SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2012; **5**:689–708
Published online 23 August 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.364

RESEARCH ARTICLE

Self-protecting networking using dynamic *p*-cycle construction within link capacity constraint

Weiyi Zhang[1], Farah Kandah[1], Xiaojiang Du[2]* and Chonggang Wang[3]

[1] Department of Computer Science, North Dakota State University, Fargo, ND 58105, U.S.A.
[2] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, U.S.A.
[3] NEC Laboratories America, Princeton, NJ 08540, U.S.A.

ABSTRACT

The *p*-cycle design problem has been extensively studied because it can provide both ring-like fast self-protection speed and spare capacity efficiency of path protection scheme. However, *p*-cycle provisioning for dynamic traffic has not been fully addressed. Most related works have not considered link capacity in the construction of *p*-cycles, which may cause problems in practice because the protection paths may not have enough backup bandwidth. In this paper, with the consideration of link capacity, we present a *sufficient and necessary condition* that guarantees *p*-cycles for providing enough protection bandwidth. Based on this condition, we propose an effective solution to provide connections for dynamic requests with the property that each link used for a connection is protected by a *p*-cycle. Simulation results show that our dynamic *p*-cycle provisioning solution outperforms the traditional path protection scheme. Copyright © 2011 John Wiley & Sons, Ltd.

KEYWORDS
self-protecting networking; network protection and restoration; survivable path provisioning; *p*-cycle; link capacity

***Correspondence**
Xiaojiang Du, Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, U.S.A.
E-mail: dux@temple.edu

1. INTRODUCTION

The method of *preconfigured protection cycles*, namely, *p*-cycles, was proposed by Grover and Stamatelakis [1]. Intuitively, a *p*-cycle is a simple cycle in the network, which can be used to protect both on-cycle links and straddling links. A *p*-cycle can achieve ring-like fast protection speed and mesh-like high efficiency of spare capacity. This is because a *p*-cycle can provide protection not only for on-cycle spans but also for straddling spans [1–5].

Most previous works studied the *p*-cycle design problem, which is how to construct a set of *p*-cycles for a given traffic matrix so that the spare capacity in the *p*-cycles is minimized. Integer linear programming (ILP) has been widely used for solving the problem [6–10]. However, ILP is very computationally intensive when the network size is large. To reduce the computation complexity, many works proposed heuristics to solve the problem [1,11–14]. The *p*-cycle concept has found extensive applications in wavelength-division multiplexing (WDM) in recent works [3,8,15–18]. It offers a promising approach for protection and restoration in WDM networks because high capacity efficiency and fast protection switching times can be achieved [10,14,19–24].

Although the *p*-cycle design problem has been extensively studied, several issues have not been fully solved:

- Most related works did not consider the bandwidth issue in the construction of *p*-cycles for the dynamic traffic. Without the consideration of link capacity, the constructed *p*-cycles may not be practical because links of the *p*-cycle may not have enough bandwidth to provide protection. To deal with such a problem, some related work reserves half of the link capacity for each link on a *p*-cycle for protection purpose [6]. However, it is not difficult to see that link capacities are not used efficiently or effectively in the scheme.
- The *p*-cycle concept is proposed to protect all links that are either *on* or *straddling* the cycle. However, a *p*-cycle is usually generated by a link *l* and its protection path [17,25]. Although link *l* is guaranteed to be protected by the *p*-cycle, other links, which are either *on* or *straddling* the cycle, may not be protected by this cycle because the protection bandwidth for them was not considered when the *p*-cycle was constructed. As a result, the

p-cycle scheme reduces to the traditional link protection scheme and does not provide a good spare capacity utility.

In this paper, we study the *p*-cycle provisioning problem, with full consideration of the capacity of each link, for dynamic traffics. We present an effective solution to set up connections as well as to construct *p*-cycles for protection. Our approach is applicable to WDM systems, synchronous optical networking, and asynchronous transfer mode for recovery from single link failure. Moreover, each *p*-cycle constructed in this paper is useful for *all* the edges, which are either on the cycle or straddling the cycle. Note that in this work, the terms *edge* and *link* are interchangeable.

The rest of the paper is organized as follows: The network model, as well as the sufficient and necessary condition for constructing a *p*-cycle, is introduced in Section 2. We then formulate the *p*-cycle provisioning problem for dynamic traffics in this section. In Section 3, a solution to *p*-cycle provisioning problem is proposed, described, and analyzed. In Section 4, we present simulation results comparing our solution with the traditional dedicated path protection scheme. We conclude this paper in Section 5.

2. NETWORK MODEL

We model the network by a undirected two-connected graph $G=(V,E)$, where V is a set of n vertices and E is a set of m edges. We assume that the capacity of all edges is the same, denoted by C. The *p*-cycles constructed in this paper have a unique property; that is, each cycle constructed protects *all* edges on or straddling the cycle, not just some of the links on the cycle. Some definitions we would use in the rest of the paper are introduced here.

Definition 2.1. *The working load of a link i, denoted by w_i, is the amount of bandwidth already used. The free bandwidth of a link i, denoted by f_i, is the amount of bandwidth that is available. As a result, w_i+f_i equals he capacity of link i. The free bandwidth of a path is the minimum of the free bandwidths among all links on the path.*

Definition 2.2. (*feasible p-cycle*) *A feasible p-cycle p is a simple cycle in G with the following properties:*

(i) For each link $l=(u,v)$, which is on p (called *on-cycle* link), the working load of l is no greater than the free bandwidth of the u–v path on p without using l. In this case, link l is protected by the u–v path on p without using l.

(ii) For each link $l=(u,v)$, which is straddling p (l is not on p, but the two end nodes of l are on p), the working load of l is no greater than the summation of the free bandwidths of the two disjoint u–v paths on p. In this case, link l is protected by the two disjoint u–v paths on p. □

Given a feasible *p*-cycle, we say it is a *corresponding* cycle for all links, which are on or straddling it. Also, we say such links are *p*-cycle protectable or are protected by the *p*-cycle.

Definition 2.3. (*redundant free bandwidth*) *Let p be a feasible p-cycle and l be a link that is either on* or *straddling p. The redundant free bandwidth* of *link l with respect to the cycle p, denoted by $f^r_{l,p}$, is the maximum amount by which w_l can be increased while maintaining p as a feasible p-cycle. In this paper, when the p-cycle p is known, we use f^r_l to represent $f^r_{l,p}$ for simplicity.* □

We have used *feasible p-cycle*, which takes into account the capacity consideration, to distinguish from the commonly used *p-cycle*. In this paper, we are only interested in feasible *p*-cycles and will drop the word "feasible" in the rest of the paper without confusion.

We illustrate the give definitions using the network in Figure 1, which includes 8 nodes and 15 links.

C, the capacity of each link in the graph, is 20. The *free bandwidth* of each link is noted on the link. There is a *feasible p*-cycle 2-3-4-5-6-2 in the graph, marked by the thick red edges. We can see that each on-cycle link can be protected by the path composed of all other links on the cycle. For example, link (2,6), whose working load is 8, can be protected by path 2-3-4-5-6, whose free bandwidth is 10. Meanwhile, the cycle can provide protection for two straddling links, (3,5) and (2,5). For example, the working load of (2,5) is 18 and could be protected by path 2-6-5 and path 2-3-4-5. The two paths have free bandwidths 12 and 10, respectively, and totally provide at most 22 for protection.

In this *p*-cycle, the redundant free load of link (5,6) is 8. If we use more than 8 of its free bandwidth to accommodate future connections, (5,6)'s free bandwidth will be reduced to less than 10. It shall not be able to protect links (2,3) and (4,5) because both have working load of 10. Meanwhile, (5,6)'s working load will be more than 10 and cannot be protected by the path 5-4-3-2-6 whose free bandwidth is no more than 10.

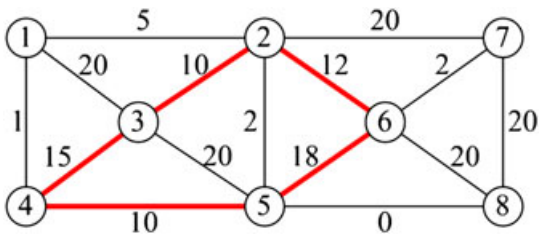


Figure 1. An example graph and free bandwidths of edges.

Security Comm. Networks 2012; **5**:689–708 © 2011 John Wiley & Sons, Ltd.
DOI: 10.1002/sec

Therefore, if we increase (5,6)'s working load by more than 8, cycle 2-3-4-5-6-2 will not be a feasible *p*-cycle any more.

2.1. Sufficient and necessary condition for feasible *p*-cycle

For *any* on-cycle link *l*, all other links on the *p*-cycle comprise a restoration path for *l*. And the free bandwidth of this restoration path must be no less than the working load of *l*. Therefore, to construct a *p*-cycle with the consideration of link capacity, the *necessary condition* is that for *any two* links, *i* and *j*, on a *feasible p*-cycle,

$$\begin{gathered} f_i \geq w_j \quad \text{and} \quad f_i \geq w_i \\ \Leftrightarrow f_i \geq C - f_i \quad \text{and} \quad f_j \geq C - f_i \\ \Leftrightarrow f_i + f_j \geq C (*) \end{gathered}$$

In addition, following this condition, it is able to guarantee that any straddling link can also be protected by its corresponding cycle. For each straddling link l whose working load is w_l, a *p*-cycle provides at most two protection paths, p_1 and p_2. Assume the links with the minimum free bandwidth on p_1 and p_2 are k and e, respectively. These two paths are able to provide f_k+f_e for protection. Because k and e are both on the cycle, we have $f_k + f_e \geq C$ (*). Therefore, p_1 and p_2 can provide protection for l because $w_l \leq C \leq f_k + f_e$.

Therefore, the *sufficient and necessary condition* for constructing a feasible *p*-cycle is that *for any two links on the cycle, the summation of their free bandwidths is no less than link capacity C*. Besides this, we also observe several interesting properties for *p*-cycles that are very important for our *p*-cycle protection design scheme:

(i) The redundant free load of a straddling link l is the same as its free bandwidth, that is, $f_l = f_l^r$.
(ii) Each on-cycle link i always has its redundant free load less than or equal to its free bandwidth, that is, $f_i^r \leq f_i$.
(iii) There is at most *one* link on a *p*-cycle whose free bandwidth is less than $C/2$.

2.2. Problem statement

Given a network with existing connections as well as corresponding *p*-cycles for the links, we consider a dynamic call-by-call system with random arrivals; that is, the requests for connection come dynamically one by one. For each new request, we know its source node *s*, destination node *d*, and the requested demand, denoted as *demd*. The objective is to find a path whose links are *p*-cycle protectable. *In other words, every link on the path is on or straddling a corresponding p-cycle with the consideration of two constraints*:

(1) Each edge is only protected by at most *one p*-cycle, which means that even an edge may appear in several *p*-cycles, but it is protected by only one of them.
(2) When accommodating the new requests, we do not modify or release the *p*-cycles for previous connections.

If we can find such a path, we accept this connection request and set up a *p*-cycle for each link on the path. Otherwise, we drop this connection request.

3. PROPOSED SOLUTION TO *P*-CYCLE PROVISIONING

In this section, we present a solution to set up a path for each coming new request as long as there exists a feasible path in the current network setup

```
1: Suppose i and j are the incident links of s and d which
   have the largest free loads, respectively;
2: if (f_i < demd) or (f_j < demd) then
3:     drop the request and return;
4: end if
5: Aux_Graph_Construction (G', G) ;
6: Find a shortest path P in G' to satisfy the connection
   request;
7: if (there is no such a path P) then
8:     drop the request and return;
9: end if
```

Algorithm 1. Connection accommodation (G, s, d, $demd$)

In Algorithm , we firstly check the link i (and j), which is incident with s (and d) and has the largest free bandwidth among all incident edges. If either i or j does not have enough free bandwidth for the demand, we drop the connection request because there is not a path between s and d with sufficient bandwidth. Otherwise, we start to construct an auxiliary graph G' composed of all *p*-cycle protectable edges and then find a shortest path in it. This path provides a connection for the coming request.

```
1: for (each link i in G) do
2:     if (i is in some existing p-cycle) then
3:         if (f_i^r ≥ demd) then
4:             add link i into G'; assign cost ε to i;
5:         end if
6:     else
7:         if (find_cycle_straddling_link(i, G)) then
8:             add link i into G'; assign cost 1 to i;
9:         else
10:            if (find_cycle_with_link(i, G)) then
11:                add link i into G'; assign cost m to i;
12:            end if
13:        end if
14:    end if
15: end for
```

Algorithm 2. Auxiliary graph construction (G', G)

Algorithm constructs an auxiliary graph G' composed of all *p*-cycle protectable links. If a link *i* is used for existing connection(s) and *has its redundant free bandwidth with respect to its corresponding p-cycle no less than demd*, then we add link *i* into G' with a small edge cost ϵ (lines 2–5). Note that with the very small value of ϵ, we prefer to reuse a link for the new connection request without constructing a new *p*-cycle for it. If link *i* has not been used, we need to find out whether there is a *p*-cycle that is able to protect *i* if *i* is used for the request. If there exists such a cycle, we add link *i* into G' (lines 6–10). If a link *i* is protected by a *p*-cycle, it must be either *on* the *p*-cycle or *straddling* it. Algorithms and check these two cases, respectively.

```
1:  if (f_i < demd) then
2:      return false;
3:  end if
4:  copy all nodes in G into a new graph G'';
5:  for (each link l in G) do
6:      if (f_l ≥ C/2) then
7:          add l into G'';
8:      end if
9:  end for
10: remove i if it is in G'';
11: if (∃ a biconnected component including end nodes of
    i in G'') then
12:     return true;
13: end if
14: for (each link l(≠ i) with f_l < C/2 in G) do
15:     remove all links in G'';
16:     add l into G'';
17:     for (each link j(≠ i)) do
18:         if (f_j ≥ C − f_l) then
19:             add j into G'';
20:         end if
21:     end for
22:     if (∃ a biconnected component including end nodes
        of
        i in G'') then
23:         return true;
24:     end if
25: end for
26: return false;
```

Algorithm 3. Find cycle straddling link(i, G)

As we mentioned, *the redundant free load of a straddling link is the same as its free bandwidth*, which means that all free bandwidths of the link are able to be used for accommodating future requests. By contrast, *on-cycle links have to reserve some free bandwidth for protecting other links on the p-cycle*. Therefore, it is preferable to use straddling links over on-cycle links for a connection. Algorithm seeks a *p*-cycle with straddling link *i*. If so, we add it into G' with cost 1. Otherwise, we seek a *p*-cycle with *i* on it in Algorithm and assign cost *m*, whose value is the number of the edges in the network, to link *i*.

3.1. Link *i* straddles some *p*-cycle

This case is considered in Algorithm 3. For link *i*, if $f_i <$ *demd*, it is not necessary to check it. Otherwise, there are two cases to be considered. In each case, we construct an auxiliary graph G'' and check whether a *p*-cycle straddled by *i* exists in G''.

3.1.1. All links on the *p*-cycle have free bandwidths ≥$C/2$.

This case is processed in lines 5–13 in Algorithm; the graph G'' is simply composed of all links (except *i*) whose free bandwidths are no less than $C/2$. Then we check biconnected components of G'' in linear time (lines 11–13) [26]. The found cycle must be a *p*-cycle because $f_l + f_e \geq C$ for any two links, *l* and *e*, on it. It takes $O(m+n)$ time to examine link *i* in this case.

3.1.2. All but one link on the *p*-cycle have free bandwidths ≥$C/2$.

From line 14 to line 25, we check each link *l* with $f_l < C/2$ to find if there is a *p*-cycle including link *l* as well as being straddled by *i*. For each link *l* with $f_l < C/2$, we construct an auxiliary graph G''. Each link in G'' other than *l* must have its free bandwidth $\geq C - f_l$ (link *i* cannot be included in G''). Then in G'', we check if there is a *p*-cycle being straddled by link *i* in linear time (line 22). The total time complexity for checking link *i* in this case is $O(m(m+n))$ because we need to repeat the mentioned operations for each link *l* whose $f_l < C/2$.

3.2. Link *i* is on some *p*-cycle

If link *i* does not *straddle* any *p*-cycle, it could be *on* a *p*-cycle. In Algorithm 4, we skip *i* if it does not have enough free bandwidth (lines 1–3). Otherwise, to find a *p*-cycle with link *i*, we set up an auxiliary graph G'' containing *i*. We still have two cases to be considered: (i) $f_i < C/2$ and (ii) $f_i \geq C/2$. Algorithm 4 checks both cases one by one.

3.2.1. f_i is less than $C/2$.

In lines 6–15, given $f_i < C/2$, for each link j ($\neq i$) on the same *p*-cycle with *i*, it must have

- $f_j > C - f_i > C/2$,
- $f_i^r \geq demd \Leftrightarrow f_j + f_i - C \geq demd \Leftrightarrow f_j \geq C + demd - f_i$

If link *i* is on some *p*-cycle, this *p*-cycle is composed of *i* and other links whose free bandwidths are no less than $C + demd - f_i$ ($> C/2$). We construct such a graph G'' and use depth first search (*DFS*) [26] starting from an end node of link *i* to find a cycle including *i* (line 12).

Both constructing G'' and finding a cycle with i in G'' by *DFS* only need linear time. Thus, for this case, only $O(m+n)$ time is needed to check link i.

3.2.2. f_i is no less than $C/2$.

If a link i with $f_i \geq C/2$ is on a p-cycle, this p-cycle must be in one of following two forms:

(1) All links have free bandwidths $\geq C/2$.
(2) One link l of this p-cycle has *free bandwidth* $< C/2$.

Form 1 is considered in lines 17–24 in Algorithm 4; the auxiliary graph G'' is composed of i and all other links whose *free bandwidths* are no less than $C/2$ and $C + demd - f_i$. In such a graph G'', we can decide whether there is a p-cycle including i by using *DFS* [26] in linear time.

From line 25 to line 39, for form 2, we check every possible graph G'' that comprises link i, a link e with $C + demd - f_i \leq f_e < C/2$, and other links with free bandwidths $\geq C - f_e (> C/2)$.

If there exists such a link e with $C + demd - f_i \leq f_e < C/2$, in linear time, we can construct G'' and try to find a p-cycle with link i and e on it. In the worst case, we will check $O(m)$ such as link e to decide whether link i is p-cycle protectable. The total time complexity for this case is $O(m(m+n))$. Overall, if link i is *on* some p-cycle, in $O(m(m+n))$ time, we can find such a p-cycle.

```
1: if (f_i < demd) then
2:     return false;
3: end if
4: copy all nodes in G into a new graph G'';
5: add link i into G'';
6: if (f_i < C/2) then
7:     for (each link l in G) do
8:         if (f_l ≥ C + demd − f_i) then
9:             add l into G'';
10:        end if
11:    end for
12:    if (∃ a cycle including i in G'') then
13:        return true;
14:    end if
15: end if
16: remove all links in G''; add i into G'';
17: for (each link l in G) do
18:    if (f_l ≥ C + demd − f_i) AND (f_l ≥ C/2) then
19:        add l into G'';
20:    end if
21: end for
22: if (∃ a cycle including i in G'') then
23:    return true;
24: end if
25: for (each link e in G) do
26:    remove all links in G'';
27:    add i into G'';
28:    if (f_e ≥ C + demd − f_i) AND (f_e < C/2) then
29:        add e into G'';
30:        for (each link l in G) do
31:            if (f_l ≥ C − f_e) then
32:                add l into G'';
33:            end if
34:        end for
35:        if (∃ a cycle including i in G'') then
36:            return true;
37:        end if
38:    end if
39: end for
40: return false;
```

Algorithm 4. Find cycle with link(i, G)

3.3. Correctness of the approach

In this section, we show the correctness of our solution by demonstrating that all accommodated connections can survive any single link failure.

Theorem 3.1. *The connections that are accommodated by Algorithm are able to survive any single link failure.* □

Proof. Initially, the network is empty; obviously, it is able to survive any single link failure. Suppose that it is still true for the first H accommodated requests.

Suppose that we have accommodated the $(H+1)$th connection by path P. Now we need to prove that network can still survive any single link failure. Assume that the broken link is e and its corresponding p-cycle is q. First, we focus on the case that e is an *on-cycle* link. Two types of single link failure cases could happen:

Case 1: *Link e is NOT on the chosen path P*. The working load on link e is still $C - f_e$ because e is not used for the new request. For each link l, which is *on* the same p-cycle with e,

(i) If l is not on the chosen path P, its free bandwidth is still f_l. $f_e + f_l \geq C$ because they both are *on* a same p-cycle. $f_l \geq C - f_e$ indicates that such a link l is able to protect the working load of e even after the new path P is set up.

(ii) If l is on path P, then its free bandwidth is reduced to $(f_l - demd)$ after accommodating the new request. Because link l is on q with e, we have $f_l + f_e \geq C + demd$, which means $f_l - demd \geq C - f_e$. So such link l on path P has enough free bandwidth to protect link e.

Thus, link e can be protected by all other links on q.

Case 2: *Link e is ALSO on the chosen path P*.

For the links that are not *on* the cycle q, we do not need to consider them because they are not used to protect e. So we only focus on the links that are *on* the cycle q. We use *dependent links* of e to represent such links in the proof. We use j to represent any dependent link of e, which is on P, and l to represent a dependent link, which is not on

P. Link e has its original free bandwidth f_e reduced to $f_e - demd$ after being used for the request. Similarly, link j also has its free bandwidth f_j decreased to $f_j - demd$. For link l, which is on the p-cycle but not on P, its free bandwidth is still f_l. Because of $f_l + f_e \geq C + demd$ and $f_e \leq C$, we have $f_l \geq demd$.

Because P contains several links that are on the same p-cycle q, with e, let us assume path P enters q at node u, goes through all nodes on both P and q, and leaves the cycle from node v. Then path P is divided into three parts: s to u, u to v, and v to d. When link e fails, only the part from u to v represented as $\{u \ldots v\}$ is affected because link e is in this part. On the p-cycle q, there are two paths from u to v. One of them, say p_r, includes link e. And the other path, say p_r', does not. After e fails, we can use p_r' to replace $\{u \ldots v\}$. Then the backup path P', which is composed of s to u, p_r', and v to d, is used for the recovery of path P. For recovery of the link e, we need to show two things:

(1) p_r' is able to accommodate the demand of P.
(2) After the recovery, the links on q still have enough free bandwidths to recover the original working load on e.

For each link on p_r', if it were on $\{u \ldots v\}$, we say it is link j. After the recovery, its free bandwidth is still $f_j - demd$, which is not less than $C - f_e$ because $f_e + f_j \geq C + demd$. This implies that it can accommodate path P as well as provide protection for the original working load, $C - f_e$, on link e. On the other hand, if the link was not on $\{u \ldots v\}$, we say it is link l. As shown before, we have $f_l \geq demd$, which means that l can accommodate path P. Moreover, we know $f_e - demd \geq C - f_l$ because e and l both are on q. Therefore, we have $f_l \geq C - f_e + demd$ and then know that l can also provide protection to the original working load on link e.

If e is a *straddling* link, following the same analysis, we can also prove the theorem. So all $H+1$ accommodated connections can survive single link failure. Therefore, we can conclude that all connections provided by our solution can survive any single link failure. ■

We use Figure 2 to illustrate our proof. For simplicity, the figure only shows part of the whole network. Each curved dashed link represents a path between two end nodes, which may go through several nodes and links in the network but is not drawn in the figure. The link capacity of the network is also 20. Each edge's current free bandwidth is noted on the link. The demand of the new request is 5.

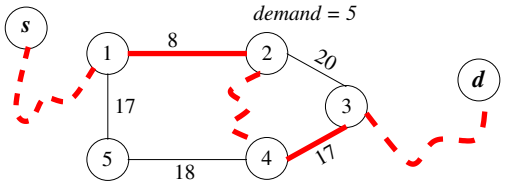


Figure 2. Illustration of the proof of Theorem 3.1.

In Figure 2, each link on the p-cycle *-2-3-4-5-1* has enough redundant free bandwidth for the request with demand 5.

Suppose that the found path P is $s \cdots 1\text{-}2 \cdots 4\text{-}3 \cdots d$, marked by the thick red edges. Two links on P, (1,2) and (4,3), are dependent links. When we set up this connection, the free bandwidth of link (1,2) will reduce to 3, and the free bandwidth of link (4,3) will decrease to 12.

Let us see an example for *case 1*. In Figure 2, if link (1,5), which is not on P, failed, its working load 3 needs to be recovered. Links (2,3) and (4,5), which are not on path P, obviously can be used for the protection. Links on P, (1,2) and (3,4), whose free bandwidth values are at least 3, can also protect the link (1,5) even after both have been used for P.

Now let us see an example for *case 2*. If link (4,3) is broken, this link failure is recovered by the following steps: Nodes 1 and 3 represent the nodes u and v in our proof, respectively. Path P is divided into *s···1, 1-2···4-3*, and *3···d*. Because link (3,4) is broken, segment *1-2···4-3* fails. We replace it with another segment *1-2-3* for the recovery of path P (with demand 5). Obviously, links (1,2) and (2,3) on *1-2-3* both can provide the protection. Besides the demand 5 of path P, the original working load 3 on (3,4) also needs protection. (1,2) and (2,3) have free bandwidth 3 and 15, respectively, and they both can protect this working load. Other links on the cycle are not used on path P and can protect (3,4)'s original working load. Therefore, (3,4) can still be protected by the p-cycle, *1-2-3-4-5-1*. It is worth noting that the protection approach in the proof extends the p-cycle concept to path segment protection, which is also presented in Ref. [27].

4. NUMERICAL EVALUATION

In this section, we implemented our solution (denoted by PC in the figures) and compared it with the traditional path protection scheme [28–30] (denoted by PP in the figures) because, to our best knowledge, this work is the first one to study dynamic p-cycle setup with link capacity consideration. It is worth noting that the goal of path protection is to protect a path from failure at any point along its routed path. Under path protection, the path of the protection path is completely disjoint from the path of the working path. The advantage of path protection is that the backup path protects the working path from all possible link and node failures along the path. Because the path selection is end-to-end, *path protection may be more efficient in terms of resource usage than link protection* [28,29]. Because we do not allow backup bandwidth sharing in our solution, dedicated path protection [30] was implemented. All tests were performed on a 2·4-GHz Linux PC with 2GB of memory. As in Ref. [31], we used random network topologies generated in Ref. [32] to study the suitability and computational time complexity of the algorithms. The edge

capacity is 20. A connection request is generated at each time unit with random source node and destination node and has a life time that is set to a random integer uniformly distributed in [1,100]. The request demands are randomly distributed between [1,20] per request. Our numerical results are presented in Figures 3-20, where each figure shows the average of 10 runs.

To measure the performance of our solution and compare it with the traditional path protection scheme, we propose our first scenario, which consists of a fixed number of nodes with a different number of edges. Figure 3 presents the satisfied ratio of the PC and the PP schemes by deploying 200 nodes with a different number of edges, ranging from 1600 to 2000 edges in the same playing field. We tested this network configuration with 2000, 3000, and 4000 requests. These results are shown in Figure 3(a, b, and c, respectively).

Our PC scheme, in all the three cases mentioned before, performed better than the traditional path protection scheme. For example, with 200 nodes with 1600 edges, by deploying our PC scheme, we can satisfy 1412 requests out of 2000 requests compared with 776 satisfied requests out of 2000 requests by the use of PP scheme. This can be seen in Figure 3(a).

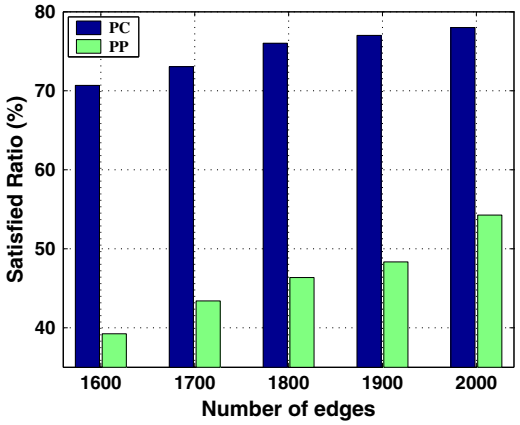
The satisfied ratio of 4000 requests is shown in Figure 3 (c). Still, it is obvious that our PC scheme can satisfy more requests compared with the PP scheme. For example, with 2000 edges, we can satisfy 32% more requests by deploying our PC scheme.

By increasing the number of edges in the network, we increase the possibility that the requests can be handled on different paths. For example, for the same number of nodes (200 nodes) with 1700 edges, by the use of our PC scheme, we can satisfy 60% compared with 68% when increasing the number of edges to 2000 edges. This can be seen in Figure 3(b), where it shows the satisfied ratio of 3000 requests.

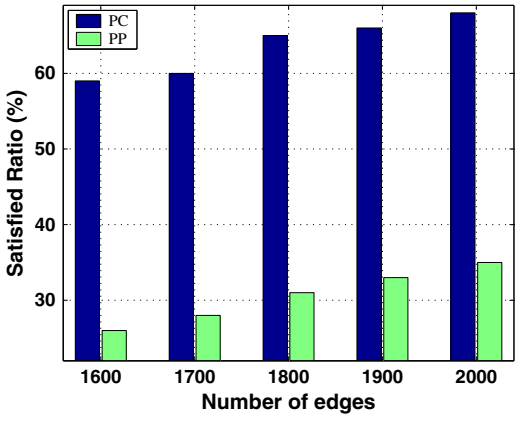
There are two main reasons for better performance of *p*-cycle solution. First, in the *p*-cycle scheme, one *p*-cycle can potentially protect multiple connections. Second, *p*-cycle scheme always prefers to use straddling links. Straddling links do not need to reserve any spare bandwidth on the existing *p*-cycles, whereas each backup path has to reserve spare bandwidth in path protection scheme. Moreover, each straddling link can be protected by at most two paths, but path protection, by contrast, generally allows only one backup path for each working path. These properties bring more flexibilities to the *p*-cycle protection, save bandwidths for future request, and consequently improve the connection satisfaction ratio.

To evaluate the performance of our PC and the PC scheme with different numbers of nodes, we studied their performance with 300, 400, and 500 nodes in Figures 4,5, and 6, respectively.

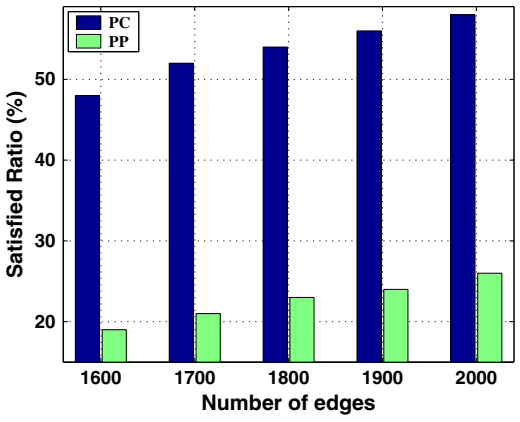
Figure 4 presents the performance evaluation of both tested schemes with their different numbers of requests. Figure 4(a) shows the satisfied ratio of the PC and the PP schemes with 300 nodes, with an edge range from 1600 to 2000 edges. In this scenario, we allow 2000 requests to come at random. The results show that by using our PC, we can satisfy more number of requests compared with that of the PP scheme. For example, in case of 2000 edges, we can satisfy 604 more requests when we use our PC scheme.



(a) 200 nodes - 2000 request

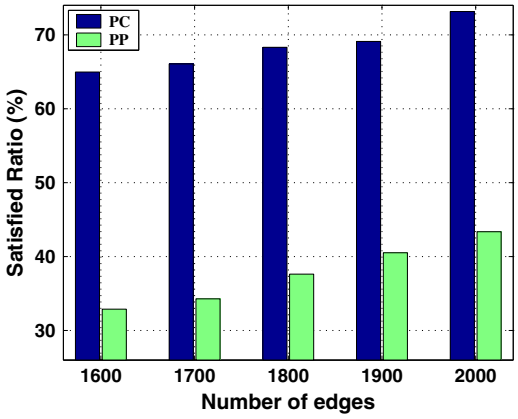


(b) 200 nodes - 3000 request

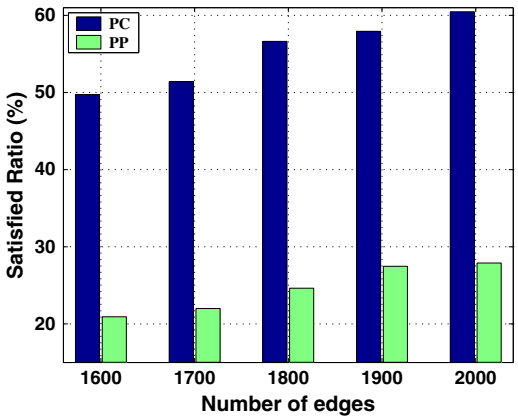


(c) 200 nodes - 4000 request

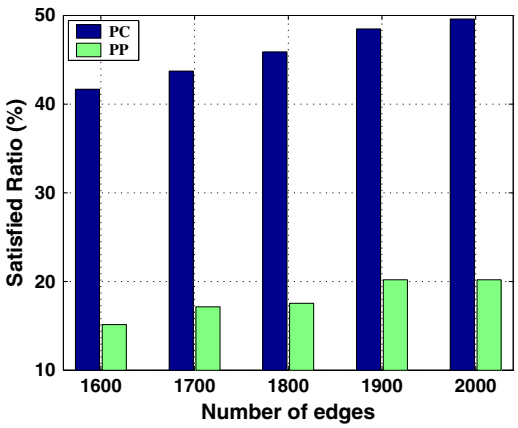
Figure 3. Performance comparisons between *p*-cycle protection and path protection—200 nodes without timeout.



(a) 300 nodes - 2000 request



(b) 300 nodes - 3000 request



(c) 300 nodes - 4000 request

Figure 4. Performance comparisons between *p*-cycle protection and path protection—300 nodes without timeout.

It can be seen from the results that our PC scheme outperforms the PP scheme in satisfying more number of requests, where in Figure 4(b), we try to satisfy 3000 requests, whereas in Figure 4(c), we try to satisfy 4000 requests. It can be seen in Figure 4 that by increasing the number of edges, we can satisfy more number of requests. For example, with 1600 edges, we can satisfy 1667 requests out of 4000 requests by the use of our PC scheme, but when we increase the number of edges to 2000 edges, we can satisfy 1984 requests.

We continue our performance testing through increasing the number of nodes to 400 nodes. These results can be seen in Figure 5. In here, we study the performance of both tested scheme with 400 nodes with the number of edges ranging from 1600 to 2000 edges to satisfy 2000, 3000, and 4000 requests. The results lead us to the same observations from the previous scenarios, where our PC scheme outperforms the PC scheme in all cases, and also, by increasing the number of edges, we can satisfy more number of requests. The results in Figure 5(a, b, and c) show the satisfied ratio to satisfy 2000, 3000, and 4000 requests, respectively.

Increasing the number of nodes in a network will provide more connectivity, through being able to have more connections between multiple node, and that would increase the network ability to handle more user's requests. But having the number of edges fixed, this will distribute the edges of more number of nodes, and that will degrade the number of satisfied requests because of the decrease of the number of paths from source to destinations. For example, using PC scheme with 400 nodes and 1600 edges, we can satisfy 8·55% less requests compared with having 300 nodes with the same number of edges. This can be seen in Figures 3(a) and 5(a).

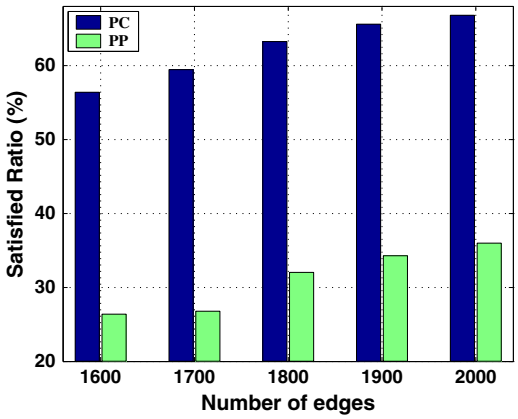
The results of increasing the number of nodes to 500 nodes and the measure of the satisfied ratio to satisfy 2000, 3000, and 4000 requests can be seen in Figure 6(a, b, and c). It is obvious that, fixing the number of nodes while increasing the number of edges, our PC scheme outperforms the traditional PP scheme. For example, using our PP scheme with the availability of 1800 edges with 500 nodes, we can satisfy 48·17% of the 3000 coming requests, whereas by the use of the PP scheme, we can satisfy 20·53% of the 3000 coming requests.

Satisfied ratio can be affected by increasing the number of nodes. To study the effect on the performance of our PC scheme and the PP scheme in satisfying coming requests, we deployed different numbers of nodes ranging from 100 to 500 nodes. Each network topology has a number of edges that equals three times the number of nodes. These results can be seen in Figure 7. Note that the letter *n* in the following figures indicated the number of nodes.

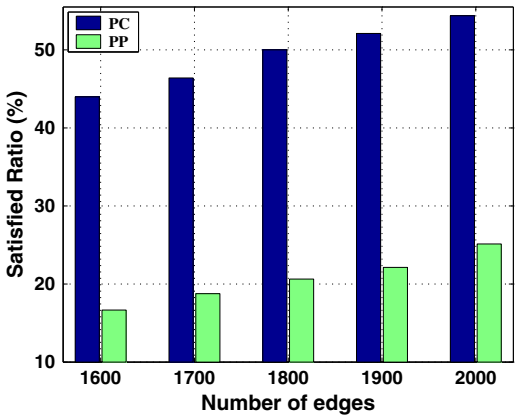
We tested this scenario with three different numbers of coming requests (2000, 3000, and 4000). This can be seen in Figure 7(a, b, and c, respectively). For the case of 2000 requests shown in Figure 7(a), we can see that increasing the density of the network will lead us to increase the satisfied ratio for both our PC and the PP schemes. For example, with 400 nodes and 1200 edges, we can satisfy 13·7% more requests compared with that with 200 nodes and 600 edges. It is still obvious that our PC scheme outperforms the PP scheme in satisfying more number of requests.

Increasing the number of nodes and the number of edges will allow the network to handle more number of user's requests. For example, with 3000 requests in 200

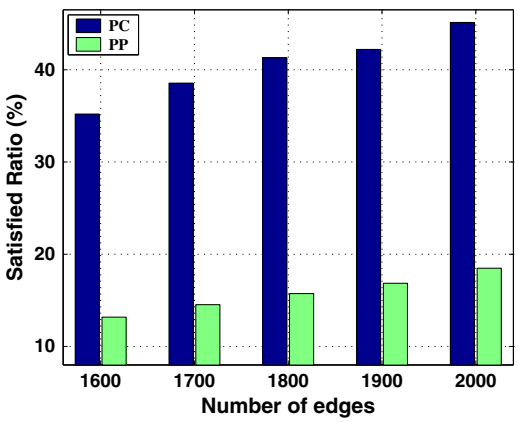
Security Comm. Networks 2012; **5**:689–708 © 2011 John Wiley & Sons, Ltd.
DOI: 10.1002/sec



(a) 400 nodes - 2000 request

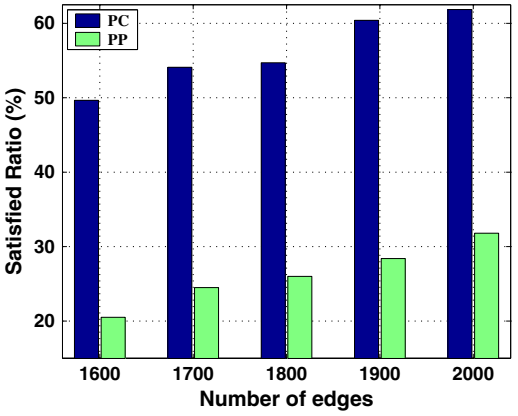


(b) 400 nodes - 3000 request

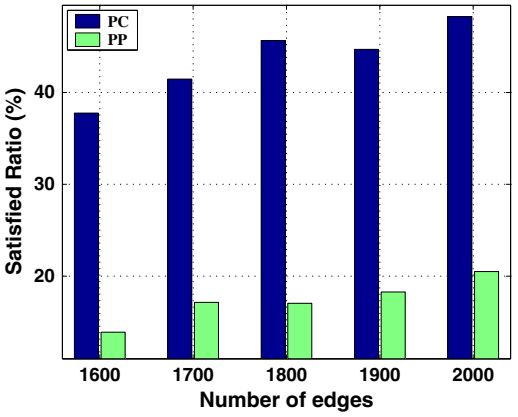


(c) 400 nodes - 4000 request

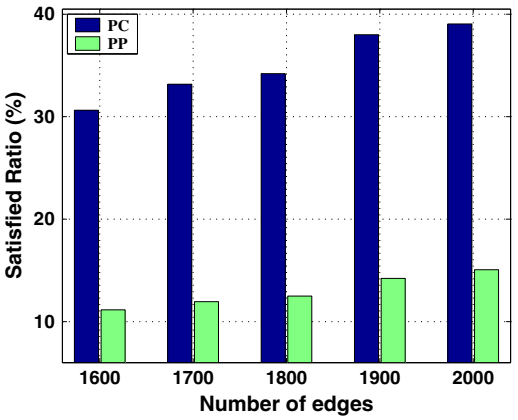
Figure 5. Performance comparisons between *p*-cycle protection and path protection—400 nodes without timeout.



(a) 500 nodes - 2000 request



(b) 500 nodes - 3000 request

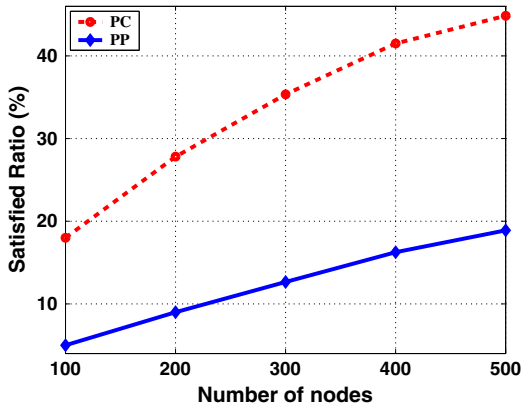


(c) 500 nodes - 4000 request

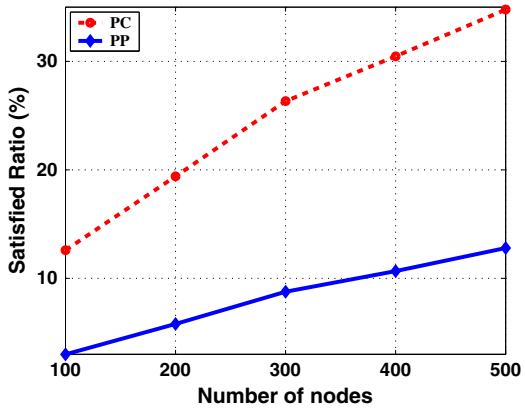
Figure 6. Performance comparisons between *p*-cycle protection and path protection—500 nodes without timeout.

nodes and 600 edges, we can satisfy 834 requests out of the 3000 coming requests with the use of our PC scheme, whereas by the use of the traditional PP, we can satisfy 270 requests out of the 3000 requests. By making the network denser, by increasing the number of nodes to 500 nodes, for example, we can satisfy 210 more number of requests by deploying our PC scheme. These results can be seen in Figure 7(b).

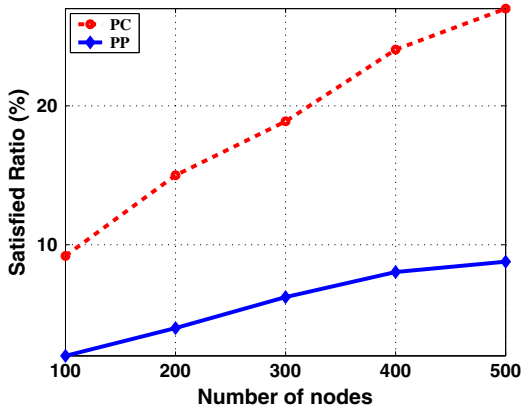
Figure 7(c) shows the satisfied ratio of both schemes for handling 4000 coming requests, with different numbers of nodes ranging from 100 to 500 nodes, whereas the number of edges equals to three times the number of nodes. It can



(a) 2000 requests, edges = $3n$



(b) 3000 requests, edges = $3n$



(c) 4000 requests, edges = $3n$

Figure 7. Performance comparisons, edges=$3n$—no timeout.

be seen that our PC scheme can satisfy more number of requests compared with the traditional PP scheme.

We study the effect when increasing the number of edges to be four or five times the number of nodes on how our PC scheme and the traditional PP scheme can handle the coming requests. The results can be seen in Figures 8 and 9, respectively.

In Figure 8, we let the number of edges to be four times the number of nodes, and we tested how both schemes can perform in satisfying 2000, 3000, and 4000 requests, which can be seen Figure 8(a, b, and c, respectively). The results show that our PC scheme still outperforms the traditional PC scheme in satisfying more number of coming requests. For example, in Figure 8(b), with 300 nodes, we satisfied 60·47% of the 3000 coming requests with our PC scheme compared with 27·9% when we use the traditional PP scheme.

Increasing the number of edges to be five times the number of node will provide us with better performance where in Figure 9(a), for example, with 300 nodes, we can satisfy 12·35% more requests from the 2000 coming requests compared with that when we had less number of edges (four times the number of nodes). It is still noticeable that our PC scheme out performs the PP scheme, where in the case of 400 nodes with 4000 coming requests, we can satisfy 1562 requests with the use of our scheme, where we can satisfy 581 requests by using the traditional path protection scheme. This can be seen in Figure 9(c).

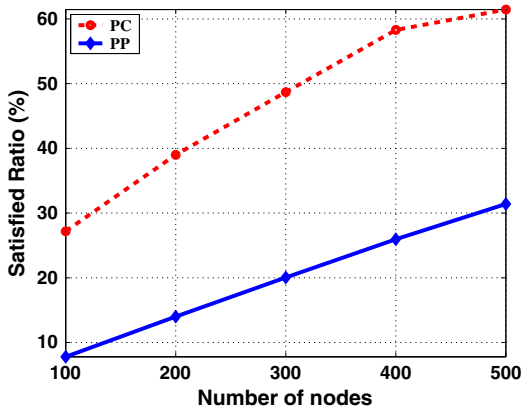
We study the effect of changing the number of edges in the network to be equal to $n \ln(n)$ in Figure 10, where n indicates the number of nodes in the network. Dashed lines show the satisfied ratio of both tested schemes with 2000 coming requests by having a number of nodes ranging from 100 to 500 nodes, and the asterisk marker indicates the PC, whereas the circle marker indicates the PP scheme. It can be seen that by increasing the number of nodes, we increase the number of edges, and that leads to increase the number of satisfied requests because of increased number of available paths. The results show that our PC scheme still out performs the traditional PP scheme.

The same observations can be seen when we allow 3000 requests to come to the network and measure how the schemes will perform by satisfying these coming requests; these results is shown by the solid lines in Figure 10. It can be seen that our scheme outperforms the traditional PP scheme. For example, in a 400-node network, we can satisfy 1861 requests with our PC scheme; on the other hand, we can satisfy 945 requests with the PP scheme. The dotted lines show the case with 4000 requests where it can be seen that our PC scheme can satisfy more requests compared with the traditional PP scheme, where with a 300-node network, we can satisfy 26·65% from the 4000 coming requests by the use of our PC scheme than the use of PP scheme.

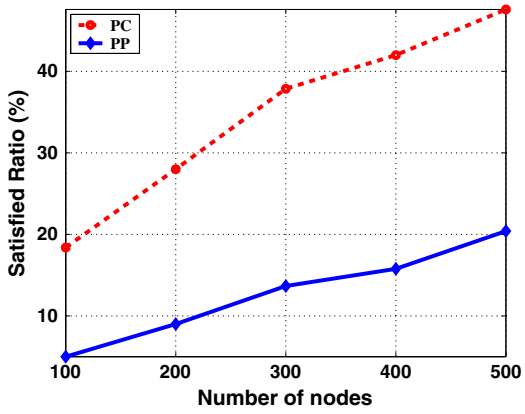
To make this study more flexible, we studied the case where all the coming requests have a life time; in other words, each request, after being satisfied, will accommodate the path until it is timed out. The life time of the request in the following discussion will be denoted as request timeout.

In this study, a connection request is generated at each time unit with random source node and destination node, and it has a life time that is set to a random integer uniformly distributed in [1,100]. The request demands are uniform but are randomly distributed in [1,20].

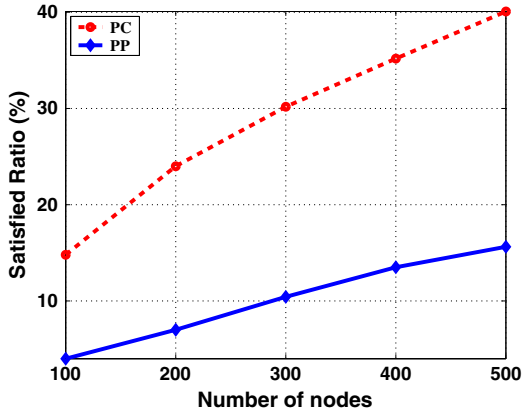
First, we will discuss the effect of having a fixed number of nodes while changing the number of edge and see



(a) 2000 requests, edges = $4n$

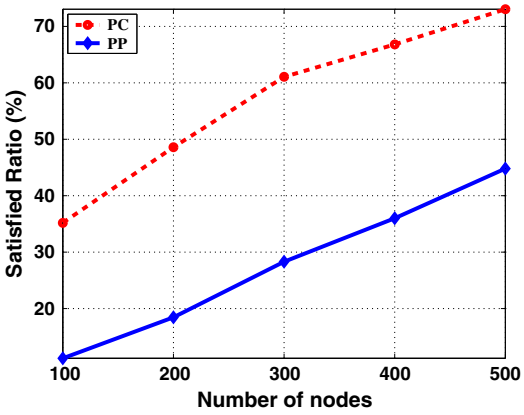


(b) 3000 requests, edges = $4n$

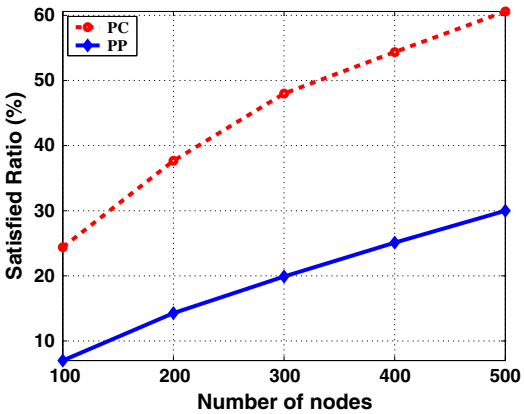


(c) 4000 requests, edges = $4n$

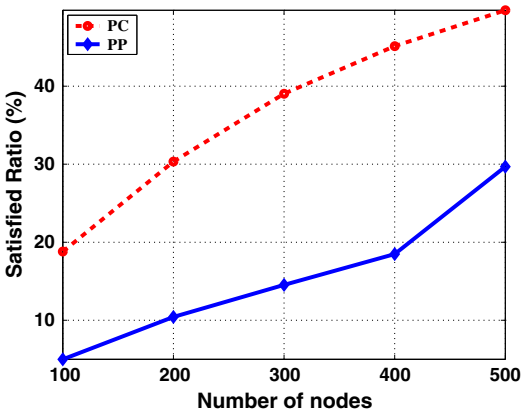
Figure 8. Performance comparisons, edges = $4n$—no timeout.



(a) 2000 requests, edges = $5n$



(b) 3000 requests, edges = $5n$



(c) 4000 requests, edges = $5n$

Figure 9. Performance comparisons, edges = $5n$—no timeout.

how the PC and PP schemes perform through satisfying the coming requests.

Our first case is shown in Figure 11, where we allow the network to have 200 nodes and allow the number of edges to range from 1600 to 2000 edges. Figure 11(a) shows the satisfied ratio of both schemes for satisfying 2000 coming requests. The results show that by providing the timeout concept for the coming requests, it is still obvious that our PC scheme outperforms the traditional PP scheme.

Increasing the number of requests allows to satisfy more number of requests because of the possibility that the source and the destination might have more paths between them. Figure 11(b and c) shows the same observations as before, but when we allow the network to handle 3000 and 4000 requests, respectively.

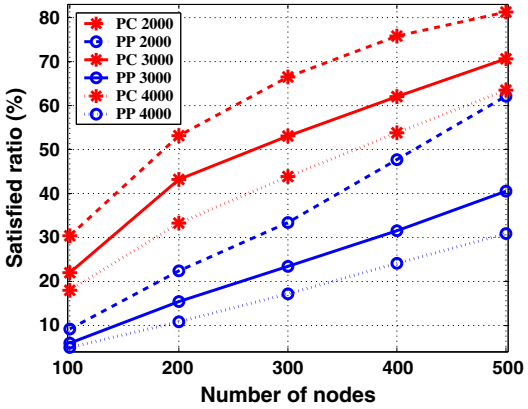


Figure 10. Performance comparisons, edges = n ln(n) — no timeout.

It can be seen that our PC scheme performs better than the PP scheme in all the cases, through providing a higher satisfied ratio. For example, in Figure 11(b) with 1800 edges, we can satisfy 41·54% more requests when we apply our PC scheme compared with that of the PP scheme.

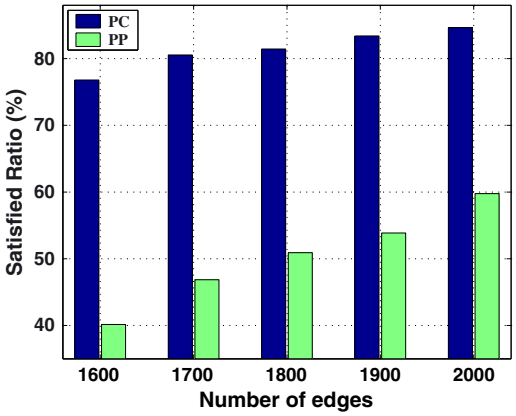
In Figure 12, we measure the satisfied ratio that can be provided with both tested schemes in a 300-node network with the number of edges ranging from 1600 to 2000 edges. By allowing each request to have a life time, we can satisfy more number of requests. After the request is timed out, it will release the path, and its previous workload will be added back to the path free bandwidth, and that will allow the network to handle more number of requests.

The same observations can be seen in Figure 12(a); having 2000 requests, we can satisfy more requests by applying our PC scheme compared with that when we apply the traditional PP scheme. For example, with 1900 edges, we can satisfy 83·4% out of the 2000 coming requests with our scheme, whereas with the PP scheme, we can satisfy 53·9% out of the 2000 coming requests.

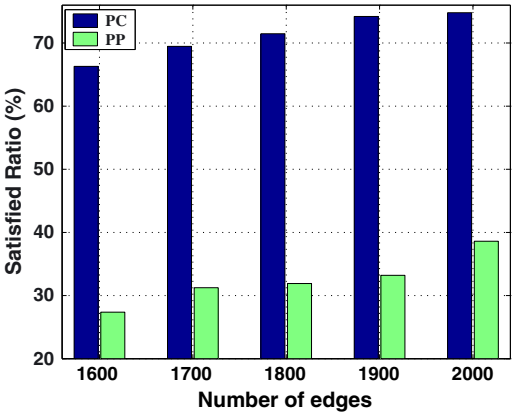
In Figure 12(b), we show the satisfied ratio when we apply our PC scheme and compared it with the traditional PP scheme with 3000 coming requests. The results show that by applying our PC scheme, we can satisfy more requests than that when we apply the traditional path protection scheme. Also, in Figure 12(c), we present the performance of both tested schemes with 4000 randomly coming requests with the timeout concept. It is obvious that our scheme still performs better than the traditional path protection scheme.

In our study, we realize that by applying our scheme in a larger network, we still can accommodate more number of requests compared with that when we apply the traditional path protection scheme. The results for 400 and 500 nodes can be seen in Figures 13 and 14, respectively.

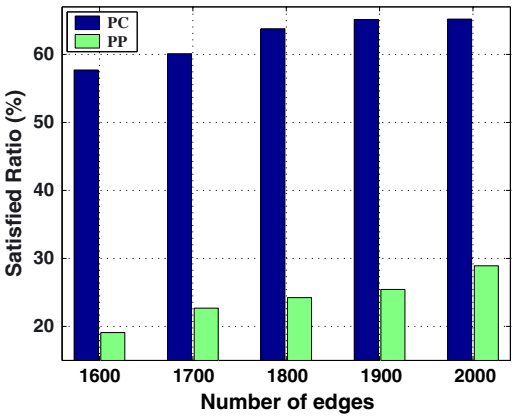
In Figure 13(a), we can see that with our PC scheme and 1800 edges, we can satisfy 35·2% more requests out of the 2000 coming requests compared with that when we use the PP scheme. Also, it can be seen that increasing the number of edges allow the network to handle more number of requests;



(a) 200 nodes - 2000 request



(b) 200 nodes - 3000 request

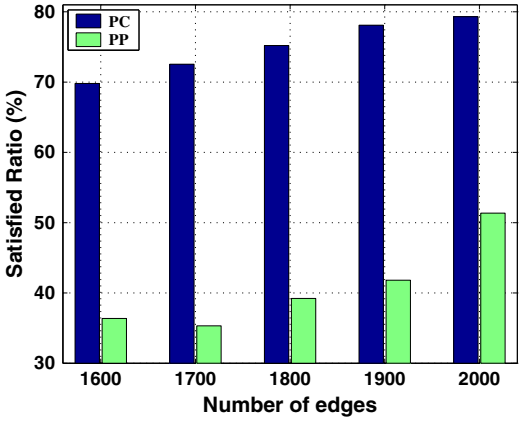


(c) 200 nodes - 4000 request

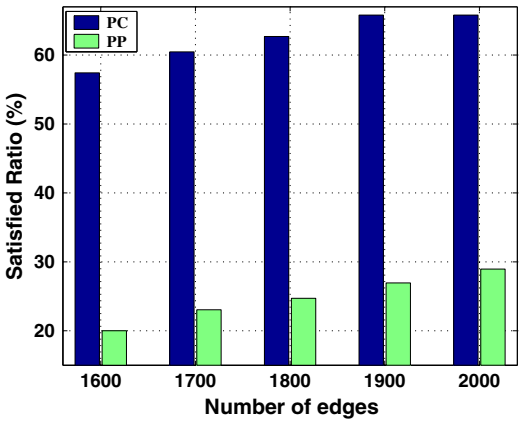
Figure 11. Performance comparisons between *p*-cycle protection and path protection—200 nodes with timeout.

for example, in Figure 14(b), with 1700 edges, we can satisfy 47·97% of the 3000 requests with our PC scheme, whereas by increasing the number of edges to 300 edges more, we can increase the satisfied ratio by 8·3%.

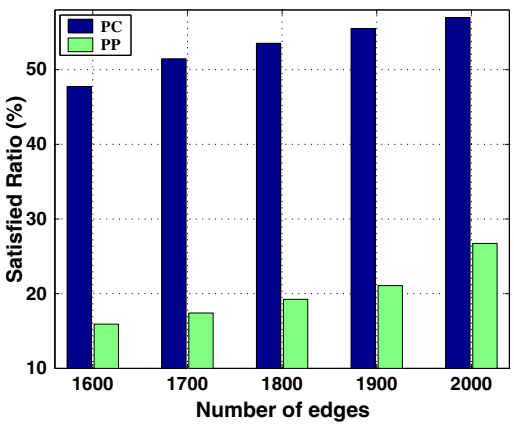
Changing the density (*the number of nodes in one square unit*) can affect the satisfied ratio in both the PC



(a) 300 nodes - 2000 request

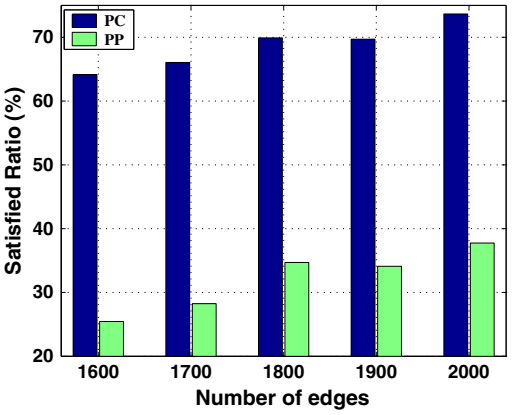


(b) 300 nodes - 3000 request

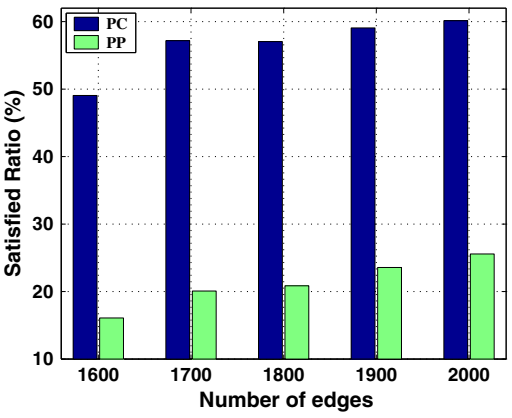


(c) 300 nodes - 4000 request

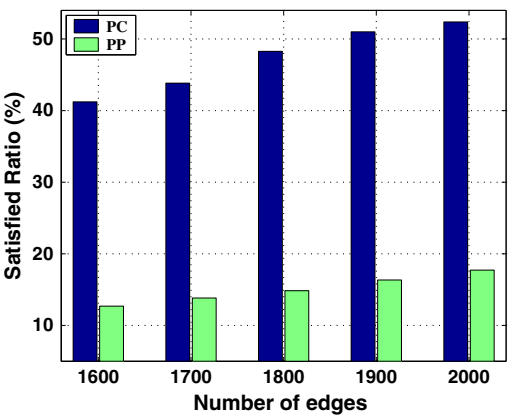
Figure 12. Performance comparisons between *p*-cycle protection and path protection—300 nodes with timeout.



(a) 400 nodes - 2000 request



(b) 400 nodes - 3000 request

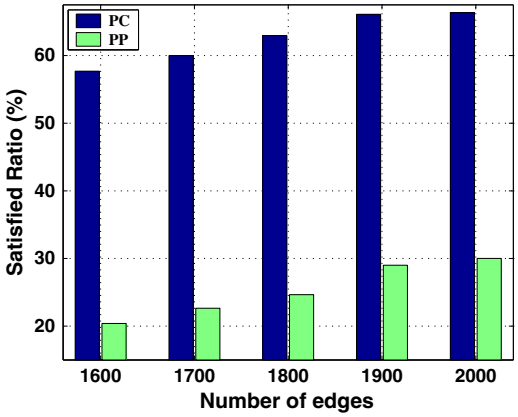


(c) 400 nodes - 4000 request

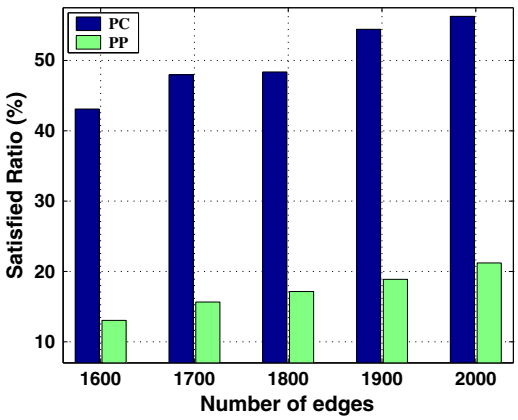
Figure 13. Performance comparisons between *p*-cycle protection and path protection—400 nodes with timeout.

and PP schemes. We studied this effect with different numbers of networks where the number of nodes ranges from 100 to 500 nodes, whereas allowing the coming requests to have a life time.

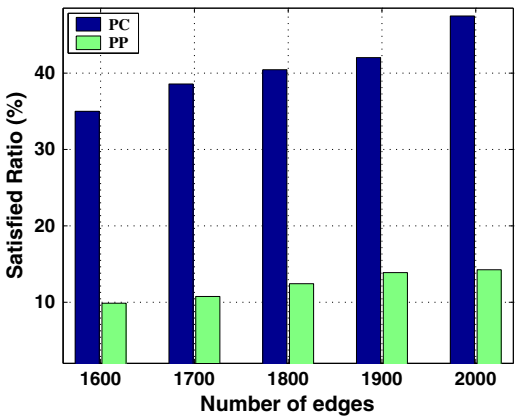
The results in Figure 15 show the satisfied ratio of both schemes in different networks with different numbers of coming requests, whereas the number of edges in each network equals to three times the number of nodes. In Figure 15(a), we present the satisfied ratio of different networks for handling 2000 requests. The results show that our PC scheme performs better than the traditional PP scheme; for example, with 300 nodes, the PC scheme



(a) 500 nodes - 2000 request

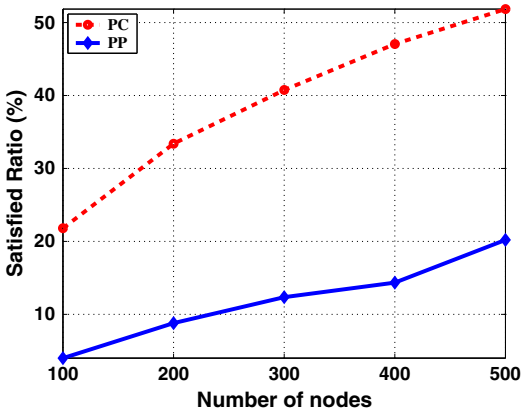


(b) 500 nodes - 3000 request

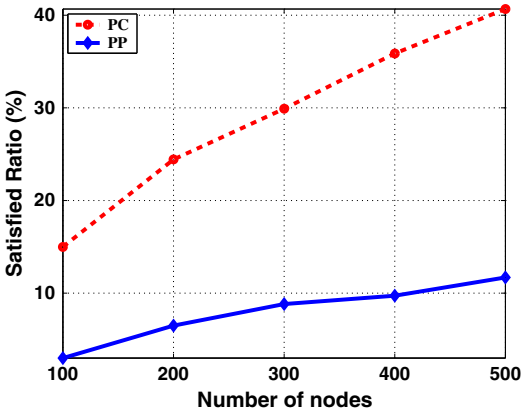


(c) 500 nodes - 4000 request

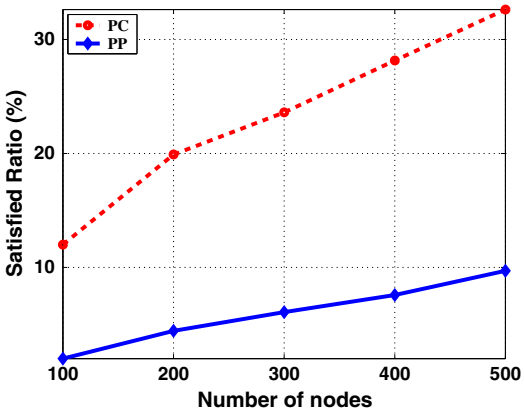
Figure 14. Performance comparisons between *p*-cycle protection and path protection—500 nodes with timeout.



(a) 2000 requests, edges = $3n$



(b) 3000 requests, edges = $3n$



(c) 4000 requests, edges = $3n$

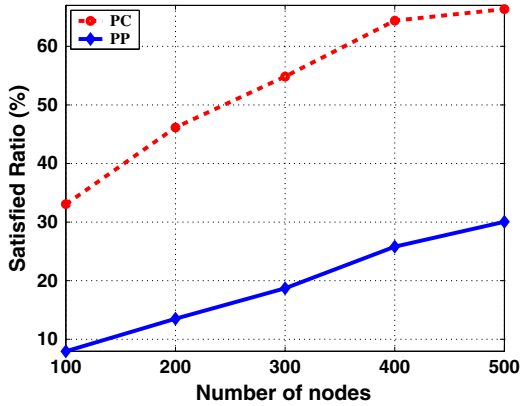
Figure 15. Performance comparisons, edges=$3n$—with timeout.

satisfied 568 more requests compared with that when we use the traditional path protection scheme.

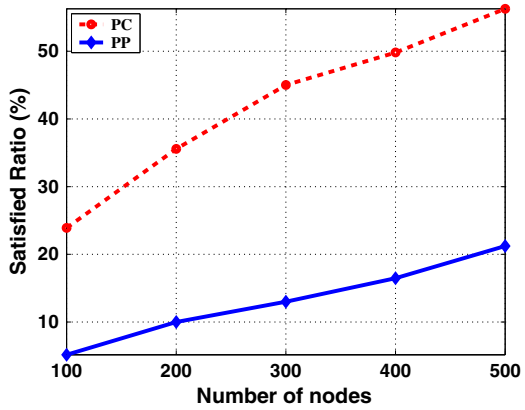
We tested both schemes to handle 3000 and 4000 requests with setting the number of edges to be equal to three times the number of nodes. and each request has a life time, where it can release the path after it timed out. The results in Figure 15(b and c) show that it is still noticeable that our scheme outperforms the PP scheme.

The case of having a request life time and more dense networks is shown in Figure 16, where for each network, we set the number of edges to be four times the number of nodes. We tested this configuration and measure its performance to handle 2000, 3000, and 4000 requests. Figure 16(a) shows the results for 2000 requests. It can

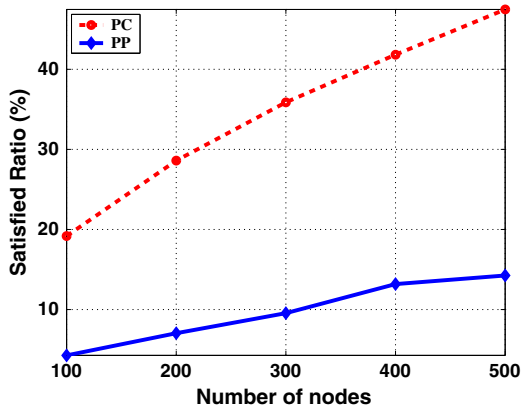
Security Comm. Networks 2012; **5**:689–708 © 2011 John Wiley & Sons, Ltd.
DOI: 10.1002/sec



(a) 2000 requests, edges = $4n$



(b) 3000 requests, edges = $4n$



(c) 4000 requests, edges = $4n$

Figure 16. Performance comparisons, edges=$4n$—with timeout.

be seen that by increasing the number of edges in the network, we can satisfy more number of requests using our PC scheme compared with the usage of the PP scheme. From the figure, it can be seen that with 500 nodes and 2000 edges, for example, we can satisfy 33·3% more requests out of the 2000 coming requests by the use of our PC scheme compared with the protection path scheme.

Our scheme in these configurations can still handle more number of requests, where in Figure 16(b), we allow 3000 requests to come to the network, and it is obvious that our PC scheme performs better than the traditional PP scheme. The results for allowing 4000 requests to come to the network are shown in Figure 16(c). For example, with 400 nodes, our scheme can satisfy 41·83% out of the 4000 requests, whereas the PP scheme can satisfy 13·18% out of the 4000 requests.

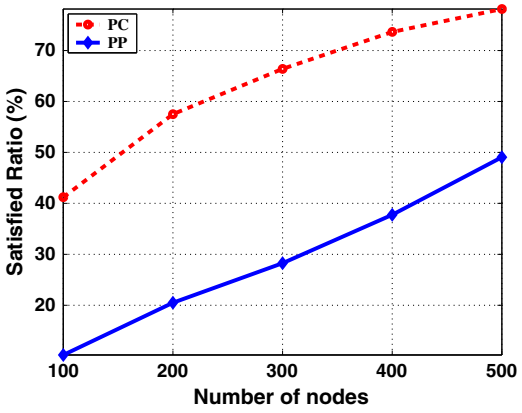
Also, it can be seen from the previous figures that by increasing the density of the network, we can satisfy more number of requests when using the same scheme, for example, in Figure 16(b) with 2000 nodes and 800 edges, when we applied our PC scheme, we can satisfy 732 requests; on the other hand, when we increase the number of nodes to 300 nodes and 1200 edges, we can satisfy 900 requests with using the same scheme.

We got the same observations when we increased the number of edges to be five times the number of nodes. These results can be seen in Figure 17. In Figure 17(a), with 2000 requests in a 200-node network, our PC scheme performs better than the PP scheme. While increasing the number of requests, we can still satisfy more coming requests by the use of our PC scheme. Figure 17(b) show the results when we allow the network to handle 3000 requests. It is obvious that our scheme still performs better than the traditional path protection scheme. With 400 nodes, our scheme can handle 1804 requests, whereas the PP scheme can handle 765 requests.

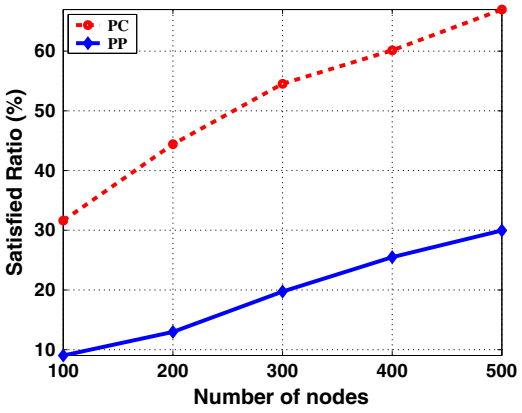
By adding more user's request to the network, our PC scheme still can satisfy more requests compared with the other scheme. In Figure 16(c), we present the satisfied ratio of both schemes when we increase the number of coming requests to 4000 requests. Increasing the density of the network will allow to increase the satisfied ratio for both the tested scheme, but still, our PC scheme outperforms the traditional PP scheme. For example, in Figure 17(b), with 200 nodes, by our PC scheme, we can satisfy 1332 requests, whereas by increasing the network density to 400 nodes with 2000 edges, we can satisfy 472 requests more.

In Figure 18, we change the number of edges in each network to be $n \ln(n)$, where n is referred to the number of nodes in the network. We study the performance of our PC scheme and compared it with the traditional PP scheme in different networks where we set the number of nodes to range from 100 to 500 nodes. We tested this configuration with 2000, 3000, and 4000 requests. We denoted our PC scheme with the asterisk marker and the PP scheme with the circle marker on the lines in Figure 18. The results with 2000 requests are shown using the dashed lines. With this configuration, our PC scheme can satisfy more requests compared with that when we use the PP scheme. For example, a 200-node network with our PC scheme can satisfy 786 more requests compared with that with the use of PP scheme.

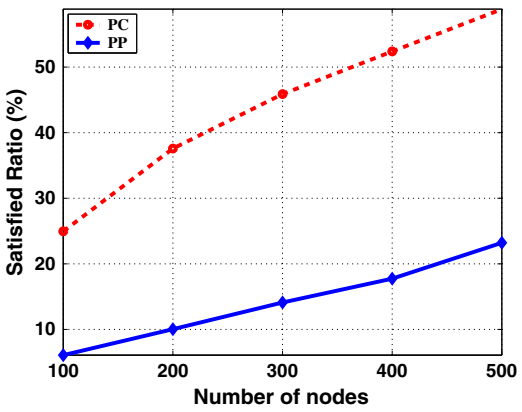
Moreover, by applying our PC scheme, we can satisfy more requests even if we increased the number of coming request. The results with 3000 and 4000 requests are shown by the solid and the dotted lines, respectively.



(a) 2000 requests, edges = $5n$



(b) 3000 requests, edges = $5n$



(c) 4000 requests, edges = $5n$

Figure 17. Performance comparisons, edges = $5n$—with timeout.

Increasing the number of the coming requests to the network has an effect on the gap between the satisfied ratio of the PC scheme and the PP scheme, where, for example, with 500 nodes and 2000 requests, our PC scheme can satisfy 18·6% more requests out of the 2000 requests, whereas this gap has been increased to 34·6% out of the 3000 requests; moreover, it had increased to 35·7% with 4000 coming requests.

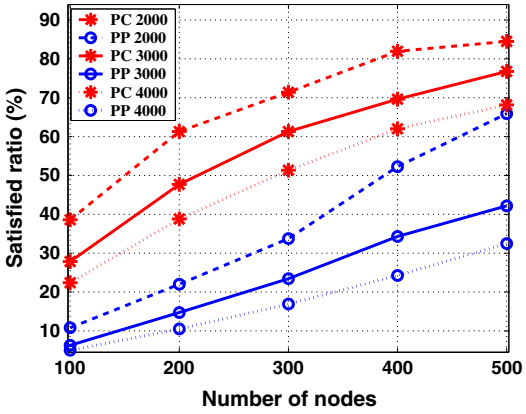


Figure 18. Performance comparisons, edges = n ln(n) — with timeout.

To make the comparison clear, we combine some of the results discussed before in Figures 19 and 20. In this discussion, we made some comparisons between our proposed PC scheme and the traditional path protection scheme in two cases. On one hand, we measure the performance of both scheme when the coming requests have no life time; in other words, if the user request has been satisfied and it accommodates a path, it will stay accommodating that path until the end of the simulating time. This case is indicated in the figures as (*no Timeout*). On the other hand, we let each coming request to have a life time that has been set randomly between 1 and 20 s, where each request, if it granted a path, will stay there until its life time runs out. This has been denoted in the figures as (*with Timeout*).

In Figure 19, we measure the performance of both schemes with 3000 randomly coming requests. Figure 19 (a) shows the performance comparisons with having 200 nodes with different numbers of edges (1600–2000). The solid lines indicated the satisfied ratio when we allow each request to have a life time, whereas the dashed line indicates the case where the requests have no life time. It can be seen that our PC scheme outperforms the traditional PP scheme in both cases. For having 2000 edges with no life timed requests, our scheme satisfies 68% out of the 3000 coming requests, whereas the PP satisfies 35% out of the 3000 requests. On the other hand, our scheme still outperforms the tractional PP scheme when allowing the requests to be timed out. For example, with 2000 requests, our scheme can satisfy 1693 requests, whereas the PP scheme satisfies 1196 requests.

Providing each request with a timeout concept, we enforce the request after being granted a path to release the path after it is timed out, and that allows us to satisfy more requests compared with that when not having a request life time. Let us consider the case when we have 1700 edges; we can see that the PC scheme with the timeout concept provides 69·47% of satisfied ratio compared with 60% of satisfied ratio when there is no timeout concept.

Security Comm. Networks 2012; **5**:689–708 © 2011 John Wiley & Sons, Ltd.
DOI: 10.1002/sec

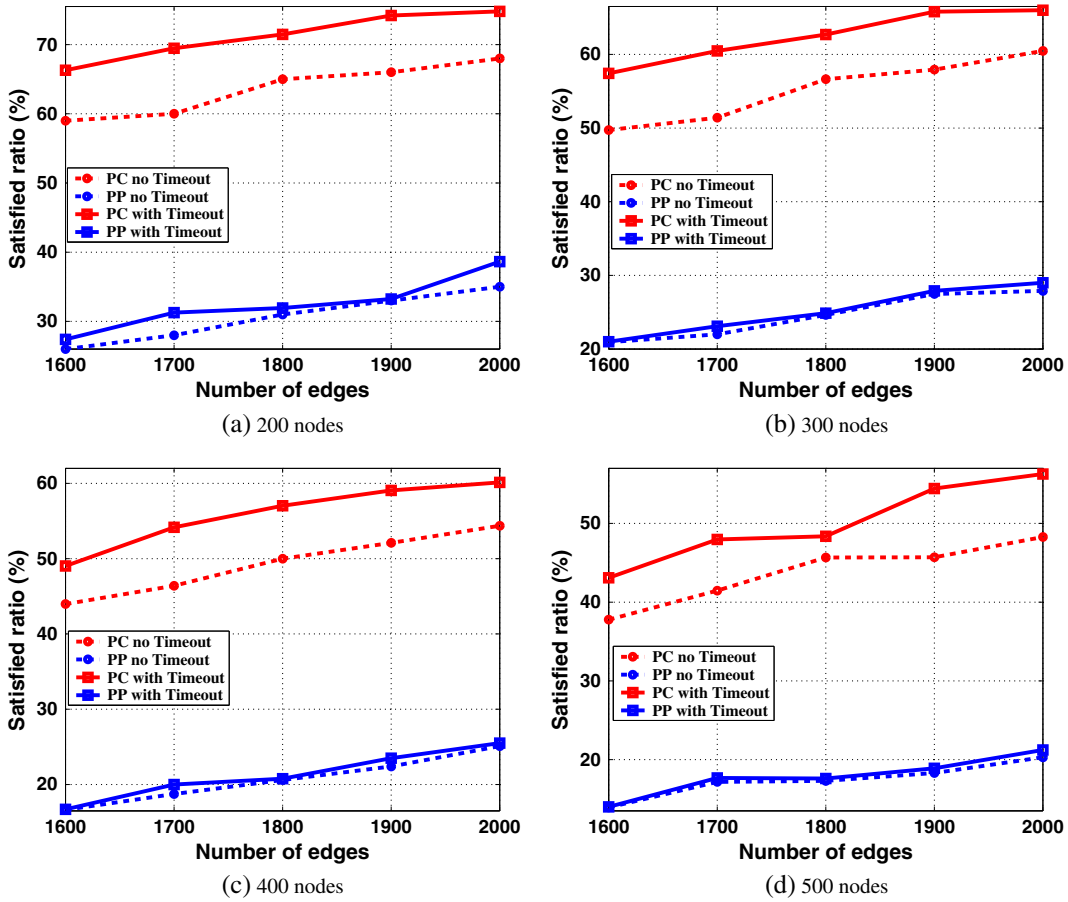


Figure 19. Performance comparisons—fixed nodes with 3000 requests.

The same observations can be seen with larger networks, where we studied the cases of having 300, 400, and 500 nodes with the same previous configurations, where we set the number of edges to range from 1600 to 2000 edges and the number of requests to be 3000 requests. These results can be seen in Figure 19(b, c, and d). In all the cases, we can see that our PC scheme outperforms the traditional path protection scheme in satisfying more number of requests in both cases, when we allow all the requests to (*have or not have*) a life time concept. For example, with 400 nodes in Figure 19(c), it can be seen that with 1800 edges, our PC scheme can satisfy 1711 requests when we allow the requests to have a life time; on the other hand, it can satisfy 1500 no timed out requests, and that outperforms the satisfied ratio provided by the traditional path protection scheme.

Our next comparison consists of making the network more dense through increasing the number of nodes and the number of edges; we tested these scenarios with 3000 coming requests. These results can be seen in Figure 20; in each scenario, we compared both our PC scheme and the PP scheme. We considered the case that each request, after being granted a path, accommodates the path until the end of the simulation, and that is denoted in the figures by (*no Timeout*). Also, we consider the case that each request has a life time, where after the request timed out, it will release the path that had been granted to it, and the workload will be added back to the free bandwidth of the links; this is denoted in the figures by (*with Timeout*).

In Figure 20(a), we set the number of edges to be three times the number of nodes, where n denoted the number of nodes. Our results show that the PC scheme outperforms the PP scheme in all the cases, and that can be improved by allowing the coming requests to have a life time, because of the path release. For example, with 300 nodes and 900 edges with no timeout concept, we can satisfy 26·33% out of the 3000 requests by using our PC scheme compared with 8·77% out of the 3000 requests with the PP scheme. On the other hand, by allowing the requests to be timed out, we can increase the satisfied ratio by 9·54% with the use of our scheme compared with the increase of 0·96% with the use of the traditional PP scheme.

Increasing the density has a better effect on both schemes, where in Figure 20(b), we increased the number of edges to be four times the number of nodes. The results show that our PC scheme still outperforms the traditional PP scheme. By allowing the requests to have a life time

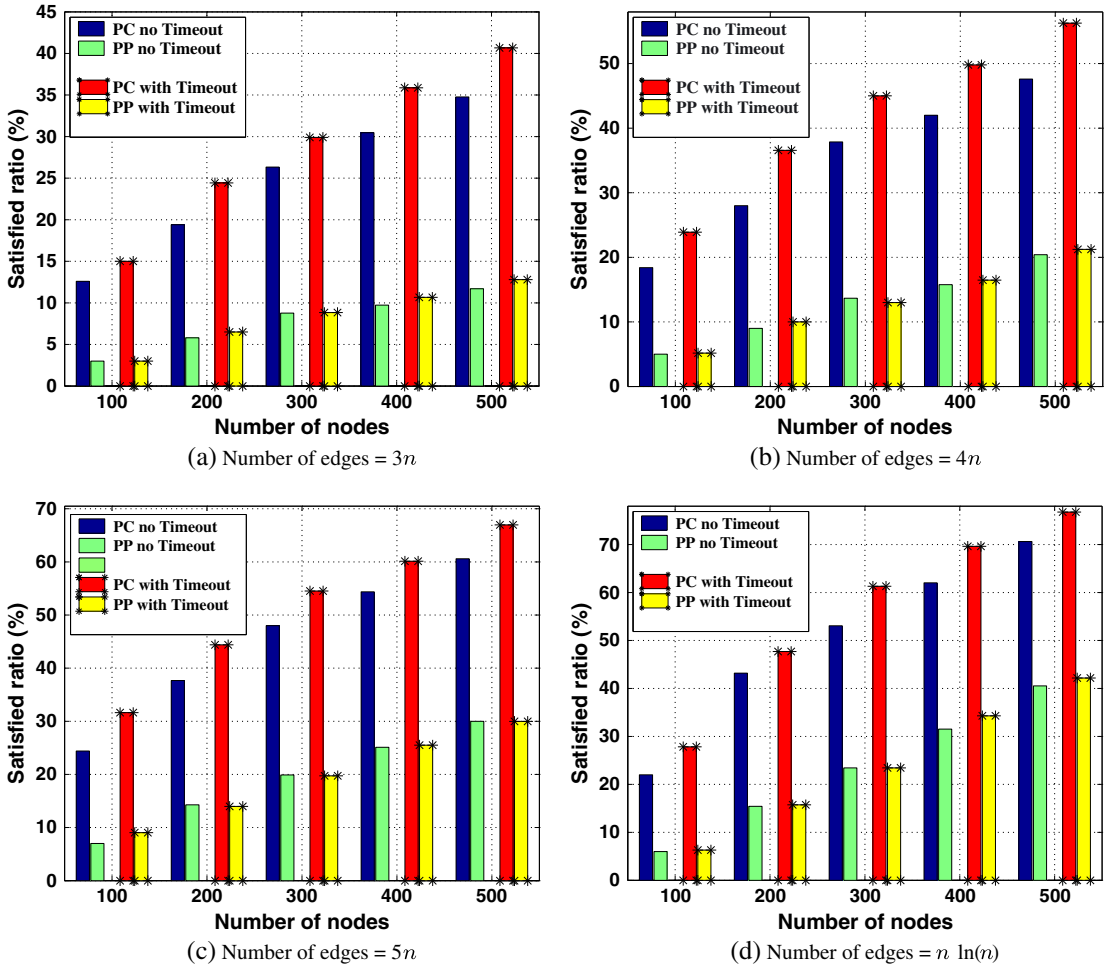


Figure 20. Performance comparisons—different nodes with 3000 requests.

in a 200-node network, by our PC scheme, we can satisfy 1098 requests, whereas with the PP scheme, we can satisfy 300 requests. These results can be improved by making the network more dense, for example, doubling the number of nodes to 400 nodes while keeping the number of edges to be four times n; with our scheme, we can satisfy 1494 requests compared with 495 requests with the PP scheme.

We increased the number of edges to be five times the number of nodes to make the network more dense; we can still have a better satisfied ratio. Whereas with 500 nodes and 2500 edges, we can satisfy 66·97% out of the 3000 requests with timeout concept when we use our PC scheme, whereas this ratio was 60·13% when we had the same number of nodes, but the number of edges were four times the number of nodes. These results can be seen in Figure 20(c and b), respectively.

With this network density, we still outperform with our PC scheme the traditional PP scheme, where in a 200-node network, we can satisfy 1131 requests out of the 3000 coming requests with no timeout scheme with our scheme compared with 428 requests with the PP scheme. And this can be even better with allowing the requests to have a timeout concept where we can increased the number of satisfied requests by 201 more requests with our PC scheme compared with 39 requests with the use of PP scheme.

Our last comparison consists of different numbers of nodes ranging from 100 to 500 nodes with setting the number of edges to be equal to n ln(n). These results can be seen in Figure 20(d). Changing the number of edges from $5n$ to n ln(n) can provide us with better satisfied ratio; for example, with 500 requests and 3000 requests with timeout concept, we can see that by setting the number of edges to be $5n$, we can satisfy 2010 requests by using our PC scheme, whereas by setting the number of edges to n ln (n), we can increase the number of satisfied requests by 294 requests.

It is still obvious that by setting the number of edges to be n ln(n) with our PC scheme, we still outperform the traditional PP scheme in both cases, where we allow the requests to stay until the end of the simulation time, or we allow them to have a specific life time, where each request after timeout will release the path to be used later to satisfy other requests. With 400 nodes and no timeout requests, we can have a satisfied ratio of 62·03% with the

Security Comm. Networks 2012; **5**:689–708 © 2011 John Wiley & Sons, Ltd.
DOI: 10.1002/sec

use of our PC scheme, whereas with the PP scheme, we have a satisfied ratio of 31·53%. These ratios can be improved by allowing the requests to release the path after they timed out, whereas with our PC scheme, we can increase the satisfied ratio by 7·6% and 2·77% by the use of the traditional path protection scheme.

5. CONCLUSIONS

In this paper, we studied the dynamic *p*-cycle provisioning problem. We presented a necessary and sufficient condition for constructing a *p*-cycle with link capacity consideration. Base on this condition, we provided an efficient approach for computing a path for each incoming connection request, with the guarantee that each link on the path is *p*-cycle protectable. Simulation results demonstrated that our *p*-cycle provisioning scheme is comparable with traditional path protection scheme. In our future work, we will study the case where each *p*-cycle protects part of the links with backup bandwidth sharing.

REFERENCES

1. Grover W, Stamatelakis D. Cycle-oriented distributed preconfiguration: ring-like speed with mesh-like capacity for self-planning network restoration. In *IEEE ICC*, Vol. **1**, 1998; 537–543.
2. Blouin F, Sack A, Grover W. Benefits of p-cycles in a mixed protection and restoration approach. In *IEEE DRCN*, 2003; 203–210.
3. Grover W. p-Cycles, ring-mesh hybrids, and mining: options for new and evolving optical transport networks. In *IEEE OFC*, 2003; 201–203.
4. Grover W, Stamatelakis D. Bridging the ring-mesh dichotomy with p-cycles. In *IEEE DRCN*, 2000; 92–104.
5. Stamatelakis D, Grover W. IP layer restoration and network planning based on virtual protection cycles. *IEEE Journal on Selected Areas in Communications* 2000; **18**: 1938–1949.
6. He W, Fang J, Somani A. A p-cycle based survivable design for dynamic traffic in WDM networks. In *IEEE GLOBECOM*, 2005; 1869–1873.
7. Kang J, Reed MJ. Bandwidth protection in MPLS networks using p-cycle structure. In *IEEE DRCN*, 2003; 356–362.
8. Li T, Wang B. Optimal configuration of p-cycles in WDM optical networks with sparse wavelength conversion. In *IEEE GLOBECOM*, 2004; 2024–2028.
9. Wu B, Yeung KL, Lui KS, Xu S. New ILP-based p-cycle construction algorithm without candidate cycle enumeration. In *IEEE ICC*, 2007; 2236–2241.
10. Zhong W, Zhang Z. Design of survivable WDM networks with shared-P-cycles. In *IEEE OFC*, Vol. **1**, 2004; 23–27.
11. Doucette J, He D, Grover W, Yang O. Algorithmic approaches for efficient enumeration of candidate p-cycles and capacitated p-cycle network design. In *IEEE DRCN*, 2003; 212–220.
12. Liu C, Ruan L. Finding good candidate cycles for efficient p-cycle network design. In *IEEE ICCCN*, 2004; 321–326.
13. Xue G, Gottapu R. Efficient construction of virtual p-cycles protecting all cycle-protectable working links. In *IEEE HPSR*, 2003; 305–309.
14. Zhang Z, Zhong W, Mukherjee B. A heuristic method for design of survivable WDM networks with p-cycles. *IEEE Communications Letters* 2004; **8**(7): 467–469.
15. Feng T, Ruan L, Zhang W. Intelligent p-cycle protection for dynamic multicast sessions in WDM networks. *IEEE/OSA Journal of Optical Communications and Networking* 2010; **2**: 389–399.
16. Schupke D, Scheffel M, Grover W. An efficient strategy for wavelength conversion in WDM p-cycle networks. In *IEEE DRCN*, 2003; 221–227.
17. Ruan L, Tang F, Liu C. Dynamic establishment of restorable connections using p-cycle protection in WDM networks. *Optical Switching and Networking* 2006; **3**(3–4): 191–201.
18. Schupke D. On Hamiltonian cycles as optimal p-cycles. *IEEE Communications Letters* 2005; **9**: 360–362.
19. Drid H, Lahoud S, Cousin B, Molnar M. A topology aggregation model for survivability in multi-domain optical networks using p-cycles. In *NPC*, 2009; 211–218.
20. Metnani A, Jaumard B. Directed p-cycle protection in dynamic WDM networks. In *ICUMT*, 2009; 1–6.
21. Schupke D. The tradeoff between the number of deployed p-cycles and the survivability to dual fibre dual failures. In *IEEE ICC*, Vol. **2**, 2003; 1428–1432.
22. Schupke D, Grover W, Clouqueur M. Strategies for enhanced dual failure restorability with static or reconfigurable p-cycle networks. In *IEEE ICC*, 2004; 1628–1633.
23. Shen G, Grover W. Design and performance of protected working capacity envelopes based on p-cycles. *OSA Journal of Optical Networking* 2005; **4**(7): 361–390.
24. Yadav R, Yadav RS, Singh HM. Quality enhancement in p-cycles using optimized restoration path (ORP) algorithm. In *ARTCom*, 2009; 549–553.
25. Zhang H, Yang O. Finding protection cycles in DWDM networks. In *IEEE ICC*, 2002; 2756–2760.
26. Tarjan R. Depth-first search and linear graph algorithms. *SIAM Journal on Computing* 1972; **1**: 146–160.
27. Shen G, Grover W. Extending the p-cycle concept to path segment protection for span and node failure

recovery. *IEEE Journal on Selected Areas in Communications* 2003; **21**(8): 1306–1319.
28. Huang C, Sharma V, Owens K, Makam V. Building reliable MPLS networks using a path protection mechanism. *IEEE Communications Magazine* 2002; **40**(3): 156–162.
29. Awduche D, Chiu A, Elwalid A, Widjaja I, Xiao X. RFC3272: Overview and Principles of Internet Traffic Engineering, May 2002.
30. Ramamurthy S, Mukherjee B. Survivable WDM mesh networks—Part I: Protection. In *IEEE INFOCOM*, 1999; 744–751.
31. Xue G, Zhang W, Tang J, Thulasiraman K. Polynomial time approximation algorithms for multi-constrained QoS routing. *IEEE/ACM Transactions on Networking* 2008; **16**: 656–669.
32. BRITE. [Online]. Available from: http://www.cs.bu.edu/brite/