backbone and access networks

Taiming Feng
Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/etd
Part of the Computer Sciences Commons

Recommended Citation
Feng, Taiming, "Survivability schemes for optical backbone and access networks" (2010). Graduate Theses and Dissertations. Paper 11698.

This Dissertation is brought to you for free and open access by the Graduate College at Digital Repository @ Iowa State University. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Digital Repository @ Iowa State University. For more information, please contact hinefuku@iastate.edu.
Survivability schemes for optical backbone and access networks

by

Taiming Feng

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

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Lu Ruan, Co-major Professor
Wensheng Zhang, Co-major Professor
David Fernández-Baca
Ahmed E. Kamal
Johnny S. Wong

Iowa State University
Ames, Iowa
2010

Copyright © Taiming Feng, 2010. All rights reserved.
DEDICATION

I would like to dedicate this thesis to my parents and my wife Hongmei Yuan. Without their support, I would not have been able to complete this work.
**TABLE OF CONTENTS**

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xi</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Survivability in WDM Optical Backbone Networks</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 Survivability Design for Double Link Failures in Unicast Communication Mode</td>
<td>3</td>
</tr>
<tr>
<td>1.1.3 Survivability Design for Single Link Failure in Multicast Communication Mode</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Survivability in Optical Access Networks</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Outline of the Dissertation</td>
<td>7</td>
</tr>
<tr>
<td>CHAPTER 2. REVIEW OF LITERATURE</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Survivability Schemes for Unicast Sessions in WDM Optical Backbone Networks</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Survivability Schemes for Multicast Sessions in WDM Optical Backbone Networks</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Survivability Schemes in Access Networks</td>
<td>12</td>
</tr>
<tr>
<td>CHAPTER 3. TWO-LINK FAILURE PROTECTION IN WDM MESH NETWORKS WITH $p$-CYCLES</td>
<td>14</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>14</td>
</tr>
<tr>
<td>3.2 Preliminaries</td>
<td>14</td>
</tr>
<tr>
<td>3.3 An ILP Model for Static Traffic Protection</td>
<td>17</td>
</tr>
<tr>
<td>3.4 Protection Schemes for Dynamic Traffic</td>
<td>20</td>
</tr>
<tr>
<td>3.4.1 Shortest Path Pair Protection Scheme</td>
<td>21</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Shortest Full Path Protection Scheme</td>
</tr>
<tr>
<td>3.5</td>
<td>Numerical Results</td>
</tr>
<tr>
<td>3.5.1</td>
<td>ILP Results for Static Traffic</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Comparison of SPPP and SFPP</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Comparison of SPPP and the Algorithms in [4]</td>
</tr>
<tr>
<td>3.6</td>
<td>Conclusion</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.2</td>
<td>The Hybrid Protection/Restoration Scheme for Two-Link Failure</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Backup Wavelength Reservation Scheme</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Dynamic Restoration Scheme for Nonsurvivable Demands</td>
</tr>
<tr>
<td>4.3</td>
<td>Numerical Results</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Results for the Backup Wavelength Reservation Scheme</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Results for the Dynamic Restoration Scheme</td>
</tr>
<tr>
<td>4.4</td>
<td>Conclusion</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>5.2</td>
<td>Overview of the IpcC Scheme</td>
</tr>
<tr>
<td>5.3</td>
<td>Finding New p-Cycles</td>
</tr>
<tr>
<td>5.4</td>
<td>Extending Existing p-Cycles</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Updating the p-Cycle Set PC</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Connection Release</td>
</tr>
<tr>
<td>5.5</td>
<td>Numerical Results</td>
</tr>
<tr>
<td>5.6</td>
<td>Conclusion</td>
</tr>
<tr>
<td>6.1</td>
<td>p-Cycle-Based Path Protection for Multicast Session in WDM Networks</td>
</tr>
</tbody>
</table>
6.1 Introduction .................................................. 68
6.2 Problem Statement ............................................ 69
   6.2.1 Problem Definition ..................................... 69
   6.2.2 Protection Strategies ................................... 70
6.3 p-Cycle-based Path Protection ($P^3$) Scheme .................... 72
   6.3.1 Overview of the $P^3$ Scheme ......................... 72
   6.3.2 Reusing Existing p-Cycles ............................ 72
   6.3.3 Computing New p-Cycles ............................. 73
6.4 Simulation Results .......................................... 76
6.5 Conclusion .................................................. 79

CHAPTER 7. PXT-BASED PATH PROTECTION FOR MULTICAST SESSIONS IN WDM NETWORKS .................................................... 80
7.1 Introduction .................................................. 80
7.2 Basic Idea ................................................... 80
7.3 PXT-Based Path Protection Scheme ................................ 82
   7.3.1 Overview of the scheme ............................... 82
   7.3.2 Reusing Existing PXTs ................................. 83
   7.3.3 Computing and Merging New PXTs .................... 85
7.4 Performance Evaluation ....................................... 87
   7.4.1 Performance of PXT Scheme .......................... 87
   7.4.2 Comparison of PXT and $P^3$ Schemes ................. 90
7.5 Conclusion .................................................. 90

CHAPTER 8. DESIGN OF SURVIVABLE HYBRID WIRELESS-OPTICAL BROADBAND-ACCESS NETWORK ............................................. 92
8.1 Introduction .................................................. 92
8.2 Protection Scheme and Problem Statement ....................... 92
   8.2.1 Protection Scheme .................................... 92
   8.2.2 Problem Statement .................................... 93
8.3 Solution Approach to the MPMC Problem ........................................ 94
  8.3.1 Construction of Graph $G$ .................................................. 95
  8.3.2 An ILP for the MCMF Problem ......................................... 96
8.4 NP-Hard Proof ................................................................. 98
8.5 A Heuristic Algorithm ....................................................... 99
8.6 Numerical Results ............................................................. 101
8.7 Conclusion ................................................................. 105

CHAPTER 9. CONCLUSIONS AND FUTURE WORK ................................. 107
  9.1 Contributions of this Work ................................................. 107
  9.2 Future Works .............................................................. 108

ACKNOWLEDGEMENTS ............................................................... 110

BIBLIOGRAPHY ................................................................. 111
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Redundancy and Computation time of ILP</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Comparison of Algorithms</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>Redundancy of DPP, SPP, and Hybrid in DISTRIBUTED network</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Average Percentage of Unaffected, Survivable, and Nonsurvivable Demands in DISTRIBUTED network</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Comparison of Average Backup Path Length</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>Computation Time(ms) under different traffic load in NSF</td>
<td>65</td>
</tr>
<tr>
<td>5.2</td>
<td>Computation Time under different traffic load in COST239</td>
<td>66</td>
</tr>
<tr>
<td>6.1</td>
<td>Redundancy comparison of $P^3$ and $IpC$</td>
<td>78</td>
</tr>
<tr>
<td>7.1</td>
<td>Comparison of redundancy in two networks</td>
<td>89</td>
</tr>
<tr>
<td>7.2</td>
<td>Average number of protected destinations per PXT in PXT scheme</td>
<td>90</td>
</tr>
<tr>
<td>7.3</td>
<td>Comparison of redundancy in 5 networks</td>
<td>91</td>
</tr>
<tr>
<td>8.1</td>
<td>Optimal solutions to different instances of the MPMC problem.</td>
<td>101</td>
</tr>
<tr>
<td>8.2</td>
<td>Heuristic solutions to different instances of the MPMC problem.</td>
<td>104</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Architecture of Wireless WDM-PON</td>
<td>6</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Two-Link Failure Protection for Link $A \rightarrow D$</td>
<td>15</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>$p$-Cycle Sharing in Two-Link Failure Protection</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>$p$-Cycles Used in SPPP Scheme</td>
<td>21</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>$p$-Cycles used in SFPP Scheme</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>The 6-node 11-edge network</td>
<td>29</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Two Test Networks</td>
<td>30</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>Wavelength usage of SPPP and SFPP in SMALLNET</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>Wavelength usage of SPPP and SFPP in COST239</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3.9</td>
<td>Protection redundancy of SPPP and SFPP in SMALLNET</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3.10</td>
<td>Protection redundancy of SPPP and SFPP in COST239</td>
<td>32</td>
</tr>
<tr>
<td>Figure 3.11</td>
<td>Reject ratio of SPPP and SFPP in SMALLNET</td>
<td>33</td>
</tr>
<tr>
<td>Figure 3.12</td>
<td>Reject ratio of SPPP and SFPP in COST239</td>
<td>33</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>An example network with three demands: AD, BC, and GH. Working paths are shown in dotted lines. Backup paths are shown in dashed lines.</td>
<td>39</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>A nonsurvivable demand affected by the failures of link $l_1 = (a, b)$ and link $l_2 = (x, y)$. Traffic between $s$ and $t$ can be restored by finding a feasible path between $x$ and $y$ (dashed line).</td>
<td>40</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Number of backup wavelengths used by DPP, SPP, and Hybrid in DISTRIBUTED network</td>
<td>42</td>
</tr>
</tbody>
</table>
Figure 4.4 Restoration ratio of local and end-to-end restoration schemes under limited capacity. ........................................ 44

Figure 5.1 Example of Finding New \( p \)-Cycles ........................................ 54
Figure 5.2 Extend an existing \( p \)-cycle to protect link \( e \). ................................. 54
Figure 5.3 Combining two \( p \)-cycles with one or more common edges. ................. 59
Figure 5.4 Combining two \( p \)-cycles with two common nodes. .......................... 59
Figure 5.5 Topology of NSF Network ................................................................. 60
Figure 5.6 Topology of COST239 Network .......................................................... 61
Figure 5.7 Wavelength Usage of IpC and DpC in NSF Network ......................... 62
Figure 5.8 Wavelength Usage of IpC and DpC in COST239 Network ................... 62
Figure 5.9 Reject Ratio of IpC and DpC in NSF Network .................................. 64
Figure 5.10 Reject Ratio of IpC and DpC in COST Network ............................... 64
Figure 6.1 (a) Tree-disjoint protection strategy. (b) Path-disjoint protection strategy. ... 69
Figure 6.2 (a) Two destination nodes \( d_1 \) and \( d_2 \) can share a \( p \)-cycle if their tree paths are link-disjoint. (b) Two destination nodes \( d_1 \) and \( d_2 \) can share a \( p \)-cycle if their protection segments are link-disjoint. ....................... 74
Figure 6.3 SMALLNET Network ................................................................. 76
Figure 6.4 Number of wavelength channels used versus number of multicast sessions in SMALLNET network. ................................................................. 77
Figure 6.5 Number of wavelength channels used versus number of multicast sessions in COST239 network. ................................................................. 77
Figure 7.1 A PXT from \( d_1 \) to \( d_3 \) that can be used to protect \( d_3 \). \( d_1 \) is a guard-node of \( d_3 \). 81
Figure 7.2 Two multicast sessions are shown in (a) and (b). Multicast trees are shown in red. The PXT \( 1 \rightarrow 3 \) can protect destination nodes \( d_2 \) (in session 1) and \( d_4 \) (in session 2) simultaneously since the working paths of the two nodes are link-disjoint. ........................................ 84
Figure 7.3 The PXT $1 \rightarrow 3 \rightarrow 0 \rightarrow 6 \rightarrow 4$ can protect destination nodes $d_2$ (in session 1) and $d_3$ (in session 2) simultaneously since the protection segments of the two nodes are link-disjoint.

Figure 7.4 Two cases of merging. New PXT is in red, existing PXT is in blue, and merged PXT is in green.

Figure 7.5 Merging two PXTs. (a) Request 1 with source $s_1$ and destinations $d_1$ and $d_2$.
(b) Request 2 with source $s_1$ and destinations $d_3$ and $d_4$.

Figure 7.6 Comparison of protection wavelength channels used in COST239 network.

Figure 7.7 Comparison of protection wavelength channels used in NSF network.

Figure 8.1 (a) An instance of the MPMC problem. (b) Graph $G$ constructed from the instance in (a).

Figure 8.2 An instance of Decision-MCPMP constructed from a Subset-Sum instance $\langle S = \{1, 2, 5, 9\}, 8 \rangle$. The link set $LS = \{(1, 5), (2, 5), (3, 5)\}$ has cost 8 and protects each segment $i$ with capacity $D(i)$.

Figure 8.3 The optimal solutions for two instances with $|V| = 20$. Left: demand is 5-fixed. Right: demand is 6-fixed.

Figure 8.4 Comparing optimal and heuristic solutions, 10-node instances.

Figure 8.5 Comparing optimal and heuristic solutions, 20-node instances.
Network survivability, reflecting the ability of a network to maintain an acceptable level of service during and after failures, is an important requirement for WDM optical networks due to the ultra-high capacity. The most common network failure is the link failure which could cause enormous data loss and lots of service disruption to Internet users. Although single-link failures are the most common failure scenarios, double-link failures can occur in some cases and cause more severe problem. Compared to unicast sessions, multicast sessions suffer more seriously from link failures because a link may carry traffic to multiple destinations rather than to a single destination. Hence, multicast sessions demand more effective and efficient protection against link failures. With the increasing demand for access bandwidth, the access networks draw more attention. The hybrid wireless-optical broadband-access network (WOBAN) is a promising architecture for future access networks because it combines the high capacity of optical communication and the flexibility and cost-effectiveness of a wireless network.

First, we consider the problem of protecting unicast connections against double link failures. The basic idea is to use two \( p \)-Cycles, with link-disjoint protection segments, to protect each working link. To utilize spare capacity more efficiently, we also propose a new hybrid protection/restoration scheme to handle two-link failures. Our scheme uses protection to ensure that most of the affected demands can be restored using the pre-planned backup paths upon a two-link failure. For the demands not restorable with protection, we use dynamic restoration to find new backup paths for them.

Second, we propose protection schemes for multicast sessions under one link failure. An intelligent \( p \)-Cycle (IpC) scheme is presented to provide \( p \)-Cycle protection for dynamic multicast sessions. When a multicast request arrives, a multicast tree is computed for it and then the IpC scheme is used to compute a set of high efficient \( p \)-Cycles on-demand to protect each link on the multicast tree. Then we
propose a p-cycle-based path protection scheme and a PXT-based path protection scheme to provide protection for dynamic multicast sessions. Basically, to protect a multicast tree, we compute one p-Cycle and one PXT for each destination node \( v \) such that the p-Cycle and the PXT can be used to restore the traffic to \( v \) when a link failure occurs on the path from the source node to \( v \).

Finally, we propose a new protection scheme for the hybrid wireless-optical broadband-access network (WOBAN). The scheme is cost-effective in that it does not require the PONs to have self-protecting capability. Based on the proposed protection scheme, we define the maximum protection with minimum cost (MPMC) problem and present one ILP solution approach to the MPMC problem. Then we prove the MPMC problem is NP-Hard and provide one heuristic algorithm for the MPMC problem.
CHAPTER 1. INTRODUCTION

The explosive growth of web based services over the Internet results in an tremendous growth in the demand for bandwidth in backbone networks and access networks. The fiber optic medium is the only one capable of providing high-bandwidth service cost-effectively and it is also less susceptible to electromagnetic interferences. Optical fibers are widely deployed in backbone networks, metropolitan and access networks. Wavelength-division multiplexing (WDM) is a technology that multiplexes multiple optical carrier signals on a single optical fibre by using different wavelengths (colours) of laser light to carry different signals and WDM optical networks are widely deployed to meet the ever increasing bandwidth demand of network users and applications. Because of the nature of large bandwidth traffic transported by WDM networks, any failure such as a fibre cut would cause enormous data loss and huge service disruption to a large number of users. Thus, survivability is a critical issue in WDM optical networks as customers require high service availability despite inevitable network element failures.

In this chapter, we first discuss some research challenges on survivability in WDM optical networks. Specifically, survivability design for unicast and multicast communication modes will be discussed. Then, we discuss the challenging survivability issues in access networks.

1.1 Survivability in WDM Optical Backbone Networks

1.1.1 Background

Compared to copper cables, optical fiber communication systems provide a tremendous bandwidth which satisfies the greatly increasing demand requirement of internet users. Wavelength-division multiplexing (WDM) is a technology which multiplexes multiple optical carrier signals onto one single optical fiber and a typical WDM link consists of a set of transmitters, optical amplifiers and receivers. The laser signals from different transmitters are multiplexed together by the multiplexer and sent to the
receivers. During the transmission, the signals need to be amplified by the optical amplifiers because of signal attenuation. At the destination, the incoming multiplexed signal is de-multiplexed into different wavelengths. Although the per-channel light signals propagating in the fiber is typically modulated at rates as 10 or 40 Gb/s in deployed backbone networks, the current laboratory fiber optic data rate record is multiplexing 155 channels, each carrying 100 Gbps over a 7000 km fiber.

In optical networks, fiber cuts are considered as the most common failures and link failures will affect a large number of communication sessions due to the huge bandwidth provided by a fiber. Therefore, it is important to design a survivable network which can protect communication sessions against link failures.

Survivable network architectures are based either on restoration or on protection[1]. Among these two type of schemes, the restoration tries to allocate spare resource to restore the communication after link failure is identified. Considering the huge data transmitted in the fiber and the long time for spare resource allocation, this scheme is not preferred. Moreover, the restoration could fail if no enough idle resource could be found in the network. On the contrary, protection schemes need to reserve the backup path together with the working path setup. Thus dedicated-resource protection has a faster restoration time and provides guarantees on the restoration ability and there are two protection methods: one is link-protection and the other is path-protection.

In link-protection, each link in the communication session is protected by a backup path. Once some link fails, the protection switches the end nodes of the failed link to protection states which will reroute the affected sessions over these backup routes. In detail, there are dedicated link protection and shared link protection. For dedicated link protection, the restoration route could be pre-connected and thus only these two nodes adjacent to the failed link need to take action. On the contrary, in the shared link protection mechanism, the restoration route can not be pre-configured because these protection capacity could be shared by many working sessions and the right connection must be set up after the link failure is identified. So shared link protection is more capacity efficient with more restoration time.

In path-protection with dedicated protection capacity, an end-to-end backup path, disjoint with the primary working path, is setup between the source and the destination node. Once failure happens, the failure information will be sent from the end nodes of the down link to the source and the destination
nodes. Thus, the source and the destination nodes will switch the traffic to the pre-configured backup path. While for the path-protection with shared protection capacity, these protection capacity can be shared and the restoration route have to be singalled and the right connections has to be set up after link failure happens. Thus shared path protection is the most capacity efficient scheme and it needs the longest restoration time.

$p$-Cycle is a promising protection technique which configures the spare capacity into pre-cross-connected cycles. Upon a link failure, protection switching is performed at the two end nodes of the failed link. Therefore, traffic restoration is extremely fast. Moreover, $p$-Cycle is also efficient in protection since it protects both on-cycle links and straddling links. Thus, $p$-Cycle can achieve two most important criteria simultaneously in survivability scheme design: fast restoration and high capacity-efficiency. Chow et al. noted in [2] that rings and p-cycles can achieve fast restoration because they provide pre-cross-connected protection paths. Based on this observation, they proposed the concept of pre-cross-connected trail (PXT). A trail is an alternating sequence of nodes and links $(v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n)$ such that for all $i$, the end nodes of $e_i$ are $v_{i-1}$ and $v_i$. A PXT is implemented by pre-cross-connecting one wavelength in each link along the trail. $p$-Cycles are special cases of PXTs where $v_0 = v_n$. Like rings and p-cycles, PXTs can provide fast restoration because they are pre-cross-connected. A similar protection scheme, called Streams, is introduced in [3].

1.1.2 Survivability Design for Double Link Failures in Unicast Communication Mode

Link failures are the dominant type of failures in WDM networks. Although single-link failures are the most common failure scenarios, double-link failure can occur in some cases. First, after a link fails, a second link may fail while the first link is being repaired. Second, two fiber links may be physically routed together for some distance and a backhoe accident may lead to the failures of both links [4]. Third, if an optical switch with two links connected to it fails, then both links fail. In this dissertation, a set of $p$-Cycle based protection schemes for two-link failures are proposed[5]. We formulate an ILP model for the $p$-Cycle design problem for static traffic. We also propose two protection schemes for dynamic traffic, namely SPPP (Shortest Path Pair Protection) and SFPP(Short Full Path Protection). Simulation results show that SFPP is more capacity efficient than SPPP under
incremental traffic. Under dynamic traffic, SPPP has lower blocking than SFPP when the traffic load is low and has higher blocking than SFPP when the traffic load is high.

We also proposed a new hybrid protection/restoration scheme to handle two-link failures. Unlike existing protection schemes that require two link-disjoint backup paths for each demand or link, our proposed scheme only requires one backup path for each demand which leads to significant saving in backup capacity. Unlike backup reprovisioning schemes, our scheme computes new backup paths for unprotected demands after the second failure occurs so that unnecessary reprovisioning is avoided.

1.1.3 Survivability Design for Single Link Failure in Multicast Communication Mode

Different to the unicast request, which has only one sender and one receiver, multicast requests normally have one source and multiple destinations. Multicasting consists of concurrently sending the same data from a source to a group of destinations in a computer or communication network [6] and it is an effective mechanism for supporting group communication. In a multicast communication, each sender transmits only one copy of each message that is replicated within the network and delivered to multiple receivers. For this reason, multicasting typically requires less total bandwidth than separately uni-casting messages to each receiver [7].

Upon the arrival of a multicast request, a unidirectional primary multicast tree is first computed and it connects the source node to all the destination nodes[8], [9], [10]. Then, backup resources are reserved for the primary tree to protect it against single link failures. In multicast applications, the failure of one link might affect the data traffic to multiple destinations; hence multicast sessions require more effective and efficient survivability protection upon link failures.

Although dedicated protection needs the minimal restoration time, the dedicated protection has low bandwidth efficiency. For example, if link-disjoint protection trees are used to provide dedicated protection, the protection redundancy will be at least 100%. Even bandwidth efficiency can be achieved through resource sharing, additional time for cross-connection at all nodes on the restoration path is needed after a link failure happens.

In this dissertation, we identify and address the challenges in applying \( p \)-Cycles for multicast session protection, and develop an intelligent \( p \)-Cycle (IpC) scheme, which forms \( p \)-Cycles gradually
according to dynamic multicast requests, and provides the protection for every link on multicast trees. Extensive simulations have been conducted to evaluate our \( P^3 \) scheme, and the results show that it outperforms existing solutions. We also propose a \( P^3 \)-based path protection scheme and a PXT-based path protection method for dynamic multicast sessions. These path-based approaches are more efficient than the traditional link-based approaches.

### 1.2 Survivability in Optical Access Networks

Passive optical network (PON) is a promising technology for broadband access as it can offer higher bandwidth to end users than other alternatives such as DSL and cable TV networks. The PON is point-to-multipoints and generally there is a single transceiver in the optical line terminal (OLT) in the central office (CO). The OLT sends information to the optical network units (ONUs) located at the subscriber end. Traditional PONs are time division multiplexing PONs (TDM-PONs), in which a single wavelength is used for all downstream transmissions and another wavelength is used for all upstream transmissions. The upstream bandwidth is shared among the users in the manner of time division multiplexing. Various TDM-PON technologies have been developed, including ATM PON (APON), Broadband PON (BPON), Gigabit PON (GPON), and Ethernet PON (EPON). As end users demand more bandwidth, there is the need to further increase the PON bandwidth using wavelength division multiplexing (WDM).

As end users demand more bandwidth, there is the need to further increase the PON bandwidth using wavelength division multiplexing (WDM). In a WDM-PON[11, 12], ONUs are assigned individual wavelengths so that each ONU can operate up to the full bit rate of a wavelength channel. WDM-PON also provides bit rate independence, protocol transparency, and excellent security and privacy.

Wireless mesh network (WMN) [13] is another promising technology for broadband access due to its low cost, ease of deployment, increased coverage, and robustness. A WMN consists of a collection of wireless routers, a few of which have wired connections to the Internet and are called the gateways. The wireless routers in a WMN form a wireless backbone to provide multi-hop connectivity between the clients and the gateways.

Recently, the hybrid wireless-optical broadband-access network (WOBAN) is presented in [14] as a
promising architecture for future access networks. The key advantage of WOBAN is that it captures the best of both the optical and wireless worlds: the reliability and high capacity of optical communication and the flexibility and cost-effectiveness of a wireless network. A WOBAN is comprised of a number of segments each containing a WMN at the front end and a PON at the back end. In a WOBAN segment, each ONU of the PON is connected to a wireless gateway in the WMN so that users within the coverage area of the WMN are connected to the CO via the WMN and the PON. Fig. 1.1 shows a WOBAN with two segments.

Although a lot of works have been done for the survivable optical backbone networks, the research on the survivability of access network is far from enough. Once one fiber is cut, especially when the fiber between the OLT and the RN is down, the damage to the network is huge: all customers connected to the RN will be affected.

In this dissertation, we propose a cost effective protection method for WOBAN that deals with network element failures in the optical part of WOBAN. We define the maximum protection with minimum cost (MPMC) problem and show that the problem can be converted to the minimum cost
maximum flow (MCMF) problem. We also present an ILP model for the MCMF problem. After proving MPMC is NP-Hard, we present a heuristic algorithm for MPMC. Numerical results are reported for applying our ILP model to obtain the optimal solutions for different instances of the MPMC problem and the heuristic solutions are close to optimal solutions.

1.3 Outline of the Dissertation

The rest of this dissertation is organized as follows:

In chapter 2, we provides the literature review of survivability schemes for optical backbone and access networks.

For the survivability design in optical backbone networks, chapter 3 considers the problem of protecting connections against two simultaneous link failures and proposes one ILP Scheme for static traffic and two heuristic schemes for dynamic traffic[5]. In this collaborative work with Long Long, my major contributions include the theoretical analysis of protection conditions, the design and implementation of two heuristic algorithms for dynamic traffic.

Chapter 4 proposes a new hybrid protection/restoration scheme to handle two-link failures[16]. Our hybrid scheme uses protection to ensure that most of the affected demands can be restored using the pre-planned backup paths upon a two-link failure. For the demands not restorable with protection, we use dynamic restoration to find new backup paths for them.

In chapter 5, we propose one $p$-Cycle based link protection scheme for dynamic multicast sessions under one link failure[17]. After computing a multicast tree for a multicast request, the IpC scheme finds the most efficient p-cycles until all links on the multicast tree are protected.

In Chapter 6, we propose a p-cycle-based path protection($P^3$) scheme for dynamic multicast sessions[18]. Given a multicast tree $T$, the $P^3$ scheme uses the path-disjoint strategy to compute a set of p-cycles on-demand to ensure every destination node in $T$ is protected.

In Chapter 7, we propose a PXT-based path protection method for dynamic multicast sessions[19]. To protect a multicast tree, we compute a PXT for each destination node $v$ such that the PXT can be used to restore the traffic to $v$ when a link failure occurs on the path from the source node to $v$. We also compare the performance of the $P^3$ scheme and the PXT based path protection scheme and
conclude that the p-Cycle based protection scheme is more suitable for dense networks according to the simulation results.

Chapter 8 studies the survivability problem in optical access networks [15]. After proposing one protection scheme for WOBAN which is one type of access networks, we define the maximum protection with minimum cost (MPMC) problem and present one optimal ILP solution and one near-optimal heuristic solution for the MPMC problem.

Finally, in Chapter 9, we conclude the dissertation and outline the plan for future research directions.
CHAPTER 2. REVIEW OF LITERATURE

In this chapter, we review the recent work on survivability in both backbone and access networks. First, we will review the unicast protection schemes in WDM optical networks. And then we review the research on survivability schemes for multicast session. At the end of this chapter, we also review some work on protection schemes designed for access networks.

2.1 Survivability Schemes for Unicast Sessions in WDM Optical Backbone Networks

It is important to protect communication sessions against link failures because link failures will affect a large number of communication sessions due to the huge bandwidth provided by a fiber. Various protection schemes have been developed for WDM networks. Ring-based protection schemes enable traffic restoration to be completed in 50-60 ms, but require at least 100% capacity redundancy. As for the path-protection, dedicated-path protection requires the backup path to be exclusively reserved by the primary working path. Some dedicated-path protection schemes are described and evaluated in [20], [21], [22]. Compared with dedicated-path protection, shared-path protection can increase the capacity efficiency[23],[24],[25]. But compared with dedicated-path protection, shared-path protection scheme requires an extra time for cross-connect once a failure occurs.

While many works have studied protection schemes for single-link failures, relatively few works have considered two-link failure scenarios where a second failure occurs before the first failure is repaired. Dual-failure restorability of span-restorable mesh networks designed to ensure 100% single-failure restorability is studied in [26]. Protection schemes for two-link failure are studied in [4, 27, 28]. In the scheme proposed in [4], two link-disjoint backup paths are computed for each link so that the network is two-link failure survivable. The scheme is slow in restoration because the backup paths are configured after link failure occurs. The schemes in [27] are link-based where each link has two
precomputed link-disjoint backup paths. Path-based protection schemes are presented in [28] where two link-disjoint backup paths are precomputed for each demand. All these protection schemes can provide 100% two-link failure restorability due to the use of two link-disjoint backup paths. However, they require a large amount of backup capacity. Furthermore, two link-disjoint backup paths may not exist for some demands/links in the network.

Since p-Cycle was first proposed in [29], it has been widely used in protection schemes against single link failures [29], [30], [31], [32], [33], [34], [35], [36], [37], [38]. The relation between the number of deployed p-Cycles and the ability to survive dual fiber duct failures is studied in [39, 40], but the schemes are not specially designed for double-link failures. In [41], a p-cycle based scheme for double-link failure protection is proposed where p-cycles are reconfigured based on the remaining spare capacity after a link failure occurs and the corresponding working paths are rerouted. This scheme cannot deal with simultaneous two-link failures. In [42], a p-cycle based multi-QoP (quality of protection) framework with five QoP service classes is proposed, where the platinum class is assured protection from all two-link failures. The protection for a platinum demand is achieved by routing it entirely over straddling links. There are also some work addressing multiple-link failure protection. The authors of [43] proposed algorithms to find $k$ disjoint p-cycles to protect each link such that the network is $k$ link-failure survivable. The author of [44] extended his work in [30] to protect multiple-link failures by using network coding and p-cycles.

Another approach to handling two-link failures is reprovisioning/reconfiguration after the first failure (RAFF) [45, 46]. In RAFF, each demand is assigned backup capacity along a backup path so that it is protected against single link failure. When the first failure occurs, affected demands are restored using the preplanned backup paths. After restoration from the first failure is complete, new backup paths are reprovisioned for those demands that may be unrecoverable using the preplanned backup capacity. This allows the affected demands to be restored quickly using the new backup paths when the second failure occurs. In [47], two backup reprovisioning schemes named MBR and GBR are proposed. In MBR, after a failure occurs, new backup paths are reprovisioned for connections that become unprotected (due to loss of primary or backup) or vulnerable (due to backup capacity sharing). In GBR, backup paths are globally rearranged for all connections after one failure occurs. Reference
[40] applies the concept of RAFF in p-cycle networks where the spare capacity can be reconfigured dynamically after the first failure to create a new set of p-cycles optimized to withstand possible second link failures. ILP models are given for two cases: complete cycle reconfiguration and incremental cycle configuration.

2.2 Survivability Schemes for Multicast Sessions in WDM Optical Backbone Networks

Researchers have proposed various multicast tree protection schemes, including tree-based [48, 49, 50, 51], ring-based [52], link-based[48], segment-based [48, 53], and path-based [48, 52, 54] schemes.

In tree-based schemes, a primary tree is protected by either a link-disjoint backup tree or an arc-disjoint backup tree. In the former case, if the primary tree uses link $u \rightarrow v$, then the backup tree can use neither link $u \rightarrow v$ nor link $v \rightarrow u$. In the latter case, however, the backup tree is allowed to use link $v \rightarrow u$, but not link $u \rightarrow v$. The drawbacks of tree-based schemes include excessive use of network resources and unavailability of link/arc-disjoint trees in some cases. The ring-based schemes are dedicated protection schemes which lead to minimal restoration time. However, their spare capacity requirement is high.

In segment-based schemes, each segment in the primary tree is protected by a path that is link-disjoint with the segment. Here, a segment is defined as the sequence of edges from the source or a splitting node on the tree to a leaf node or a downstream splitting node [48]. In path-based schemes, each destination $d_i$ in the multicast session is protected by a backup path that is link-disjoint with the path from $s$ to $d_i$ on the primary tree. Segment-based and path-based schemes are more capacity efficient than tree-based schemes since a backup path can share capacity with the primary tree as well as with the other backup paths. Segment-based and path-based schemes are capacity efficient since a backup path can share capacity with the primary tree as well as with the other backup paths. However, these schemes require long restoration time since some nodes need to reconfigure their switches to set up the backup segment or path when a link failure occurs.

In [55], Integer Linear Program (ILP) methods are proposed for p-cycle based protection of static multicast sessions. In [56], Kodian and Grover propose the concept of failure-independent path-protecting (FIPP) p-cycle, which extends the p-cycle concept to provide end-to-end failure independent path protection. An ILP model is given in [56] to solve the FIPP p-cycle network design problem for a
given set of unicast demands. A heuristic method for FIPP p-cycle design is given in [57]. Variations of FIPP p-cycles are proposed in [58] to provide tree protection (SOPT) and segment protection (SOPS) for multicast sessions. ILP based heuristic algorithms are given to minimize the spare capacity of SOPT and SOPS for a given set of multicast sessions. These ILP based algorithms are time consuming and not suitable for protecting dynamic multicast sessions. In addition, the performance of SOPT and SOPS are worse than p-cycle-based link protection scheme since a subset of the disjoint tree/segment sets, instead of all possible disjoint sets, are used in the ILP models of SOPT/SOPS.

Although many p-cycle based schemes have been proposed for unicast protection [59, 60, 34], applying p-cycles for multicast protection of dynamic traffic has been barely studied. To use p-Cycles to protect a multicast tree against single link failures, every link on the multicast tree should be protected by a p-Cycle. Meanwhile, the p-Cycles used to protect the tree links should consume as few network resources as possible. This results in a challenging problem of finding a set of p-Cycles that can protect all links on the multicast tree and use the minimum number of wavelength channels. This problem has been studied by Zhong et al. in [55, 61]. Specifically, they proposed Integer Linear Program based methods [55] to protect static multicast sessions and the dynamic p-Cycle (DpC) scheme [61], extended from [62], to protect dynamic multicast sessions. The proposed dynamic p-Cycle (DpC) scheme [61], extended from [62], prefers short cycles, which may not always be a good choice because longer cycles may introduce more straddling links and therefore provide better protection efficiency. The DpC scheme chooses p-Cycles from a set of pre-computed candidate cycles, which cannot adapt to dynamic incoming multicast requests.

### 2.3 Survivability Schemes in Access Networks

Various survivable PON architectures have been proposed in literature. For example, [63] proposes two self-protecting architectures for WDM-PON. The first architecture protects FF failures by connecting adjacent remote nodes with a fiber. The second architecture protects both FF failures and DF failures by duplicating the distribution fibers. Both architectures double the wavelength requirement in order to provide protection. In [64], a protection scheme is proposed for hybrid WDM/TDM PONs. The scheme employs protection feeder fibers and fibers interconnecting pairs of ONUs to pro-
vide protection to both FF and DF failures. Unlike the scheme in [63], no additional wavelengths are required for providing protection. [65] proposes a self-survivable WDM-PON architecture that can protect FF/DF failures, RN failures, and failures of transmitters in CO and ONUs. In all these schemes, at least N additional fibers need to be laid in order to protect N ONUs against FF and DF failures. This may result in capital expenditure that is too high for the cost-sensitive access networks. [66] converts the problem of designing survivable access network as a simplex cover problem and claims that once one terminal node is protected once it is connected with some other terminal node. But [66] does not consider the capacity of each terminal node. In fact, it is possible that the protection capacity of one terminal node is limited.

Due to the existence of alternative routes in a mesh network, the front-end WMNs in a WOBAN are self-healing. However, the back-end PONs cannot survive network element failures because a tree topology is used. One way to provide survivability in WOBAN is to employ survivable PON architectures.

The authors of [67] propose an approach to WOBAN survivability that does not require the PONs to have self-protecting capability. The idea is to reroute the traffic around the failure. Specifically, if an ONU in a segment fails, the traffic will be rerouted to another gateway in the same segment that is connected to a live ONU. If an OLT in a segment fails, the traffic will be rerouted to a gateway in another segment that has a live OLT. This scheme assumes that every wireless router in one segment can find a multi-hop path to a gateway in another segment. This assumption is not true when the WMNs in different segments of a WOBAN are separated by a large distance so that no wireless router in one segment can communicate with a wireless router in another segment. In this case, the rerouting scheme proposed in [67] does not work.
CHAPTER 3. TWO-LINK FAILURE PROTECTION IN WDM MESH NETWORKS WITH $p$-CYCLES

3.1 Introduction

In this chapter, we consider the problem of protecting connections against two simultaneous link failures. Our basic idea is to use two $p$-cycles with link-disjoint protection segments to protect each working link. Since $p$-cycles are pre-configured using the spare capacity in the network, extremely fast restoration can be achieved. We formulate an ILP model for the $p$-cycle design problem for static traffic and we also propose two protection schemes SPPP and SFPP for dynamic traffic. Compared with the other methods, the worst-case and average number of optical cross connects that need to be configured upon a double-link failure in the SPPP scheme are less and thus SPPP scheme has a faster restoration speed.

The rest of this chapter is organized as follows. In Section 3.2, we present two theorems about double-link failure protection. An ILP model for the $p$-cycle design problem for static traffic is given in Section 3.3. In Section 3.4, we propose two double-link failure protection schemes for dynamic traffic. Numerical results are presented in Section 3.5. A conclusion is given in Section 3.6.

3.2 Preliminaries

We use a directed graph $G = (V, E)$ to represent a WDM optical network. A bidirectional communication link between nodes $u$ and $v$ are represented by two directed edges $u \rightarrow v \in E$ and $v \rightarrow u \in E$. Connections are unidirectional and each connection requires one unit of capacity (i.e., the capacity of a wavelength). We use unidirectional $p$-cycles to protect connections. A unidirectional $p$-cycle consumes one unit of capacity on each unidirectional on-cycle link; it can protect one unit of working capacity on
any straddling link and any link in the opposite direction of an on-cycle link.

Figure 3.1 Two-Link Failure Protection for Link $A \rightarrow D$

In [43], two link-disjoint $p$-cycles are computed to protect a working link against two link failures. However, we do not have to enforce the link-disjoint requirement on the two $p$-cycles in order to protect a link against two link failures. In fact, when a link $e$ is protected by a $p$-cycle $p$, only part of the $p$-cycle is used for protection. We name the part of $p$ that carries the traffic when $e$ fails as the protection segment for $e$ on $p$, which is denoted by $p(e)$. Fig. 3.1 shows two $p$-cycles $p_1$ and $p_2$, both of which can protect link $A \rightarrow D$. $p_1(A \rightarrow D) = A \rightarrow F \rightarrow G \rightarrow D$ is the protection segment for link $A \rightarrow D$ on $p_1$ and $p_2(A \rightarrow D) = A \rightarrow E \rightarrow D$ is the protection segment for link $A \rightarrow D$ on $p_2$. Although $p_1$ and $p_2$ are not link-disjoint (they share links $D \rightarrow C$, $C \rightarrow B$, and $B \rightarrow A$), they can still protect link $A \rightarrow D$ against two link failures since $p_1(A \rightarrow D)$ and $p_2(A \rightarrow D)$ are link-disjoint.

The following theorem gives the sufficient condition for a working link to be protected against any two-link failure.

**Theorem 1.** A working link $A \rightarrow B$ can be protected against any two-link failure if there exist two $p$-cycles $p_1$ and $p_2$ such that the following conditions are met.

1. $p_1$ can protect link $A \rightarrow B$;
2. $p_2$ can protect link $A \rightarrow B$;
3. $p_1(A \rightarrow B)$ is link-disjoint with $p_2(A \rightarrow B)$. 

Proof. The three conditions ensure that there are three link-disjoint paths from $A$ to $B$: one is the direct link from $A$ to $B$, the other two are $p_1(A \rightarrow B)$ and $p_2(A \rightarrow B)$. When any two links in the network fail, there must exist at least one path from $A$ to $B$ that is intact. Therefore, link $A \rightarrow B$ is protected against any two-link failure. ⊓⊔

According to Theorem 1, we can use two protection-segment-disjoint $p$-cycles to protect a working link against two link failures. However, using two $p$-cycles to protect each working link requires a large amount of protection capacity. To reduce the capacity requirement, we allow two working links to share a $p$-cycle. Let $e_1$ and $e_2$ be two working links. Let $S_1$ be a set of two protection-segment-disjoint $p$-cycles for $e_1$ and $S_2$ be a set of two protection-segment-disjoint $p$-cycles for $e_2$. If $|S_1 \cap S_2| = 1$, then $e_1$ and $e_2$ share one $p$-cycle. If $|S_1 \cap S_2| = 2$, then $e_1$ and $e_2$ share two $p$-cycles. When two links share one or two $p$-cycles, it’s possible that the failure of these two links will leave one or both of them unprotected. In this case, we say the sharing is invalid. On the other hand, we say the sharing is valid if the two links are still protected when both of them fail simultaneously. In the following, we present a theorem that gives the sufficient condition for a valid sharing.

**Theorem 2.** Let $e_1$ and $e_2$ be two working links that share one or two $p$-cycles (i.e., $S_1 \cap S_2 \neq \emptyset$). The sharing is valid if the following conditions are met.

1. For link $e_1$, there exists a $p$-cycle $p_1 \in S_1$ such that $e_2 \notin p_1(e_1)$.
2. For link $e_2$, there exists a $p$-cycle $p_2 \in S_2$ such that $e_1 \notin p_2(e_2)$;
3. $p_1(e_1)$ is link-disjoint with $p_2(e_2)$ if $p_1 = p_2$.

Proof. Suppose both $e_1$ and $e_2$ fail. Conditions 1) and 2) ensure that both $p_1(e_1)$ and $p_2(e_2)$ are not affected by the failures. If $p_1 \neq p_2$, then $e_1$ can be protected by $p_1$ and $e_2$ can be protected by $p_2$. Therefore, the sharing is valid. If $p_1 = p_2$, then $p_1(e_1)$ is link-disjoint with $p_1(e_2)$ according to condition 3). Thus, $p_1$ has two protection segments that can provide protection to $e_1$ and $e_2$ simultaneously. Therefore, the sharing is valid. ⊓⊔

Fig. 3.2 shows two examples of $p$-cycle sharing. In the left example, two working links $e_1 = A \rightarrow B$ and $e_2 = C \rightarrow D$ are protected by the same two $p$-cycles $p_1$ and $p_2$, where both $e_1$ and $e_2$ are

straddling links of $p_1$ and on-cycle links of $p_2$. That is, $S_1 = S_2 = \{p_1, p_2\}$. When both $e_1$ and $e_2$ fail, $p_2$ can protect neither of them since $e_2 \in p_2(e_1)$ and $e_1 \in p_2(e_2)$. $p_1$ can be used to protect either $e_1$ or $e_2$ but not both because $p_1(e_1) = A \rightarrow D \rightarrow F \rightarrow C \rightarrow B$ and $p_1(e_2) = C \rightarrow B \rightarrow E \rightarrow A \rightarrow D$ are not link-disjoint. Therefore, $e_1$ and $e_2$ cannot validly share the $p$-cycles $p_1$ and $p_2$. We now consider the example shown on the right side of Fig. 3.2, where everything is the same except that the direction of $p$-cycle $p_1$ is reversed. In this case, $p_1(e_1) = A \rightarrow E \rightarrow B$ does not contain $e_2$, $p_1(e_2) = C \rightarrow F \rightarrow D$ does not contain $e_1$, and $p_1(e_1)$ and $p_1(e_2)$ are link-disjoint. According to Theorem 2, $e_1$ and $e_2$ can validly share $p_1$ and $p_2$.

### 3.3 An ILP Model for Static Traffic Protection

In this section, we present an ILP model for the following $p$-cycle design problem: given a network $G = (V, E)$, and the working capacity $d_{ab}$ on each link $a \rightarrow b \in E$, compute a set of $p$-cycles to protect the working capacity against two-link failures such that the total capacity required by the $p$-cycles is minimized.

Objective:

$$\text{Minimize } \sum_p \sum_{(m,n) \in E} e_{mn}^p$$
Notations:

- \( P \): the maximum no. of \( p \)-cycles in the solution.
- \( p \): \( p \)-cycle index where \( p \in \{1, 2, \ldots, P\} \).
- \( d_{ab} \): integer, total amount of working capacity on link \( a \rightarrow b \).
- \( e_{mn}^p \): binary variable, 1 if \( p \)-cycle \( p \) uses link \( m \rightarrow n \) as an on-cycle link.
- \( x_{ab,k}^p \): binary variable, 1 if \( p \)-cycle \( p \) protects the \( k^{th} \) working capacity on link \( a \rightarrow b \).
- \( z_n^p \): binary variable, 1 if node \( n \) is on \( p \)-cycle \( p \).
- \( f_{mn}^{p,(ab,k)} \): binary variable, 1 if \( p \)-cycle \( p \) protects the \( k^{th} \) working capacity on link \( a \rightarrow b \) and the protection segment traverses link \( m \rightarrow n \).
- \( v_{cd}^{p,(ab,k)} \): binary variable, 1 if \( p \)-cycle \( p \) protects the \( k^{th} \) working capacity on link \( a \rightarrow b \) and the protection segment does not use link \( c \rightarrow d \) or \( d \rightarrow c \).
- \( AB_{cd,l}^{p,(ab,k)} \): binary variable, it equals \( |v_{cd}^{p,(ab,k)} - v_{ab}^{p,(cd,l)}| \).
- \( C_{cd,l}^{p,(ab,k)} \): binary variable, used in the absolute value constraints for \( AB_{cd,l}^{p,(ab,k)} \).

Capacity Constraints:

\[
\sum_{p} x_{ab,k}^p \geq 2, \quad \forall (a, b) \in E, \quad \forall k \leq d_{ab}; \quad (3.1)
\]
\[
\sum_{k} x_{ab,k}^p \leq 1, \quad \forall p, \quad \forall (a, b) \in E; \quad (3.2)
\]

Cycle Constraints:

\[
\sum_{(m, n) \in E} e_{mn}^p = \sum_{(n, l) \in E} e_{nl}^p = z_n^p, \quad \forall p, \forall n \in V; \quad (3.3)
\]
\[
e_{mn}^p + e_{nm}^p \leq 1, \quad \forall p, \forall (m, n) \in E \quad (3.4)
\]
Link Protection Constraints:

\[
\sum_m^{f_{mn}^p(ab,k)} - \sum_l^{f_{nl}^p(ab,k)} = \begin{cases} 
  x_{ab,k}^p & \text{if } n = b \\
  -x_{ab,k}^p & \text{if } n = a \\
  0 & \text{otherwise}
\end{cases}
\] (3.5)

\[\forall p, \forall (a, b) \in E, \forall n \in V, \forall k \leq d_{ab};\]

\[
\sum_m^{f_{ma}^p(ab,k)} = \sum_n^{f_{bn}^p(ab,k)} = 0
\] (3.6)

\[\forall p, \forall (a, b) \in E, \forall k \leq d_{ab};\]

\[f_{mn}^p(ab,k) \leq v_{mn}^p
\] (3.7)

\[\forall (a, b) \in E, (m, n) \in E, (a, b) \neq (m, n), \forall p, \forall k \leq d_{ab};\]

Protection Segment Disjointness Constraints:

\[fp_{mn}^p(ab,k) + fq_{nm}^p(ab,k) \leq 1
\] (3.8)

\[fp_{mn}^p(ab,k) + f_{nm}^q(ab,k) \leq 1
\] (3.9)

\[\forall (a, b) \in E, (m, n) \in E, (a, b) \neq (m, n), (a, b) \neq (n, m),
\]

\[\forall p, q, p \neq q, \forall k \leq d_{ab};\]

Absolute Value Constraints:

\[v_{cd}^{p(ab,k)} = (a_{ab,k}^p - f_{cd}^{p(ab,k)} - f_{dc}^{p(ab,k)})
\] (3.10)

\[\forall (a, b), (c, d) \in E, (a, b) \neq (c, d), \forall p, \forall k \leq d_{ab}
\]

\[AB_{cd,l}^{p(ab,k)} \geq v_{cd}^{p(ab,k)} - v_{ab}^{p(cd,l)}
\] (3.11)

\[AB_{cd,l}^{p(ab,k)} \geq -(v_{cd}^{p(ab,k)} - v_{ab}^{p(cd,l)})
\] (3.12)

\[AB_{cd,l}^{p(ab,k)} \leq v_{cd}^{p(ab,k)} - v_{ab}^{p(cd,l)} + 2c_{cd,l}^{p(ab,k)}
\] (3.13)

\[AB_{cd,l}^{p(ab,k)} \leq -(v_{cd}^{p(ab,k)} - v_{ab}^{p(cd,l)}) + 2(1 - C_{cd,l}^{p(ab,k)})
\] (3.14)

\[\forall (a, b), (c, d) \in E, (a, b) \neq (c, d),
\]

\[\forall p, \forall k \leq d_{ab}, l \leq d_{cd}.
\]
$p$-Cycle Sharing Constraints:

$$f_{mn}^{p,(ab,k)} + f_{mn}^{p,(cd,l)} + v_{cd}^{p,(ab,k)} + v_{ab}^{p,(cd,l)} \leq$$

$$\sum_p v_{cd}^{p,(ab,k)} + \sum_p v_{cd}^{p,(cd,l)} + \sum_p AB_{cd}^{p,(ab,k)} + 1 \quad (3.15)$$

$$\forall (a,b), (c,d) \in E, (a,b) \neq (c,d),$$

$$\forall p, \forall (m,n) \in E, \forall k \leq d_{ab}, l \leq d_{cd}.$$

Constraint (1) ensures that each unit of working capacity on a link is protected by at least two $p$-cycles. Constraint (2) ensures that a $p$-cycle can protect only one unit of working capacity on a link. Constraints (3) and (4) define $p$-cycle $p$ by ensuring that the in-degree and out-degree of each node on $p$ is 1 and $p$ cannot contain both link $(m,n)$ and $(n,m)$. Constraints (5)-(7) ensure that the $k$th working capacity on link $a \to b$ can be protected by $p$-cycle $p$ only if a unit flow can be sent from $a$ to $b$ using the links on $p$. In fact, the links traversed by the unit flow form the protection segment. Constraints (8) and (9) ensure that the two protection segments that protect a unit of working capacity on a link are link-disjoint. Constraint (10) defines $v_{cd}^{p,(ab,k)}$. Constraints (11)-(14) define $AB_{cd}^{p,(ab,k)}$. Constraint (15) ensures that all $p$-cycle sharings are valid based on Theorem 2. It takes the following three cases into the consideration. If $\sum_p AB_{cd}^{p,(ab,k)} \geq 1$, then link $(a,b)$ and link $(c,d)$ can be protected by two different $p$-cycles when both links fail. If $\sum_p AB_{cd}^{p,(ab,k)} = 0$, then there are two cases. If $\sum_p v_{cd}^{p,(ab,k)} + \sum_p v_{cd}^{p,(cd,l)} \geq 4$, then link $(a,b)$ and link $(c,d)$ share the same two $p$-cycles, and both links are straddling links of the two $p$-cycles. In this case, one of the two $p$-cycles can protect $(a,b)$ and the other one can protect $(c,d)$ when both links fail. Otherwise, we have $\sum_p v_{cd}^{p,(ab,k)} + \sum_p v_{cd}^{p,(cd,l)} = 2$ and only one $p$-cycle $p$ can be used to protect link $(a,b)$ and link $(c,d)$ when both of them fail. In this case, we must have $f_{mn}^{p,(ab,k)} + f_{mn}^{p,(cd,l)} + v_{cd}^{p,(ab,k)} + v_{ab}^{p,(cd,l)} \leq 3$ to ensure that $p(a,b)$ and $p(c,d)$ are link-disjoint. Constraint (15) combines all three cases to ensure that all $p$-cycle sharings are valid.

### 3.4 Protection Schemes for Dynamic Traffic

In this section, we study the problem of two-link failure protection for dynamic traffic. We assume that the working path for a connection is given. The problem is to compute a set of $p$-cycles to protect the working path against any two-link failure so that the total capacity required by the $p$-cycles is
minimized. We present two heuristic algorithms for this problem. Both algorithms are designed to achieve efficient protection by employing $p$-cycle sharing.

3.4.1 Shortest Path Pair Protection Scheme

We propose the Shortest Path Pair Protection (SPPP) scheme in this section. Given the working path $P$ of a connection, SPPP computes a set of $p$-cycles to protect $P$ as follows. For each link on $P$, we compute two $p$-cycles to protect the link so that the two $p$-cycles are protection-segment-disjoint. Whenever possible, we reuse the $p$-cycles that have been previously created to minimize the total protection capacity.

![Figure 3.3 p-Cycles Used in SPPP Scheme.](image)

Fig.3.3 illustrates how SPPP protects a working path from $s$ to $d$ that traverses link 1 through link 4. For each link on the working path, SPPP computes two $p$-cycles with link-disjoint protection segments to protect the link. As shown in the figure, $pc_1$ and $pc'_2$ are used to protect link $i$, for $1 \leq i \leq 4$. To save capacity, we allow a $p$-cycle to be shared by different working links if sharing is allowed according to Theorem 2. For example, suppose link 3 can share $pc_2$ with link 2, then $pc_3 = pc_2$ and only one new $p$-cycle (i.e., $pc'_3$) needs to be created for link 3; suppose link 4 can share $pc_1$ with link 1 and can share $pc'_2$ with link 2, and $pc_1(link4)$ is link-disjoint with $pc'_2(link4)$, then $pc_4 = pc_1$, $pc'_4 = pc'_2$, and no new $p$-cycle needs to be created for link 4.

We now explain the detail of SPPP. SPPP uses a boolean function $check\_share(pc_1, pc_2, e)$, where $pc_1$ and $pc_2$ are two $p$-cycles and $e$ is a working link. Both $pc_1$ and $pc_2$ can protect $e$, and $pc_1(e)$ and $pc_2(e)$ are link-disjoint. $check\_share(pc_1, pc_2, e)$ returns true if $e$ can share $pc_1$ with all other working links protected by $pc_1$ and false otherwise. That is, $check\_share(pc_1, pc_2, e)$ returns true if for every...
working link \( e' \neq e \) that is protected by \( pc_1 \), \( e \) and \( e' \) can share \( pc_1 \). (Note that Theorem 2 can be used to check whether \( e \) and \( e' \) are allowed to share \( pc_1 \)).

Given a working link \( e \), the set of existing \( p \)-cycles that can protect \( e \) is denoted by \( PC_e \). That is, \( PC_e \) contains all existing \( p \)-cycles that have \( e \) as an on-cycle link or a straddling link. For each link \( e \) on the working path, SPPP computes two \( p \)-cycles for \( e \) as follows. We first check whether there exist two \( p \)-cycles in \( PC_e \) such that they can be reused to protect \( e \). If so, no new \( p \)-cycle needs to be created for \( e \). This check can be done by using the \( \text{check}_\text{share} \) function. Specifically, if we can find two \( p \)-cycles \( pc_i \) and \( pc_j \) in \( PC_e \) such that 1) \( pc_i(e) \) and \( pc_j(e) \) are link-disjoint, and 2) both \( \text{check}_\text{share}(pc_i, pc_j, e) \) and \( \text{check}_\text{share}(pc_j, pc_i, e) \) return true, then \( pc_i \) and \( pc_j \) can be reused to protect \( e \). Otherwise, we try to reuse one \( p \)-cycle in \( PC_e \) to protect \( e \). To reuse a \( p \)-cycle \( pc_i \) in \( PC_e \) to protect \( e \), we need to compute a second \( p \)-cycle \( pc_j \) for \( e \) such that \( pc_i(e) \) and \( pc_j(e) \) are link-disjoint and \( \text{check}_\text{share}(pc_i, pc_j, e) \) returns true. If this can be done, then \( e \) is protected by reusing \( pc_i \) and creating a new \( p \)-cycle \( pc_j \). Finally, if none of the \( p \)-cycles in \( PC_e \) can be reused to protect \( e \), then we create two new \( p \)-cycles for \( e \) such that the protection segments for \( e \) on these two \( p \)-cycles are link-disjoint. To compute such two \( p \)-cycles for \( e \), we first use Bhandari’s algorithm [68] to compute two link-disjoint paths between the two endnodes of \( e \) with minimum total length. We then obtain two \( p \)-cycles for \( e \) by combining each path with \( e \) in the reverse direction. Clearly, these two \( p \)-cycles can provide link-disjoint protection segments for \( e \).

The pseudocode of the SPPP scheme is shown in Algorithm 1. The input is a working path \( P \), the output is a set \( PC \) of \( p \)-cycles that protect \( P \). The algorithm computes two \( p \)-cycles for each link \( e \) in \( P \) in the for loop from line 1 to line 16.

Line 3 checks whether there are two \( p \)-cycles \( pc_i \) and \( pc_j \) in \( PC_e \) that can be reused to protect \( e \). If yes, \( e \) needs no new \( p \)-cycle for protection and \( flag \) is set to 0 in line 4. In line 5, \( pc_i \) and \( pc_j \) are removed from \( PC_e \) since they can no longer be used to protect other connections that traverse \( e \).

Line 7 checks if we can reuse a \( p \)-cycle \( pc_i \) in \( PC_e \) to protect \( e \), which requires a new \( p \)-cycle \( pc_j \) to be created for \( e \) with certain conditions satisfied. If yes, \( flag \) is set to 1 in line 8 and \( pc_i \) is removed from \( PC_e \) in line 9. In line 10, for every link \( e' \neq e \) that can be protected by \( pc_j \) (i.e., \( e' \) is either an on-cycle link or a straddling link of \( pc_j \)), we add \( pc_j \) into \( PC_e \) so that \( pc_j \) can be exploited for reuse...
Algorithm 1  SPPP Scheme

1: for (every $e \in P$) do
2:   $flag = 2$;
3:   if ($\exists pc_i, pc_j \in PC_e$ such that $pc_i(e) \cap pc_j(e) = \phi$ and $check\_share(pc_i, pc_j, e) = true$ and $check\_share(pc_j, pc_i, e) = true$) then
4:      $flag = 0$;
5:      $PC_e = PC_e - \{pc_i, pc_j\}$;
6:   else
7:      if ($\exists pc_i \in PC_e$ and we can create a new $p$-cycle $pc_j$ for $e$ such that $pc_i(e) \cap pc_j(e) = \phi$ and $check\_share(pc_i, pc_j, e) = true$) then
8:         $flag = 1$;
9:         $PC_e = PC_e - \{pc_i\}$;
10:        $\forall e' \neq e$ that can be protected by $pc_j$, $PC_{e'} = PC_{e'} \cup \{pc_j\}$;
11:       $PC = PC \cup \{pc_j\}$;
12:   end if
13: end if
14: if ($flag = 2$) then
15:   Use Bhandari’s Algorithm to obtain two protection-segment-disjoint $p$-cycles $pc_1$ and $pc_2$ for $e$;
16:   $\forall e' \neq e$ that can be protected by $pc_1$, $PC_{e'} = PC_{e'} \cup \{pc_1\}$;
17:   $\forall e' \neq e$ that can be protected by $pc_2$, $PC_{e'} = PC_{e'} \cup \{pc_2\}$;
18:   $PC = PC \cup \{pc_1, pc_2\}$;
19: end if
20: Return $PC$;
in the future. In line 11, \( pc_j \) is added into \( PC \).

Line 12-16 deal with the case where no existing \( p \)-cycle can be reused to protect \( e \). In line 13, we compute two shortest protection-segment-disjoint \( p \)-cycles \( pc_1 \) and \( pc_2 \) to protect \( e \) using Bhandari’s Algorithm. In line 14 and 15, for every link \( e' \neq e \) that can be protected by \( pc_1/pc_2 \), we add \( pc_1/pc_2 \) into \( PC_{e'} \). Finally, the two new \( p \)-cycles, \( pc_1 \) and \( pc_2 \), are added into \( PC \).

We now analyze the time complexity of SPPP. For each \( e \in P \), the algorithm computes two \( p \)-cycles for \( e \). The time of this computation is dominated by the computation in line 3. The running time of function \( \text{check}\_\text{share}(pc_i, pc_j, e) \) is \( O(|E| \times |V|^2) \) because it needs to check each working link protected by \( pc_i \) to see if it can share \( pc_i \) with \( e \), and the time of the check is \( O(|V|^2) \). Assuming \( |PC_e| \) is upper bounded by a constant, the running time of line 3 is \( O(|E| \times |V|^2) \). Since line 3 is executed for each edge in the working path \( P \) and the number of edges in \( P \) is upper bounded by \( |V| \), the complexity of SPPP is \( O(|E| \times |V|^3) \).

The advantage of the SPPP scheme is that it can save plenty of protection capacity by exploiting \( p \)-cycle sharing. However, SPPP always creates short \( p \)-cycles, which are less efficient than long \( p \)-cycles as shown in [17] since short \( p \)-cycles tend to have less straddling links. In the next, we present another protection scheme that makes use of long \( p \)-cycles for connection protection.

### 3.4.2 Shortest Full Path Protection Scheme

In this section, we present the Shortest Full Path Protection (SFPP) Scheme. Given the working path \( P \) of a connection, SFPP computes a set of \( p \)-cycles to protect \( P \) as follows. First, we compute one short \( p \)-cycle for each link on \( P \). Next, we compute a long \( p \)-cycle that contains all links on \( P \) and is link-disjoint with the protection segments of all the working links computed in the first step. Clearly, the long \( p \)-cycle can protect every link in \( P \). Therefore, each working link is still protected by two \( p \)-cycles (one short and one long) with link-disjoint protection segments. Like SPPP, SFPP reuses existing \( p \)-cycles whenever possible to save protection capacity.

Fig. 3.4 illustrates how SFPP protects a working path from \( s \) to \( d \) that traverses link 1 through link 4. Four short \( p \)-cycles, \( pc_1 \) to \( pc_4 \), are first found to protect link 1 to link 4. These short \( p \)-cycles can be shared by the working links. For example, if link 3 can share \( pc_1 \) with link 1, then \( pc_3 = pc_1 \) and no new
short \( p \)-cycle needs to be created for link 3. In the second step, we find a long \( p \)-cycle, labeled as \( pc_5 \), to cover the entire working path. \( pc_5 \) must be link-disjoint with the protection segments \( pc_1(\text{link}1) \) to \( pc_4(\text{link}4) \) to ensure that each working link is protected by two protection-segment-disjoint \( p \)-cycles.

We now explain the detail of SFPP. Let \( PC_e \) denote the set of existing \( p \)-cycles that can protect link \( e \). We first find one short \( p \)-cycle for every link \( e \) on the working path \( P \). During this process, existing \( p \)-cycles will be reused if sharing is possible. Specifically, when we process link \( e \), we first check whether there is a \( p \)-cycle in \( PC_e \) that can be reused to protect \( e \). A \( p \)-cycle \( pc \) can be reused to protect \( e \) if 1) \( pc \) does not contain any edge \( e' \neq e \) in \( P \), and 2) for every edge \( e' \neq e \) in \( P \) that is protected by \( pc \), \( pc(e) \) and \( pc(e') \) are link-disjoint. The first condition is needed because if \( pc \) contains \( e' \), then \( pc \) and the long \( p \)-cycle will not be protection-segment-disjoint since they both contain \( e' \). The second condition is needed for the following reason. When both \( e' \) and \( e \) fail, the long \( p \)-cycle can protect neither of them since the protection segment of one link contains the other link. So, both links have to be protected by \( pc \). According to Theorem 2, \( pc(e) \) and \( pc(e') \) must be link-disjoint. We define a function \( \text{check\_share1}(pc, e) \) that returns true when the two conditions are satisfied. That is, if \( \text{check\_share1}(pc, e) \) returns true, then \( pc \) can be reused to protect \( e \). Otherwise, \( pc \) cannot be reused to protect \( e \). In this case, we need to compute a new \( p \)-cycle for \( e \) with the requirement that it does not contain any edge \( e' \neq e \) in \( P \). After we have found a short \( p \)-cycle for each link \( e \) in \( P \), we compute a long \( p \)-cycle as follows. We first remove all links in \( P \) and all links belong to the protection segments (provided by the short \( p \)-cycles) of all the working links in \( P \). We then compute the shortest path \( SP \) from the source \( s \) to the destination \( d \). Finally, we combine \( SP \) with the reverse direction of \( P \) to form a long \( p \)-cycle \( pc_f \). After \( pc_f \) is obtained, we check whether there is any invalid
sharing of $p$-cycles as follows. For each link $e$ in $P$ and each link $e'$ that is not in $P$, if $e$ and $e'$ share a short $p$-cycle $pc$, we check whether the sharing is valid according to Theorem 2. We define a function \text{check\_share2}(e, e', pc) that performs the check. The function returns true if the sharing of $pc$ by $e$ and $e'$ is valid. If the function returns false, the long $p$-cycle $pc_f$ must contain link $e'$ and the sharing would be valid if $pc_f$ does not contain $e'$. (We will explain why in the next.) So, we remove $e'$ from $G$. After all troublesome links are removed, we recompute a long $p$-cycle $pc_f$. We then repeat the process of checking $p$-cycle sharing validity and computing the long $p$-cycle until no invalid $p$-cycle sharing can be found.

We now explain why an invalid sharing of $pc$ by $e$ and $e'$ is caused by the inclusion of $e'$ in $pc_f$. Let $pc'$ be the second $p$-cycle that protects $e'$. (The first $p$-cycle that protects $e'$ is $pc$, which is shared by $e$.) In order for $e$ and $e'$ to validly share $pc$, we have to make sure that when both links fail, at least one of $pc$ and $pc'$ can protect $e'$ and at least one of $pc$ and $pc_f$ can protect $e$. We know at least one of the protection segments $pc'(e')$ and $pc'(e')$ does not contain $e$ since $pc(e')$ and $pc'(e')$ must be link-disjoint. Therefore, there are three cases to consider.

1. \textit{Both $pc(e')$ and $pc'(e')$ do not contain $e$:} Clearly, $e'$ can be protected by $pc'$ when both $e$ and $e'$ fail since $pc'(e')$ does not contain $e$. In addition, one of $pc$ and $pc_f$ can protect $e$ because $pc(e)$ and $pc_f(e)$ are link-disjoint and therefore at least one of them does not contain $e'$. So, $e$ and $e'$ can validly share $pc$.

2. \textit{$pc(e')$ contains $e$ and $pc'(e')$ does not contain $e$:} $e$ and $e'$ can validly share $pc$ for the same reason given in the previous case.

3. \textit{$pc'(e')$ contains $e$ and $pc(e')$ does not contain $e$:} $e'$ has to be protected by $pc$ when both $e$ and $e'$ fail since $pc'(e')$ contains $e$. As for $e$, it can be protected by $pc_f$ if $pc_f(e)$ does not contain $e'$. Therefore, if $pc_f(e)$ does not contain $e'$, then the sharing is valid.

As can be seen from the above three cases, if we know $e$ and $e'$ cannot validly share $pc$, then it must be the case that $pc_f$ contains $e'$. And we can turn the sharing into a valid one by making sure that $pc_f$ does not contain $e'$. 

1: \( PC_{\text{temp}} = \emptyset, \ PC = \emptyset; \ \text{flag}=1; \)
2: \textbf{for} (\( \forall e \in P \)) \textbf{do}
3: \( \text{protected} = \text{false}; \)
4: \textbf{if} (\( \exists pc \in PC_e, \text{check\_share\_1}(pc, e) = \text{true} \)) \textbf{then}
5: \( PC_{\text{temp}} = PC_{\text{temp}} \cup \{pc(e)\}; \)
6: \( PC_e = PC_e \setminus \{pc\}; \)
7: \( \text{protected} = \text{true}; \)
8: \textbf{if} (\( \neg \text{protected} \)) \textbf{then}
9: \quad \text{Find the shortest } p\text{-cycle } pc_e \text{ such that } pc_e \text{ does not contain any link in } P \text{ except } e; \)
10: \quad \( PC = PC \cup \{pc_e\}; \)
11: \quad \( PC_{\text{temp}} = PC_{\text{temp}} \cup \{pc_e(e)\}; \)
12: \quad \( \forall e' \neq e \text{ that can be protected by } pc_e, \)
13: \quad \quad \( PC_{e'} = PC_{e'} \cup \{pc_e\}; \)
14: \quad \text{Remove all links in } P \text{ and all links belong to the protection segments in } PC_{\text{temp}} \text{ from } G; \)
15: \textbf{for} (\( \forall e \in P \)) \textbf{do}
16: \textbf{for} (\( \forall e' \notin P \text{ that share } pc \text{ with } e \)) \textbf{do}
17: \quad \textbf{if} (\( \text{check\_share\_2}(e, e', pc) = \text{false} \)) \textbf{then}
18: \quad \quad \text{remove } e' \text{ from } G; \)
19: \quad \textbf{if} (\( \text{flag} = 1 \)) \textbf{then}
20: \quad \quad \text{flag}=0; \)
21: \quad \textbf{if} (\( \text{flag} = 0 \)) \textbf{then}
22: \quad \quad \text{Find the shortest path } SP \text{ from } s \text{ to } d \text{ and combine it with reversed } P \text{ to form a long } p\text{-cycle } pc_f; \)
23: \quad \quad \text{flag} = 1; \)
24: \quad \quad \textbf{goto} 15; \)
25: \quad \( PC = PC \cup \{pc_f\}; \)
26: \quad \( \forall e' \neq e \text{ that can be protected by } pc_f \)
27: \quad \quad \( PC_{e'} = PC_{e'} \cup \{pc_f\}; \)
28: \quad \text{Return } PC; \)

\textbf{Algorithm 2} SFPP Scheme
The pseudocode of the SFPP scheme is shown in Algorithm 2. The input is a working path $P$, the output is a set $PC$ of $p$-cycles that protect $P$. Set $PC_{\text{temp}}$ stores the protection segments that are used to protect the links on $P$.

The for loop in line 2-12 computes a short cycle for each link $e$ in $P$. Line 4 checks if there is a $p$-cycle $pc$ in $PC_e$ that can be reused to protect $e$. If so, $pc(e)$ is added into $PC_{\text{temp}}$ in line 5 and $pc$ is removed from the set $PC_e$ in line 6. If we cannot find an existing $p$-cycle to protect $e$, we compute a new $p$-cycle $pc_e$ to protect $e$ in line 9. $pc_e$ must not contain any link in $P$ except $e$ to ensure that it is protection-segment-disjoint with the long $p$-cycle. $pc_e$ is added to $PC$ in line 10 and $pc_e(e)$ is added into $PC_{\text{temp}}$ in line 11. In line 12, we update $PC_e$ for every link $e' \neq e$ that can be protected by $pc_e$.

In the next, the algorithm computes the long $p$-cycle $pc_f$, which must be link-disjoint with $P$ and the protection segments stored in $PC_{\text{temp}}$. Therefore, we remove all links in $P$ and all links belong to the protection segments in $PC_{\text{temp}}$ from $G$ in line 13. In line 14, we obtain the long $p$-cycle $pc_f$ by combining the shortest path $SP$ from $s$ to $d$ and the reversed path $P$.

The nested for loop in line 15-20 does the $p$-cycle sharing validity check. For each link $e$ in $P$, if it shares $pc$ with another link $e'$ not in $P$, we check the sharing validity according Theorem 2. If $\text{check\_share2}(e, e', pc)$ returns false in line 17, then $pc_f$ must contain $e'$. So, we remove $e'$ from $G$ in line 18 and set flag to 0 if its current value is 1. If flag is 0 after the nested for loop is executed, we recompute $pc_f$ in line 22. We then set flag to 1 in line 23 and repeat the $p$-cycle sharing validity check and computation of $pc_f$ until no invalid $p$-cycle sharing can be found.

After we find a $pc_f$ that ensures all $p$-cycle sharings are valid, we add it into $PC$ in line 25. For each edge $e' \neq e$ that can be protected by $pc_f$, the set $PC_{e'}$ is updated in line 26.

The time complexity of SFPP is dominated by the computation in lines 15-24. The complexity of function $\text{check\_share2}(e, e', pc)$ is $O(|V|^2)$, so the complexity of lines 15-20 is $O(|V|^2|E||V|^2) = O(|E|^3|V|^3)$. This block of code would be executed at most $|E|$ times because at most $|E|$ edges can be removed from $G$. Therefore, the complexity of SFPP is $O(|E|^2|E||V|^3) = O(|E|^2|V|^3)$.

Since SFPP makes use of long $p$-cycles, when failures occur in the network, some rerouted working paths may pass through redundant nodes and links since protection switching is done at the two endnodes of the failed link. This problem can be solved using the algorithm given in [70], which
removes the loop backs and release the redundant capacity by reconfiguring the restored paths.

3.5 Numerical Results

3.5.1 ILP Results for Static Traffic

We use ILOG CPLEX 10.1.0 to implement the ILP on a computer with four Intel Xeon 2.40GHz CPUs and 4BG of memory. A small test network with 6 nodes and 11 edges (shown in Fig. 3.5) is used. Table 3.1 shows the working capacity, the protection capacity (computed by the ILP), the protection redundancy (ratio of protection capacity to working capacity), and the running time for different number of connections. Each data point is the average of ten test cases.

![Figure 3.5 The 6-node 11-edge network.](image)

Table 3.1 Redundancy and Computation time of ILP

<table>
<thead>
<tr>
<th>Number of connections</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working capacity</td>
<td>1.2</td>
<td>2.1</td>
<td>3.7</td>
<td>4.9</td>
<td>6.1</td>
</tr>
<tr>
<td>Protection capacity</td>
<td>7.1</td>
<td>9.4</td>
<td>13.8</td>
<td>15</td>
<td>18.4</td>
</tr>
<tr>
<td>Protection Redundancy</td>
<td>592%</td>
<td>448%</td>
<td>373%</td>
<td>306%</td>
<td>302%</td>
</tr>
<tr>
<td>Running Times (s)</td>
<td>0.034</td>
<td>0.91</td>
<td>59.8</td>
<td>1304</td>
<td>11684.2</td>
</tr>
</tbody>
</table>

The table shows that as the number of connections increases from 1 to 5, the protection redundancy decreases from 592% to 302%. This is expected because p-cycle sharing can be better exploited when more connections exist in the network. On the other hand, the running time increases exponentially as the number of connections increases.
3.5.2 Comparison of SPPP and SFPP

We conduct simulations to compare the performance of SPPP and SFPP under incremental traffic and dynamic traffic. Two networks, the SMALLNET network and the COST239 network (Fig. 3.6), are used in the simulations. In each simulation run, a set of randomly generated connection requests are loaded to the network. For each connection request, the working path is routed along the shortest path between the source and the destination.

In the first set of simulations, we consider incremental traffic. That is, a demand never terminates once it is satisfied. The capacity of the network link is set to infinity. The total number of wavelength channels used by all the working paths and by all the \(p\)-cycles are recorded for each simulation run. In Fig. 3.7, we show the performance of SPPP and SFPP under different traffic load in SMALLNET network. The results shows that SFPP uses less wavelength channels for protection than SPPP under all traffic loads. Specifically, SFPP achieves a 16.4%-18.3% reduction in wavelength usage over SPPP. The reason for SFPP to ourperform SPPP is that SFPP uses long \(p\)-cycles that have more straddling links so that higher protection efficiency can be achieved.

In Fig. 3.8, we show the performance of SPPP and SFPP in COST239 network. Again, SFPP uses less wavelength channels for protection than SPPP under all traffic loads. Specifically, SFPP achieves a 21.5%-24.5% reduction in wavelength usage over SPPP. The improvement of SFPP over SPPP is bigger than that in SMALLNET network. This is because the COST239 network is denser. So, long \(p\)-cycles tend to have higher protection efficiency due to the inclusion of more straddling links.
Figure 3.7  Wavelength usage of SPPP and SFPP in SMALLNET.

Figure 3.8  Wavelength usage of SPPP and SFPP in COST239.

Figure 3.9  Protection redundancy of SPPP and SFPP in SMALLNET.
Fig. 3.9 and Fig. 3.10 compare the protection redundancy of SPPP and SFPP for SMALLNET and COST239, respectively. Both figures show that the protection redundancy of SPPP and SFPP drop slightly as the number of connections increases, which is consistent with the ILP results. The redundancy of SFPP is much lower than that of SPPP. For SMALLNET, SFPP achieves 16.4%-18.3% reduction in redundancy over SPPP; For COST239, SFPP achieves 23.0% -24.5% reduction in redundancy over SPPP.

In the second set of simulations, we consider dynamic traffic. In each simulation run, 5000 randomly generated connection requests are loaded to the network and the reject ratio is recorded. The arrival of traffic follows Poisson distribution with $\lambda$ connection requests per second and the duration of the request is exponentially distributed with a mean of $1/\mu$. The traffic load measured in erlangs is $\lambda/\mu$. The capacity of the network link is set to 10 wavelengths.

In Fig 3.11, we compare the reject ratio of SFPP and SPPP under different traffic loads (in erlangs) in SMALLNET network. The results show that SFPP performs better than SPPP when traffic load is above 32 erlangs. However, SFPP performs worse than SPPP when traffic load is below 32 erlangs. This can be explained as follows. When the traffic load is low, there is not enough connections to fully utilize the protection capacity provided by the long $p$-cycles. However, when the traffic load becomes high, the long $p$-cycles can be fully utilized and they can provide more efficient protection than those $p$-cycles created by SPPP.
Figure 3.11  Reject ratio of SPPP and SFPP in SMALLNET.

Figure 3.12  Reject ratio of SPPP and SFPP in COST239.
In Fig 3.12, we compare the reject ratio of SFPP and SPPP under different traffic loads in COST239 network. Again, the results show that SFPP performs better than SPPP under high traffic loads and performs worse than SPPP under low traffic loads.

3.5.3 Comparison of SPPP and the Algorithms in [4]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>I</th>
<th>II</th>
<th>MADPA</th>
<th>SPPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protection ratio</td>
<td>100%</td>
<td>100%</td>
<td>98.8%</td>
<td>100%</td>
</tr>
<tr>
<td>Protection redundancy</td>
<td>200%</td>
<td>200%</td>
<td>200%</td>
<td>259%</td>
</tr>
<tr>
<td>$XC_{\text{max}}$ with signaling</td>
<td>26</td>
<td>18</td>
<td>N/A</td>
<td>6</td>
</tr>
<tr>
<td>$XC_{\text{avg}}$ with signaling</td>
<td>9.34</td>
<td>8.64</td>
<td>N/A</td>
<td>4.4</td>
</tr>
<tr>
<td>$XC_{\text{max}}$ w/o signaling</td>
<td>N/A</td>
<td>N/A</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>$XC_{\text{avg}}$ w/o signaling</td>
<td>N/A</td>
<td>N/A</td>
<td>7.3</td>
<td>4</td>
</tr>
</tbody>
</table>

We compare SPPP with the three approaches–Method I, Method II, and MADPA–proposed in [4] as shown in Table 3.2. The network topology used is the 20-node 32-link ARPANET network. Protection ratio is the percentage of double-link failures that can be protected. Protection redundancy is the ratio of the total protection capacity to the total working capacity. $XC_{\text{max}}$ and $XC_{\text{avg}}$ denote the worst-case and average number of optical cross connects that need to be configured upon a double-link failure. When a link fails, Methods I and II require that all nodes in the network are informed of the failure through signaling. However, this is not required for MADPA. SPPP can operate with or without signaling of the failure event. If, upon a link failure, the traffic on the link is sent on both $p$-cycles simultaneously, then signaling is not required. In this case, a total of 4 cross connections are needed to recover from any double-link failure because the two endnodes of each failed link need configure their cross connects to direct the traffic onto the $p$-cycles. On the other hand, if only one $p$-cycle is used to restore the traffic upon a link failure, then signaling of failure is required and a total of 6 cross connections are needed to recover from a double-link failure in the worst case. The worst case occurs when the second failure affects the $p$-cycle used to protect the first failure. In this case, when the first link fails, both endnodes configure their cross connects to direct the traffic onto the first $p$-cycle for this link. When the second link fails, the endnodes of the link configure their cross connects to direct the
traffic onto one of the two $p$-cycles that is not affected by the first link failure. After the endnodes of the first failed link learn that the second failure affects the $p$-cycle being used, they reconfigure their cross connects to direct the traffic onto the second $p$-cycle for this link. Thus, a total of 6 cross connections are needed. The results in Table 3.2 show that while SPPP has higher protection redundancy than the other three methods, the number of cross connections required is much less. Since $p$-cycles are pre-configured, SPPP requires only the endnodes of the failed links to configure their cross connects. On the other hand, cross connects have to be configured by every node along the protection path in the other three methods. Thus, SPPP is much faster in restoration than the other methods. Basically, SPPP trades off protection redundancy for restoration speed. Compared with the other methods, SPPP’s gain in restoration speed is much larger than its loss in protection redundancy.

3.6 Conclusion

In this chapter, we consider the problem of protecting connections against two-link failures. The basic idea is to protect each working link with two $p$-cycles with link-disjoint protection segments. We present an ILP model to compute the optimal set of $p$-cycles for protecting a set of static demands. We also propose two protection schemes – SPPP and SFPP – for dynamic demands. The numerical results show that SFPP is more capacity efficient than SPPP under incremental traffic and SPPP has slightly better failure recovery performance than SFPP. Under dynamic traffic, SPPP has lower blocking than SFPP when the traffic load is low and has higher blocking than SFPP when the traffic load is high. Compared with the algorithms proposed in [4], SPPP trades off protection redundancy for fast restoration speed.
CHAPTER 4. A HYBRID PROTECTION/RESTORATION SCHEME FOR TWO-LINK FAILURE IN WDM MESH NETWORKS

4.1 Introduction

In this chapter, we propose a new hybrid protection/restoration scheme to handle two-link failures. Unlike existing protection schemes that require two link-disjoint backup paths for each demand or link, our scheme only requires one backup path for each demand which leads to significant saving in backup capacity. Unlike backup reprovisioning schemes, our scheme computes new backup paths for unprotected demands after the second failure occurs so that unnecessary reprovisioning is avoided. The key ideas of our scheme are the following:

- Each demand is assigned a single backup path. Backup capacity is reserved to ensure all the demands whose working path is affected by the two-link failure and whose backup path is not affected by the two-link failure can be restored using the pre-planned backup paths.

- For those demands whose working path and backup path are both affected by the two-link failure, dynamic restoration is used to find new backup paths for the demands after the second failure occurs.

Basically, our scheme uses protection to ensure that most of the affected demands can be restored using the pre-planned backup paths upon a two-link failure. For the demands not restorable with protection, we use dynamic restoration to find new backup paths for them. Our scheme has the following advantages. First, most demands are fully protected against two-link failures with only one backup path pre-planned for each demand. Our backup capacity reservation method exploits backup capacity sharing under two-link failures. As a result, our scheme is capable of restoring the same set of demands as Dedicated Path Protection (DPP) with significantly less backup capacity. (In DPP, each working
path has a dedicated backup path. No backup capacity sharing is allowed.) Second, for demands not protected against two-link failures, they are dynamically restored using the available backup capacity upon second link failure. Our simulation results show that over 95% of these demands can be restored with pre-reserved backup capacity.

The remainder of the chapter is organized as follows. In Section 4.2, we describe the hybrid protection/restoration scheme for two-link failures. In Section 4.3, we present simulation results to demonstrate the efficiency of our scheme. Section 4.4 concludes this chapter.

### 4.2 The Hybrid Protection/Restoration Scheme for Two-Link Failure

In this section, we present a hybrid protection/restoration scheme for two-link failure where a two-link failure consists of two sequential link failures where the second failure occurs before the first failure is repaired. We assume the network has full wavelength-conversion capability and each demand requires one full wavelength capacity.

Our scheme works as follows. When a demand arrives at the network, the shortest pair of link-disjoint paths for the demand is computed using the Bhandari algorithm [68]. The shorter path is established as the working path and the longer path is reserved as the backup path, i.e, backup capacity is reserved on the backup path but the backup path is not set up. When a two-link failure $F$ occurs, we can divide the current demands in the network into the following three sets:

- $S_u$: it contains the demands whose working path is not affected by $F$. These demands are called \textit{unaffected demands}.
- $S_s$: it contains the demands whose working path is affected by $F$ and whose backup path is not affected by $F$. These demands are called \textit{survivable demands}.
- $S_n$: it contains the demands whose working path and backup path are both affected by $F$. These demands are called \textit{nonsurvivable demands}.

When $F$ occurs, demands in $S_u$ need no restoration since their working paths are not affected. Demands in $S_s$ lose their working paths because of $F$, but their backup paths are intact. If enough
backup capacity is reserved on their backup paths, these demands can be restored. In section 4.2.1, we present a backup capacity reservation scheme that ensures all demands in $S_s$ can be restored upon any two-link failure. The demands in $S_s$ lose both their working paths and their backup paths when $F$ occurs. So new backup paths need to be found for these demands to restore the traffic. In section 4.2.2, we describe a dynamic restoration scheme to restore these demands.

### 4.2.1 Backup Wavelength Reservation Scheme

To compute the number of backup wavelengths needs to be reserved on each link to ensure the recovery of demands in $S_s$, we introduce a new scheme based on the link-vector scheme proposed in [69]. The link-vector scheme in [69] can explore the backup-sharing potential between different demands and determine the minimum number of backup wavelengths required on each link to ensure full protection against any single-link failure. However, as will be illustrated below, this scheme cannot guarantee all demands in $S_s$ can be restored upon a two-link failure. In this section, we propose a new link-vector scheme which provides this guarantee.

We first introduce the original link-vector scheme in [69]. In this scheme, each link in the network is associated with a vector of $|E|$ elements, where $E$ is the set of links in the network. Let $\nu_e$ denote the link-vector for link $e$, an element $\nu_{e'}^e (e' \in E)$ of $\nu_e$ is an integer indicating the number of demands whose working path traverses $e'$ and whose backup path traverses $e$. To protect all the demands against any single-link failure, the number of backup wavelengths needs to be reserved on link $e$ is

$$\nu^*_e = \max_{\forall e' \in E} \nu_{e'}^e$$  \hspace{1cm} (4.1)

Although reserving $\nu^*_e$ backup wavelengths on every link $e$ in the network ensures all affected demands can be restored upon a single link failure, some demands in $S_s$ can not be restored when a second failure occurs due to insufficient backup wavelengths. This is illustrated in Fig. 4.1. There are three demands AD, BC, and GH. Their respective working paths are routed over A-B-C-D, B-C, and G-H, as indicated by the dotted lines. Their respective backup paths are routed over A-G-H-D, B-E-F-C, and G-E-F-H, as indicated by the dashed lines. To protect against any single-link failure, only one backup wavelength needs to be reserved on each link used by the three backup paths. In particular, only one
backup wavelength is needed on link $e_2$ even though it is used by two backup paths. That is, we have $\nu_{e_2}^* = 1$ because $\nu_{e_2}^{e_1} = \nu_{e_2}^{e_3} = 1$ and all other elements of $\nu_{e_2}$ are 0. Consider a two-link failure event where $e_1$ fails first and then $e_3$ fails. When $e_1$ fails, the AD demand and the BC demand can be restored using their backup paths. When $e_3$ fails, the GH demand (a survivable demand) cannot be restored using its backup path G-E-F-H because only one backup wavelength is reserved on E-F and this backup wavelength has been used to restore the BC demand.

![Figure 4.1 An example network with three demands: AD, BC, and GH. Working paths are shown in dotted lines. Backup paths are shown in dashed lines.](image)

We now present a new link-vector scheme that reserves enough backup wavelengths on each link to ensure the restoration of all survivable demands. In the new scheme, the number of backup wavelengths needs to be reserved on link $e \in E$, denoted by $\nu_e^*$, is computed as follows:

$$
\nu_e^* = \max_{\forall e_i, e_j, e_i \neq e_j} (\nu_{e_i}^{e_i} + \nu_{e_j}^{e_j} - n(e_i, e_j))
$$

(4.2)

where $n(e_i, e_j)$ is the number of working paths that traverse both $e_i$ and $e_j$. Note that $\nu_{e_i}^{e_i} + \nu_{e_j}^{e_j} - n(e_i, e_j)$ is the number of backup wavelengths required on $e$ to restore all the survivable demands when both $e_i$ and $e_j$ fail, which is equal to the number of demands whose working path traverses either $e_i$ or $e_j$ (or both). The minus $n(e_i, e_j)$ term is needed to avoid double counting the demands whose working path traverses both $e_i$ and $e_j$. By considering all pairs of $e_i$ and $e_j$ and taking the maximum value of $\nu_{e_i}^{e_i} + \nu_{e_j}^{e_j} - n(e_i, e_j)$, equation (2) ensures that the number of backup wavelengths reserved on link $e$ is the minimum required to allow all the survivable demands to be restored upon any two-link failure. Consider the example in Fig. 4.1. According to equation (2), we have $\nu_{e_2}^* = \nu_{e_2}^{e_1} + \nu_{e_2}^{e_3} - n(e_1, e_3) =$
\[1 + 1 - 0 = 2\]. So we need to reserve two backup wavelengths on \(e_2\) to ensure all survivable demands can be restored upon a two-link failure. Suppose \(e_1\) and \(e_3\) both fail, then we have two survivable demands: BC and GH. Both of them can be restored when two backup wavelengths are reserved on \(e_2\).

### 4.2.2 Dynamic Restoration Scheme for Nonsurvivable Demands

In this section, we describe a scheme for dynamically restoring the nonsurvivable demands upon a two-link failure.

Let \(d\) be a nonsurvivable demand with source \(s\) and destination \(t\). Let \(l_1 = (a, b)\) be the first failed link, which affects path \(p_1\) of \(d\). Let \(l_2 = (x, y)\) be the second failed link, which affects path \(p_2\) of \(d\). Note that if \(p_1\) is the working (or backup) path, then \(p_2\) is the backup (or working) path. Fig. 4.2 shows a nonsurvivable demand affected by two link failures. We note that the traffic of \(d\) must be carried on path \(p_2\) when \(l_2\) fails. This can be seen as follows. If \(p_1\) is the working path, then the failure of \(l_1\) causes the traffic to be switched to the backup path \(p_2\). On the other hand, if \(p_1\) is the backup path, then the failure of \(l_1\) will not affect the working path \(p_2\) so that \(p_2\) continues to carry the traffic after \(l_1\) fails. In both cases, \(p_2\) carries the traffic when \(l_2\) fails. If we can find a backup path between \(x\) and \(y\) that does not use \(l_1\) and \(l_2\), then the traffic can be quickly restored using this backup path.

![Figure 4.2](image)

**Figure 4.2** A nonsurvivable demand affected by the failures of link \(l_1 = (a, b)\) and link \(l_2 = (x, y)\). Traffic between \(s\) and \(t\) can be restored by finding a feasible path between \(x\) and \(y\) (dashed line).

We define a feasible path between two nodes as a path that satisfy two conditions: 1) the path does not contain \(l_1\) and \(l_2\); 2) each link on the path has a free wavelength (a backup wavelength is considered free if it is not used by any activated backup path). Our restoration scheme works as follows. When link \(l_2\) fails, we find the shortest feasible path \(p\) between \(x\) and \(y\) (shown in dashed line in Fig. 4.2) and
use this path to route traffic around link \((x, y)\). Thus, the restoration path for \(d\) consists of the path from \(s\) to \(x\), path \(p\), and the path from \(y\) to \(t\). Note that if a feasible path between \(x\) and \(y\) cannot be found, then \(d\) cannot be restored. In this restoration scheme, the source node does not need to be informed of the failure of \(l_2\). When \(l_2\) fails, our restoration scheme will compute the backup path between \(x\) and \(y\), set up the backup path using a signaling protocol, and switch the traffic onto the backup path.

An alternative restoration scheme for \(d\) is to compute the shortest feasible path between source \(s\) and destination \(t\) and switch the traffic onto this path when \(l_2\) fails. We call this scheme the *end-to-end restoration scheme*. In contrast, our scheme is a *local restoration scheme* that reroutes the traffic around the failed link instead of finding an end-to-end restoration path. The advantage of our local restoration scheme is that it provides faster restoration than the end-to-end restoration scheme. This is due to two reasons. First, in the end-to-end restoration scheme, the failure detecting node \(x\) needs to send a message to source \(s\) to notify it of the failure of \(l_2\) because \(s\) is responsible for switching the traffic onto the backup path. This is not needed in our local restoration scheme. Second, our local restoration scheme requires shorter time to set up the backup path. This is because the backup path used by our local restoration scheme is found between the end nodes of the second failed link; this backup path is generally shorter than the end-to-end backup path used in the end-to-end restoration scheme.

### 4.3 Numerical Results

In this section, we present some numerical results to show the performance of our hybrid protection/restoration scheme. We use the 47-node 98-link DISTRIBUTED network given in [71]. We first study the capacity efficiency of our backup wavelength reservation scheme and then study the performance of our local restoration scheme for nonsurvivable demands.

#### 4.3.1 Results for the Backup Wavelength Reservation Scheme

First, we compare the backup capacity requirement of dedicated path protection (DPP), shared path protection (SPP), and our hybrid scheme. All three schemes assign a single backup path for each working path. However, they differ in the backup capacity reservation strategy used. DPP does not allow backup capacity sharing, so it can protect all survivable demands against two-link failures.
SPP reserves backup capacity according to equation (1), which exploits backup capacity sharing to protect all demands against single-link failures. Although more capacity efficient than DPP, SPP cannot restore all survivable demands due to backup capacity contention upon a two-link failure. Our hybrid scheme reserves backup capacity according to equation (2), which exploits backup capacity sharing while ensuring all survivable demands can be restored upon a two-link failure.

Fig. 4.3 shows the number of backup wavelengths used by DPP, SPP, and our hybrid scheme for demand sets of different sizes in DISTRIBUTED network. We observe that Hybrid requires less capacity than DPP and requires more capacity than SPP for all demand sets. Hybrid exploits backup capacity sharing to reserve the minimum amount of backup capacity needed to restore all survivable demands. This leads to 25%-27% reduction in backup capacity compared to DPP that does not exploit backup capacity sharing.

Table 4.1 Redundancy of DPP, SPP, and Hybrid in DISTRIBUTED network

<table>
<thead>
<tr>
<th>No. of Demands</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
<th>2500</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPP</td>
<td>0.61</td>
<td>0.59</td>
<td>0.58</td>
<td>0.59</td>
<td>0.58</td>
</tr>
<tr>
<td>Hybrid</td>
<td>0.93</td>
<td>0.90</td>
<td>0.90</td>
<td>0.90</td>
<td>0.89</td>
</tr>
<tr>
<td>DPP</td>
<td>1.23</td>
<td>1.23</td>
<td>1.23</td>
<td>1.23</td>
<td>1.23</td>
</tr>
</tbody>
</table>

Table 4.1 compares the redundancy of DPP, SPP, and Hybrid in DISTRIBUTED network, where redundancy is defined as the ratio of total backup capacity to total working capacity. We observe that
Hybrid has lower redundancy than DPP and higher redundancy than SPP. Hybrid is highly capacity efficient as its redundancy is less than 1.

Next, we consider all possible two-link failures for each demand set and record the percentage of demands that belong to $S_u$, $S_s$, and $S_n$. The results are presented in Table 4.2 where the data shown is the average taken over all possible two-link failures. We observe that for all demand sets, about 92% of the demands are not affected by the two-link failure, less than 7.5% of the demands are survivable demands, and less than 0.5% of the demands are nonsurvivable demands. This shows that on average only about 8% of the demands are affected by a two-link failure. Survivable demands accounts for 94% of the affected demands and can be restored by our hybrid scheme using the pre-planned backup paths. The remaining 6% of the affected demands are nonsurvivable demands that can be restored using our dynamic restoration scheme.

<table>
<thead>
<tr>
<th>No. of Demands</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
<th>2500</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_u$</td>
<td>92.04</td>
<td>92.12</td>
<td>92.14</td>
<td>92.09</td>
<td>92.10</td>
</tr>
<tr>
<td>$S_s$</td>
<td>7.47</td>
<td>7.40</td>
<td>7.39</td>
<td>7.43</td>
<td>7.42</td>
</tr>
<tr>
<td>$S_n$</td>
<td>0.49</td>
<td>0.48</td>
<td>0.47</td>
<td>0.48</td>
<td>0.48</td>
</tr>
</tbody>
</table>

The above results show that our hybrid scheme can protect 99.5% of the demands against two-link failures using pre-reserved backup capacity. This is achieved with a low redundancy of 0.9.

4.3.2 Results for the Dynamic Restoration Scheme

In this section, we compare the performance of our local restoration scheme and the end-to-end restoration scheme in restoring nonsurvivable demands. We consider two scenarios: limited capacity and unlimited capacity. In the limited capacity scenario, we set the link capacity based on our backup capacity reservation scheme. That is, the capacity of a link is equal to the total working capacity on the link plus the total backup capacity reserved on the link. With limited capacity, it may not be possible to restore all the nonsurvivable demands. In the unlimited capacity scenario, we set the link capacity to infinity so that there are enough spare capacity to restore the nonsurvivable demands. Note that even when there is enough capacity, some nonsurvivable demands may not be restored due to
topology reason, i.e., no backup path can be found in the topology after two links have failed. For the DISTRIBUTED network used in our study, failure of restoration due to topology does not occur. That is, both the end-to-end and the local restoration schemes are able to restore all the nonsurvivable demands under the unlimited capacity scenario.

Fig. 4.4 shows the restoration ratio of nonsurvivable demands for the two restoration schemes under limited capacity, where the restoration ratio is defined as the ratio of the number of nonsurvivable demands that can be restored to the total number of nonsurvivable demands. We observe that the end-to-end scheme has higher restoration ratio than the local scheme. This is expected since the end-to-end scheme considers all feasible paths between the source and the destination while the local scheme is restricted to use the intact part of the traffic carrying path as part of the restoration path. We also observe that both schemes have very high restoration ratio: 0.995-0.997 for the end-to-end scheme and 0.950-0.967 for the local scheme. This indicates that the amount of backup capacity reserved by our hybrid scheme not only allows all survivable demands to be restored using pre-planned backup paths but also allows almost all nonsurvivable demands to be restored using dynamic restoration.

Table 4.3 shows the average backup path length of nonsurvivable demands for the two restoration schemes under limited capacity and unlimited capacity. Note that the backup path used in the end-to-end scheme is between the source and the destination while the backup path used in the local scheme is between the end nodes of the second failed link. The data shown in the table is the average taken over
Table 4.3 Comparison of Average Backup Path Length

<table>
<thead>
<tr>
<th>Number of Demands</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
<th>2500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local (Limited Cap.)</td>
<td>3.83</td>
<td>3.82</td>
<td>3.81</td>
<td>3.81</td>
<td>3.82</td>
</tr>
<tr>
<td>Local (Unlimited Cap.)</td>
<td>3.28</td>
<td>3.28</td>
<td>3.29</td>
<td>3.29</td>
<td>3.28</td>
</tr>
<tr>
<td>End-to-End (Limited Cap.)</td>
<td>7.18</td>
<td>7.11</td>
<td>7.07</td>
<td>7.10</td>
<td>7.10</td>
</tr>
<tr>
<td>End-to-End (Unlimited Cap.)</td>
<td>7.16</td>
<td>7.10</td>
<td>7.07</td>
<td>7.10</td>
<td>7.08</td>
</tr>
</tbody>
</table>

all possible two-link failures. We observe that the local scheme has much shorter backup path length than the end-to-end scheme in both capacity settings. This means that the end-to-end scheme takes longer to set up the backup path compared to the local scheme. In addition, the end-to-end scheme suffers the failure notification delay due to the need to notify the source. Thus, our local restoration scheme provides much faster recovery than the end-to-end restoration scheme. We also observe that both schemes have shorter average backup path length in the unlimited capacity case compared to the limited capacity case. This is expected because some links in the limited capacity case cannot be used by a backup path due to lack of spare capacity. This will cause the backup path to take a longer path than the shortest path.

In practice, optical backbone network links are often over-provisioned. In this case, the local restoration scheme will be a better choice than the end-to-end restoration scheme since it can achieve the same restoration ratio as the end-to-end scheme while providing faster restoration speed.

4.4 Conclusion

In this chapter, we propose a hybrid protection/restoration scheme for handling two-link failures in WDM mesh networks. Our scheme associates one backup path for each working path and reserves the minimum amount of backup capacity required to ensure that all survivable demands can be successfully restored using the pre-planned backup paths. For nonsurvivable demands, our local restoration scheme can quickly restore them after the second link failure by computing a backup path around the failed link. The numerical results show that our hybrid scheme can protect all survivable demands against two-link failures with significantly less backup capacity than DPP. And our local restoration scheme provides much faster restoration than the end-to-end restoration scheme for nonsurvivable demands by
establishing shorter backup paths and eliminating the need to notify the source node of the failure.
CHAPTER 5. INTELLIGENT $p$-CYCLE PROTECTION FOR MULTICAST SESSIONS IN WDM NETWORKS

5.1 Introduction

In this chapter, we focus on the problem of protecting dynamic multicast sessions in WDM networks. The dynamic $p$-Cycle (D$p$C) scheme [61] chooses $p$-Cycles from a set of pre-computed short candidate cycles, which cannot adapt to dynamic incoming multicast requests, and has low protection efficiency. We propose an intelligent $p$-Cycle (I$p$C) scheme to provide $p$-cycle protection for dynamic multicast sessions. When a multicast request arrives, a multicast tree is computed for it (using any known algorithm) and then the I$p$C scheme is used to compute a set of high efficiency $p$-cycles on-demand to protect the multicast tree. The proposed I$p$C schemes has the following attractive features.

- It provides fast restoration since pre-configured $p$-cycles are used to protect the multicast tree links.

- It makes efficient use of spare capacity since a set of high efficiency $p$-cycles are computed on demand to protect the multicast tree links.

- Both intra-session sharing and inter-session sharing are achieved since a $p$-cycle can provide protection to links belonging to not only the same multicast tree, but also different multicast trees.

- The capacity efficiency is further improved by combining the existing $p$-cycles whenever possible.

- Assuming sufficient capacity is available in the network, a set of $p$-cycles can always be found to protect any multicast tree as long as the network is 2-edge-connected. This is not true for
tree-based, segment-based, and path-based protection schemes. (Note that segment-based and path-based schemes suffer from the trap topology problem where a backup path cannot be found for a tree segment or tree path even though the network is 2-edge-connected.)

In this chapter, we assume each node is equipped with wavelength converter and capable of converting any input wavelength to any output wavelength. According to this assumption, any lightpath passing through the node may use a converter if necessary and the wavelength assignment is not the key research topic in this work.

### 5.2 Overview of the IpC Scheme

A WDM optical network is represented by a graph $G = (V, E)$, where $V$ and $E$ represent the sets of nodes and links, respectively. A multicast session $R$ is denoted as $\{s, d_1, ..., d_k\}$, where $s$ is the source and $d_i$ is the $i^{th}$ destination. $T$ denotes the multicast tree associated with multicast session $R$. The set of all links on $T$ is denoted as $E_T$ and the set of all nodes on $T$ is denoted as $V_T$. We use directed $p$-cycles to protect a multicast tree since multicast traffic is directed. A directed $p$-cycle can protect a directed link $u \rightarrow v$ if $u \rightarrow v$ is a straddling link of the $p$-cycle or the directed link $v \rightarrow u$ (not $u \rightarrow v$!) is on the $p$-cycle. In either case, the $p$-cycle segment from $u$ to $v$ can be used to route the traffic around the link $u \rightarrow v$ when it fails. $C(u \rightarrow v)$ denotes the free capacity on the directional link $u \rightarrow v$.

Given a multicast tree $T$ and a $p$-cycle $c$ that can protect some link(s) on $T$, we define the efficiency ratio (ER) of $c$ as the ratio of $|PE(c)|$ to $|c|$, where $PE(c)$ denotes the set of links in $E_T$ that are protected by $c$ and $|c|$ denotes the number of links on $c$. It is also possible that this $p$-cycle $c$ can provide protection for upcoming multicast requests. We use $PA(c)$ to denote the links in all multicast requests protected by $c$ ($PE(c) \subseteq PA(c)$). Note that $|c|$ is equal to the number of wavelength channels used by $c$. Clearly, the larger is ER, the more efficient is $c$ in protecting the tree links.

Given a multicast tree $T$, our IpC algorithm, formally presented in Algorithm 3, is used to find a set $PC$ of $p$-cycles to protect $T$ so that every link in $E_T$ is protected by some $p$-cycle in $PC$. The framework of the algorithm is as follows.
(1) For every link in $E_T$, there are two options to protect it: finding a new $p$-cycle for it, or extending an existing $p$-cycle in $PC$ to protect it. Hence, we can find at most $2^*|E_T|$ $p$-cycles for all links in $E_T$.

(2) Let $p$ be the $p$-cycle with the maximum ER among all the $p$-cycles found in (1). We add $p$ to $PC$ and remove all links in $E_T$ that can be protected by $p$.

(3) We combine $p$ with the other $p$-cycles in $PC$ to reduce the wavelength usage of the $p$-cycles.

(4) If $E_T$ becomes empty, $PC$ is returned; otherwise, the above steps are repeated.

Three algorithms are used by our Ipc algorithm. Algorithm 2 and Algorithm 3 are used in Step (1) to compute a new $p$-cycle and an extended $p$-cycle to protect a link in $E_T$, respectively. Algorithm 4 is used in Step (3) to combine $p$ with the other $p$-cycles in $PC$. In the following, we discuss the detail of Algorithm 1. We then describe Algorithm 2, Algorithm 3, and Algorithm 4 in the next three subsections.

Algorithm 3 Find $p$-Cycles to Protect Multicast Tree $T$

1: $PC = \phi$
2: Remove every link in $E_T$ that can be protected by an existing $p$-cycle
3: while ($E_T \neq \phi$) do
4: $Temp = \phi$
5: for every $e \in E_T$ do
6: Find a new $p$-cycle $p_{new}$ for $e$ using Algorithm 4 and add $p_{new}$ to $Temp$
7: if $PC \neq \phi$ then
8: Find an extended $p$-cycle $p_{ext}$ for $e$ using Algorithm 5 and add $p_{ext}$ to $Temp$
9: if $Temp = \phi$ then
10: Return NULL
11: Find $p$ in $Temp$ with the maximum ER and add $p$ to $PC$
12: if $p$ is extended from a $p$-cycle $p_i$ in $PC$ then
13: Remove $p_i$ from $PC$
14: Remove the links in $EP(p)$ from $E_T$
15: Update $PC$ based on $p$ using Algorithm 6
16: Return $PC$

The purpose of Algorithm 1 is to find a set $PC$ of $p$-cycles such that every link in $E_T$ is protected by some $p$-cycle in $PC$. Since some of the links in $E_T$ may be protected by some existing $p$-cycles
formed for existing multicast/unicast sessions, we first remove all links that can be protected by reusing the existing \( p \)-cycles from \( E_T \) (in line 2). \( \forall e \in E_T, \) if \( p \)-cycle \( c \) can protect \( e \), \( PC(c) = PC(c) \cup e \).

Then we start the process of iteratively building \( p \)-cycles for every link \( e \) in \( E_T \). We use set \( PC \) to store newly built \( p \)-cycles.

In line 6, we find a new \( p \)-cycle for link \( e \) using Algorithm 4. Basically, Algorithm 4 finds a set of \( p \)-cycles that can protect \( e \) and returns the \( p \)-cycle with the maximum ER. In lines 7-9, if \( PC \) is not empty, we find an extended \( p \)-cycle for \( e \) using Algorithm 5. Basically, Algorithm 5 finds the maximum ER \( p \)-cycle that is extended from a \( p \)-cycle in \( PC \). After the for loop in lines 5-10 are executed, every link \( e \) in \( E_T \) has at most two candidate \( p \)-cycles, \( p_{\text{new}} \) and \( p_{\text{ext}} \). All these \( p \)-cycles are stored in set \( Temp \).

In line 11-13, if the set \( Temp \) is empty, then NULL is returned. This occurs when no \( p \)-cycles could be found due to lack of spare capacity in the network. As a result, \( T \) cannot be protected.

In line 14-17, we choose \( p \)-cycle \( p \) with the maximum ER from \( Temp \) and add \( p \) into \( PC \). Furthermore, if \( p \) is extended from a \( p \)-cycle in \( PC \), we remove the original \( p \)-cycle from \( PC \). Since \( p \) may protect one or more links in \( E_T \), we remove all these links from \( E_T \) in line 18.

In line 19, Algorithm 6 is used to update \( PC \) based on \( p \). Specifically, Algorithm 6 combines \( p \) with the other \( p \)-cycles in \( PC \) to reduce the wavelength usage of the \( p \)-cycles without affecting the protection of the links in \( E_T \).

When \( E_T \) becomes empty, the algorithm returns \( PC \), which contains a set of \( p \)-cycles that protect all the links in \( E_T \). \( \forall p \in PC, \forall e \in p \), the free capacity of directional link \( e \) needs to be decreased by one. \( \forall p \in PC \), we also need to update the protected link set \( PA(p) \).

### 5.3 Finding New \( p \)-Cycles

We now present Algorithm 4, which finds a new \( p \)-cycle for link \( e = n_1 \rightarrow n_2 \) in \( E_T \). The new \( p \)-cycle contains link \( n_2 \rightarrow n_1 \) and therefore can protect \( e \). The basic idea is to perform breath-first searches from \( n_1 \) and \( n_2 \) at the same pace until these two searches arrive at some common node(s), indicating the finding of one or more \( p \)-cycles. Among the found \( p \)-cycles, the one with the maximum ER is returned.
The algorithm uses the following notations:

- $G_1$ and $G_2$: storing all nodes that have been reached by the breadth first searches from nodes $n_1$ and $n_2$, respectively. Initially, $G_1 = \{n_1\}$ and $G_2 = \{n_2\}$.

- $G^t_1$ and $G^t_2$: storing nodes which were added into $G_1$ and $G_2$ in the most recent step of the breadth first searches. Initially, $G^t_1 = \{n_1\}$ and $G^t_2 = \{n_2\}$.

- $PL_i$: node $n_i$’s parent list, storing the nodes through which $n_i$ is connected to $n_1$ or $n_2$. The first node in the list is called the primary parent and the other nodes in the list are called secondary parents.

The detail of the algorithm is explained as follows. To find a $p$-cycle that includes link $n_2 \rightarrow n_1$, there must be a free wavelength on this link. Line 2-4 check whether this condition is met. If not, NULL is returned to indicate that we cannot find a $p$-cycle to protect link $n_1 \rightarrow n_2$.

Before performing the Breadth First Searches (BFS), we remove link $(n_1, n_2)$ from $G$ in line 5 to make sure BFS does not consider this link.

Lines 6-11 perform the BFS starting from $n_1$ and $n_2$, respectively, until (1) some node(s) is found to be in both $G_1$ and $G_2$ (i.e. $G_1 \cap G_2 \neq \emptyset$), which indicates at least one cycle has been found, or, (2) the BFS could not continue ($G^t_1 = \emptyset$ or $G^t_2 = \emptyset$), which means there is not enough spare capacity to create a $p$-cycle. During the breadth-first search, we need to make sure that the link has the free capacity on the correct direction. Specifically, in line 7, $\forall n_i \in G^t_1$, to run breadth-first search for one step to access node $n_j$, we need to make sure there is free capacity in link $n_i \rightarrow n_j$. And in line 9, $\forall n_i \in G^t_2$, to run breadth-first search for one step to access node $n_j$, we need to make sure there is free capacity in link $n_j \rightarrow n_i$.

In line 12-14, NULL is returned if there is no common node between $G_1$ and $G_2$, which indicates no $p$-cycle could be found due to lack of spare capacity.

Lines 15-26 update the parent list for every $n_i \in G_1 \cap G_2$ as follows. If $n_i$’s primary parent is in $G_1$ (or $G_2$), then we consider every node $n_j$ in the set $G^t_2 - (G_1 \cap G_2)$ (or $G^t_1 - (G_1 \cap G_2)$). If there is an edge between $n_i$ and $n_j$ in $G$, then $n_j$ is added to $n_i$’s parent list. This is because the facts that $n_j$ has been reached in the most recent step of BFS from $n_2$ (or $n_1$) and there is an edge between $n_i$ and
Algorithm 4  Find a new $p$-cycle for link $e = n_1 \rightarrow n_2$

1: $G_1' = G_1 = \{n_1\}$; $G_2' = G_2 = \{n_2\}$  
   \hspace{1cm} $PL_i = \phi$ for all node $i \in V$
2: if link $n_2 \rightarrow n_1$ has no free wavelength then  
   3: Return NULL
4: Remove link $(n_1, n_2)$ from $G$
5: repeat
6: Run breadth-first search for one step for $\forall n_i \in G_1'$
7: Update $G_1$ and $G_1'$; Update $PL_j$ for $\forall n_j \in G_1'$
8: Run breadth-first search for one step for $\forall n_i \in G_2'$
9: Update $G_2$ and $G_2'$; Update $PL_j$ for $\forall n_j \in G_2'$
10: until $G_1 \cap G_2 \neq \emptyset$ or $G_1' = \emptyset$ or $G_2' = \emptyset$
11: if $G_1 \cap G_2 = \emptyset$ then  
12: Return NULL
13: for $(\forall n_i \in G_1 \cap G_2)$ do
14: if $n_i$’s primary parent $\in G_1$ then  
15: \hspace{1cm} $tmpSet = G_2' - (G_1 \cap G_2)$
16: else
17: \hspace{1cm} $tmpSet = G_1' - (G_1 \cap G_2)$
18: for $(\forall n_j \in tmpSet)$ do
19: \hspace{2cm} if link $(n_i, n_j) \in E$ then  
20: \hspace{3cm} $PL_i = PL_i \cup \{n_j\}$
21: for $(\forall n_i \in G_1 \cap G_2)$ do
22: \hspace{1cm} $Path_i = GetPath(n_i)$
23: for $(\forall secondary parent n_{i}^{j} \in PL_i)$ do
24: \hspace{2cm} $Path_{i}^{j} = \{n_i\} \cup GetPath(n_{i}^{j})$;
25: Combine $Path_i$ and $Path_{i}^{j}$ to build cycle $C_{i}^{j}$
26: $p_{new} = \text{ the } p\text{-cycle with the maximum ER among all the } p\text{-cycles built}$
27: Return $p_{new}$
28: Function $GetPath(n_t)$
29: $Path = \{n_t\}$
30: while $(n_t$’s parent list $PL \neq \phi$) do
31: \hspace{1cm} $n_t = \text{ the primary parent in } PL_t$.
32: \hspace{1cm} $Path = Path \cup \{n_t\}$
33: Return $Path$
In Lines 27-33, for every \( n_i \in G_1 \cap G_2 \), if its primary parent is in \( G_1 \) (or \( G_2 \)), then we find its only path \( \text{Path}_i \) to root \( n_1 \) (or \( n_2 \)) via its primary parent and \( |\text{PL}_i| \) paths from \( n_i \) to the other root \( n_2 \) (or \( n_1 \)) via its secondary parents. After we get all these paths, we combine \( \text{Path}_i \) with each of the other \( |\text{PL}_i| - 1 \) paths to form \( |\text{PL}_i| - 1 \) \( p \)-cycles. Note that all formed \( p \)-cycles have link \( n_2 \rightarrow n_1 \) as an on-cycle link. Finally, Lines 34-35 select the \( p \)-cycle \( p_{\text{new}} \) with the maximum ER from all the formed \( p \)-cycles and return \( p_{\text{new}} \).

Function \( \text{GetPath} \) is used by Algorithm 4 and is defined in Lines 37-43. This function finds the path from the input node to one root (\( n_1 \) or \( n_2 \)) by following the node’s primary parent step by step, and returns the path.

We illustrate Algorithm 4 using the example shown in Fig. 5.1. To find a new \( p \)-cycle for link \( n_1 \rightarrow n_2 \), we first remove link \( (n_1, n_2) \) from the graph. Next, we perform BFS from \( n_1 \) and \( n_2 \) at the same pace. After one step of BFS, \( G_1 = \{n_1, n_3\} \) and \( G_2 = \{n_2, n_5, n_6, n_7\} \). After two steps of BFS, \( G_1 = \{n_1, n_3, n_4\} \) and \( G_2 = \{n_2, n_5, n_6, n_7, n_4, n_8\} \). We stop the BFS now since \( G_1 \cap G_2 = \{n_4\} \neq \phi \). At this time, \( \text{PL}_4 = \{n_3, n_5, n_6\} \), where \( n_3 \) is \( n_4 \)'s primary parent and \( n_5, n_6 \) are \( n_4 \)'s secondary parents. Next, \( n_8 \) is added to \( \text{PL}_4 \) according to lines 15-26. Thus, \( n_8 \) becomes the third secondary parent of \( n_4 \). Since \( n_4 \)'s primary parent \( n_3 \) is in \( G_1 \), there is only one path \( \text{Path}_4 = n_4 \rightarrow n_3 \rightarrow n_1 \) from \( n_4 \) to \( n_1 \). On the other hand, since \( n_4 \) has three secondary parents (\( n_5, n_6, n_8 \)), it has three paths to \( n_2 \), which are \( \text{Path}^1_4 = n_4 \rightarrow n_5 \rightarrow n_2 \), \( \text{Path}^2_4 = n_4 \rightarrow n_6 \rightarrow n_2 \), and \( \text{Path}^3_4 = n_4 \rightarrow n_8 \rightarrow n_7 \rightarrow n_2 \). Therefore, we can find three \( p \)-cycles for link \( n_1 \rightarrow n_2 \) by combining \( \text{Path}_4 \) with \( \text{Path}^1_4 \), \( \text{Path}^2_4 \) and \( \text{Path}^3_4 \), respectively. The resulting \( p \)-cycles are \( \{n_1 \rightarrow n_3 \rightarrow n_4 \rightarrow n_5 \rightarrow n_2 \rightarrow n_1\} \), \( \{n_1 \rightarrow n_3 \rightarrow n_4 \rightarrow n_6 \rightarrow n_2 \rightarrow n_1\} \) and \( \{n_1 \rightarrow n_3 \rightarrow n_4 \rightarrow n_8 \rightarrow n_7 \rightarrow n_2 \rightarrow n_1\} \). Among these
Figure 5.1 Example of Finding New \( p \)-Cycles

Figure 5.2 Extend an existing \( p \)-cycle to protect link \( e \)

three \( p \)-cycles, the one with the maximum ER will be returned by Algorithm 4.

### 5.4 Extending Existing \( p \)-Cycles

In this section, we present Algorithm 5, which finds a \( p \)-cycle that is extended from a \( p \)-cycle in \( PC \) to protect a link \( e = n_1 \rightarrow n_2 \) in \( ET \).

The basic idea of Algorithm 5 is as follows. For every \( p \)-cycle \( p \in PC \), if \( p \) can be extended to include link \( n_2 \rightarrow n_1 \) or include nodes \( n_1 \) and \( n_2 \), then extension is performed to produce a new \( p \)-cycle that can protect \( e \), which is added to set \( EPC \). After all \( p \)-cycles in \( PC \) have been considered, the algorithm chooses from \( EPC \) the \( p \)-cycle with the maximum ER and returns it.

Consider a \( p \)-cycle \( p \) and a link \( e \), we can extend \( p \) to protect \( e \) according to the following two cases.

**Case I: One endnode of \( e \) is already on \( p \)**

The left graph in Fig. 5.2 shows an example, where link \( e = n_1 \rightarrow n_2 \) needs to be protected and one endnode of the link (i.e., \( n_2 \)) is already on \( p \). If \( p \) can be extended to also include the other endnode
Algorithm 5  Find an extended $p$-cycle for link $e = n_1 \rightarrow n_2$

1: $EPC = \emptyset$;
2: for $(\forall p_j \in PC)$ do
3:     for $(\forall n_k$ on $p_j)$ do
4:         $u = n_k$, $v =$ next-hop node of $u$ on $p_j$;
5:         while (no node on segment $u \rightarrow v$, excluding $u$ and $v$, belongs to the multicast tree) $\land$ $(v \neq u)$ do
6:             if neither endnode of $e$ is on $p_j$ then
7:                 $n_1 =$ the virtual node representing $e$
8:         Find $Path_1$: the shortest path from $u$ to $n_1$ that is link-disjoint with $p_j$
9:         Find $Path_2$: the shortest path from $n_1$ to $v$ that is link-disjoint with $p_j$ and $Path_1$
10:        Goto 16 if $Path_1$ or $Path_2$ cannot be found
11:       Extend $p_j$ to cycle $p_{u,v}^j$ by replacing segment $u \rightarrow v$ with the concatenation of $Path_1$ and $Path_2$
12:       if $p_{u,v}^j$ can protect $e$ then
13:           Add $p_{u,v}^j$ to $EPC$
14:       $v =$ the next-hop node of $v$ on $p_j$
15:       if $EPC = \emptyset$ then
16:           Return NULL
17:       $p_{ext} =$ the $p$-cycle with the maximum ER among all $p$-cycles in $EPC$.
18:       Return $p_{ext}$
of the link (i.e., \( n_1 \)), then \( e \) will be protected. We extend \( p \) in the following way. Selecting two nodes \( u \) and \( v \) on \( p \) and two paths \( u \rightarrow n_1 \) and \( n_1 \rightarrow v \) such that the following rules are followed. Then replacing segment \( u \rightarrow v \) on \( p \) with the concatenation of \( u \rightarrow n_1 \) and \( n_1 \rightarrow v \).

(R1) Path \( u \rightarrow n_1 \) and path \( n_1 \rightarrow v \) should be link-disjoint with each other, and also link-disjoint with \( p \). Otherwise, replacing segment \( u \rightarrow v \) with the concatenation of \( u \rightarrow n_1 \) and \( n_1 \rightarrow v \) will not result in a \( p \)-cycle.

(R2) After the extension, link \( n_1 \rightarrow n_2 \) should not become an on-cycle link. Otherwise, if this link fails, an alternative path from \( n_1 \) to \( n_2 \) cannot be provided by the \( p \)-cycle. On the other hand, if link \( n_1 \rightarrow n_2 \) becomes a straddling link or link \( n_2 \rightarrow n_1 \) becomes an on-cycle link after the extension, then the extended \( p \)-cycle can protect link \( e \).

(R3) There should be no multicast tree nodes appearing on segment \( u \rightarrow v \). Otherwise, replacing this segment may cause some links on the multicast tree to lose protection.

In Algorithm 5, the for loop from line 3 to line 18 computes a set of \( p \)-cycles extended from an existing \( p \)-cycle \( p_j \in PC \) to protect link \( e = n_1 \rightarrow n_2 \). Here, we consider every possible pair of \( u \) and \( v \) on \( p_j \), one by one. Line 5 checks if there is any multicast tree node appearing on segment \( u \rightarrow v \) to ensure rule (R3) is followed. Lines 9 and 10 check if the paths \( u \rightarrow n_1 \) and \( n_1 \rightarrow v \) are link-disjoint, and if they are link-disjoint with \( p_j \), to ensure rule (R1) is followed. When searching the shortest path, we just need consider the directed links with free capacity. Specifically, for line 9, all directional links on the path(from \( u \) to \( n_1 \)) must have free capacity. In line 10, we also check that all directional links on the path(from \( n_1 \) to \( v \)) have free capacity. Line 13 checks if rule (R2) is followed. All \( p \)-cycles extended from \( p_j \) that have passed the above checks are put in set \( EPC \).

**Case II: No endnode of \( e \) is on \( p \)**

The right graph in Fig. 5.2 shows an example, where link \( e = n_1 \rightarrow n_2 \) needs to be protected and none of its endnodes is on \( p \). We deal with this case by viewing \( e \) as a virtual node and adding this virtual node into an existing \( p \)-cycle using the method described in Case I. Lines 6-8 in Algorithm 5 handles this case by setting \( n_1 \) to be the virtual node. In this case, the extended \( p \)-cycle must have link
n_2 \rightarrow n_1 as an on-cycle link in order to protect e. That is, the direction of the extended p-cycle must be opposite to the direction of e.

In Algorithm 5, for every p-cycle p_j \in PC, a set of extended p-cycles are computed and put into EPC based on the two cases described above. If no extended p-cycles can be found due to lack of spare capacity, then NULL is returned. This case is dealt with in lines 20-22. Otherwise, the p-cycle with the maximum ER among all p-cycles in EPC is selected to protect e and is returned in lines 23-24.

5.4.1 Updating the p-Cycle Set PC

In this section, we describe Algorithm 6, which is used by Algorithm 3 to update the p-cycle set PC after a p-cycle p is added to PC. The update involves combining p with the other p-cycles in PC in a way that reduces the wavelength usage of the p-cycles while not affecting the existing protections of the links in ET. The combining of the p-cycles continue repeatedly until no more combinations can be done.

Consider two p-cycles p and p_i, we can combine them to create a new p-cycle according to the following two cases.

**Case I: p and p_i have one or more common edges**

In this case, p and p_i have one or more common edges with opposite directions. An example is shown in Fig. 5.3. In this example, p = n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_5 \cdots \rightarrow n_j \cdots \rightarrow n_m \rightarrow n_1 and p_i = n_3 \rightarrow n_2 \rightarrow n_1 \rightarrow n_k \cdots \rightarrow n_i \cdots \rightarrow n_4 \rightarrow n_3 share two common edges with opposite directions. p and p_i can be combined to obtain a new p-cycle p_c = n_3 \rightarrow n_5 \cdots \rightarrow n_j \cdots \rightarrow n_m \rightarrow n_1 \rightarrow n_k \cdots \rightarrow n_i \cdots \rightarrow n_4 \rightarrow n_3. If p_c can protect all the tree edges that are protected by either p or p_i (i.e., PE(p) \cup PE(p_i) \subseteq PE(p_c)), then p_c can provide the same protection with less wavelength usage. Since the new p-cycle p_c is more efficient, we will add p_c into PC and remove p and p_i from PC.

**Case II: p and p_i have two common nodes**

When p and p_i have two common nodes, they can be combined to create a new p-cycle as shown in Fig. 5.4. In this Figure, p = n_1 \rightarrow \ldots n_6 \rightarrow \ldots n_3 \rightarrow \ldots n_5 \cdots \rightarrow n_j \cdots \rightarrow n_m \cdots \rightarrow n_1
and \( p_i = n_3 \rightarrow \ldots n_2 \rightarrow n_1 \rightarrow \ldots n_k \rightarrow \ldots n_4 \rightarrow \ldots n_1 \) have two common nodes: \( n_1 \) and \( n_3 \). Two different new \( p \)-cycles can be obtained by combining \( p \) and \( p_i \). The first one is 
\[
p_{c1} = n_1 \rightarrow \ldots n_6 \rightarrow \ldots n_3 \rightarrow \ldots n_2 \rightarrow \ldots n_1.
\]
The second one is 
\[
p_{c2} = n_1 \rightarrow \ldots n_k \rightarrow \ldots n_i \rightarrow \ldots n_4 \rightarrow \ldots n_3 \rightarrow \ldots n_5 \rightarrow \ldots n_j \rightarrow \ldots n_m \rightarrow \ldots n_1.
\]
If one of the two new \( p \)-cycles can protect all the tree edges that are protected by either \( p \) or \( p_i \), then it can provide the same protection as \( p \) and \( p_i \) with less wavelength usage. Since this new \( p \)-cycle is more efficient, we will add it into \( PC \) and remove \( p \) and \( p_i \) from \( PC \).

**Algorithm 6 Update set \( PC \) based on the newly added \( p \)-cycle \( p \)**

1: \( Size_{Be} = |PC|; Size_{Af} = 0; \)
2: \( \textbf{while} \ Size_{Be} > Size_{Af} \textbf{ do} \)
3: \( \quad Size_{Be} = |PC|, NC = true, i = 1; \)
4: \( \quad \textbf{while} NC \text{ and } i < |PC| \textbf{ do} \)
5: \( \quad \quad \textbf{if} \ p \text{ and } p_i \text{ have one or more common edges then} \)
6: \( \quad \quad \quad p_e = \text{combination of } p \text{ and } p_i \)
7: \( \quad \quad \quad \textbf{if} \ (PE(p) \cup PE(p_i) \subseteq PE(p_c)) \text{ \&\& Simple}(p_c) \textbf{ then} \)
8: \( \quad \quad \quad \quad NC = false; \)
9: \( \quad \quad \quad \textbf{if} \ NC \text{ and } p \text{ and } p_i \text{ have two common nodes then} \)
10: \( \quad \quad \quad \quad p_{c1} = \text{First combination of } p \text{ and } p_i \)
11: \( \quad \quad \quad \quad p_{c2} = \text{Second combination of } p \text{ and } p_i \)
12: \( \quad \quad \quad \quad \textbf{if} \ (PE(p) \cup PE(p_i) \subseteq PE(p_{c1})) \text{ \&\& Simple}(p_{c1}) \textbf{ then} \)
13: \( \quad \quad \quad \quad \quad NC = false; \)
14: \( \quad \quad \quad \quad p_c = p_{c1} \)
15: \( \quad \quad \textbf{else} \)
16: \( \quad \quad \quad \quad \textbf{if} \ (PE(p) \cup PE(p_i) \subseteq PE(p_{c2})) \text{ \&\& Simple}(p_{c2}) \textbf{ then} \)
17: \( \quad \quad \quad \quad \quad NC = false; \)
18: \( \quad \quad \quad \quad p_c = p_{c2} \)
19: \( \quad \quad \quad \textbf{if} \ !NC \textbf{ then} \)
20: \( \quad \quad \quad \quad PC = PC - \{p\} - \{p_i\} + \{p_c\} \)
21: \( \quad \quad \quad i++; \)
22: \( \quad Size_{Af} = |PC| \)
23: \( \quad p = p_c \)
24: \( \textbf{Return} \ PC \)

In our algorithm, non-simple \( p \)-cycle is not allowed and each combined \( p \)-cycle found in the loop will be checked to guarantee it is a simple \( p \)-cycle by function \( \text{Simple}(p) \). That means, for each directional link \( u \rightarrow v \in p \), it can only show up once and meanwhile directional link \( v \rightarrow u \) can not show
Figure 5.3 Combining two $p$-cycles with one or more common edges.

Figure 5.4 Combining two $p$-cycles with two common nodes.

up in $p$-cycle $p$.

In Algorithm 6, the while loop in lines 4-28 checks whether $p$-cycle $p$ can be combined with another $p$-cycle $p_i$ in $PC$ by considering the two cases described above. (Lines 5-10 check Case I and lines 11-23 check Case II.) If a $p$-cycle $p_i$ can be found to combine with $p$, then the combined new $p$-cycle $p_c$ is added to $PC$, replacing $p$ and $p_i$ (Lines 24-26). Once a combination is performed, we assign $p_c$ to $p$ (in line 30) and repeat the above process until no more combinations could be done.
5.4.2 Connection Release

Once the requested service R completes, the created multicast connection needs to be released. To tear down a working multicast tree $T$, which is created corresponding to R, basically we need two steps. The first step is to release the capacity allocated by the multicast tree $T$; the second step is to update p-Cycles originally providing the protection to $T$.

Step 1: $\forall e \in T$, release the wavelength reserved by $T$ and the free capacity of $e$ needs to be increased by one.

Step 2: $\forall e \in T$, there exists one p-cycle $p_e$ originally protecting $e$. Suppose the set of all links protected by $p_e$ is $PA(p_e)$, the link $e$ will be removed from set $PA(p_e)$. If the set $PA(p_e)$ is empty, then we need to tear down the p-cycle $p_e$. That is to say, the p-cycle $p_e$ is torn down only when it is not used to protect any working wavelength. To tear down p-cycle $p_e$, $\forall e \in p_e$, we need to release the wavelength reserved by this $p_e$ and the free capacity of directional link $e$ will be increased by one.

5.5 Numerical Results

We conduct simulations to compare the performance of our proposed IpC scheme with another p-cycle based multicast protection scheme DpC [61], where each link has two pre-selected p-cycles. Two networks, the NSF network (Fig. 5.5) and the COST239 network (Fig. 5.6), are used in the simulations. In each simulation run, a set of randomly generated multicast requests are loaded to the network to compare the performance of IpC and DpC. For each multicast request, the source node and the desti-
nation nodes are randomly selected. For NSF network, the number of destination nodes are randomly generated in the range $[3, 6]$. For COST239 network, the number of destination nodes are randomly generated in the range $[2, 5]$. The algorithm for computing a multicast tree for a given multicast request is given in the appendix. The weight of each link is 1. The performance of the IpC and DpC are compared under two simulation settings: unlimited link capacity and limited link capacity. We also study the computation time for each multicast request under different traffic load in these two networks. The simulation results are presented in the next three subsections.

5.5.0.1 Unlimited Link Capacity

First, we compare the performance of the two algorithms when the network link capacity is set to infinity. In this case, all multicast requests can be satisfied. The performance metric we use is the total number of wavelength channels used by all the $p$-cycles for protecting the multicast sessions.

In Fig 5.7, we show the performance of IpC and DpC under different traffic load for NSF network, where the traffic load varies from 1000 to 6000 multicast requests. The figure shows that IpC uses significantly less wavelength channels than DpC under all traffic loads. Specifically, IpC achieves a 24.5%-24.8% reduction in wavelength usage over DpC. The reason IpC performs better than DpC is two fold. First, IpC computes $p$-cycles on demand while DpC chooses $p$-cycles from pre-computed $p$-cycles. Second, IpC always selects high efficiency cycles while DpC uses short cycles which tend to have low efficiency since short cycles tend to have few straddling links.

In Fig. 5.8, we compare the performance of IpC and DPC for COST239 Network. The results
Figure 5.7  Wavelength Usage of IpC and DpC in NSF Network

Figure 5.8  Wavelength Usage of IpC and DpC in COST239 Network
again show that IpC is much more capacity efficient than DpC. The capacity saving of IpC over DpC ranges from 28.5% to 29.2%. For IpC, the number of wavelength channels used to protect the multicast trees is even less than that used by the multicast trees, which is not the case in NSF network. This is because COST239 network is denser than NSF network. Consequently, the p-cycles calculated by IpC have a higher probability of containing more straddling links which leads to better protection efficiency.

5.5.0.2 Limited Link Capacity

Next, we compare the performance of the two algorithms when the capacity of the directional link in the network is set to 16. That is, every directional link supports 16 wavelength channels. In this case, some multicast requests may be blocked because either the multicast tree cannot be established or the p-cycles for protecting the tree links cannot be created due to lack of wavelengths. The performance metric we use is the reject ratio, which is defined as the number of rejected multicast requests to the total number of multicast requests.

In each simulation run, 5000 randomly generated multicast requests are loaded to the network and the reject ratio is computed at the end of the simulation run. The arrival of multicast requests follows Poisson distribution with \( \lambda \) requests per second and the duration of the request is exponentially distributed with a mean of \( 1/\mu \). The traffic load measured in erlangs is \( \lambda/\mu \). For each traffic load, 10 simulations are conducted and the average reject ratio is plotted in Fig 5.9 and Fig 5.10.

In Fig 5.9, we compare the reject ratio of IpC and DpC under different traffic load in NSF network. The results show that IpC achieves lower reject ratio than DpC under all traffic loads. The reason IpC performs better than DpC is that IpC computes p-cycles on demand and prefers long p-cycles while DpC chooses p-cycles from short pre-computed p-cycles. When the capacity of the network link is limited, the long p-cycles used by IpC tend to spread the wavelength usage across the whole network. While the short p-cycles used by DpC tend to consume the wavelengths in areas of heavy traffic, which blocks future multicast requests. The maximum difference between the reject ratio of DpC and the reject ratio of IpC is 17.3%, which occurs at the load of 40 erlangs. The average difference between the two reject ratios is 10.9%.

In Fig 5.10, we compare the reject ratio of IpC and DpC under different traffic load in COST239
Network. Again, the performance of IpC is better than that of DpC. In addition, the performance improvement of IpC over DpC is higher than that in NSF network. This is because the COST239 Network is denser so that there exists more high efficiency cycles which could be found by IpC. Thus, the $p$-cycles selected by IpC will provide even larger advantage than the short $p$-cycles used by DpC.

**5.5.0.3 Computation Time**

In this section, we study the computation time in millisecond for each request in both networks with different traffic load. We use java language to implement the IpC on a computer with Intel 3.0GHZ
Table 5.1 Computation Time(ms) under different traffic load in NSF

<table>
<thead>
<tr>
<th>Erlang</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>125</td>
<td>265</td>
<td>156</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>187</td>
<td>188</td>
<td>141</td>
</tr>
<tr>
<td>Mean</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>13</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>125</td>
<td>265</td>
<td>156</td>
<td>125</td>
<td>125</td>
<td>94</td>
<td>187</td>
<td>188</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>94</td>
<td>16</td>
<td>62</td>
<td>109</td>
<td>94</td>
<td>63</td>
<td>125</td>
<td>16</td>
<td>46</td>
<td>141</td>
</tr>
<tr>
<td>3</td>
<td>47</td>
<td>78</td>
<td>94</td>
<td>125</td>
<td>47</td>
<td>109</td>
<td>25</td>
<td>62</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>31</td>
<td>47</td>
<td>109</td>
<td>0</td>
<td>16</td>
<td>31</td>
<td>15</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>0</td>
<td>15</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>31</td>
<td>16</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>31</td>
<td>15</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>31</td>
<td>16</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>31</td>
<td>0</td>
<td>16</td>
<td>16</td>
<td>15</td>
<td>31</td>
<td>16</td>
<td>47</td>
<td>62</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>31</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>47</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>31</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>15</td>
<td>63</td>
<td>16</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>16</td>
<td>15</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>78</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

The result in NSF network is shown in table 5.1. We collect the maximum and the mean computation time for one multicast request. We also record the computation time for the first 10 multicast requests. As we can see from the table, the maximum computation time for one request always occurs among the first 3 requests. That is because few p-Cycles exists at the beginning and most links in these multicast requests cannot be protected by existing p-Cycles. So for each link in the multicast trees, we need to find a new p-Cycle to protect it, which needs more time. Once IP found enough p-Cycles, most of the links in the following multicast request can be protected by these existing p-Cycles. So the computation time of one request will decrease with the increased number of requests. For some requests, the computation time is even 0 because all links in the multicast request can be protected by existing p-cycles.

The result in COST239 network is shown in table 5.2. The maximum and mean computation time for one multicast request and the computation time for the first 10 multicast requests are recorded. As we can see from the table, the maximum computation time for one request always occurs among the first 3 requests. Compared with the mean computation time in NSF network, the mean computation time in COST239 network is less because the size of multicast request in COST239 is smaller.

Algorithm 7 is used to generate the multicast tree.

In each round, we find the shortest path $SP$ between any node in set $T_s$ and any node in set $UT_d$. We then add all nodes on $SP$ to $T_s$, add all edges on $SP$ to $P$, and remove the last node on $SP$ from CPU and 1.5GB of memory.
Table 5.2 Computation Time under different traffic load in COST239

<table>
<thead>
<tr>
<th>Erlang</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>125</td>
<td>172</td>
<td>109</td>
<td>109</td>
<td>187</td>
<td>141</td>
<td>219</td>
<td>125</td>
<td>109</td>
<td>125</td>
</tr>
<tr>
<td>Mean</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>125</td>
<td>63</td>
<td>109</td>
<td>109</td>
<td>187</td>
<td>141</td>
<td>219</td>
<td>125</td>
<td>78</td>
<td>125</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>172</td>
<td>32</td>
<td>32</td>
<td>16</td>
<td>47</td>
<td>16</td>
<td>63</td>
<td>94</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>31</td>
<td>15</td>
<td>15</td>
<td>62</td>
<td>94</td>
<td>47</td>
<td>47</td>
<td>109</td>
<td>63</td>
</tr>
<tr>
<td>4</td>
<td>78</td>
<td>47</td>
<td>63</td>
<td>32</td>
<td>94</td>
<td>15</td>
<td>15</td>
<td>78</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>0</td>
<td>62</td>
<td>62</td>
<td>0</td>
<td>0</td>
<td>31</td>
<td>15</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>47</td>
<td>0</td>
<td>63</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>46</td>
<td>15</td>
<td>31</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>31</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>47</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>16</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>16</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

Algorithm 7 Find a Multicast Tree for a Multicast Session \( R = \{ s, d_1, d_2, \ldots, d_k \} \)

1: \( T_s = \{ s \} \), \( UT_d = \{ d_1, d_2, \ldots, d_k \} \), \( P = \phi \)
2: \( \forall s_i \in T_s, \forall d_j \in UT_d \), find the shortest path between \( s_i \) and \( d_j \).
3: Among all paths found above, select the shortest one: \( SP = \{ s_i \rightarrow \cdots \rightarrow d_j \} \)
4: \( \forall \text{node } n_i \in SP, T_s = T_s \cup \{ n_i \}, UT_d = UT_d - \{ d_j \} \)
5: \( \forall \text{edge } n_i \rightarrow n_j \in SP, P = P \cup \{ n_i \rightarrow n_j \} \).
6: if \( UT_d \neq \emptyset \) then
7: goto Step 2.
8: Return \( P \).
When $UT_d$ becomes empty, we have found a multicast tree for $R$, where the edges of the tree are stored in $P$.

5.6 Conclusion

In this chapter, we propose an Intelligent $p$-Cycle (IpC) scheme to provide $p$-cycle based protection for dynamic multicast sessions. The main feature of IpC is that it dynamically computes high-efficiency $p$-cycles to protect multicast sessions as they arrive so that spare capacity is used efficiently. The capacity efficiency is further improved by reusing existing $p$-cycles to protect a new multicast session and combining existing $p$-cycles whenever possible. The numerical results show that IpC has significantly better performance than D$p$C, which is an existing $p$-cycle based multicast protection scheme. In addition, IpC performs better in denser networks since denser networks contain more high efficiency cycles which could be utilized by IpC.
CHAPTER 6.  \( p \)-CYCLE-BASED PATH PROTECTION FOR MULTICAST SESSION
IN WDM NETWORKS

6.1 Introduction

In this chapter we propose a \( p \)-cycle-based path protection (\( P^3 \)) scheme to provide protection for dynamic multicast sessions. Given a multicast tree \( T \), the \( P^3 \) scheme computes a set of \( p \)-cycles on-demand to ensure every destination node in \( T \) is protected. The scheme has three features that make it capacity efficient. First, it reuses existing \( p \)-cycles to protect as many destination nodes in the current multicast session as possible. Second, when new \( p \)-cycles need to be created to protect some destinations, the scheme creates \( p \)-cycles with high protection efficiency. Third, only one \( p \)-cycle is needed to protect a destination against any link failure along the tree path from the source to the destination. This path-based approach is more efficient than the traditional link-based approach where \( p \)-cycles are used to protect individual links on the tree. The \( P^3 \) scheme also provides fast restoration since \( p \)-cycles are preconfigured. Thus, there is no need to configure the protection path upon a link failure. We conduct extensive simulations to evaluate the performance of the \( P^3 \) scheme. The results show that it has much higher capacity efficiency than a \( p \)-cycle-based link protection scheme.

The rest of this chapter is organized as follows. In section 6.2, we give the problem statement and describe the \( p \)-cycle based path protection strategy. In section 6.3, the \( P^3 \) scheme is presented. Simulation results are given in section 6.4. Finally, we conclude the paper in Section 6.5.
6.2 Problem Statement

6.2.1 Problem Definition

Let graph $G = (V, E)$ represent a WDM optical network, where $V$ and $E$ represent the set of nodes and links, respectively. A multicast session is denoted by $R = \{s, D, T\}$, where $s$ is the source node, $D$ is the set of destination nodes, and $T$ is the multicast tree for the multicast session. The set of all nodes on $T$ is denoted by $V_T$. For each node $u \in V_T$, $p_u$ denotes the path from $s$ to $u$ on $T$. Given a destination node $d_j \in D$, a node $v \in V_T$ is called the guard-node of $d_j$ if $p_{d_j}$ and $p_v$ are link-disjoint. The set of guard-nodes of $d_j$ is denoted by $N_{d_j}$. According to the definition, $s$ is the guard-node for every $d_j \in D$. Fig. 6.1(a) shows a multicast tree with source $s$ and three destinations $d_1, d_2, d_3$. Destination node $d_3$ has three guard-nodes $s, d_1, d_2$ since path $s - u - d_3$ and path $s - d_1 - d_2$ are link-disjoint.

Let $v$ be a guard-node of $d_j \in D$. If there is a directed path $p(v, d_j)$ in $G$ from $v$ to $d_j$ that is link-disjoint with $p_{d_j}$, then $v$ can provide protection for $d_j$ upon any single link failure in $p_{d_j}$ because $v$ can restore the traffic for $d_j$ by sending it along the path $p(v, d_j)$. Based on this observation, to protect $d_j$, we can create a directed p-cycle $pc$ such that 1) $pc$ contains $d_j$ and a node $v \in N_{d_j}$ and 2) the segment of $pc$ from $v$ to $d_j$ is link-disjoint with $p_{d_j}$. When a link on $p_{d_j}$ is down, $v$ can restore the traffic to $d_j$ using the segment from $v$ to $d_j$ on $pc$. We denote the segment on $pc$ providing protection for $d_j$ as...
An example of such p-cycle-based protection is given in Fig. 6.1(a). The directed p-cycle $pc_{d_3}$ (in dashed line) contains $d_3$ and one of its guard-nodes $d_1$, and the segment from $d_1$ to $d_3$ on $pc_{d_3}$ (i.e. $seg_{pc_{d_3}}$) is link-disjoint with path $p_{d_3} = s - u - d_3$. If a link on path $p_{d_3}$ is down, the guard-node $d_1$ can send the traffic to $d_3$ through $pc_{d_3}$ (using the segment from $d_1$ to $d_3$). Thus, $d_3$ can be protected by the p-cycle $pc_{d_3}$. This protection approach is path-based since the p-cycle provides an alternate path to the destination being protected. And the p-cycle can protect the destination against any single link failure on the tree path from the source to the destination.

To protect a multicast session, we need to find a set of p-cycles so that each destination is protected by a p-cycle. Thus, we consider the following problem in this paper: Given a graph $G = (V, E)$ and a multicast session $R = \{s, D, T\}$, find a set of directed p-cycles to provide path-based protection for all nodes in $D$ against any single link failure while minimizing the total capacity used by the p-cycles.

### 6.2.2 Protection Strategies

Given a multicast session $R = \{s, D, T\}$, in order to protect $d_j \in D$, we need to create a directed p-cycle $pc_{d_j}$ such that 1) $pc_{d_j}$ contains $d_j$ and a node $v \in N_{d_j}$ and 2) the segment of $pc_{d_j}$ from $v$ to $d_j$ is link-disjoint with $p_{d_j}$. We consider two strategies of creating $pc_{d_j}$ as follows.

#### 6.2.2.1 Tree-Disjoint Strategy

One simple way of creating $pc_{d_j}$ is to remove all links on tree $T$ and then find a directed p-cycle that contains $d_j$ and a guard-node $v$ of $d_j$. The p-cycle can be constructed by combining a directed path $p_1$ from $v$ to $d_j$ and a directed path $p_2$ from $d_j$ to $v$ where $p_1$ and $p_2$ are link-disjoint.

An important observation about the tree-disjoint strategy is that when a p-cycle $pc_{d_j}$ is created to protect $d_j$, every destination $d_i \neq d_j \in pc_{d_j}$ is also protected by $pc_{d_j}$.

This can be proved as follows. The p-cycle $pc_{d_j}$ is found after removing all links on $T$. Thus for each destination $d_i \in pc_{d_j}$, $p_{d_i}$ is link-disjoint with $pc_{d_j}$. In addition, $pc_{d_j}$ must contain a guard-node for every $d_i \in pc_{d_j}$. This is because $p_{d_j}$ and $p_v$ are link-disjoint, therefore at least one of $d_j$ and $v$ is in $N_{d_i}$. Let $w$ be a guard-node of $d_i$ on $pc_{d_j}$. Then $w$ can provide the protection for $d_i$ through the segment from $w$ to $d_i$ on $pc_{d_j}$.
Consider the example in Fig. 6.1(a). Suppose p-cycle $pc_{d_3}$ is created to protect $d_3$ using the tree-disjoint strategy (i.e., $pc_{d_3}$ does not contain any link in $T$), then $d_1 \in pc_{d_3}$ can also be protected by $pc_{d_3}$. Specifically, $d_3$ is a guard-node of $d_1$ and it can protect $d_1$ using the segment from $d_3$ to $d_1$.

Clearly, tree-disjoint strategy provides capacity efficient protection since a single p-cycle can protect all destinations on it.

### 6.2.2.2 Path-Disjoint Strategy

A drawback of the tree-disjoint strategy is that we may not be able to find a p-cycle that contains $d_j$ and one of its guard-node after all links on $T$ are removed from $G$. In this case, we can use a path-disjoint strategy based on the fact that a p-cycle $pe$ can protect destination $d_j$ as long as $seg_{pe}^{d_j}$ is link-disjoint with $p_{d_j}$. With path-disjoint strategy, we do not remove all links in $T$ to find a p-cycle for $d_j$. Instead, we only need to guarantee that the protection segment for $d_j$ on the p-cycle is link-disjoint with $p_{d_j}$. An example showing the path-disjoint strategy is given in Fig. 6.1(b). The p-cycle $pc_{d_3}$ is created to protect $d_3$. It shares three links ($u-d_3, s-u, s-d_1$) with $T$ but the segment from $d_1$ to $d_3$ is link-disjoint with path $p_{d_3} = s-u-d_3$. So $pc_{d_3}$ can provide protection for $d_3$ using guard-node $d_1$ and the segment from $d_1$ to $d_3$. Unlike the example in Fig. 6.1(a), $pc_{d_3}$ cannot protect $d_1$ against the failure of link $s-d_1$. Thus, path-disjoint strategy is not as efficient as tree-disjoint strategy.

Based on the above two strategies, we develop our p-cycle-based path protection scheme and present the detail of the scheme in the next section.

The following is a list of notations used in the rest of the paper.

- $R = (s, D, T)$: a multicast session with source $s$, destination node set $D$, and multicast tree $T$.
- $p_u$: the path from source $s$ to node $u$ on $T$.
- $N_{d_i}$: the set of guard-nodes of destination $d_i \in D$.
- $pc_{d_i}$: the p-cycle found to protect destination $d_i \in D$.
- $seg_{pc_{d_i}}^{d_i}$: the protection segment protecting destination $d_i$ on p-cycle $pc_{d_i}$.
- $e_{pc_i}$: protection efficiency of p-cycle $pc_i$. It is the ratio of the number of destinations protected by $pc_i$ over the number of links in $pc_i$. 
• $PC$: the set of p-cycles that have been created.

6.3 p-Cycle-based Path Protection ($P^3$) Scheme

6.3.1 Overview of the $P^3$ Scheme

The $P^3$ Scheme is presented in Algorithm 8. Given a new multicast session $R = \{s, D, T\}$ and a set $PC$ of p-cycles that have been created to protect existing multicast sessions, Algorithm 8 computes a set of p-cycles to protect all the destination nodes in $D$. The algorithm consists of two steps. First, we find the nodes in $D$ that can be protected by some existing p-cycles in $PC$ and remove them from $D$ (line 1-3). Then we call Algorithm 9 repeatedly until all nodes in $D$ are protected (line 4-5). Each time Algorithm 9 is called, it computes a new p-cycle to protect one or more nodes in $D$ and remove the protected nodes from $D$.

Algorithm 8 p-Cycle-based Path Protection

1: for all $d \in D$ do
2:  if $d$ can be protected by a p-cycle in $PC$ then
3:       $D = D - \{d\}$
4: while $D \neq \phi$ do
5:       Call Algorithm 9

6.3.2 Reusing Existing p-Cycles

In Algorithm 8, we first find the nodes in $D$ that can be protected by some existing p-cycles in $PC$. A node $d_i \in D$ can be protected by an existing p-cycle $pc_j$, which already protects a set of nodes $N_{pc_j}$, if the following two conditions are met:

1. $d_i \in pc_j$ and $pc_j \cap N_{d_i} \neq \phi$ and $p_{d_i} \cap seg_{pc_j}^{d_i} = \phi$

2. $\forall u \in N_{pc_j}$, either $p_u \cap p_{d_i} = \phi$ or $p_u \cap p_{d_i} \neq \phi$ but $seg_{pc_j}^u \cap seg_{pc_j}^{d_i} = \phi$

Condition (1) ensures that $pc_j$ can provide protection for $d_i$ because $pc_j$ contains $d_i$ and a guard-node of $d_i$, and the segment from the guard-node to $d_i$ on $pc_j$ is link-disjoint with $p_{d_i}$. Condition (2) ensures that using $pc_j$ to protect $d_i$ will not conflict with nodes already protected by $pc_j (N_{pc_j})$. 
Specifically, for each protected node \( u \in N_{pc_j} \), if \( p_u \) is link-disjoint with \( p_{d_i} \) (\( p_u \cap p_{d_i} = \phi \)), then \( d_i \) does not conflict with \( u \). As shown in Fig. 6.2(a), two tree paths \( s_1 \rightarrow d_1 \) (for destination \( d_1 \)) and \( s_2 \rightarrow d_2 \) (for destination \( d_2 \)) are link-disjoint. The first tree path can be protected by the segment from \( g_1 \) to \( d_1 \) on the anti-clockwise p-cycle (in dashed line), and the second tree path can be protected by the segment from \( g_2 \) to \( d_2 \) on the same p-cycle. (\( g_1 \) and \( g_2 \) are the guard-nodes of \( d_1 \) and \( d_2 \) respectively.) Although \( d_1 \) and \( d_2 \)'s protection segments share common links from \( g_1 \) to \( d_2 \) on the p-cycle, they can share the p-cycle. This is because a link failure will affect at most one of the tree paths. On the other hand, if \( p_u \) is not link-disjoint with \( p_{d_i} \) (\( p_u \cap p_{d_i} \neq \phi \)), but their protection segments are link-disjoint (\( seg_{pc_j}^{u} \cap seg_{pc_j}^{d_i} = \phi \)), \( d_i \) will not conflict with \( u \) and they can share the same p-cycle \( pc_j \). As shown in Fig. 6.2(b), two tree paths \( s_1 \rightarrow d_1 \) (for destination \( d_1 \)) and \( s_2 \rightarrow d_2 \) (for destination \( d_2 \)) share a common link \( u - v \). Path \( s_1 \rightarrow d_1 \) is protected by the segment from \( g_1 \) to \( d_1 \) on the anti-clockwise p-cycle and path \( s_2 \rightarrow d_2 \) is protected by the segment from \( g_2 \) to \( d_2 \) on the same p-cycle. (\( g_1 \) and \( g_2 \) are the guard-nodes of \( d_1 \) and \( d_2 \) respectively.) Since the two protection segments are link-disjoint, they can be used simultaneously when both \( d_1 \) and \( d_2 \) are affected by the failure of link \( u - v \). Thus, \( d_1 \) and \( d_2 \) can share the protection of the same p-cycle.

By checking for the two conditions listed above, we can find all nodes in \( D \) that can be protected by reusing existing p-cycles in \( PC \). For the rest of the nodes in \( D \), we need to compute new p-cycles to protect them using Algorithm 9.

### 6.3.3 Computing New p-Cycles

We now describe the detail of Algorithm 9. Given a set of un-protected destination nodes \( D \), Algorithm 9 computes a p-cycle with high protection efficiency to protect one or more nodes in \( D \).

Given a multicast session \( R \) and a p-cycle \( pc \), we define the protection efficiency of \( pc \), denoted by \( e_{pc} \), to be the ratio of the number of destinations of \( R \) protected by \( pc \) over the number of links in \( pc \). According to this definition, a p-cycle with higher protection efficiency is more efficient in protecting \( R \).

Algorithm 9 works as follows. First, it finds one efficient candidate p-cycle for each destination \( d_i \in D \). To find the candidate p-cycle for \( d_i \), we consider every guard-node of \( d_i \). For each guard-node
Figure 6.2  (a) Two destination nodes $d_1$ and $d_2$ can share a p-cycle if their tree paths are link-disjoint. (b) Two destination nodes $d_1$ and $d_2$ can share a p-cycle if their protection segments are link-disjoint.

$v$ of $d_i$, we find a p-cycle that contains $d_i$ and $v$. Among all these p-cycles, we choose the one with the highest protection efficiency as the candidate p-cycle for $d_i$. After we find a candidate p-cycle for each $d_i$, we have a total of $|D|$ p-cycles and the most efficient one among them is selected and added to $PC$.

The detailed steps of Algorithm 9 are given below.

Line 1 initializes $Best_{pc}$ which will store the most efficient p-cycle and $Best_e$ which will store the protection efficiency of $Best_{pc}$. Next, for every $d_i \in D$, we find (in line 4-21) the most efficient candidate p-cycle for $d_i$, which is stored in $cpc_{d_i}$. And $e_{pc}$ records the protection efficiency of p-cycle $cpc_{d_i}$. To find the most efficient p-cycle for $d_i$, we consider every $u \in N_{d_i}$ to find a p-cycle for $d_i$ using $u$ as the guard-node. Tree-disjoint strategy is tried first in line 5-10. Line 5 removes all links in $T$ and then line 6 finds the shortest path $p_1$ from $u$ to $d_i$. The shortest path $p_2$ from $d_i$ to $u$ is found in line 8 after line 7 removes all links in $p_1$. Then the p-cycle $pc_{temp}$ is formed by combining $p_1$ and $p_2$ in line 9. Line 10 restores all edges removed.

If a p-cycle cannot be found using the tree-disjoint strategy, we use the path-disjoint strategy in line 11-18. Line 12 removes all links on path $p_{d_i}$ and line 13 finds the shortest path $p_1$ from $u$ to $d_i$. The protection segment $p_1$ found in this way will be link-disjoint with $p_{d_i}$. After $p_1$ is found, the removed links are restored in line 14. After line 15 removes links on path $p_1$, line 16 finds the shortest path $p_2$
Algorithm 9 Computing a New p-Cycle for $D$

1: $Best_{pc} = null; Best_e = 0$
2: for all $(d_i \in D)$ do
3: \hspace{1 em} $cpc_{d_i} = null; e_{cpc} = 0;$
4: \hspace{1 em} for all $(u \in N_{d_i})$ do
5: \hspace{2 em} $E = E - T$
6: \hspace{2 em} Find shortest path $p_1$ from $u$ to $d_i$
7: \hspace{2 em} $E = E - \{e | e \in p_1\}$
8: \hspace{2 em} Find shortest path $p_2$ from $d_i$ to $u$
9: \hspace{2 em} $pc_{temp} = p_1 + p_2$
10: \hspace{2 em} $E = E \cup \{e | e \in p_1\} \cup T$
11: \hspace{2 em} if $pc_{temp} == null$ then
12: \hspace{3 em} $E = E - \{e | e \in p_{d_i}\}$
13: \hspace{3 em} Find shortest path $p_1$ from $u$ to $d_i$
14: \hspace{3 em} $E = E \cup \{e | e \in p_{d_i}\}$
15: \hspace{3 em} $E = E - \{e | e \in p_1\}$
16: \hspace{3 em} Find shortest path $p_2$ from $d_i$ to $u$
17: \hspace{3 em} $E = E \cup \{e | e \in p_1\}$
18: \hspace{3 em} $pc_{temp} = p_1 + p_2$
19: \hspace{3 em} if $e_{pc_{temp}} > e_{cpc}$ then
20: \hspace{4 em} $cpc_{d_i} = pc_{temp}$
21: \hspace{4 em} $e_{cpc} = e_{pc_{temp}}$
22: \hspace{3 em} if $e_{cpc} > Best_e$ then
23: \hspace{4 em} $Best_{pc} = cpc_{d_i}$
24: \hspace{4 em} $Best_e = e_{cpc}$
25: for all $(d_i \in D)$ do
26: \hspace{1 em} if $d_i$ can be protected by $Best_{pc}$ then
27: \hspace{2 em} $D = D - \{d_i\}$
28: $PC = PC \cup \{Best_{pc}\}$
from $d_i$ to $u$. Then line 17 restores the links in $p_1$. $pc_{temp}$ is formed by combining $p_1$ and $p_2$ in line 18.

Lines 19-21 store the current best candidate p-cycle for $d_i$ in $cpc_{di}$ and the corresponding protection efficiency in $e_{cpc}$.

Once we find the most efficient p-cycle for $d_i$ after finishing the loop in line 4-21, we compare its protection efficiency with that of $Best_{pc}$. The p-cycle with higher protection efficiency is stored in $Best_{pc}$ (line 23). The corresponding protection efficiency is stored in $Best_e$ (line 24).

Finally, in line 25-27 we remove from $D$ all nodes that can be protected by $Best_{pc}$. We also add $Best_{pc}$ to the set $PC$ in line 28.

### 6.4 Simulation Results

We run simulations with dynamic multicast requests on SMALLNET network (Fig 6.3) and COST239 network (Fig 5.6). In each simulation run, a set of randomly generated multicast requests are loaded to the network to compare our $P^3$ scheme with $IpC$ [17], which is a p-cycle-based link protection scheme. For each multicast request, the source node and the destination nodes are randomly selected and the bandwidth requested is one wavelength. For SMALLNET network, the number of destination nodes are randomly generated in the range [3, 5]. For COST239 network, the number of destination nodes are randomly generated in the range [2, 5]. The capacity of the network link is set to infinity.

The total number of wavelength channels used by all multicast trees and by all p-cycles are recorded for each simulation run.

![Figure 6.3 SMALLNET Network](image)

In Fig 6.4, we compare the performance of $P^3$ and $IpC$ under different traffic load in SMALLNET.
network. (Traffic load varies from 100 to 600 multicast requests.) The results show that $P^3$ uses significantly less wavelength channels for multicast tree protection than $IpC$ under all traffic loads. Specifically, $P^3$ achieves a 28.5%-46.6% reduction in wavelength usage over $IpC$. The reduction becomes larger as the number of multicast sessions increases because more multicast sessions provides more opportunity for different destinations to share a p-cycle. In all our simulations, $P^3$ can find p-cycles to protect all multicast requests. If we only use the tree-disjoint protection strategy, then 12.7%-14.1% of the multicast requests cannot be protected.

![Figure 6.4](image1.jpg)

**Figure 6.4** Number of wavelength channels used versus number of multicast sessions in SMALLNET network.

![Figure 6.5](image2.jpg)

**Figure 6.5** Number of wavelength channels used versus number of multicast sessions in COST239 network.

In Fig 6.5, we show the performance of $P^3$ and $IpC$ in COST239 network. The results again show that $P^3$ outperforms $IpC$ under all traffic loads. Specifically, $P^3$ achieves a 34.7%-58.6% reduction in
Table 6.1 Redundancy comparison of $P^3$ and $IpC$

<table>
<thead>
<tr>
<th>Demands</th>
<th>COST239 $IpC$</th>
<th>$P^3$</th>
<th>SMALLNET $IpC$</th>
<th>$P^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1.01</td>
<td>0.66</td>
<td>0.97</td>
<td>0.69</td>
</tr>
<tr>
<td>200</td>
<td>0.98</td>
<td>0.55</td>
<td>0.95</td>
<td>0.61</td>
</tr>
<tr>
<td>300</td>
<td>0.97</td>
<td>0.50</td>
<td>0.94</td>
<td>0.56</td>
</tr>
<tr>
<td>400</td>
<td>0.97</td>
<td>0.46</td>
<td>0.93</td>
<td>0.55</td>
</tr>
<tr>
<td>500</td>
<td>0.96</td>
<td>0.45</td>
<td>0.93</td>
<td>0.53</td>
</tr>
<tr>
<td>600</td>
<td>0.94</td>
<td>0.43</td>
<td>0.92</td>
<td>0.52</td>
</tr>
</tbody>
</table>

wavelength usage over $IpC$. We also found that about 1.2%-1.7% of all requests cannot be protected if only the tree-disjoint protection strategy is used. This failure ratio is much lower than in SMALLNET network for two reasons. First, COST239 is denser than SMALLNET. Second, the number of destination nodes for each multicast request in SMALLNET is chosen in the range [3, 5], which is a bit larger than the range [2, 5] used in COST239 network. Both factors make it more likely to find a p-cycle in COST239 network after removing the links in the multicast tree.

Table 6.1 compares the redundancy of $P^3$ and $IpC$ in two networks under different number of demands, where redundancy is defined as the total number of wavelength channels used by the p-cycles to the total number of wavelength channels used by the multicast trees. As shown in Table 6.1, the redundancy of $P^3$ is much lower than that of $IpC$. The redundancy of $P^3$ can be as low as 0.43 in COST239 network and as low as 0.52 in SMALLNET network. We also observe that as the number of demands increases, the redundancy of both schemes decreases. However, the redundancy of $P^3$ decreases much faster than that of $IpC$. For example, in COST239 network, the redundancy of $IpC$ decreases by 6.6% while the redundancy of $P^3$ decreases by 35% as the number of demands increases from 100 to 600. This shows that $P^3$ can better exploit the opportunity for p-cycle sharing as the number of multicast sessions in the network grows.

In summary, $P^3$ has much higher capacity efficiency than $IpC$ in protecting multicast sessions. This is because $P^3$ protects each destination node in the multicast tree using path-based protection, as opposed to protecting individual links in the multicast tree. Meanwhile, the use of p-cycles ensure that the protection segments on the p-cycles for protecting the destination nodes are pre-cross-connected,
which lead to fast restoration upon a link failure in the network.

6.5 Conclusion

We present the p-cycle-based path protection ($P^3$) scheme for dynamic multicast sessions in WDM networks. The $P^3$ scheme uses p-cycles to provide path-based protection to the destination nodes on the multicast tree. The key idea is to use tree-disjoint strategy whenever possible to increase the protection efficiency and use path-disjoint strategy when tree-disjoint strategy fails to find a p-cycle. Simulation results show that $P^3$ is much more efficient than a p-cycle-based link protection scheme named $IpC$. The $P^3$ scheme also provides fast restoration speed since the protection paths provided by the p-cycles are preconfigured.
CHAPTER 7. PXT-BASED PATH PROTECTION FOR MULTICAST SESSIONS IN WDM NETWORKS

7.1 Introduction

In this chapter, we propose a PXT-based path protection scheme for dynamic multicast sessions. To protect a multicast tree, we compute a PXT for each destination node \( v \) such that the PXT can be used to restore the traffic to \( v \) when a link failure occurs on the path from the source node to \( v \). To further improve capacity efficiency, our scheme reuses existing PXTs to protect a new multicast tree whenever possible. Our scheme also provides fast restoration since PXTs are pre-cross-connected structures. Simulation results show that our scheme has much higher capacity efficiency than IpC [17] - a p-cycle-based link protection scheme. We also compare the performance of the p-Cycle based path protection and the PXT based path protection.

The rest of this chapter is organized as follows. In section 7.2, we describe the basic idea of the PXT-based path protection method for multicast sessions. In section 7.3, we present the detail of our PXT-based path protection scheme. Simulation results comparing the performance of our scheme and the p-Cycle protection schemes(IpC and \( P^3 \)) are presented in section 7.4. Finally, we conclude this chapter in 7.5.

7.2 Basic Idea

Let graph \( G=(V, E) \) represents a WDM optical network, where \( V \) and \( E \) are the sets of nodes and links, respectively. A multicast session \( R \) is denoted by \( \{s, D, T\} \), where \( s \) is the source, \( D \) is the set of destinations, and \( T \) is the (directed) multicast tree that connects \( s \) to all destinations. For each node \( u \) on \( T \), \( p_u \) denotes the path from \( s \) to \( u \) on \( T \). The set of all nodes on \( T \) is denoted by \( V_T \).
Given a destination node \( d_j \in D \), a node \( v \in V_T \) is called the guard-node of \( d_j \) if \( p_{d_j} \) and \( p_v \) are link-disjoint. The set of guard-nodes of \( d_j \) is denoted by \( N_{d_j} \). According to the definition, source \( s \) is the guard-node of every \( d_j \in D \).

Let \( v \) be a guard-node of \( d_j \in D \), if we can create a directed PXT from \( v \) to \( d_j \) that is link-disjoint with \( p_{d_j} \), then the PXT can provide protection for \( d_j \) upon any single link failure in path \( p_{d_j} \) as follows. If a link on \( p_{d_j} \) fails, \( v \) can send the multicast traffic to \( d_j \) via the PXT. This PXT-based path protection scheme is shown in Fig. 7.1. The figure shows a multicast tree with source \( s \) and three destinations \( d_1, d_2, \) and \( d_3 \). Nodes \( s, d_1, \) and \( d_2 \) are three guard-nodes of \( d_3 \) because path \( s - u - d_3 \) and path \( s - d_1 - d_2 \) are link-disjoint. \( pxt_{d_3} \) (dashed line) is a PXT that does not contain any link in \( p_{d_3} \), so it can be used to protect destination \( d_3 \). If a link in path \( s - u - d_3 \) fails, the guard-node \( d_1 \) can send the traffic to \( d_3 \) through \( pxt_{d_3} \).

To protect a multicast session, we need to find a PXT to protect each destination in the multicast session. Thus we consider the following problem in this chapter: Given a graph \( G = (V, E) \) and a multicast session \( R = \{s, D, T\} \) where the traffic demand from \( s \) to every destination in \( D \) is one wavelength, find a set of PXTs to protect all destinations in \( D \) against any single link failure while minimizing the total protection capacity.

The following is a list of notations used in this chapter.

- \( R = (s, D, T) \): a multicast session with sources \( s \), a set of destination nodes \( D \), and multicast tree \( T \).
• $p_v$: the path from source $s$ to node $v$ on $T$.

• $N_{d_i}$: the set of guard-nodes of destination $d_i \in D$.

• $seg_{pxt}^{d_i}$: the protection segment protecting destination $d_i$ on $pxt$ ($pxt$ is a PXT that can protect $d_i$). The protection segment begins at a guard-node of $d_i$ and ends at $d_i$.

• $e_{pxt}$: efficiency of PXT $pxt$, which is the ratio of the number of destinations protected by $pxt$ to the number of links in $pxt$.

• $P$: the set of PXTs that have been created.

### 7.3 PXT-Based Path Protection Scheme

#### 7.3.1 Overview of the scheme

Our PXT-based path protection scheme is presented in Algorithm 10. Given a new multicast session $R = (s, D, T)$ and a set $P$ of PXTs (initially empty) that have been created to protect existing multicast sessions, Algorithm 10 computes a set of PXTs to protect all $d_i \in D$. The algorithm consists of two steps. First, we find the destination nodes in $D$ that can be protected by existing PXTs in $P$ and remove them from $D$ (line 1-3). Then we call Algorithm 11 repeatedly until all destinations in $D$ are protected (line 4-5). Each time Algorithm 11 is called, it first finds a new PXT with high efficiency to protect one or more nodes in $D$ and remove the protected nodes from $D$. It then tries to merge the new PXT with one existing PXT in $P$ such that the resulting PXT can produce the highest efficiency.

---

**Algorithm 10  PXT-based Path Protection Scheme**

1: for all $d \in D$ do
2:     if $d$ can be protected by a PXT in $P$ then
3:         $D = D - \{d\}$
4: while $D \neq \emptyset$ do
5:     Call Algorithm 11
7.3.2 Reusing Existing PXTs

In Algorithm 10, we first find the nodes in $D$ that can be protected by an existing PXT in $P$. A node $d_i \in D$ can be protected by an existing PXT $p_{xt_j}$, which already protects a set of nodes $N_{p_{xt_j}}$, if the following four conditions are met.

1. $p_{xt_j}$ contains $d_i$ and a guard-node $u$ of $d_i$.
2. The direction of $p_{xt_j}$ is from $u$ to $d_i$.
3. $seg_{p_{xt_j}}^{d_i}$ (the segment from $u$ to $d_i$ on $p_{xt_j}$) is link-disjoint with $p_{d_i}$.
4. $\forall v \in N_{p_{xt_j}}$, $p_v$ is link-disjoint with $p_{d_i}$ or $seg_{p_{xt_j}}^v$ is link-disjoint with $seg_{p_{xt_j}}^{d_i}$

Condition (1) ensures that one guard-node of $d_i$ and $d_i$ itself are on the existing PXT $p_{xt_j}$. Condition (2) ensures that the direction of the protection segment is correct as PXTs are directed. Condition (3) ensures that the working path of $d_i$ is link-disjoint with its protection segment. Basically, these three conditions are required to ensure that $p_{xt_j}$ can protect $d_i$. Condition (4) ensures that using $p_{xt_j}$ to protect $d_i$ in current multicast session will not conflict with nodes already protected by $p_{xt_j} (N_{p_{xt_j}})$. Specifically, for each node $v \in N_{p_{xt_j}}$, if $p_v$ is link-disjoint with $p_{d_i}$, then $d_i$ does not conflict with $v$. In this case, the working path of $v$ and the working path of $d_i$ will not fail simultaneously upon a single link failure, so $p_{xt_j}$ can protect $v$ and $d_i$ simultaneously. On the other hand, if $p_v$ is not link-disjoint with $p_{d_i}$, but the protection segment for $v$ is link-disjoint with the protection segment for $d_i$, then $d_i$ does not conflict with $v$. In this case, a single link failure may affect both $v$ and $d_i$. However, $p_{xt_j}$ can protect both nodes simultaneously since the two protection segments on $p_{xt_j}$ are link-disjoint.

Fig. 7.2 shows an example of using a PXT to protect two destination nodes when their working paths are link-disjoint. Part (a) shows a multicast session with source $s_1$ and destinations $d_1$ and $d_2$. Part (b) shows a multicast session with source $s_2$ and destinations $d_3$ and $d_4$. The PXT $1 \rightarrow 3$ (in green) can be used to protect both $d_2$ and $d_4$ since the working paths of $d_2$ and $d_4$ are link-disjoint.

Fig. 7.3 shows an example of using a PXT to protect two destination nodes when their protection segments are link-disjoint. There are two multicast sessions. The first session has source $s_1$ and destinations $d_1$ and $d_2$. The second session has source $s_2$ and destinations $d_3$ and $d_4$. (Note that
Figure 7.2 Two multicast sessions are shown in (a) and (b). Multicast trees are shown in red. The PXT $1 \rightarrow 3$ can protect destination nodes $d_2$ (in session 1) and $d_4$ (in session 2) simultaneously since the working paths of the two nodes are link-disjoint.

$s_1 = s_2$. In this example, PXT $1 \rightarrow 3 \rightarrow 0 \rightarrow 6 \rightarrow 4$ can protect both $d_2$ and $d_3$ even though their working paths share a common link $2 - 4$. This is because the protection segment for $d_2$ ($1 \rightarrow 3$) and the protection segment for $d_3$ ($6 \rightarrow 4$) are link-disjoint.

Figure 7.3 The PXT $1 \rightarrow 3 \rightarrow 0 \rightarrow 6 \rightarrow 4$ can protect destination nodes $d_2$ (in session 1) and $d_3$ (in session 2) simultaneously since the protection segments of the two nodes are link-disjoint.

By checking the four conditions, we can find all nodes in $D$ that can be protected by reusing existing PXTs in $P$. For the rest of the nodes in $D$, we need to compute new PXTs to protect them using Algorithm 11.
7.3.3 Computing and Merging New PXTs

Given a set of destination nodes $D$, Algorithm 11 first computes a PXT with high efficiency to protect one or more nodes in $D$ and remove the protected nodes from $D$. It then tries to merge the new PXT with an existing PXT in $P$ to further improve the efficiency.

The detail of Algorithm 11 is given below.

Algorithm 11 Computing a new PXT and merging it with $P$

1: $\text{Best}_{\text{pxt}} = \text{null}; \text{Best}_e = 0$
2: for all $d_i \in D$ do
3:     for all $u \in N_{d_i}$ do
4:         $E = E - \{e | e \in p_{d_i}\}$
5:         Find shortest path $p$ from $u$ to $d_i$
6:         $E = E \cup \{e | e \in p_{d_i}\}$
7:         if $e_p > \text{Best}_e$ then
8:             $\text{Best}_{\text{pxt}} = p$
9:             $\text{Best}_e = e_p$
10:    for all $d_i \in D$ do
11:        if $d_i$ can be protected by $\text{Best}_{\text{pxt}}$ then
12:            $D = D - \{d_i\}$
13:       Best'_{\text{pxt}} = \text{null}; \text{Best}'_e = 0; pxt' = \text{null}$
14:    for all $\text{pxt} \in P$ do
15:        temp = doMerge($\text{Best}_{\text{pxt}}, \text{pxt}$)
16:        if temp! = \text{null and } e_{\text{temp}} > \text{Best}'_e \text{ then}
17:            Best'_{\text{pxt}} = temp
18:            pxt' = \text{pxt}
19:        Best'_e = e_{\text{temp}}
20:    if Best'_{\text{pxt}}! = \text{null then}
21:        $P = P - \{\text{pxt}'\}$
22:    $P = P \cup \{\text{Best}'_{\text{pxt}}\}$
23: else
24:    $P = P \cup \{\text{Best}_{\text{pxt}}\}$

Line 1 initializes two variables $\text{Best}_{\text{pxt}}$ and $\text{Best}_e$, which store the current most efficient PXT and its corresponding efficiency, respectively. In line 2-9, we compute a high efficiency PXT to protect some nodes in $D$. Specifically, for every $d_i \in D$ and every guard-node $u \in N_{d_i}$, we find the shortest path from $u$ to $d_i$ that is link-disjoint with $p_{d_i}$. The path is a candidate PXT for protecting $d_i$. Among all the candidate PXTs, the most efficient one is stored in $\text{Best}_{\text{pxt}}$, and its efficiency is stored in $\text{Best}_e$. 
After we find the most efficient PXT, we remove all destination nodes protected by this PXT from $D$ in line 10 - 12.

Next, we try to merge $Best_{pxt}$ with a PXT in $P$ in line 13 - 24. In line 13, we initialize three variables $Best'_{pxt}$, $Best'_{e}$, and $pxt'$. $Best'_{pxt}$ and $Best'_{e}$ are used to store the current most efficient PXT resulted from merging and its corresponding efficiency. $pxt'$ is used to store the existing PXT in $P$ whose merging with $Best_{pxt}$ creates $Best'_{pxt}$. In line 14-19, we merge $Best_{pxt}$ with each PXT $pxt$ in $P$ and choose the one that produces the highest efficiency. In line 15, we use a temporary variable $temp$ to store the PXT resulted from merging. The function $doMerge$ will return $null$ if $pxt$ cannot be merged with $Best_{pxt}$. In line 20-24, we replace the existing PXT ($pxt'$) with the PXT resulted from merging ($Best'_{pxt}$) if $Best'_{pxt}$ is not $null$; otherwise, we add the new PXT $Best_{pxt}$ into $P$.

**Merging Two PXTs** In line 15, function $doMerge$ is called to merge a new PXT ($Best_{pxt}$) with an existing PXT ($pxt$). There are two cases in which the new PXT can merge with the existing PXT: the rear part of the new PXT and the front part of the existing PXT share a common segment as shown in Fig. 7.4(a), or, the front part of the new PXT and the rear part of the existing PXT share a common segment as shown in Fig. 7.4(b). In either case, we can merge the two PXTs together, resulting in a longer PXT shown in Fig. 7.4(c). Note that after several rounds of merging a node may appear multiple times in a PXT, but we do not allow a link to appear more than once in a PXT.

When merging two PXTs together, we need to ensure that the new PXT can simultaneously protect all the destination nodes protected by the two old PXTs. It’s easy to verify that the new PXT can protect every node originally protected by the two old PXTs. Hence, we only need to make sure there will be no conflict by checking condition (4) given in Section III-B. Specifically, for every pair of nodes $u, v$ where $u$ is protected by one old PXT and $v$ is protected by the other old PXT, we should ensure that
either the working paths of $u$ and $v$ are link-disjoint or the protection segment of $u$ on the new PXT is link-disjoint with the protection segment of $v$ on the new PXT. Fig. 7.5 shows an example. For Request 1 in part (a), we find a PXT $3 \rightarrow 4 \rightarrow 2$ to protect $d_1$. For Request 2 in part (b), we find a new PXT $4 \rightarrow 2 \rightarrow 5$ to protect $d_3$. Since the working path of $d_1$ ($0 \rightarrow 1 \rightarrow 2$) is link-disjoint with the working path of $d_3$ ($6 \rightarrow 5$), merging the two PXTs together will not cause any conflict, i.e., the resulting PXT $3 \rightarrow 4 \rightarrow 2 \rightarrow 5$ can protect $u$ and $v$ simultaneously.

7.4 Performance Evaluation

7.4.1 Performance of PXT Scheme

We run simulations with dynamic multicast requests on COST239 network (Fig 5.6) and NSF network (Fig 5.5). In each simulation run, a set of randomly generated multicast requests are loaded to the network to compare our PXT-based path protection scheme with IpC [17], which is a p-cycle-based link protection scheme. For each multicast request, the source node and the destination nodes are randomly selected. For COST239 network, the number of destination nodes is randomly generated in the range [2,5]. For NSF network, the number of destination nodes is randomly generated in the range [2,3]. The capacity of the network link is set to infinity. The total number of wavelength channels used by all the multicast trees, by all the PXTs (for our PXT-based scheme), and by all the p-cycles (for IpC scheme) are recorded for each simulation run.

In Fig 7.6, we show the wavelength usage of $PXT$ and $IpC$ schemes under different traffic load
in COST239 network. (Traffic load varies from 100 to 600 multicast sessions.) The result shows that $PXT$ scheme uses significantly less wavelength channels for protection than $IpC$ scheme under all traffic loads. Specifically, $PXT$ scheme achieves 21.8%-50% reduction in wavelength usage over $IpC$ scheme. Also, the number of protection wavelength channels required by $PXT$ scheme is much less than the number of working wavelength channels.

![Figure 7.6 Comparison of protection wavelength channels used in COST239 network](image)

In Fig 7.7, we compare the performance of $PXT$ scheme and $IpC$ scheme under different traffic load in NSF network. The figure shows that the $PXT$ scheme also performs very well by using significantly less protection wavelength channels than $IpC$ scheme under all traffic loads. Specifically, the $PXT$ scheme achieves a 31.1%-56.8% reduction in wavelength usage over $IpC$ scheme.

![Figure 7.7 Comparison of protection wavelength channels used in NSF network](image)
Table 7.1 Comparison of redundancy in two networks

<table>
<thead>
<tr>
<th>Demands</th>
<th>COST239</th>
<th></th>
<th>NSF</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IpC</td>
<td>PXT</td>
<td>IpC</td>
<td>PXT</td>
</tr>
<tr>
<td>100</td>
<td>101%</td>
<td>78.9%</td>
<td>130%</td>
<td>85%</td>
</tr>
<tr>
<td>200</td>
<td>97.8%</td>
<td>62.9%</td>
<td>124%</td>
<td>68.2%</td>
</tr>
<tr>
<td>300</td>
<td>97.1%</td>
<td>57%</td>
<td>122.2%</td>
<td>58.5%</td>
</tr>
<tr>
<td>400</td>
<td>96.7%</td>
<td>50.8%</td>
<td>122.9%</td>
<td>54.9%</td>
</tr>
<tr>
<td>500</td>
<td>95.6%</td>
<td>48.8%</td>
<td>121%</td>
<td>53%</td>
</tr>
<tr>
<td>600</td>
<td>94.4%</td>
<td>47.5%</td>
<td>120%</td>
<td>50%</td>
</tr>
</tbody>
</table>

Table 7.1 compares the redundancy of PXT scheme and IpC scheme in COST239 network and NSF network under different number of multicast requests. Redundancy is defined as the ratio of the total number of wavelength channels used for protection to the total number of wavelength channels used by the multicast trees. As shown in Table 7.1, the redundancy of PXT is much lower than that of IpC for both networks. When the number of demands is 600, PXT scheme can achieve 47.5% redundancy in COST239 network and 50% redundancy in NSF network, which is very impressive. We also observe that as the number of demands increases, the redundancy of both schemes decreases. However, the redundancy of PXT scheme decreases much faster than that of IpC scheme. Specifically, in COST239 network, the redundancy of IpC scheme decreases 6.6% and the redundancy of PXT scheme decreases 31.4% as the number of multicast sessions increases from 100 to 600. In NSF network, the redundancy of IpC scheme decreases 10% while the redundancy of PXT scheme decreases 35%. This shows that PXT scheme can reuse PXTs more efficiently as the number of multicast sessions in the network increases. Finally, we note that PXT scheme achieves lower redundancy in COST239 network than in NSF network because the former is denser than the latter.

Table 7.2 shows the average number of protected destinations per PXT in our PXT scheme. We can see that the average number of protected destinations per PXT increases with the increase of the traffic load. This is because more multicast sessions offer more opportunities for PXT reuse.

In summary, the PXT scheme has much higher capacity efficiency than that of IpC scheme in protecting multicast sessions. This is because PXT scheme protects each destination node in the multicast tree using path-based protection with the help of the guard nodes, as opposed to protecting
individual links in the multicast tree as done by the \( IpC \) scheme. In addition, PXTs are pre-cross-connected structures, which greatly reduces the recovery time compared with conventional shared path protection schemes.

### 7.4.2 Comparison of PXT and \( P^3 \) Schemes

In this section, we compare the performance of PXT and \( P^3 \) schemes in the variances of COST239 network. These networks are generated from COST239 by removing 2 and 4 links and adding 2 and 4 links. According to our simulations, the protection capacity redundancy of both schemes drops with the increase of network density and drops with the increase of number of traffic. In detail, the protection capacity redundancy of \( P^3 \) in COST239+4 network drops 26.4% in average compared with that in COST239-4 network. Meanwhile, compared with the redundancy in COST239-4 network, the protection capacity redundancy of \( PXT \) scheme in COST239+4 network drops 8.4% in average. In general, the protection capacity redundancy of \( P^3 \) is much lower than that of \( PXT \) in dense networks. For example, in COST239+4 network, the protection capacity redundancy of \( P^3 \) is 16.7% less than that of \( PXT \). The protection capacity redundancy of \( P^3 \) is 10.6% less than that of \( PXT \) in COST239+2 network. So \( P^3 \) is more suitable in dense networks.

### 7.5 Conclusion

In this chapter, we present the PXT-based path protection scheme to protect dynamic multicast sessions against single link failures in WDM optical networks. The scheme is capacity efficient in that it provides path-based protection for multicast destinations and reuses PXTs whenever possible. The scheme also provides fast restoration as PXTs are pre-cross-connected structures. Our simulation results show that our scheme is significantly more capacity efficient than \( IpC \), a p-cycle-based link pro-
Table 7.3 Comparison of redundancy in 5 networks

<table>
<thead>
<tr>
<th>Demands</th>
<th>COST239-4 P&lt;sup&gt;1&lt;/sup&gt;</th>
<th>COST239-2 P&lt;sup&gt;1&lt;/sup&gt;</th>
<th>COST239 P&lt;sup&gt;1&lt;/sup&gt;</th>
<th>COST239+2 P&lt;sup&gt;1&lt;/sup&gt;</th>
<th>COST239+4 P&lt;sup&gt;1&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>78%</td>
<td>80%</td>
<td>71%</td>
<td>66%</td>
<td>79%</td>
</tr>
<tr>
<td>200</td>
<td>64%</td>
<td>66%</td>
<td>58%</td>
<td>55%</td>
<td>63%</td>
</tr>
<tr>
<td>300</td>
<td>58%</td>
<td>59%</td>
<td>54%</td>
<td>50%</td>
<td>57%</td>
</tr>
<tr>
<td>400</td>
<td>56%</td>
<td>53%</td>
<td>52%</td>
<td>46%</td>
<td>51%</td>
</tr>
<tr>
<td>500</td>
<td>55%</td>
<td>50%</td>
<td>51%</td>
<td>49%</td>
<td>49%</td>
</tr>
<tr>
<td>600</td>
<td>53%</td>
<td>46%</td>
<td>48%</td>
<td>46%</td>
<td>43%</td>
</tr>
</tbody>
</table>

tection scheme. We also compare the performance of the p-Cycle and PXT based protection schemes and the p-Cycle based protection scheme is more suitable for dense networks.
CHAPTER 8. DESIGN OF SURVIVABLE HYBRID WIRELESS-OPTICAL BROADBAND-ACCESS NETWORK

8.1 Introduction

In this chapter, we propose a new protection scheme for WOBAN. The scheme is cost-effective in that it does not require the PONs to have self-protecting capability. In addition, it does not assume every wireless router in one segment can find a multi-hop path to a gateway in another segment. Instead, we make the general assumption that the wireless routers can send traffic to the gateways in the same segment but cannot send traffic to the gateways in other segments. Based on the proposed protection scheme, we define the maximum protection with minimum cost (MPMC) problem and present an ILP solution and a heuristic approach to the MPMC problem. The proposed protection scheme is much more cost-effective than employing self-protecting PON architectures and our heuristic algorithm is very effective in obtaining near-optimal solutions according to the numerical results.

The rest of this chapter is organized as follows. In Section 8.2, we describe the proposed protection scheme for WOBAN and formally define the MPMC problem. In Section 8.3, we describe our solution approach to the MPMC problem. We then prove the decision problem of MPMC is NP-hard in Section 8.4 and a heuristic algorithm for MPMC is given in Section 8.5. We present the numerical results in Section 8.6. Finally, Section 8.7 concludes this chapter.

8.2 Protection Scheme and Problem Statement

8.2.1 Protection Scheme

We propose a scheme to deal with DF/FF/ONU/OLT failures in the optical part of a WOBAN. A DF failure is equivalent to an ONU failure because the ONU attached to the failed DF loses its connection
to the OLT. An FF failure is equivalent to an OLT failure because the OLT attached to the failed FF can no longer drive the ONU s in its segment. Therefore, we only consider ONU failures and OLT failures in this paper.

Our proposed protection scheme works as follows. In each segment of the WOBAN, one of the ONUs is designated as the backup ONU. We connect selected pairs of backup ONUs with fibers so that each backup ONU is connected to at least one backup ONU in another segment. Two backup ONUs are called neighbors if they are connected by a fiber. Two segments are called neighbors if their backup ONUs are neighbors. When the OLT in segment $i$ fails, all traffic in segment $i$ will be sent to the segment’s backup ONU, which then sends the traffic to the neighbor backup ONUs. The backup ONU in a neighbor segment will distribute the traffic to all the ONUs in its segment via the wireless gateways so that each ONU in the segment handles the traffic using its spare capacity. Since the traffic in segment $i$ is handled by the spare capacity in the neighbor segments, full protection can be achieved if the sum of the spare capacity in the neighbor segments is greater than or equal to the amount of traffic in segment $i$. If an ONU in segment $i$ fails, then the traffic normally handled by the failed ONU will be handled by the other ONUs in segment $i$ if they have enough spare capacity to handle the affected traffic. Otherwise, the affected traffic that cannot be handled within segment $i$ will be sent to the neighbor segments by the backup ONU in segment $i$.

### 8.2.2 Problem Statement

An important design problem arising from the proposed protection scheme is to determine the pairs of backup ONUs to be connected with fibers so that 1) the amount of traffic that can be protected upon an OLT/ONU failure is maximized and 2) the cost of connecting the backup ONUs is minimized. We refer to this problem as the maximum protection with minimum cost (MPMC) problem.

We now give the formal definition of the MPMC problem. An instance of the MPMC problem is represented by $< V, d, cap, c >$. $V$ is a set of nodes where each node represents a segment in the WOBAN. $d$ is a function from $V$ to the set of positive integers. $cap$ is a function from $V$ to the set of nonnegative integers. For each node $i \in V$, $d(i)$ is the traffic demand in segment $i$ and $cap(i)$ is the spare capacity in segment $i$. $c$ is a function from $V \times V$ to the set of positive integers. For
94

each unordered pair of nodes \(i, j \in V\), \(c(ij)\) is the cost of laying a fiber between the backup ONUs in segment \(i\) and segment \(j\). The MPMC problem is to compute a set \(S\) of (unordered) node pairs such that \(\sum_{j \in V} \min(\sum_{(i,j) \in S} \text{cap}(i), \text{d}(j))\) is maximized and \(\sum_{(i,j) \in S} c(ij)\) is minimized. Note that \((i, j) \in S\) means the backup ONUs in segment \(i\) and segment \(j\) should be connected so that the two segments become neighbors. \(\sum_{(i,j) \in S} \text{cap}(i)\) is the total spare capacity in the neighbor segments of segment \(j\) and \(d(j)\) is the amount of traffic in segment \(j\) that needs to be protected when the OLT in segment \(j\) fails. Thus, the amount of traffic in segment \(j\) that can be protected upon the OLT failure is \(\min(\sum_{(i,j) \in S} \text{cap}(i), \text{d}(j))\). Considering all possible OLT failures, our goal is to maximize \(\sum_{j \in V} \min(\sum_{(i,j) \in S} \text{cap}(i), \text{d}(j))\). (This also maximizes the amount of traffic that can be protected upon an ONU failure since the amount of traffic that needs to be protected upon an ONU failure is less than that upon an OLT failure.) The other goal is to minimize the total cost of protection, i.e. \(\sum_{(i,j) \in S} c(ij)\).

Fig. 8.1(a) shows an instance of the MPMC problem. \(V\) represents a WOBAN with 3 segments \(a\), \(b\), and \(c\). The traffic demand of \(a\) is 4 and the spare capacity of \(a\) is 2. The traffic demand of \(b\) is 4 and the spare capacity of \(b\) is 4. The traffic demand of \(c\) is 3 and the spare capacity of \(c\) is 3. The optimal solution for the MPMC problem is \(S = \{(a, b), (b, c)\}\). By laying fibers between the backup ONUs in \(a\) and \(b\) and between the backup ONUs in \(b\) and \(c\), full traffic protection can be achieved. Specifically, the traffic of \(a\) can be protected by \(b\) since \(\text{cap}(b) = \text{d}(a)\). The traffic of \(b\) can be protected by \(a\) and \(c\) since \(\text{cap}(a) + \text{cap}(c) > \text{d}(b)\). The traffic of \(c\) can be protected by \(b\) since \(\text{cap}(b) > \text{d}(c)\). The cost of \(S\) is \(c(ab) + c(bc) = 3 + 2 = 5\). This is the minimum cost solution among all solutions that achieve full protection.

### 8.3 Solution Approach to the MPMC Problem

Given an instance \(I = < V, d, \text{cap}, c >\) of the MPMC problem, we can find the optimal solution to \(I\) in two steps. First, we create a graph \(G\) based on \(I\) and solve the minimum cost maximum flow (MCMF) problem on \(G\). Second, we convert the optimal solution to the MCMF problem on \(G\) to the optimal solution to \(I\).
8.3.1 Construction of Graph $G$

To obtain the optimal solution to $I$, we first construct a directed graph $G$ where each edge in $G$ is associated with a cost and a capacity. The vertices of $G$ are created as follows. First, we create a source vertex $S$ and a sink vertex $T$. Second, for each pair of nodes $u, v \in V$, if $\text{cap}(u) > 0$ or $\text{cap}(v) > 0$, we create a vertex $I_{uv}$ in $G$ (such a vertex is called an I-vertex). Third, for every node $v \in V$, we create a vertex $J_v$ in $G$ (such a vertex is called a J-vertex). The edges of $G$ are created as follows. For each I-vertex $I_{uv}$, we create a directed edge from $S$ to $I_{uv}$. The cost of this edge is $c(uv)$ and the capacity of this edge is infinity. For each J-vertex $J_v$ in $G$, we create a directed edge from $J_v$ to $T$. The cost of this edge is 0 and the capacity of this edge is $d(v)$. Finally, for each I-vertex $I_{uv}$, if $\text{cap}(u) > 0$, we create $m$ directed edges from $I_{uv}$ to $v$ where $m = \text{cap}(u)$; if $\text{cap}(v) > 0$, we create $n$ directed edges from $I_{uv}$ to $u$, where $n = \text{cap}(v)$. All these edges have a cost of 0 and a capacity of 1.

Fig. 8.1(b) shows the graph $G$ constructed from an instance of the MPMC problem given in Fig. 8.1(a). $G$ contains the source vertex $S$, the sink vertex $T$, three I-vertices $I_{ab}$, $I_{ac}$, and $I_{bc}$, and three J-vertices $J_a$, $J_b$, and $J_c$. There is one directed edge from $S$ to each of the three I-vertices $I_{ab}$, $I_{ac}$ and $I_{bc}$. The costs of these edges are $c(ab) = 3$, $c(ac) = 4$, and $c(bc) = 2$, respectively. And all these edges have a capacity of infinity. There is a directed edge from each of the three J-vertices $J_a$, $J_b$, and $J_c$ to $T$. The capacity of these edges are $d(a)=4$, $d(b)=4$, and $d(c)=3$, respectively. The costs of these edges are all 0. The edges from the I-vertices to the J-vertices are created according to the rule given earlier. For example, since $\text{cap}(a) = 2$ and $\text{cap}(b) = 4$, there are two edges from $I_{ab}$ to $J_a$ and four edges from $I_{ab}$ to $J_a$, each of which has a cost of 0 and a capacity of 1.

The minimum cost maximum flow (MCMF) problem on $G$ is to compute a maximum flow from $S$ to $T$ such that the total cost of the flow is minimum where the cost of a flow is the sum over $\text{cost}(e)$ for all edge $e$ with a nonzero flow. We now show that the optimal solution to $I$ can be obtained from the optimal solution to the MCMF problem on $G$.

Let $f$ be the minimum cost maximum flow from $S$ to $T$ in $G$ and $F$ is the value of $f$. Based on $f$, we can obtain the optimal solution $S_I$ to $I$ as follows. $S_I$ is empty initially. For each I-vertex $I_{uv}$, if $f(S \rightarrow I_{uv}) > 0$, add $(u, v)$ to $S_I$. (The existence of a nonzero flow on the edge from $S$ to $I_{uv}$ indicates that at least one of the segments $u$ and $v$ needs to use the spare capacity in the other
segment to protect its traffic. Therefore, \((u, v)\) should be included in \(S_I\). It’s easy to verify that
\[
F = \sum_{j \in V} \min(\sum_{i,j} \in S_I \cap(i), d(j))
\]
(since the capacity assignment of the edges in \(G\) ensures that the maximum amount of flow that can enter a \(J\)-vertex \(J_j\) is equal to \(\sum_{i \neq j} \in V \cap(i)\) and the maximum amount of flow that can leave \(J_j\) is \(d(j))\). Thus, \(S_I\) achieves the maximum traffic protection for \(I\). \(S_I\) is also the minimum cost solution to \(I\). This is because \(\text{cost}(S_I) = \sum_{(u,v) \in S_I} c(uv) = \sum_{f(S \rightarrow I_{uv}) > 0} \text{cost}(S \rightarrow I_{uv}) = \text{cost}(f)\).

Fig.8.1 (b) shows the minimum cost maximum flow \(f\) in \(G\), which has a value of 11. The cost of \(f\) is \(\text{cost}(S \rightarrow I_{ab}) + \text{cost}(S \rightarrow I_{bc}) = 3 + 2 = 5\). Since the edge from \(S\) to \(I_{ab}\) and the edge from \(S\) to \(I_{bc}\) have nonzero flows, the optimal solution to the instance shown in Fig.8.1 (a) is \\{(a, b), (b, c)\}. This solution achieves maximum traffic protection (11) with minimum cost (5).

### 8.3.2 An ILP for the MCMF Problem

In the previous section, we have shown that the optimal solution to \(I\) can be obtained from the optimal solution to the MCMF problem on \(G\). In this section, we describe how to solve the MCMF problem on \(G\).

To compute the minimum cost maximum flow in \(G\), we first find the maximum flow in \(G\) using the Ford-Fulkerson Algorithm. Let \(F\) be the value of the maximum flow. After that, we need to find the
minimum cost flow with a value of $F$. This problem can be formulated by the following ILP model.

- $S, T$ the source vertex and the sink vertex
- $I_i, J_j$ the $i_{th}$ I-vertex and the $j_{th}$ J-vertex
- $\text{CAP}_{J_j T}$ integer, capacity of edge $J_j \to T$
- $\text{CAP}_{I_i J_j}$ integer, capacity of edge $I_i \to J_j$
- $C_{SI_i}$ integer, cost of edge $S \to I_i$
- $X_{SI_i}$ binary variable, 1 means edge $S \to I_i$ carries a positive flow
- $X_{I_i J_j k}$ binary variable, 1 means the $k_{th}$ edge from $I_i$ to $J_j$ carries one unit of flow
- $f_{I_i J_j}$ integer variable, flow from $I_i$ to $J_j$
- $f_{J_j T}$ integer variable, flow from $J_j$ to $T$

**Objective:**

$$\text{Minimize} \sum_i X_{SI_i} \cdot \text{cost}(S \to I_i)$$

**Constraints:**

1. $f_{I_i J_j} = \sum_k X_{I_i J_j k} \quad \forall i, j$ (8.1)
2. $\sum_i f_{I_i J_j} = f_{J_j T} \quad \forall j$ (8.2)
3. $f_{J_j T} \leq \text{CAP}_{J_j T} \quad \forall j$ (8.3)
4. $\sum_j f_{J_j T} = F$ (8.4)
5. $X_{I_i J_j k} \leq X_{SI_i} \quad \forall i, j, k$ (8.5)
6. $X_{SI_i} \leq \sum_j \sum_k X_{I_i J_j k} \quad \forall i$ (8.6)

The objective is to minimize the total cost of the edges that carry a positive flow. Note that only the edges from $S$ to the $I$-vertices have nonzero cost, so the objective function considers only those edges.

Constraint (1) ensures that the total flow from $I_i$ to $J_j$ is equal to the number of edges from $I_i$ to $J_j$ that carry one unit of flow. Constraint (2) ensures that the total flow coming into $J_j$ is equal to the total flow going out of $J_j$. Constraint (3) ensures that the flow on edge $J_j \to T$ is bounded by its capacity. Constraint (4) ensures that the total flow entering $T$ is equal to the maximum flow value $F$. Constraint (5) ensures that if an edge from $I_i$ to $J_j$ carries one unit of flow, then $X_{SI_i}$ will be 1. This ensures that if there is a nonzero flow from $I_i$ to $J_j$, then the cost of edge from $S$ to $I_i$ will be counted in the
objective function. Constraint (6) ensures that if no flow is sent out of $I_i$, then $X_{SI_i}$ will be 0 and the cost of edge $S \rightarrow I_i$ will not be counted in the objective function.

In [72], an algorithm for the classic MCMF problem is given. However, the algorithm cannot be used to solve our MCMF problem because our problem is different from the classic problem. In the classic MCMF problem, the cost of a flow on an edge $e$ is defined as $c(e) \times f(e)$, where $c(e)$ is the cost of a unit flow and $f(e)$ is the flow on edge $e$. In our MCMF problem, the cost of a flow on an edge $e$ is equal to the cost of $e$ if $f(e) > 0$ and is equal to zero if $f(e) = 0$.

### 8.4 NP-Hard Proof

In this section, we prove that the MPMC problem is NP-Hard.

The MPMC problem seeks to find the maximum protection for all segments with the minimum cost. The maximum protection for segment $i$ is $D(i) = \min\{d(i), \sum_{j \neq i, j \in [1,...,n]} \text{cap}(j)\}$ because segment $i$ only needs $d(i)$ protection and it can get at most $\sum_{j \neq i, j \in [1,...,n]} \text{cap}(j)$ protection from the other segments. Next, we prove that the decision problem of finding the minimum cost protection that achieves the maximum protection (Decision-MCPMP) is NP-complete.

We define Decision-MCPMP as follows: Given $\langle V, D, \text{cap}, c, C \rangle$, where $V$ is a set of $n$ segments, $D(i)$ specifies the required maximum protection for segment $i \in V$, $\text{cap}$ and $c$ are the same functions defined in Section 8.2-B, and $C$ is an integer, determine whether there is a link set $LS$ of (unordered) segment pairs representing connections between segments such that each segment $i$ is protected with capacity $D(i)$ and $\sum_{(i,j) \in LS} c(ij) = C$.

First, we show that Decision-MCPMP is in NP. Given a set $LS$, we can check if the total cost equals $C$ and if each segment $i$ could be protected with $D(i)$ in $O(n^2)$. Thus, Decision-MCPMP is in NP.

Next, we reduce the NP-complete problem Subset-Sum to Decision-MCPMP. In the Subset-Sum problem, a set $S = \{s_1, s_2, ..., s_n\}$ of integers and an integer $C$ are given, the problem is to determine whether there is a subset $S' \subseteq S$, s.t. $\sum_{s_i \in S'} s_i = C$. Given an instance of the Subset-Sum problem $\langle S = \{s_1, s_2, ..., s_n\}, C \rangle$, we can build an instance of Decision-MCPMP $\langle V, D, \text{cap}, c, C \rangle$ as follows: $V = \{1, 2, ..., n, n + 1\}$ is a set of $n + 1$ segments; $D(i) = 0, \forall i \in [1,...,n]$ and $D(n + 1) = C$; $\text{cap}(i) = s_i, \forall i \in [1,...,n]$ and $\text{cap}(n + 1) = 0$; $c(i, n + 1) = s_i, \forall i \in [1,...,n]$ and $c(i, j) = \infty$, 

∀ i, j ∈ [1, ..., n]. In this instance, segment n + 1 needs to be protected with capacity C and other segments need no protection.

We prove that there exists \( S' \subseteq S \) whose sum equals C if and only if there is a link set \( LS \) with cost C and each segment i is protected with capacity \( D(i) \) for the corresponding instance of Decision-MCPMP.

⇒ If \( \exists S' \subseteq S \), s.t. \( \sum_{s_i \in S'} s_i = C \), we build \( LS = \{(i, n + 1) | s_i \in S', i \in [1, ..., n]\} \) for the instance of Decision-MCPMP. Thus, the cost of the link set \( LS \) equals \( \sum_{s_i \in S'} c(i, n + 1) = \sum_{s_i \in S'} s_i = C \); the protection for segment \( n + 1 \) equals \( \sum_{s_i \in S'} \text{cap}(i) = \sum_{s_i \in S'} s_i = C = D(n + 1) \) and the protection for segment \( i \) equals \( D(i) = 0 \) for \( 1 \leq i \leq n \).

⇐ If there is a link set \( LS \) with cost C and each segment i is protected with capacity \( D(i) \) for the instance of Decision-MCPMP, then segment \( n + 1 \) is protected by the set of segments \( \{i | (i, n + 1) \in LS, i \in [1, ..., n]\} \). We have \( \sum_{s_i \in S'} s_i = \sum_{(i, n+1) \in LS} s_i = \sum_{(i, n+1) \in LS} \text{cap}(i) = C \). That is, \( S' \) is a subset of \( S \) whose sum equals C.

Figure 8.2 shows an instance of Decision-MCPMP \( \langle V, D, \text{cap}, c, 8 \rangle \) that is constructed from a Subset-Sum problem instance \( \langle S = \{1, 2, 5, 9\}, 8 \rangle \). In the Decision-MCPMP problem instance, \( V \) contains 5 elements \( \{1, 2, 3, 4, 5\} \) and the function values for \( D, \text{cap} \) and \( c \) are given in the figure. For the Subset-Sum problem instance, \( S \) has a subset \( S' = \{1, 2, 5\} \) whose sum equals 8. Correspondingly, for the Decision-MCPMP problem instance there is a link set \( LS = \{(1, 5), (2, 5), (3, 5)\} \) that has cost 8 and protects each segment \( i \) with capacity \( D(i) \). Specifically, segment 5 is protected with capacity 8 and other segments receive no protection.

### 8.5 A Heuristic Algorithm

Although the ILP model presented in Section 8.3 obtains the optimal solution for MPMC, it is not practical for large network design due to its long running time. In this section, we present a heuristic algorithm for MPMC. The algorithm consists of two steps. The first step is to compute the maximum protection requirement \( D(i) \) for each segment \( i \), where \( D(i) = \min \{d(i), \sum_{j \not= i, j \in [1, ..., n]} \text{cap}(j)\} \) as
discussed in Section 8.4. The second step is to find a low cost protection by repeatedly selecting two segments to connect until the protection requirement for each segment is satisfied.

The pseudocode of the second step is shown in Algorithm 12. This algorithm selects a pair of segments to connect based on the metric \( M = \frac{D_i^j + D_i^j}{c(i,j)} \). In this metric, \( c(i,j) \) is the cost of connecting segments \( i \) and \( j \). \( D_i^j \) identifies the valid protection provided by segment \( j \) for segment \( i \). If segment \( i \) is already fully protected by other segments, then connecting segments \( j \) and \( i \) will not provide any valid protection for segment \( i \), i.e., \( D_i^j = 0 \); otherwise \( D_i^j \) will be equal to \( \min(D(i), \text{cap}(j)) \). Thus the metric \( M \) measures the cost efficiency of connecting segments \( i \) and \( j \) and the algorithm chooses the most efficient connection between two segments until we satisfy the protection requirements of all segments. Note that the value of \( D(i) \) needs to be updated to reflect the remaining protection requirement of segment \( i \) after segment \( i \) is connected with another segment.

The input of the algorithm will be three sets of integers \( DEMAND = \{D(i)|i = 1, \ldots, n\} \), \( CAP = \{\text{cap}(i)|i = 1, \ldots, n\} \), \( COST = \{c(i,j)|i,j = 1, \ldots, n\} \) and a set of candidate connections \( CANDIDATE = \{(a,b)|a,b = 1, \ldots, n \text{ and } a < b\} \). The output will be a set of connected segments \( OUTPUT = \{(a,b)|a,b = 1, \ldots, n \text{ and } a < b\} \).

The while loop keeps running until all segments get the required protection. Line 3 finds the
Algorithm 12 Heuristic Algorithm

1: \( \text{OUTPUT} = \Phi \)
2: \( \textbf{while} \ \exists D(i) \neq 0, \forall i \in \{1, \ldots, n\} \ \textbf{do} \)
3: \( M_j^i = \max(D_i^a + D_j^b), \forall (a, b) \in \text{CANDIDATE}; \)
4: \( \text{OUTPUT} = \text{OUTPUT} \cup \{(i, j)\}, \text{CANDIDATE} = \text{CANDIDATE} - \{(i, j)\} \)
5: \( D_i = \max(D_i - \text{cap}(j), 0), D_j = \max(D_j - \text{cap}(i), 0); \)
6: \( \text{Return } \text{OUTPUT} \)

Table 8.1 Optimal solutions to different instances of the MPMC problem.

| Demands | \( |V| = 10 \) | \( |V| = 20 \) |
|---------|------------|------------|
|         | cost | #links | \( R \) | cost | #links | \( R \) |
| 5-random | 1700 | 11 | 1.1 | 2492 | 26 | 1.3 |
| 6-random | 2730 | 16 | 1.6 | 2946 | 26 | 1.3 |
| 7-random | 4472 | 22 | 2.2 | 6485 | 40 | 2 |
| 8-random | 8062 | 34 | 3.4 | 12536 | 70 | 3.5 |
| 5-fixed  | 632  | 5 | 0.5 | 842  | 10 | 0.5 |
| 6-fixed  | 1440 | 10 | 1 | 2118 | 20 | 1 |
| 7-fixed  | 2283 | 15 | 1.5 | 3700 | 30 | 1.5 |
| 8-fixed  | 3390 | 20 | 2 | 5563 | 40 | 2 |

segment pair with the largest metric value and line 4 adds the selected segment pair \((i, j)\) into the set \(\text{OUTPUT}\) and removes \((i, j)\) from the set \(\text{CANDIDATE}\). Line 5 updates the remaining needed protection for segments \(i\) and \(j\).

8.6 Numerical Results

We solved different instances of the MPMC problem using both the ILP approach in Section 8.3 and the heuristic algorithm in Section 8.5; we report the numerical results in this section.

An instance \(< V, d, \text{cap}, c >\) of the MPMC problem is generated as follows. We randomly distribute \(|V|\) nodes in a 600x600 square area. \(|V|\) is set to 10 and 20 in different instances. Demand type is either random or fixed. For random demand, each node \(i\) has a \(d(i)\) value randomly chosen in the range \([\text{min}, 10]\), where \(\text{min}\) is set to 5, 6, 7, and 8 in different instances. For fixed demand, \(d(i)\) is set to a constant \(k\) for all nodes \(i\), where \(k\) is set to 5, 6, 7, and 8 in different instances. All nodes have a capacity of 10, so the spare capacity of node \(i\) is \(\text{cap}(i) = 10 - d(i)\). For each pair of nodes \(i\) and \(j\), \(c(ij)\) is set to the Euclidean distance between \(i\) and \(j\), rounded to the nearest integer.
First, we compute the optimal solution for different instances of the MPMC problem using the approach given in Section 8.3. ILOG CPLEX 8.0 is used to solve the ILP for the MCMF problem. The results are given in Table 8.1. For each instance, the cost of the optimal solution, the number of links needed by the optimal solution, and the ratio of number of links to number of nodes (denoted by $R$) are shown. The number of links needed is equal to the number of node pairs in the optimal solution, which is the number of fibers need to be laid to provide protection. In the table, ‘$k$-random’ means random demand with a $\min$ value of $k$. ‘$k$-fixed’ means fixed demand with a value of $k$. For all instances shown in the table, full traffic protection is achieved by the optimal solution. This is because for all instances, $\sum_{i \neq j \in V} \text{cap}(i) \geq d(j)$ for all node $j \in V$. As shown in the table, the cost, the number of links, and $R$ all increase as the demand increases for both random demand and fixed demand. However, $k$-fixed always has lower cost and requires fewer links than $k$-random. This is because the total traffic demand of all nodes is lower when demand is fixed than when demand is random.

For random demand, $R$ increases as the demand increases, but it does not increase as $|V|$ increases. In fact, $R$ is similar for $|V| = 10$ and $|V| = 20$. The $R$ value of the proposed protection scheme is much lower than that of the self-protecting PON architectures proposed in [63][64][65]. In those architectures, $N$ fibers need to be laid to protect a PON with $N$ ONUs. Therefore, $R$ is equal to the number of ONUs in a PON, which is typically 32. For our protection method, $R$ does not depend on the number of ONUs. Instead, it depends on the traffic demand of the WOBAN. Even in the high demand case where every segment has a demand of at least 80% of its capacity, $R$ is as low as 3.4 for $|V| = 10$ and 3.5 for $|V| = 20$. Thus, the proposed protection scheme is much more cost-effective than employing self-survivable PONs.

For fixed demand, Table 8.1 shows that 10-node instance and 20-node instance always have the same $R$ value given a certain demand value. In fact, for $k$-fixed demand, the number of links needed is $\lceil k/(10 - k)\rceil |V|/2$. This is because when each node has a fixed demand of $k$, the spare capacity of each node is $10 - k$. So, in order to achieve full protection, each node needs $\lceil k/(10 - k)\rceil$ neighbors. Thus, the number of links needed is $\lceil k/(10 - k)\rceil |V|/2$. For example, when $|V| = 10$ and demand is 7-fixed, the number of links needed is $\lceil 7/(10 - 7)\rceil \times 10/2 = 15$. For all instances shown in the table, the number of links needed by the optimal solution is equal to $\lceil k/(10 - k)\rceil |V|/2$. However, this
is not true in general because the goal of the MPMC problem is not to minimize the number of links needed. So, the optimal solution may require more links than $\lceil k/(10 - k) \rceil |V|/2$. For the instances shown in Table 8.1, it happens that the optimal solution also minimizes the number of links needed. Furthermore, when the number of links is $\lceil k/(10 - k) \rceil |V|/2$, $R$ is $\lceil k/(10 - k) \rceil /2$, which does not depend on $|V|$. This explains why a 10-node $k$-fixed demand instance and a 20-node $k$-fixed demand instance have the same $R$ value in the table.

Fig. 8.3 shows the optimal solutions for two instances with $|V| = 20$. The left figure shows the optimal solution when each node has a fixed demand of 5. The links in the figure are drawn between node pairs in the optimal solution. Since each node has a fixed demand of 5, each node has 5 units of spare capacity. Thus, once two nodes are connected, each can provide full protection to the other. The left figure shows that each node has exactly one neighbor and a total of five links are needed. The right figure shows the optimal solution when each node has a fixed demand of 6. In this case, each node has a spare capacity of 4. So, if a node is connected to two other nodes, then it can be fully protected. Thus, the number of links needed to achieve full protection is 20. The right figure shows that the optimal solution requires 20 links and each node has exactly two neighbors.

We also run the heuristic algorithm on the same set of problem instances and the results are reported in Table 8.2. The results show that the heuristic solutions are close to optimal solutions in terms of both cost and number of links required for protection. For the case of 20-node with 6-random demand, the heuristic algorithm even finds the optimal solution. Moreover, the running time of the heuristic
Table 8.2  Heuristic solutions to different instances of the MPMC problem.

| Demands | |V| = 10 | |V| = 20 |
|---------|-------------------|-------------------|
|         | cost     | #links | #links/| cost     | #links | #links/|
| 5-random| 1790     | 13     | 1.3   | 3026     | 32     | 1.6   |
| 6-random| 2968     | 18     | 1.8   | 2946     | 26     | 1.3   |
| 7-random| 4546     | 23     | 2.3   | 6923     | 43     | 2.15  |
| 8-random| 8204     | 35     | 3.5   | 12644    | 75     | 3.75  |
| 5-fixed | 736      | 6      | 0.6   | 389      | 11     | 0.55  |
| 6-fixed | 1509     | 11     | 1.1   | 2276     | 22     | 1.1   |
| 7-fixed | 2411     | 16     | 1.6   | 3884     | 32     | 1.6   |
| 8-fixed | 3584     | 21     | 2.1   | 5845     | 42     | 2.1   |

algorithm for all problem instances is only tens of milliseconds. On the other hand, solving the ILP model takes a few hours for problem instances with 20 nodes and the running time increases to days for 30-node problem instances.

In Figures 8.4 and 8.5, we compare the cost of the optimal and heuristic solutions for different problem instances. The top bar chart in Figure 8.4 compares the cost of optimal and heuristic solutions in the 10-node instances with fixed demands and the bottom bar chart in Figure 8.4 compares that under
random demands. The average cost increase of the heuristic solution over optimal solution with fixed demands is 7.36\% and the increase under random demands is 4.1\%. In Figure 8.5, we compare the cost of optimal and heuristic solutions in 20-node instances under both fixed and random demands. With fixed demands, the cost of heuristic solution is on the average 5.4\% more than that of optimal solution. And for random demands, the cost increase is 6.98\%. The results show that the heuristic algorithm performs very well in obtaining near-optimal solutions.

8.7 Conclusion

In this chapter we propose a protection scheme for the hybrid wireless-optical broadband-access network (WOBAN). The idea is to connect the backup ONUs in different segments so that the traffic in one segment can be protected by the spare capacity in neighbor segments. We define the maximum protection with minimum cost (MPMC) problem and show that the optimal solution to an instance of the MPMC problem can be obtained by solving the minimum cost maximum flow (MCMF) problem on a graph constructed from the instance. We prove that the decision problem of MPMC is NP-Hard.
and propose a heuristic algorithm for MPMC. The numerical results show that the proposed protection scheme is much more cost-effective than employing self-protecting PON architectures. In addition, the heuristic algorithm is very effective in obtaining near-optimal solutions.
CHAPTER 9. CONCLUSIONS AND FUTURE WORK

9.1 Contributions of this Work

In this dissertation, we first study two link failures for unicast sessions in backbone networks. After proving two theorems about double-link failure protection, we propose one ILP model for the \( p \)-cycle design problem for static traffic. The basic idea of this ILP model is to use two link-disjoint protection segments to protect each working link. Since the ILP model is only suitable for static traffic, we present two heuristic algorithms to provide the protection against double-link failures for dynamic traffic. According to the numerical results, compared with the other methods, SPPP’s gain in restoration speed is much larger than its loss in protection redundancy. To decrease the protection capacity, we present a new hybrid protection/restoration scheme to handle two-link failures. Basically, our hybrid scheme uses protection to ensure that most of the affected demands can be restored using the pre-planned backup paths upon a two-link failure. For the demands not restorable with protection, we use dynamic restoration to find new backup paths for them. Our scheme is capable of restoring the same set of demands as Dedicated Path Protection (DPP) with significantly less backup capacity.

Next we propose three schemes to protect dynamic multicast sessions against single link failures. The \( p \)-Cycle-based link protection scheme, intelligent \( p \)-Cycle (IpC) scheme, provides \( p \)-Cycle protection against single link failure for dynamic multicast sessions. After the multicast tree is computed for one multicast request, the IpC scheme computes a set of high efficient \( p \)-cycles to protect every link on the multicast tree. The efficiency of one \( p \)-cycle is defined as the ratio of the number of protected capacity to the number of reserved capacity on this \( p \)-cycle. We continue to search the most efficient \( p \)-cycles until all links on the multicast tree are protected. With IpC, both intra-session sharing and inter-session sharing are achieved since a \( p \)-cycle can provide protection to links belonging to not only the same multicast tree, but also different multicast trees. This link protection scheme has short restoration time and
is more capacity efficient compared with existing link protection schemes. We also propose two path protection schemes for dynamic multicast sessions: a p-cycle-based path protection ($P^3$) scheme and a PXT-based path protection scheme. Given a multicast tree $T$, the $P^3$ scheme uses the path-disjoint strategy to compute a set of p-cycles on-demand to ensure every destination node in $T$ is protected. $P^3$ creates new efficient p-cycles only if existing p-cycles are not sufficient to protect destination nodes in the current multicast session. A similar idea is used in the PXT-based path protection scheme for dynamic multicast sessions. To protect a multicast tree, we compute a PXT for each destination node $v$ such that the PXT can be used to restore the traffic to $v$ when a link failure occurs. The performance comparison of the $P^3$ scheme and the PXT based path protection scheme shows that the p-Cycle based protection scheme is more capacity efficient in dense networks.

Lastly, we propose a new protection scheme for the hybrid wireless-optical broadband-access network (WOBAN). The scheme is cost-effective and requires pairs of backup ONUs to be connected with fibers so that each backup ONU is connected to at least one backup ONU in another segment. Basically, once the OLT in segment $i$ fails, all traffic in segment $i$ will be switched to the neighbor backup ONUs which will distribute the traffic via the wireless gateways so that each ONU in the segment handles the traffic using its spare capacity. If an ONU in segment $i$ fails, then the traffic normally handled by the failed ONU will be handled by the other ONUs in segment $i$ if they have enough spare capacity to handle the affected traffic. Otherwise, the affected traffic that cannot be handled within segment $i$ will be switched to the neighbor segments by the backup ONU. Based on the proposed protection scheme, we formalize the maximum protection with minimum cost (MPMC) problem and present one optimal ILP solution for the MPMC problem. We prove the MPMC problem is NP-Hard and provide one heuristic algorithm. The numerical results show that the heuristic algorithm is very effective in obtaining near-optimal solutions.

### 9.2 Future Works

In this dissertation, we studied the survivability schemes against single and double link failures for unicast and multicast sessions in WDM optical networks. In fact, node failure is another type of failure in WDM optical networks. Normally one node failure will cause multiple link failures and has
much more severe impact compared to single link failure. As for protecting unicast connections against single node failure, it has already been well studied and we just need to reserve one backup path which is node-disjoint with the working path. But protecting multicast connections against node failures has not drawn much research attention. With the increase of multicast applications in the Internet, this could be a challenging and interesting research topic.

All protection mechanisms presented in this dissertation are designed for single-domain network and we assume each node in the network has a complete vision of the network, which is not realistic in multi-domain networks. A multi-domain network is a network composed of several independent single-domain networks and every single-domain network has separate rules of operation and management. Thus it is not possible to directly apply our proposed protection schemes in multi-domain networks. So it is interesting and practical to extend our proposed protection schemes across multi-domain networks.
ACKNOWLEDGEMENTS

I would like to thank all people who have helped and inspired me during my study and research in Iowa State University.

I especially want to express my sincere appreciation to my advisors, Lu Ruan and Wensheng Zhang, for their guidance, help and support throughout this research. Their enthusiasm in research motivated me in the past four years. In addition, they were always accessible for our research discussion. I also would like to thank my committee members Ahmed E. Kamal, David Fernandez-Baca and Johnny S. Wong for their help in completing this work.

All my friends in the Iowa State University made it a great place to live and study. It has been a great experience to collaborate with Hsinyi Jiang, Long Long, Hua Qing, Chuang Wang and Wei Zhang, from whom I have learned a passion for life and research. I would also take this opportunity to express my thanks to Yetian Chen, Ru He, Chang Liu, Jia Tao, Jiang Tian, Xia Wang, Liyuan Xiao, Ming Xu, Toby Xu, Xinyuan Zhao and Fuchao Zhou. Their friendship and help inspired me in research and made Ames such a pleasant place to be in for all these years.

Support for this research has been provided in part by Iowa State University, and in part by the National Science Foundation.
BIBLIOGRAPHY


[16] Lu Ruan and Taiming Feng, “A Hybrid Protection/Restoration Scheme for Two-Link Failure in WDM Mesh Networks,” in *GLOBECOM2010*, Dec. 6-10, 2010, MIAMI, USA.


