# Two-Link Failure Protection in WDM Mesh Networks with $p$-Cycles

Taiming Feng[a], Long Long[b], Ahmed E. Kamal[b], Lu Ruan[a]

[a]*Department of Computer Science, Iowa State University, Ames, IA 50011*
[b]*Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011*

**Abstract**

In WDM networks, it is important to protect connections against link failures due to the high bandwidth provided by a fiber link. Although many $p$-cycle based schemes have been proposed for single-link failure protection, protection against two-link failures have not received much attention. In this paper, we propose $p$-cycle based protection schemes for two-link failures. 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.

*Key words:*
WDM , Survivability , p-Cycle , Two-link Failure

## 1. Introduction

Network survivability is an important requirement for WDM optical networks due to their ultra-high capacity. Various protection schemes have been developed for WDM networks. Ring-based protection schemes enable traffic recovery to be completed in 50-60 ms, but require at least 100% capacity redundancy. On the other hand, mesh-based protection schemes are much more capacity efficient, attributed to diverse traffic routing and protection capacity sharing among different connections. However, more complicated protection switching process leads to much longer recovery time. $p$-Cycle is a promising protection technique as it

*Email addresses:* taiming@iastate.edu (Taiming Feng), longlong@iastate.edu (Long Long), kamal@iastate.edu (Ahmed E. Kamal), ruanlu@iastate.edu (Lu Ruan)

achieves the speed of ring with the efficiency of mesh [1], [2]. $p$-Cycles are established by configuring the spare capacity into pre-cross-connected cycles. Upon a link failure, protection switching is performed at the two endnodes of the failed link and other switching nodes will not be reconfigured. Therefore, traffic recovery is extremely fast. $p$-Cycle is also efficienct in protection since it protects both on-cycle links and straddling links. As shown in Fig. 1, a-b-c-d-f-a is a $p$-Cycle. For the on-cycle link a-b, the $p$-Cycle provides one protection path a-f-d-c-b. For the straddling link a-c, the $p$-Cycle provides two protection paths: a-b-c and a-f-d-c. Thus, $p$-Cycle can protect one unit of working capacity on every on-cycle link and protect two units of working capacity on every straddling link. Moreover, the $p$-Cycle has the property to provide protection for multiple failures. For example, if f-b and f-c both fail, f-a-b can provide protection for f-b and f-d-c can provide the protection for f-c.

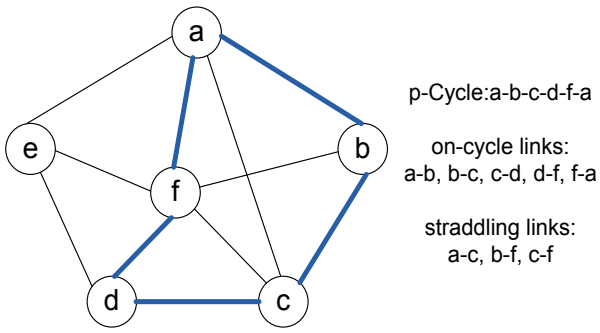


Figure 1: A $p$-Cycle Example

The concept of $p$-cycle was first proposed in [3] and subsequently many works in literature study the $p$-cycle design problem for protecting against single-link failures. Most of these works assume the demands have been routed and seek to find the optimal set of $p$-cycles to protect the given working capacity [3], [4], [5]. The joint optimization of the working path routes and the $p$-cycles is studied in [6]. An extension of the basic $p$-cycle concept called Failure Independent Path-Protecting (FIPP) $p$-cycle is proposed in [7], which leads to more capacity efficient network designs than link protecting $p$-cycle. Recently, the author of [8] introduced a new $1 + N$ protection scheme against single-link failures by combining network coding and $p$-cycles.

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 [9]. Third, if an optical switch with two links connected to it fails, then both links fail. Double-link failure protection has been addressed in some works. In [10], 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 the scheme proposed in [9], two link-disjoint backup paths are computed for each link so that the network is two-link failure survivable. The similar scheme was also proposed in [11] and the problem was formulated as an Integer Linear Program. The scheme is slow in recovery because the backup paths are configured after link failure occurs. In [12], 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 [13] proposed algorithms to find $k$ disjoint $p$-cycles to protect each link such that the network is $k$ link-failure survivable. The author of [14] extended his work in [8] to protect multiple-link failures by using network coding and $p$-cycles.

In this paper, 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 preconfigured using the spare capacity in the network, extremely fast recovery can be achieved. We formulate an ILP model for the $p$-cycle design problem for static traffic model in which the set of connections to be established is given a priori. We propose two protection schemes for dynamic traffic. In the dynamic traffic model, connection requests arrive at the network one by one and the network knows nothing about the bandwidth requirement, source and destination node of incoming requests. Thus, primary and backup lightpaths need to be computed online according to the utilization of the current network resources. We also study the performance in incremental traffic which is a special case of dynamic traffic and the demand never terminates once it is satisfied.

The rest of this paper is organized as follows. In Section 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. In Section 4, we propose two double-link failure protection schemes for dynamic traffic. Numerical results are presented in Section 5 and conclusions are given in Section 6.

## 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.

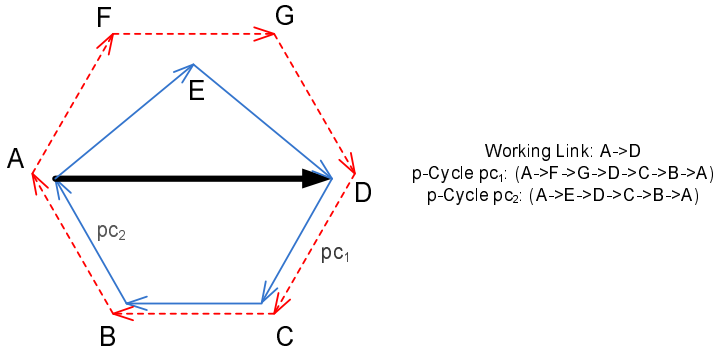


Figure 2: Two-Link Failure Protection $p$-Cycles for Link $A{\rightarrow}D$

In [13], 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 $pc$, only part of the $p$-cycle is used for protection. We name the part of $pc$ that carries the traffic when $e$ fails as the *protection segment* for $e$ on $pc$, which is denoted by $pc(e)$. Fig. 2 shows two $p$-cycles $pc_1$ and $pc_2$, both of which can protect link $A{\rightarrow}D$. $pc_1(A{\rightarrow}D){=}(A{\rightarrow}F{\rightarrow}G{\rightarrow}D)$ is the protection segment for link $A{\rightarrow}D$ on $pc_1$ and $pc_2(A{\rightarrow}D){=}(A{\rightarrow}E{\rightarrow}D)$ is the protection segment for link $A{\rightarrow}D$ on $pc_2$. Although $pc_1$ and $pc_2$ 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 $pc_1(A{\rightarrow}D)$ and $pc_2(A{\rightarrow}D)$ are link-disjoint.

The following theorem gives the sufficient condition for the traffic on 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 $pc_1$ and $pc_2$ such that the following conditions are met.*

1. $pc_1$ *can protect link* $A \rightarrow B$*;*
2. $pc_2$ *can protect link* $A \rightarrow B$*;*
3. $pc_1(A \rightarrow B)$ *is link-disjoint with* $pc_2(A \rightarrow B)$.

**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 $pc_1(A \rightarrow B)$ and $pc_2(A \rightarrow B)$. 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 the protection of a common $p$-cycle. Let $e_1$ and $e_2$ be two working links. Let $S_1$ and $S_2$ be a pair of protection-segment-disjoint $p$-cycles for $e_1$ and $e_2$, respectively. 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 $e_2$ are protected by the same pair of $p$-cycles. When two links share one or two $p$-cycles, it is 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* $pc_1 \in S_1$ *such that* $e_2 \notin pc_1(e_1)$.
2. *For link* $e_2$*, there exists a* $p$*-cycle* $pc_2 \in S_2$ *such that* $e_1 \notin pc_2(e_2)$*;*
3. $pc_1(e_1)$ *is link-disjoint with* $pc_2(e_2)$ *if* $pc_1 = pc_2$.

**Proof** Upon a two-link failure, we consider the case that only one of $e_1$ and $e_2$ fails, say $e_1$, so $e_2$ is still working and only $e_1$ needs to be protected. Based on Theorem 1, there must still exist one protection segment for $e_1$ that is not affected by another link failure, since $e_1$ is protected by two link-disjoint segments. Thus, both $e_1$ and $e_2$ can survive regardless of protection sharing.

Thus, we only need to focus on the case where both $e_1$ and $e_2$ fail. Conditions 1 and 2 ensure that both $pc_1(e_1)$ and $pc_2(e_2)$ are not affected by the failures. If $pc_1 \neq pc_2$, then $e_1$ and $e_2$ are protected by $pc_1$ and $pc_2$, respectively. In this case, the sharing is valid. If $pc_1 = pc_2$, then $pc_1(e_1)$ has to be link-disjoint with $pc_1(e_2)$ described in condition 3. Otherwise, one unit protection capacity provided by a $p$-cycle is not enough to protect two failed link at a time. Thus, $pc_1(e_1)$ and $pc_1(e_2)$ have to be link-disjoint to validate the sharing.

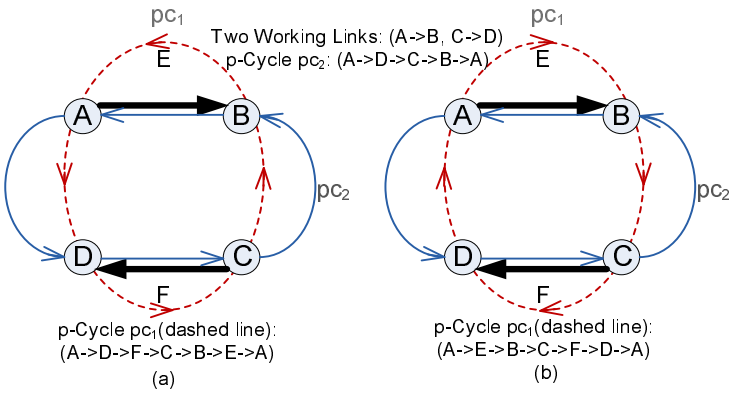


Figure 3: (a). $pc_1, pc_2$ fail to protect $e_1, e_2$ simultaneously; (b). $pc_1, pc_2$ protect $e_1, e_2$ simultaneously.

Fig. 3 shows two examples of $p$-cycle sharing. In the example shown in Fig. 3(a), two working links $e_1$=$A$→$B$ and $e_2$=$C$→$D$ are protected by the same pair of $p$-cycles, $pc_1$ and $pc_2$, where both $e_1$ and $e_2$ are straddling links of $pc_1$ and on-cycle links of $pc_2$. That is, $S_1 = S_2 = \{pc_1, pc_2\}$. When both $e_1$ and $e_2$ fail, $pc_2$ can protect neither of them since $e_2 \in pc_2(e_1)$ and $e_1 \in pc_2(e_2)$. $pc_1$ can be used to protect either $e_1$ or $e_2$ but not both because $pc_1(e_1)$=($A$→$D$→$F$→$C$→$B$) and $pc_1(e_2)$=($C$→$B$→$E$→$A$→$D$) are not link-disjoint. Therefore, $e_1$ and $e_2$ cannot validly share the $p$-cycles $pc_1$ and $pc_2$. We now consider the example shown in Fig. 3(b), where everything is the same except that the direction of $p$-cycle $pc_1$ is reversed. In this case, $pc_1(e_1) = (A \rightarrow E \rightarrow B)$ does not contain $e_2$, $pc_1(e_2) = (C \rightarrow F \rightarrow D)$ does not contain $e_1$, and $pc_1(e_1)$ and $pc_1(e_2)$ are link-disjoint. According to Theorem 2, $e_1$ and $e_2$ can validly share $pc_1$ and $pc_2$.

## 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 $G = (V, E)$, the working capacity $d_{ab}$ on each link $(a \rightarrow b) \in E$, and the maximum number of $p$-cycles needed, 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.

In the input parameters, $P$ is the upper bound of the number of $p$-cycles needed and can be computed according to the static demands. Suppose the static demands need $M$ links with one unit of capacity reserved on each link, a total of $2M$ $p$-cycles will be required in the worst case to protect any double-link failures since

each link requires two $p$-cycles. In the results obtained by solving an ILP, variables $e^p_{mn}$ $(\forall(m{\rightarrow}n){\in}E)$ identify the configuration of those computed $p$-cycles. Given a $p$-cycle $pc_p$, if all the corresponding variables $e^p_{mn}$ equal 0, this $p$-cycle is not used to protect any link in the solution.

Input Parameters:

| | |
|---|---|
| $P$ | 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 $a{\rightarrow}b$. |

Variable Notations:

| | |
|---|---|
| $e^p_{mn}$ | binary variable, 1 if $p$-cycle $p$ uses link $m{\rightarrow}n$ as an on-cycle link. |
| $z^p_n$ | binary variable, 1 if node $n$ is on $p$-cycle $p$. |
| $x^p_{ab\_k}$ | binary variable, 1 if $p$-cycle $p$ protects the $k^{th}$ working capacity on link $a{\rightarrow}b$. |
| $f^{p,(ab\_k)}_{mn}$ | binary variable, 1 if $p$-cycle $p$ protects the $k^{th}$ working capacity on link $a{\rightarrow}b$ and the protection segment traverses link $m{\rightarrow}n$. |
| $v^{p,(ab\_k)}_{cd}$ | 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 $d{\rightarrow}c$. |
| $AB^{p,(ab\_k)}_{cd\_l}$ | binary variable, it equals $\lvert v^{p,(ab\_k)}_{cd} - v^{p,(cd\_l)}_{ab} \rvert$. |
| $C^{p,(ab\_k)}_{cd\_l}$ | binary variable, used in the absolute value constraints for $AB^{p,(ab\_k)}_{cd\_l}$. |

Objective:

$$\text{Minimize} \sum_{p} \sum_{(m,n)\in E} e^p_{mn}$$

The objective function sums the total capacity used by all the active $p$-cycles.

1) Capacity Constraints:

$$\sum_{p} x^p_{ab\_k} \geq 2, \quad \forall(a{\rightarrow}b) \in E,\ \forall k \leq d_{ab}; \tag{1}$$

$$\sum_{k} x^p_{ab\_k} \leq 1, \quad \forall p,\ \forall(a{\rightarrow}b) \in E; \tag{2}$$

Equation (1) ensures that each working unit on a link should be protected by at least two $p$-cycles. Equation (2) ensures that a unidirectional $p$-cycle can protect only one unit capacity on a given link.

2) Cycle Constraints:

$$\sum_{(m\to n)\in E} e^p_{mn} = \sum_{(n\to l)\in E} e^p_{nl} = z^p_n,\ \forall p,\ \forall n \in V; \tag{3}$$

$$e^p_{mn} + e^p_{nm} \le 1, \quad \forall p,\ \forall (m\to n) \in E; \tag{4}$$

Equations (3) is the flow conservation constraint for any simple cycle by ensuring that the in-degree and out-degree of every on-cycle node is 1. Equation (4) ensures that each unidirectional $p$-cycle $p$ cannot traverse the same link twice in both directions.

3) Link Protection Constraints:

$$\sum_m f^{p,(ab\_k)}_{mn} - \sum_l f^{p,(ab\_k)}_{nl} = \begin{cases} x^p_{ab\_k} & \text{if } n = b \\ -x^p_{ab\_k} & \text{if } n = a \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$\forall p,\ \forall (a\to b) \in E,\ \forall n \in V,\ \forall k \le d_{ab};$$

$$\sum_m f^{p,(ab\_k)}_{ma} = \sum_n f^{p,(ab\_k)}_{bn} = 0,\ \ \forall p,\ \forall (a\to b) \in E,\ \forall k \le d_{ab}; \tag{6}$$

$$f^{p,(ab\_k)}_{mn} \le e^p_{mn},\ \ \forall (a\to b), (m\to n) \in E,\ (a\to b) \ne (m\to n),\ \forall p,\ \forall k \le d_{ab}; \tag{7}$$

Equation (5) and (6) are the flow conservation constraints for each protection segment provided by a $p$-cycle $p$ to protect the $k^{th}$ working capacity on link $a\to b$. In this case, a unit of protection flow should be reserved from node $a$ to $b$ using the on-cycle links of $p$. But there is no incoming flow of source node $a$ and outgoing flow of $b$ along the protection segment. Equation (7) ensures that any link $(m\to n)$ used by a protection segment should be an on-cycle link of the protection $p$-cycle.

4) Protection Segment Disjointness Constraints:

$$f^{p,(ab\_k)}_{mn} + f^{q,(ab\_k)}_{mn} \le 1, \tag{8}$$

$$f^{p,(ab\_k)}_{mn} + f^{q,(ab\_k)}_{nm} \le 1, \tag{9}$$

$$\forall (a\to b), (m\to n) \in E, (a\to b) \ne (m\to n) \text{ or } (n\to m),\ \forall p, q, p \ne q,\ \forall k \le d_{ab};$$

Any link should be link-disjoint with its protection segment in any direction, which is guaranteed by Equation (8) and (9).

5) Absolute Value Constraints:

$$v_{cd}^{p,(ab\_k)} = (x_{ab\_k}^{p} - f_{cd}^{p,(ab\_k)} - f_{dc}^{p,(ab\_k)}), \tag{10}$$
$$\forall (a{\to}b),(c{\to}d) \in E,\ (a{\to}b) \neq (c{\to}d),\ \forall p, \forall k \leq d_{ab};$$
$$AB_{cd\_l}^{p,(ab\_k)} \geq v_{cd}^{p,(ab\_k)} - v_{ab}^{p,(cd\_l)}, \tag{11}$$
$$AB_{cd\_l}^{p,(ab\_k)} \geq -(v_{cd}^{p,(ab\_k)} - v_{ab}^{p,(cd\_l)}), \tag{12}$$
$$AB_{cd\_l}^{p,(ab\_k)} \leq v_{cd}^{p,(ab\_k)} - v_{ab}^{p,(cd\_l)} + 2C_{cd\_l}^{p,(ab\_k)}, \tag{13}$$
$$AB_{cd\_l}^{p,(ab\_k)} \leq -(v_{cd}^{p,(ab\_k)} - v_{ab}^{p,(cd\_l)}) + 2(1 - C_{cd\_l}^{p,(ab\_k)}), \tag{14}$$
$$\forall (a{\to}b),(c{\to}d) \in E,\ (a{\to}b) \neq (c{\to}d),\ \forall p,\ \forall k \leq d_{ab}, l \leq d_{cd}.$$

Equation (10) defines $v_{cd}^{p,(ab\_k)}$, which will be used for protection sharing between link $ab\_k$ and $cd\_l$. The absolute value $AB_{cd\_l}^{p,(ab\_k)}$ is defined by equations (11) - (14). Eq. (11) and (12) make sure that the absolute value is always greater and equal to 0. However, they are not enough. When both $v$ variables equal each other, the absolute value should be 0. But it can still be either 0 or 1. In order to make it equal 0, we have to introduce a new binary variable, $C_{cd\_l}^{p,(ab\_k)}$. Eq. (13) and (14) ensure that when both $v$ variables equal 0 or 1 at the same time, the absolute value $AB$ can only be 0 by randomly choosing $C$ as either 0 or 1. Meanwhile, equation (11) and (12) are not violated.

6) $p$-Cycle Sharing Constraints:

$$f_{mn}^{p,(ab\_k)} + f_{mn}^{p,(cd\_l)} + v_{cd}^{p,(ab\_k)} + v_{ab}^{p,(cd\_l)}$$
$$\leq \sum_{p} v_{cd}^{p,(ab\_k)} + \sum_{p} v_{ab}^{p,(cd\_l)} + \sum_{p} AB_{cd\_l}^{p,(ab\_k)} + 1, \tag{15}$$
$$\forall (a{\to}b),(c{\to}d){\in}E,\ (a{\to}b){\neq}(c{\to}d),\ \forall p,\ \forall (m{\to}n){\in}E,\ \forall \{k,l\}{\leq}d_{ab}.$$

Constraint (15) ensures that all $p$-cycle sharing are valid based on Theorem 2. It takes the following three cases into the consideration when both link $(a{\to}b)$ and $(c{\to}d)$ fail simultaneously.

If $\sum_p AB_{cd\_l}^{p,(ab\_k)} \geq 1$, link $(a{\to}b)$ and $(c{\to}d)$ can be protected by two different $p$-cycles, because there exist at least one $p$ such that $|v_{cd}^{p,(ab\_k)} - v_{ab}^{p,(cd\_l)}| = 1$. Assume that $v_{cd}^{p,(ab\_k)} = 1, v_{ab}^{p,(cd\_l)} = 0$, then $p$ can be used to protect link $(a{\to}b)$ without traversing link $(c{\to}d)$. Meanwhile, there must exist another $p'$ that can protect link $(c{\to}d)$ without traversing link $(a, b)$, since there are two link-disjoint protection segments for $(c{\to}d)$.

If $\sum_p AB_{cd\_l}^{p,(ab\_k)} = 0$, both link $(a{\to}b)$ and $(c{\to}d)$ share the same two $p$-cycles. There are two cases to be discussed as follows:

1) If $\sum_p v_{cd}^{p,(ab\_k)} + \sum_p v_{ab}^{p,(cd\_l)} = 4$, both link $(a{\rightarrow}b)$ and $(c, d)$ are straddling links of the two protection $p$-cycles. In this case, one of the two $p$-cycles can protect $(a{\rightarrow}b)$ and the other one can protect $(c{\rightarrow}d)$ when both links fail.

2) If $\sum_p v_{cd}^{p,(ab\_k)} + \sum_p v_{ab}^{p,(cd\_l)} = 2$, one of the protection $p$-cycles actually traverses both links and cannot be used for protection anymore. Thus, only one $p$-cycle $p$ can be used to protect them. 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 the protection segment $p(a{\rightarrow}b)$ and $p(c{\rightarrow}d)$ are link-disjoint.

Constraint (15) combines all three cases together to ensure that all $p$-cycle sharing are valid. Note that the condition $\sum_p v_{cd}^{p,(ab\_k)} + \sum_p v_{ab}^{p,(cd\_l)} \geq 2$ always holds. Thus, if $\sum_p AB_{cd\_l}^{p,(ab\_k)} \geq 1$, the sharing is valid. There is no need to address the remaining two cases and Eq. (15) should not be violated. If $\sum_p AB_{cd\_l}^{p,(ab\_k)} = 0$, Eq. (15) ensures that one of the last two cases will occur.

Current objective is to minimize the spare capacity required by the $p$-cycles. However, this objective could be easily modified to achieve different goals. For instance: 1. If the goal is to minimize the maximum total capacity (working and spare capacity) used on any link, we can introduce a new variable $\zeta$ and a new constraint: $\zeta \geq d_{mn} + \sum e_{mn}^{p},\ \forall(m{\rightarrow}n) \in E$. In this newly added constraint, $\zeta$ is the maximal aggregated amount of capacity reserved by working capacity and protection $p$-cycles on every link. Accordingly, the new objective will be: *Minimize* $\zeta$. 2. In the second case, if the total amount of capacity provided by each link $(m{\rightarrow}n)$ is upper bounded by a number, denoted by $\lambda_{mn}$, we can introduce a new constraint: $\lambda_{mn} \geq d_{mn} + \sum e_{mn}^{p},\ \forall(m{\rightarrow}n) \in E$ to ensure that the total capacity reserved on each link will not exceed the limit. Therefore, our ILP model can be modified in a flexible fashion to adapt to various network scenarios with different design goals.

## 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. The notations and some functions used in the algorithms are explained in Table 1.

Table 1: Notations used in the algorithms

| Notations | Meaning |
|---|---|
| $\mathbb{P}$ | The set of links used by the working path of a given connection |
| $pc_p$ | A $p$-cycle indexed by $p$ |
| $pc_p(e)$ | the protection segment on $pc_p$ that actually protect link $e$ |
| $\mathbb{C}$ | The set of all the existing $p$-cycles in the network |
| $\mathbb{C}(e)$ | The set of existing $p$-cycles that can protect link $e$ |
| $\mathbb{C}_{temp}$ | The set of protection segments that protect a set of links |
| $cycle\_build(e, n)$ | Construct $n$ new cycles that can protect $e$ |
| $check\_share(pc_i, pc_j, e)$ | Check whether $p$-cycle $pc_i$ and $pc_j$ can protect link $e$ simultaneously in Algorithm 1 |
| $check\_disjoint(pc_i, e)$ | Check whether an existing $p$-cycle $pc_i$ can protect a working link $e$ in Algorithm 2 |
| $check\_share2(e, e', pc_i)$ | Check whether link $e$ and $e'$ can share the protection of $pc_i$ validly in Algorithm 2 |

### *4.1. Shortest Path Pair Protection Scheme*

We propose the Shortest Path Pair Protection (SPPP) scheme in this section. Given the working path $\mathbb{P}$ of a connection, SPPP computes a set of $p$-cycles to protect $\mathbb{P}$ as follows. For each link on $\mathbb{P}$, 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 provisioned to minimize the total protection capacity.

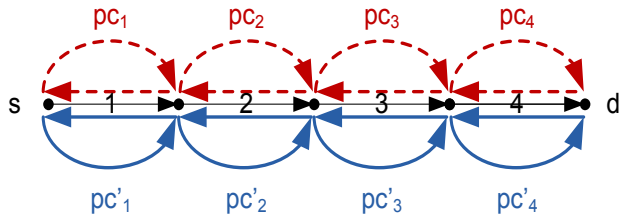


Figure 4: $p$-Cycles $pc_i, pc_i'$ protect link i (i=1,2,3,4)

Fig. 4 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, $pc_i$ and $pc'_i$ 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.

SPPP uses a boolean function $check\_share(pc_1, pc_2, e)$ to check whether two $p$-cycles can be used to protect a working link $e$. The checking procedure consists of three steps. First, it checks whether $e$ can be protected by $pc_1$ and has valid sharing with other links also protected by $pc_1$; Second, check whether $e$ can be protected by $pc_2$ with valid sharing; Third, whether $pc_1(e)$ and $pc_2(e)$ are link-disjoint. $check\_share(pc_1, pc_2, e)$ returns true if all three steps are passed. The rules are actually based on Theorem 1 and 2. Given a working link $e$, the set of existing $p$-cycles that can protect $e$ is denoted by $\mathbb{C}(e)$. That is, $\mathbb{C}(e)$ contains all existing $p$-cycles that have $e$ as an on-cycle link or a straddling link. For each link $e \in \mathbb{P}$, SPPP computes two $p$-cycles for $e$ as follows:

We try to use as many existing $p$-cycles as possible to maximize the sharing. We first check whether there exist two $p$-cycles in $\mathbb{C}(e)$ 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 $check\_share$ function. If two $p$-cycles cannot be found in $\mathbb{C}(e)$, we try to reuse at least one $p$-cycle. By checking the protection condition of a $p$-cycle, say $pc_i$, we need to construct the second $p$-cycle $pc_j$ for $e$ such that $pc_i(e)$ and $pc_j(e)$ are link-disjoint. If $check\_share(pc_i, pc_j, e)$ returns true, then $e$ is protected against double-link failure. Finally, if none of the $p$-cycles in $\mathbb{C}$ can be reused to protect $e$, then we construct two new $p$-cycles for $e$ such that the protection segments for $e$ on these two $p$-cycles are link-disjoint.

The function $cycle\_build(e, n)$ is used to construct new cycles where $n$ is the number of newly constructed cycles. If $n = 1$, the function finds the shortest path that is link-disjoint with $e$ using Dijkstra's algorithm and then combines the same link $e$ in the opposite direction to form a cycle, which can be used to protect $e$. If $n = 2$, we first use Bhandari's algorithm [15] to obtain a pair of link-disjoint paths between the two end nodes of $e$ with minimum total length by temporarily marking it invalid. We can 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 pseudo-code of the SPPP scheme is shown in Algorithm 1. The input is a working path $\mathbb{P}$ and the set of the existing $p$-cycles, the output is a new set $\mathbb{C}$ of $p$-cycles that protect $\mathbb{P}$. The algorithm computes two $p$-cycles for each link $e \in \mathbb{P}$

**Algorithm 1**: SPPP Scheme

**Input**: Working Path $\mathbb{P}$, the set of existing $p$-cycles, $\mathbb{C}$
**Output**: A updated set $\mathbb{C}$

1 **for** $\forall e \in \mathbb{P}$ **do**
2  flag=2;
3  **if** $\exists\{pc_i, pc_j\} \in \mathbb{C}(e)$ *s.t.* $check\_share(pc_i, pc_j, e)$==*true* **then**
4  flag=0;
5  $\mathbb{C}(e) = \mathbb{C}(e) - \{pc_i, pc_j\}$;
6  **end**
7  **else if**
   $(\exists pc_i \in \mathbb{C}(e)) \wedge (pc_j = cycle\_build(e, 1)) \wedge (check\_share(pc_i, pc_j, e))$
   **then**
8  flag = 1;
9  $\mathbb{C}(e) = \mathbb{C}(e) - \{pc_i\}$;
10  $\forall e' \neq e$ that can be protected by $pc_j$, $\mathbb{C}(e') = \mathbb{C}(e') \bigcup \{pc_j\}$;
11  $\mathbb{C} = \mathbb{C} \bigcup \{pc_j\}$;
12  **end**
13  **if** *flag*==*2* **then**
14  construct $pc_1$ and $pc_2$ for $e$ by running $cycle\_build(e, 2)$;
15  $\forall e' \neq e$ that can be protected by $pc_1$, $\mathbb{C}(e') = \mathbb{C}(e') \bigcup \{pc_1\}$;
16  $\forall e' \neq e$ that can be protected by $pc_2$, $\mathbb{C}(e') = \mathbb{C}(e') \bigcup \{pc_2\}$;
17  $\mathbb{C} = \mathbb{C} \bigcup \{pc_1, pc_2\}$;
18  **end**
19 **end**

in the for loop from line 1 to line 19. The first step of the procedure is executed by Lines 3-6, in which we try to find a pair of existing $p$-cycles to protect a given link $e$. The second step is conducted by Lines 7-12, in which we reuse a $p$-cycle $pc_i$ and construct a new one, $pc_j$ to provide protection. The final step is to construct two new $p$-cycles such that their protection segments are link-disjoint, which is realized by Lines 13-18. It is worth noting that each $p$-cycle can only protect a link once. Once a link $e$ has been protected by a given $p$-cycle, say $pc_i$, this $p$-cycle is valid to protect $e$ any more in the future. This is the reason that in Lines 5 and 9, we need to remove the $p$-cycles that have been used to protect $e$ from $\mathbb{C}(e)$.

We now analyze the time complexity of SPPP. First, we check the running time of function $check\_share(pc_i, pc_j, e)$ is $O(|E||V|^2)$ 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 checking process is $O(|V|^2)$. For each $e \in \mathbb{P}$, the algorithm computes two $p$-cycles

for $e$. The time of this computation is dominated by the computation in line 3. We assume $|\mathbb{C}(e)|$ is upper bounded by a constant. Since line 3 is executed for each edge in the working path $\mathbb{P}$ and the number of edges in $\mathbb{P}$ is upper bounded by $|V|$, the complexity of SPPP is $O(|E||V|^3)$.

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 [16] 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.

### *4.2. Shortest Full Path Protection Scheme*

In this section, we present the Shortest Full Path Protection (SFPP) Scheme. Given the working path $\mathbb{P}$ of a connection, SFPP computes a set of $p$-cycles to protect $\mathbb{P}$ as follows. First, we compute one short $p$-cycle for each link on $\mathbb{P}$. Next, we compute a long $p$-cycle that contains all links on $\mathbb{P}$ 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 $\mathbb{P}$. 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.

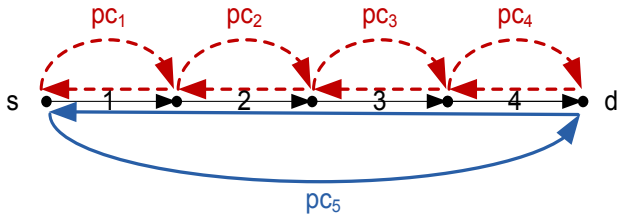


Figure 5: $p$-Cycle $pc_i$ protects link $i$ (i=1,2,3,4) and $pc_5$ protects links 1,2,3, and 4.

Fig. 5 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 $pc_1(link1)$ to $pc_4(link4)$ to ensure that each working link is protected by two protection-segment-disjoint $p$-cycles.

We now explain the details of SFPP. We first find one short $p$-cycle for every link $e$ on the working path $\mathbb{P}$. 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 $\mathbb{C}(e)$ that can be reused to protect $e$. A $p$-cycle $pc_i$ can be reused to protect $e$ if 1) $pc_i$ does not contain any edge $e' \neq e \in \mathbb{P}$, and 2) for every link $e' \neq e \in \mathbb{P}$ that is protected by $pc_i$, $pc_i(e)$ and $pc_i(e')$ are link-disjoint. The first condition is needed because if $pc_i$ 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 $pc_i$. We define a function $check\_disjoint(pc_i, e)$ to check whether those two conditions given a link $e$ and an existing $p$-cycle $pc_i$. It returns true if both conditions are satisfied.

If the function returns false, a new $p$-cycle should be constructed for $e$ with the requirement that it does not contain any edge $e' \neq e$ in $\mathbb{P}$. After each link $e \in \mathbb{P}$ has been protected by a $p$-cycle, we construct a long $p$-cycle as follows. We first mark all the links in $\mathbb{P}$ as invalid as well as all links on the protection segments (provided by the short $p$-cycles) of all the working links in $\mathbb{P}$. We can simply substitute $e$ in the function $cycle\_build(e, n)$ by $\mathbb{P}$ and then run $cycle\_build(\mathbb{P}, 1)$ to form a long $p$-cycle $pc_f$ by combining two link-disjoint paths. One, denoted by $\mathbb{P}'$, starts from the source $s$ to the destination $d$ and the other one is $\mathbb{P}$ in the opposite direction. Hence, each link on the path $\mathbb{P}$ is protected against double-link failure.

However, constructing this new long $p$-cycle may destruct the validity of sharing between any link $e \in \mathbb{P}$ with any link $e' \notin \mathbb{P}$ if $e$ and $e'$ share a short $p$-cycle. We have to make sure that after the construction of $pc_f$, every link $e \in \mathbb{P}$ is still protected and every $p$-cycle sharing is valid. We define a function $check\_share2(e, e', pc_i)$ to perform the checking process. The function returns true if the sharing of a $p$-cycle $pc_i$ by $e$ and $e'$ is valid based on Theorem 2. If the function returns false, we have to reconstruct the long $p$-cycle $pc_f$ because it must contain link $e'$ (We will explain why in the next.) We could make the sharing valid if the new $p$-cycle $pc_f$ does not contain $e'$. Hence, we need to temporarily remove $e'$ from $G$ before construct the new $pc_f$. 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 $pc_i$ by $e$ and $e'$ is caused by the inclusion of $e'$ in $pc_f$. Let $pc'_f$ be the second $p$-cycle that protects $e'$. (The first $p$-cycle that protects $e'$ is $pc_i$, which is shared by $e$.) In order for $e$ and $e'$ to validly share $pc_i$, we have to make sure that when both links fail, at least one of $pc_i$ and $pc'_f$ can protect $e'$ and at least one of $pc_i$ and $pc_f$ can protect $e$. We know that at least one of the protection segments $pc_i(e')$ and $pc'_f(e')$ does not contain $e$ since they must be link-disjoint. Therefore, there are three cases to consider as follows:

1. *Neither* $pc_i(e')$ *nor* $pc'_f(e')$ *contain* $e$: Clearly, $e'$ can be protected by $pc'_f$ since $pc'_f(e')$ is not affected by the failure. In addition, one of $pc_i$ and $pc_f$ can protect $e$ because $pc_i(e)$ 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 $pc_i$.
2. $pc_i(e')$ *contains* $e$ *and* $pc'_f(e')$ *does not contain* $e$: $e$ and $e'$ can validly share $pc_i$ for the same reason given in the previous case.
3. $pc'_f(e')$ *contains* $e$ *and* $pc_i(e')$ *does not contain* $e$: $e'$ has to be protected by $pc_i$ when both $e$ and $e'$ fail. $e$ can be protected by $pc_f$ if $pc_f(e)$ does not contain $e'$. Therefore, the sharing is valid only if $pc_f(e)$ does not contain $e'$.

**Algorithm 2**: SFPP Scheme

**Input**: Working path $\mathbb{P}$, $p$-cycle set $\mathbb{C}$, $\mathbb{C}_{temp} = \phi$ and flag=1
**Output**: A updated set $\mathbb{C}$

1 **for** $\forall e \in \mathbb{P}$ **do**
2  **if** $\exists pc_i \in \mathbb{C}(e)$ *and* $check\_disjoint(pc_i, e)$==*true* **then**
3  link $e$ is protected once;
4  **end**
5  **else**
6  mark $\mathbb{P} \backslash e$ invalid and obtain $pc_i$ by running $cycle\_build(e, 1)$;
7  $\mathbb{C} = \mathbb{C} \bigcup \{pc_i\}$;
8  update $\mathbb{C}(e')$ for all $e'$ that can be protected by $pc_i$;
9  **end**
10  $\mathbb{C}_{temp} = \mathbb{C}_{temp} \cup pc_i(e)$;
11 **end**
12 mark $e \in \mathbb{P}$ and $e' \in \mathbb{C}_{temp}$ invalid in $G$;
13 construct $pc_f$ by running $cycle\_build(\mathbb{P}, 1)$;
14 **for** $\forall e \in \mathbb{P}$ *and* $e' \notin \mathbb{P}$ *that share* $pc_i \in \mathbb{C}(e)$ **do**
15  **if** $check\_share2(e, e', pc_i)$ == *false* **then**
16  mark $e'$ invalid and flag=0;
17  **end**
18 **end**
19 **if** *flag == 0* **then**
20  repeat Line 13;
21  flag = 1 and goto Line 14;
22 **end**
23 Add $pc_f$ to $\mathbb{C}$ and $\mathbb{C}(e')$ for all $e' \in E$ that can be protected by $pc_f$;

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 $e'$.

The pseudo-code of the SFPP scheme is shown in Algorithm 2. The input is a working path $\mathbb{P}$ and the set of existing $p$-cycles $\mathbb{C}$, the output is the updated set $\mathbb{C}$ of $p$-cycles. Set $\mathbb{C}_{temp}$ stores the protection segments that are used to protect the links on $\mathbb{P}$. The first loop from line 1-11 tries to find an existing $p$-cycle to protect each link $e \in \mathbb{P}$. If no existing $p$-cycle in $\mathbb{C}$ can protect a given $e$, then construct a new $p$-cycle. We need to make sure that certain capacity sharing conditions have to be satisfied. The second part from line 12-13 is to construct the long $p$-cycle $pc_f$ for the whole path $\mathbb{P}$. However, $pc_f$ may cause invalid sharing between any link on $\mathbb{P}$ and the links not in $\mathbb{P}$ based on our previous analysis if it traverses any link in $\mathbb{C}_{temp}$. Hence, we need to use function $check\_share2$ to check the validity of $pc_f$. If the function returns false, we need to temporarily remove the link and reconstruct a new $pc_f$ repeatedly until a valid long $p$-cycle is found. This process is described by line 14-22. After we find a $pc_f$ that ensures all $p$-cycle sharing is valid, we add it into set $\mathbb{C}$ and also update $\mathbb{C}(e')$ for each edge $e' \neq e$ that can be protected by $pc_f$ in line 23.

The time complexity of SFPP is dominated by the repeated construction process in lines 15-22. The complexity of function $check\_share2(e, e', pc)$ is $O(|V|^2)$, so the complexity of lines 15-18 is $O(|V||E||V|^2) = O(|E||V|^3)$. 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 $O(|E|^2|V|^3)$.

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 [17], which removes the loop backs and release the redundant capacity by reconfiguring the restored paths.

## 5. Numerical Results

### *5.1. Metrics and Methodology*

We use ILOG CPLEX 10.1.0 to implement the ILP on a computer with four Intel Xeon 2.40GHz CPUs and 4G of memory. The ILP scheme provides the optimal solution for static traffic demands. The simulations of SPPP and SFPP are implemented on a computer with a Intel 3.0GHz CPU and 1.5G of memory. We measure the performance of our schemes from following metrics:

- *the protection redundancy:* defined as the ratio of protection capacity to working capacity.

- *the reject ratio:* defined as the ratio of the number of requests rejected to the number of requests received.
- *the number of wavelength channels:* one wavelength channel is defined as a wavelength on some link. For example, A 3-hop path uses 3 wavelength channels.
- *the number of XC:* this metric denotes the number of optical cross connects that need to be reconfigured upon a double-link failure.

When studying the performance of SPPP and SFPP, we first consider the incremental traffic, that is, a demand never terminates once it is satisfied. In this traffic model, the capacity of the network link is set to infinity. Then we study the performance of these two schemes in dynamic traffic model.

In both traffic models, 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. For dynamic traffic, 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 $\lambda/\mu$.

### *5.2. ILP Results for Static Traffic*

A small test network with 6 nodes and 11 edges (shown in Fig. 6) is used to test our ILP scheme. Table 2 shows the protection redundancy and the running time of ILP for different number of connections. Each data point is the average of ten test cases.

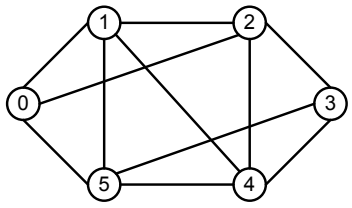


Figure 6: The 6-node 11-edge network.

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. We also use the SFPP and SPPP under the same scenarios and

Table 2: Redundancy and Computation time of ILP and heuristic algorithms

| Number of connections: | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| ILP Run Times (s) | 0.034 | 0.91 | 59.8 | 1304 | 11684.2 |
| Protection Redundancy(ILP): | 592% | 448% | 373% | 306% | 302% |
| Protection Redundancy(SPPP): | 618% | 546% | 544% | 534% | 515% |
| Protection Redundancy(SFPP): | 600% | 561% | 535% | 498% | 502% |

calculate their redundancy performance. As expected, and as shown in Table 2, SFPP and SPPP are not as efficient as the ILP under static traffic because they deal with connection requests one by one and without considering the future incoming connection requests.

When ILP is used to find the optimal protection strategy for static demands, there will be sufficient time between the planning and provisioning processes even the ILP has long run-time. Moreover, ILP can provide the baseline to evaluate the performance of heuristic algorithms.

### *5.3. Comparison of SPPP and SFPP*

We conduct simulations to compare the performance of SPPP and SFPP under incremental traffic and dynamic traffic. Three networks, the SMALLNET network , the COST239 network(Fig. 7) and the DISTRIBUTED network(Fig. 8)[18], which consists of 47 nodes and 98 links, are used in the simulations.

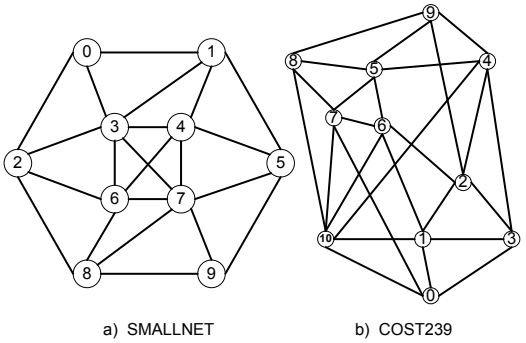


Figure 7: SMALLNET and COST239 Networks.

In the first set of simulations, we consider incremental traffic. The total number of wavelength channels used by all the working paths and by all the $p$-cycles are recorded for each simulation run.

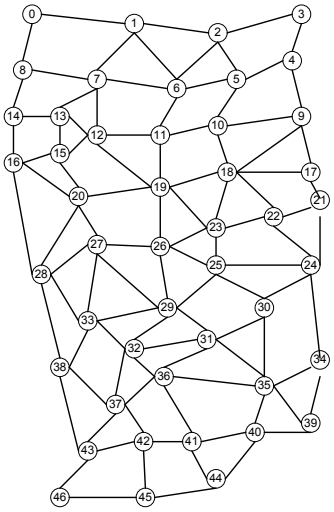


Figure 8: Distributed Network.

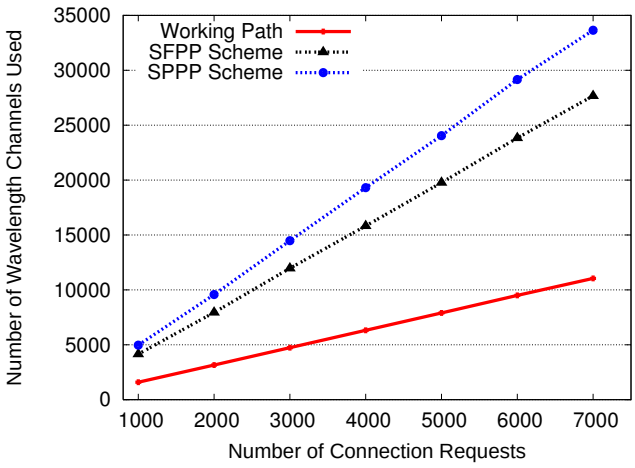


Figure 9: Wavelength usage of SPPP and SFPP in SMALLNET.

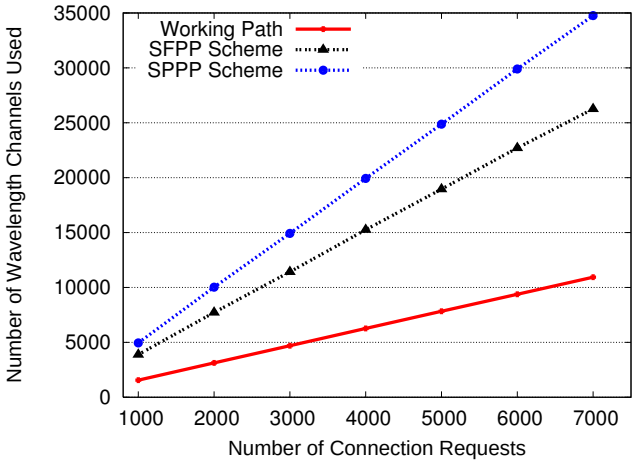


Figure 10: Wavelength usage of SPPP and SFPP in COST239.

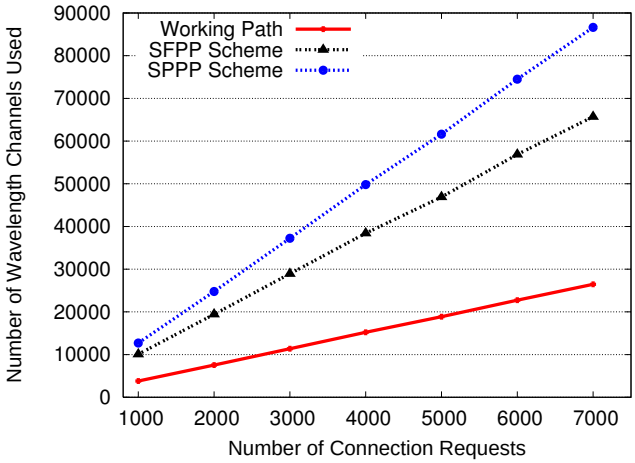


Figure 11: Wavelength usage of SPPP and SFPP in DISTRIBUTED.

In Fig. 9, 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. 10, 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.

In Fig. 11, we show the performance of SPPP and SFPP in DISTRIBUTED network. Again, SFPP uses less wavelength channels for protection than SPPP under all traffic loads. Specifically, SFPP achieves a 20.6%-24.2% reduction in wavelength usage over SPPP. We can also observe that the number of wavelength channels used is much higher than that used in SMALLNET and COST239 because DISTRIBUTED is larger. With the increase of wavelength channels used by the working path, the wavlength channels reserved by these p-Cycles will also increase.

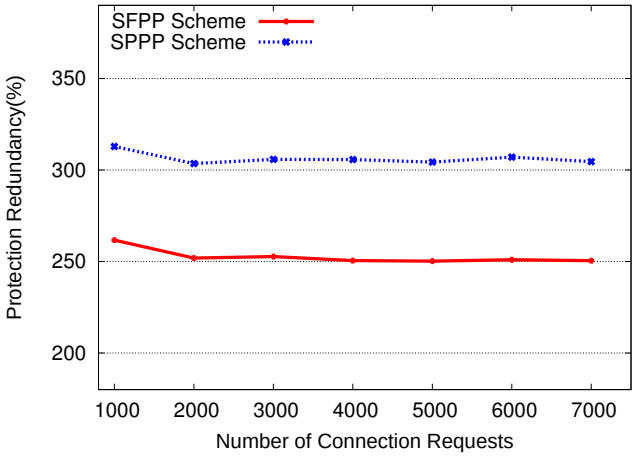


Figure 12: Protection redundancy of SPPP and SFPP in SMALLNET.

Fig. 12, Fig. 13, and Fig. 14 compare the protection redundancy of SPPP and SFPP for SMALLNET, COST239 and DISTRIBUTED, respectively. These figures show that the protection redundancy of SPPP and SFPP drop slightly as

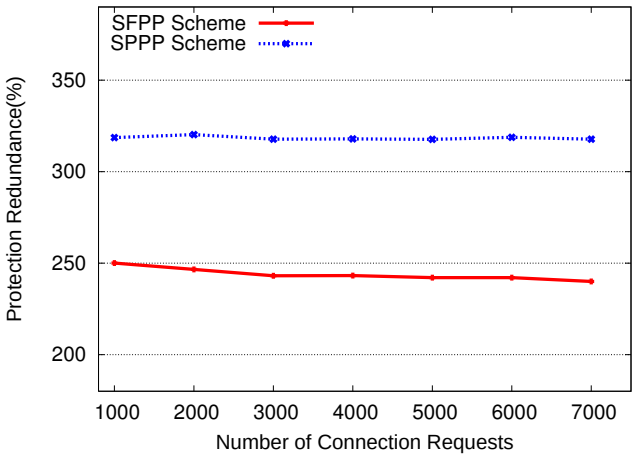


Figure 13: Protection redundancy of SPPP and SFPP in COST239.

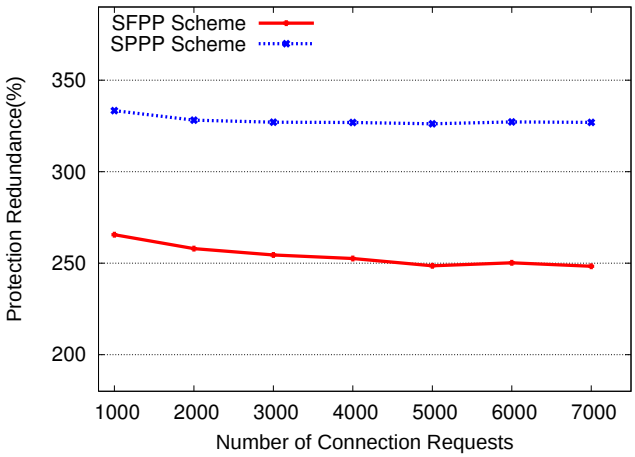


Figure 14: Protection redundancy of SPPP and SFPP in DISTRIBUTED.

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; For DISTRIBUTED, SFPP achieves 23% -24% reduction in redundancy over SPPP.

As shown in Table 3, we also study the efficiency of p-Cycles constructed by SFPP and SPPP under incremental traffic in three different networks. The p-Cycle efficiency of p-Cycle $pc_i$ is defined as the ratio of working wavelength channels protected by $pc_i$ over the wavelength channels reserved on p-Cycle $pc_i$. SFPP has better efficiency performance than SPPP because SFPP tends to use longer $p$-cycles that can protect more straddling links. Thus, each $p$-cycle has higher capacity efficiency in average and this also explains why SFPP has lower redundancy.

Table 3: Comparison of p-Cycle Efficiency

| Networks | SMALLNET | COST239 | DISTRIBUTED |
|---|---|---|---|
| SFPP Efficiency | 0.75 | 0.78 | 0.71 |
| SPPP Efficiency | 0.65 | 0.63 | 0.6 |

In the second set of simulations, we consider dynamic traffic. Each simulation has 10 rounds and 5000 randomly generated connection requests are loaded to the network in each round and the average reject ratio is recorded. The capacity of the network link is set to 10 wavelengths.

In Fig 15, we compare the reject ratio of SFPP and SPPP under different traffic loads (in erlangs) in SMALLNET network. The results show that SFPP performs worse than SPPP when traffic load is low. 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. We also observe that the long $p$-cycles can be fully utilized and they can provide more efficient protection than those $p$-cycles created by SPPP when the traffic load becomes high. According to our simulation, SFPP performs better than SPPP when traffic load is above 32 erlangs. However, the reject ratio is high and it is not practical.

In Fig 16, we compare the reject ratio of SFPP and SPPP under different traffic loads in COST239 network. Again, the results show that SFPP performs worse than SPPP under low traffic loads. Similarly, SFPP will perform better than SPPP when the traffic load is above 30 erlangs.

In Fig 17, we compare the reject ratio of SFPP and SPPP under different traffic loads in DISTRIBUTED network. Again, the results show that SPPP performs much better than SFPP under low traffic loads. Which also shows that our SPPP has good scalability performance.

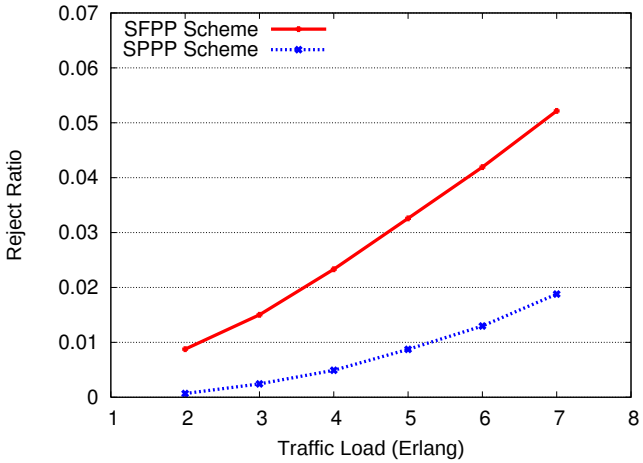


Figure 15: Reject ratio of SPPP and SFPP in SMALLNET.

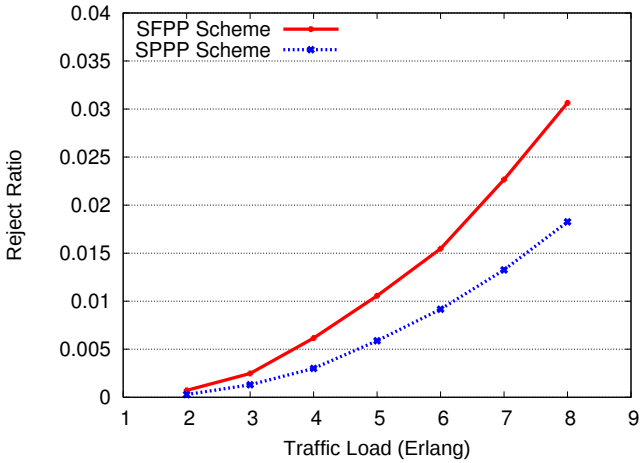


Figure 16: Reject ratio of SPPP and SFPP in COST239.

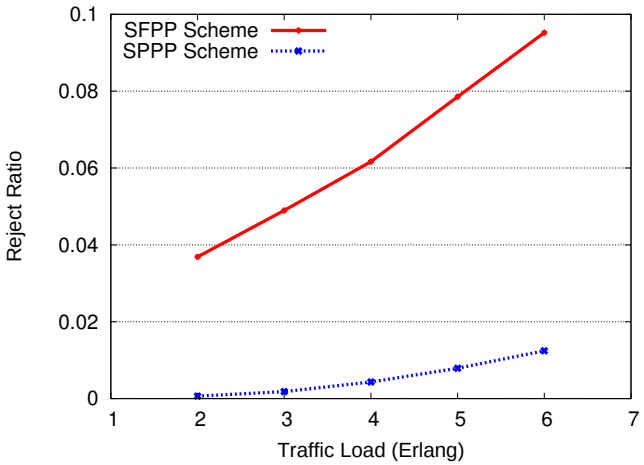


Figure 17: Reject ratio of SPPP and SFPP in DISTRIBUTED.

We also observe that the reject ratio is 0 in all of these three networks when there is no protection provided.

In Table 4, we compare the average computation time for a single demand using SPPP and SFPP in two test networks. As the table shows, it only takes SPPP 1.55 milliseconds on average to compute the $p$-Cycles for a demand in SMALLNET. Meanwhile, SPPP runs faster than SFPP in both networks which corresponds well with previous time complexity analysis. SFPP just needs 2.45 milliseconds to compute the $p$-Cycle in COST239. Thus, the proposed schemes are suitable for dynamic demands. Both algorithms need more run time in DISTRIBUTED, which is expected because the DISTRIBUTED network has much more edges and nodes.

Table 4: Average Computation Time(milliseconds): 10 rounds, 5000 demands in each round

| Networks | SMALLNET | COST239 | DISTRIBUTED |
|---|---|---|---|
| SPPP | 1.55 | 1.9 | 39.86 |
| SFPP | 2.1 | 2.45 | 69.56 |

### *5.4. Comparison of SPPP and the Algorithms in [9]*

We compare SPPP with the three approaches–Method I, Method II, and MADPA–proposed in [9] as shown in Table 5. 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

Table 5: Comparison of Algorithms in ARPANET

| Algorithm | I | II | MADPA | SPPP |
|---|---|---|---|---|
| Protection ratio | 100% | 100% | 98.8% | 100% |
| Protection redundancy | 200% | 200% | 200% | 259% |
| $XC_{max}$ with signaling | 26 | 18 | N/A | 6 |
| $XC_{avg}$ with signaling | 9.34 | 8.64 | N/A | 4.4 |
| $XC_{max}$ w/o signaling | N/A | N/A | 24 | 4 |
| $XC_{avg}$ w/o signaling | N/A | N/A | 7.3 | 4 |

capacity to the total working capacity. $XC_{max}$ and $XC_{avg}$ denote the worst-case and average number of optical cross connects that need to be configured upon a double-link failure. In [9], the authors assume the working capacity reserved on each link is one unit such that the schemes proposed in the paper require fixed protection capacity on each link, which equals 200%.

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 5 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. Basi-

cally, 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.

## 6. Conclusions

In this paper, 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. The ILP can provide the optimal solution for static demands and provide a baseline for evaluating the performance of heuristic algorithms. Realizing that ILP is not suitable for online provisioning process, 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. Under dynamic traffic, SPPP has lower blocking than SFPP in most practical cases. In addition to the time complexity analysis, we run simulations to show that the average computation time for a single demand is at the milisecond-level. The time complexity analysis also shows the good scalability of our proposed SFPP and SPPP schemes. Compared with the algorithms proposed in [9], SPPP trades off protection redundancy for fast restoration speed. In the future, we plan to develop heuristic algorithms to solve the static traffic provisioning problem and design more efficient algorithms for dynamic traffic in terms of running time and protection redundancy. We will also consider multiple-link failure protection.

## Acknowledgments

This work is supported by NSF ANI-0237592 and CNS-0626741.

## References

[1] R. Yadav, R. S. Yadav, H. M. Singh, A review of survivable transport networks based on p-cycles, in: International Journal of Computer Sciences and Engineering Systems, Vol. 1, 2007.

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

[3] W. Grover, D. Stamatelakis, Cycle-oriented distributed preconfiguration: Ring-like speed with mesh-like capacity for self-planning network restoration, in: Proc. IEEE ICC'98, 1998, pp. pages 537–543.

[4] D. Schupke, C. Gruber, A. Autenrieth, Optimal configuration of p-cycles in wdm networks, in: Proc. of IEEE ICC, 2002.

[5] B. Wu, K. L. Yeung, S. Xu, Ilp formulation for p-cycle construction based on flow conservation, in: proceedings of the IEEE GLOBECOM, 2007.

[6] W. Grover, J. Doucette, Advances in optical network design with p-cycles: Joint optimization and preselection of candidate p-cycles, in: Proc. of IEE LEOS Summer Topical Meeting, 2002.

[7] W. D. Grover, A. Kodian, Failure-independent path protection with p-cycles: Efficient, fast and simple protection for transparent optical networks, in: proceedings of the ICTON, 2005, pp. 363–369.

[8] A. E. Kamal, 1+n network protection for mesh networks: Network coding-based protection using p-cycles, in: IEEE/ACM Transactions on Networking, Vol. 18, 2010, pp. 67–80.

[9] H. Choi, S. Subramaniam, H.-A. Choi, Loopback recovery from double-link failures in optical mesh networks, in: IEEE/ACM TRANSACTIONS ON NETWORKING, Vol. 12, 2004, pp. 1119–1130.

[10] D. Schupke, Multiple failure survivability in wdm networks with p-cycles, in: Proceedings of the International Symposium on Circuits and Systems, 2003.

[11] W. He, M. Sridharan, A. K. Somani, Capacity optimization for surviving double-link failures in mesh-restorable optical networks, Photonic Network Communications 9 (1) (2005) 99–111.

[12] A. Kodian, W. D. Grover, Multiple-quality of protection classes including dual-failure survivable services in p-cycle networks, in: proceedings of the Broadnets, 2005, pp. 231–240.

[13] H. Wang, H. T. Mouftah, P-cycles in multi-failure network survivability, in: Proceedings of International Conference on Transparent Optical Networks, 2005.

[14] A. E. Kamal, 1+n protection against multiple faults in mesh networks, in: proceedings of the IEEE International Conference on Communications (ICC), 2007.

[15] R. Bhandari, Survivable networks, algorithms for diverse routing, Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[16] T. Feng, L. Ruan, W. Zhang, Intelligent p-cycle protection for multicast sessions in wdm networks, in: proceedings of the IEEE International Conference on Communications (ICC), 2008.

[17] R. Asthana, Y. Singh, Second phase reconfiguration of restored path for removal of loop back in p-cycle protection, in: Communications Letters, IEEE, Vol. 11, 2007, pp. 201–203.

[18] P. Baran, On distributed communications networks, in: IEEE Trans. Commun.,, Vol. COM-12, 1964, pp. 1–9.