

Disjoint Multi-Path Routing and Failure Recovery

Yong Oh Lee, A. L. Narasimha Reddy
Dept. of Electrical and Computer Engineering
Texas A&M University
{hisfy205,reddy}@ece.tamu.edu

ABSTRACT

Applications such as Voice over IP and video delivery require continuous network service, requiring fast failure recovery mechanisms. Proactive Fast failure recovery mechanisms have been recently proposed to improve network performance during the failure transients. The proposed mechanisms need extra infrastructural support in the form of routing table entries, extra addresses etc. In this paper, we study if the extra infrastructure support can be exploited to build disjoint paths in those frameworks, while keeping the recovery path lengths close to the original paths. Furthermore, we propose a new framework that enables the construction of nearly-disjoint recovery paths with a lower infrastructure cost. Our evaluations show that it is possible to extend the fast recovery mechanisms to provide support for nearly-disjoint paths.

Keywords

Fast failure recovery, Disjoint multi-path routing, Multi-topology, Hamiltonian cycle, Set of cycle

1. INTRODUCTION

Applications such as voice, interactive games and video conferencing are increasingly relying on Internet. These applications require more robust service from the network. Transients in routing table convergence during link/node failures in the network can cause severe disruptions for these applications. Several mechanisms have been recently proposed to provide fast recovery from failures, to provide continuous service even during the failure transients. In these schemes, backup paths are precomputed before a failure. The failure-discovering router employs the backup next-hop after

a failure, until the new routing tables are computed (to take the failure into account). As a result, the fast recovery mechanisms provide an almost instantaneous response to a failure. Such mechanisms are particularly useful with transient failures, which are common in IP networks today [10].

A number of proactive recovery schemes [16] [4] [12] [21] [24] [24] [18] [26] have been proposed to tolerate link and node failures, some of them being considered for adoption at the IETF. We focus on some of these schemes for our discussion here: Multiple Routing Configurations (MRC) [16] and NotVia [4]. MRC employs multiple topologies of the network. In addition to normal routing configuration with no failures, the additional topologies are designed to cover the loss of some nodes and links. After a failure, an appropriate network topology is used to tolerate this failure for routing packets. In NotVia, routers are provided additional IP addresses. These additional addresses are used during a failure to route around the failed link or node. The node closest to the failure encapsulates the packet with a NotVia address as a destination to route around the failure. Each NotVia address is designated for tolerating an individual failure and the routing tables are computed accordingly to route packets destined to these NotVia addresses, ahead of time.

The proactive recovery schemes require additional infrastructure to provide the fast recovery from failures. This additional support ranges from extra routing table entries, space in the packet headers (to indicate the topology employed for example), and/or extra addresses depending on the employed scheme. The proactive recovery schemes focus on quickly recovering from the failure and may or may not employ some of the links of the original (primary) path (before the failure) in the recovery (backup) path (during the failure transient). Ideally, the length of the recovery path is not much longer than the original path.

In this paper, we study an additional issue of whether the recovery paths can be made disjoint, when possible, from the original path. *We explore if the original path and recovery path can be made disjoint, such that*

the additional infrastructure put in place for failure recovery, could be used potentially for disjoint multi-path routing during normal times when no failures occur. We consider the problem of building recovery paths disjoint from the original paths while keeping the length of recovery paths close to the length of primary paths. We do not constrain the construction of primary paths and hence any routing algorithm can be employed to construct the primary paths. We approach this problem in two stages.

First, we study if the already proposed MRC [16], LFA [3], and NotVia [4] approaches for fast recovery can be enhanced to build disjoint recovery paths in those frameworks, while keeping the recovery path lengths close to the original paths. To this end, we develop techniques for disjoint multi-path computation using disjoint loop free alternatives (D-LFA) and disjoint fast recovery schemes: disjoint multiple routing configuration (D-MRC) and disjoint NotVia (D-NotVia).

Second, we propose a new framework that enables the construction of nearly-disjoint recovery paths with a lower infrastructure cost in terms of routing tables, packet header information and new addresses. We propose fast recovery schemes based on Hamiltonian circuits (HC) and set of cycles (SOC). HC and SOC are used for failure recovery in optical networks [8] [9], and in IP networks [25]. Intuitively, HC and SOC arrange nodes in the network either in a Hamiltonian cycle or as a union of a set of cycles. After a failure, the packets are routed around the failure by sending the packet along the cycles until the destination is reached.

We evaluate the MRC and NotVia based schemes and the cycle based approaches in building fast recovery paths and in enabling the construction of disjoint paths. We show through evaluations that MRC and NotVia can be enhanced to provide nearly disjoint paths, and the schemes based on cycles can provide an alternative tradeoff for the construction of disjoint paths at a lower infrastructure cost.

This paper is organized as follows. In §2, we briefly review the existing fast recovery schemes. In §3, we introduce the disjoint LFA (D-LFA) (in §3.1). Then we propose the methods of computing disjoint recovery path by modified MRC (D-MRC) (in §3.2) and by NotVia (in §3.3). §4 introduces HC and SOC for failure recovery. §5 shows the simulation results. In §6, we compare the different schemes. §7 concludes the paper.

2. FAST RECOVERY SCHEME

Traditional routing in IP networks involves computing a forwarding link for each destination, referred to as the primary forwarding link. When a packet is received at a node, it is forwarded via the primary forwarding link corresponding to the destination address in the packet. To recover from the failure of a link

or node along the forwarding path, a failure-detecting node (which discovers the link or node failure) must reroute the packet over a different link, referred to as the backup forwarding link. The backup forwarding link at different nodes in the network must be chosen in a consistent manner to avoid looping.

The MRC approach [16] divides the network into multiple auxiliary graphs, such that each link is removed in at least one of the auxiliary graphs and each auxiliary graph is connected. Every node maintains one routing table entry corresponding to each auxiliary graph for every destination. If the primary forwarding link fails, a packet is routed over the auxiliary graph referred as back-up topology where the primary link was removed. The auxiliary graph over which the packet is forwarded is carried in the header of every packet. In order to minimize the number of back-up topologies, greedy algorithms are employed where as many nodes and links as possible are removed in a single back-up topology. Each back-up topology results in extra infrastructure support at each node and a larger number of topologies also needs a larger number of bits in the packet header. The focus on decreasing the number of back-up topologies can result in backup paths with longer path lengths.

Current IPFRR standard [3] proposes to use NotVia [4] with Loop Free Alternative (LFA) [3]. LFA is one of the proactive fast recovery schemes. The failure detecting node forwards to LFA next-hop (instead of the primary next-hop) in order to route around the failed link/node. After forwarding to the LFA next-hop, the LFA next-hop uses the primary path to route the packets to the destination. To prevent routing loops and to ensure that the failed node/link is not used in routing, the LFA next-hop is one of the neighbor nodes which satisfy (1).

$$d(N^i, d) < d(i, N^i) + d(i, g) \quad (1)$$

where $d(i, j)$ is the distance between node i and node j , N^i is the neighbor node of node i , and d is the destination.

LFA is less complex than other fast recovery schemes. Only distances ($d(N^i, d)$, $d(i, N^i)$, and $d(i, d)$) are used for finding LFA next hop. In addition, LFA can reduce the backup path length because LFA uses the primary path again after reaching the LFA next-hop. However, there is no guarantee that LFA next hops exist at any node for all destinations.

NotVia is introduced to complement LFAs to achieve full failure coverage. NotVia uses tunneling. When a node detects a failure, that node encapsulates the packet headed in the direction of the failure and sends it to the NotVia address of the failed component (link/node). In the case of a link failure, NotVia address is destined to the next-hop of the primary path. In the case of a

node failure, NotVia address is destined to the next-hop of next-hop of the primary path. NotVia can tolerate any single link or node failure. However, NotVia increases the path length due to the increased hop count between the failure detecting node and the next-hop of the primary path. Moreover, the backup path is not disjoint from the primary path because the packet is expected to continue on the original path after reaching the NotVia address.

3. BUILDING DISJOINT PATHS USING EXISTING FAST RECOVERY SCHEMES

In this paper, we define *disjointness* of two paths between two nodes i and j as follows (similar to the *novelty* measure in [20]). Let $P_{primary} = (i, p_1, p_2, \dots, j)$ be denoted as the primary path between nodes i and j , constructed by the routing scheme. Let $P_{backup} = (i, b_1, b_2, \dots, j)$ be the backup path. The disjointness of the backup path with respect to the primary path is measured as

$$disjointness = 1 - \frac{|P_{primary} \cap P_{backup}|}{|P_{primary}|} \quad (2)$$

We also define *stretch*, of backup as

$$stretch = \frac{|P_{backup}|}{|P_{primary}|} \quad (3)$$

In this paper, we try to construct backup/recovery paths whose disjointness is as close to 1 as possible, while keeping stretch as small as possible. We measure the success of the failure recovery schemes based on these two measures of disjointness and stretch. We also compare the necessary infrastructure support of the different schemes.

3.1 D-LFA

LFA is light-weight failure recovery scheme, but LFA does not guarantee a disjoint path from the primary path. To figure out whether LFA next-hop guarantees a disjoint backup path, we compute the sink routing tree. LFA routes packets to neighbors whose paths to the destination remain intact after a failure and do not cause routing loops after rerouting from the point of failure. We propose to restrict this choice to neighbor nodes whose paths are disjoint from the primary path. For example, in Figure 1, while node j can be an LFA next-hop, the path from j to destination d has a considerable overlap with the original path from node i . In order to make the backup path disjoint from the primary path, we restrict LFA candidates to nodes in different subtrees from the subtree of the failure-detecting node. Only nodes m and n are chosen as D-LFA candidates.

In D-LFA, we need to figure out whether any two nodes are in the same subtree or not rather than to

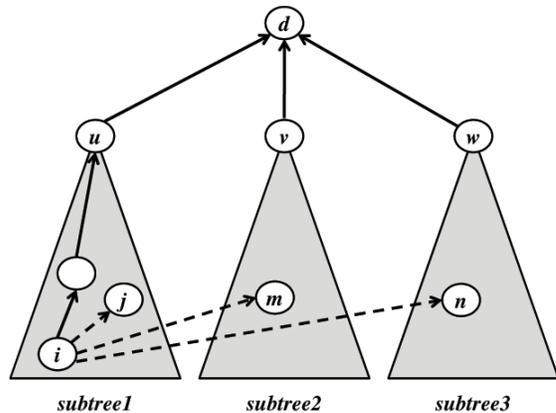


Figure 1: Sink routing tree destined to d

Table 1: Definition for example in Figure 1

	Definition	Example in Fig. 1
$child_d$	the children of the destination in sink routing tree	$child_d = \{u, v, w\}$
$subroot_d^i$	the root of the subtree containing node i	$subroot_d^i = u$

compute which subtrees contain the two nodes. So, we just need to compute subroots of two nodes to figure out whether the two nodes are in the same subtree or not.

In addition to (1), D-LFA next-hop should satisfy (4) for a disjoint backup path.

$$subroot_d^i \neq subroot_d^{N^i} \quad (4)$$

After reaching the D-LFA next-hop, packets are forwarded along the shortest path to the destination.

It is easy to show that D-LFA next-hop selected by (1) and (4) provides a disjoint backup path from the failure detecting node to the destination. D-LFA prevents routing loops and routes around the failed node/link by (1). Since D-LFA next-hop is not in the same subtree of the sink routing tree as that of the failure detecting node by (4), paths from D-LFA next-hop and from the failure-detecting node to the destination have no common links and no common nodes before reaching the destination.

To break a tie among multiple neighbor nodes which satisfy (1) and (4), the neighbor node which has the shortest distance to the destination is selected as the D-LFA node. If the shortest distances to destination from several neighbor nodes are the same, we randomly choose among them.

For example, node i has three neighbor nodes, nodes j , m and n in Figure 1. Suppose all the neighbor nodes satisfy (1). However, node j does not satisfy (4), because $subroot_d^i = u$ and $subroot_d^j = u$. Nodes m and

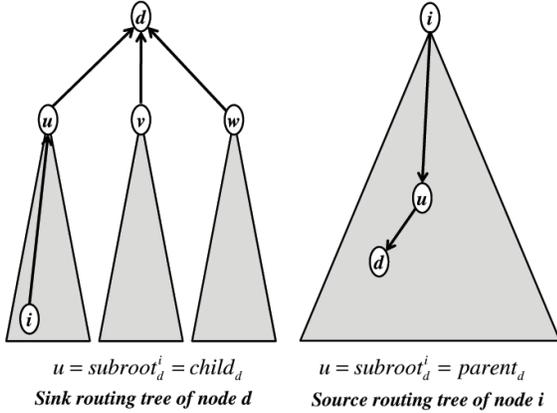


Figure 2: Computing subroot from the routing tree

n satisfy (4), because $subroot_d^m = v$ and $subroot_d^n = w$. Suppose $d(i, m) + d(m, d) < d(i, n) + d(n, d)$, node m is selected as the D-LFA node.

From the perspective of implementation, we need a method to compute subroots of the neighbor nodes from source routing trees. In OSPF and IS-IS routing algorithms, all the nodes have the network topology and the source routing trees to all the destinations (left figure in Figure 2) instead of sink routing trees (right figure in Figure 2). $subroot_d^i$ is the child of node d in the subtree containing node i in the sink routing tree of node d . However, $subroot_d^i$ is the parent of node d in the source routing tree of node i . For D-LFA, each node computes source routing tree of its neighbor nodes or the neighbor nodes can communicate their subroot information with each other. The parent node of the destination d in the source routing tree is the same as the subroot of the sink routing tree toward the destination.

However, D-LFA next hop does not always exist at a node to all the destinations. According to an analysis on real ISP topologies, over 40% links and nodes are not protected by LFA [7]. With the more restrictive D-LFA condition, this percentage is expected to be lower. We need other fast recovery routing mechanisms for the case when D-LFA next-hop does not exist. We combine other the fast recovery routing schemes with D-LFA to achieve full failure coverage and disjointness close to 1 in the next sections.

3.2 D-MRC

3.2.1 Computing D-MRC

Links and nodes can be *isolated* or *restricted* in a backup topology. Packets cannot be routed through an isolated node to another node in a backup topology. The links connected with the isolated node are either isolated or restricted. The isolated link never delivers packets. For this purpose, the weight of the isolated link

is set as infinite. To prevent the last hop problem [16], the restricted link could deliver only the packets headed to the isolated node. The weight of the restricted link is set as a very high value (at least the sum of the weights of all the links [16])

In order to compute disjoint backup paths in the MRC framework, we employ the following ideas. One way to improve MRC is to restrict the maximum number of isolated nodes in a single back-up topology. This is expected to potentially provide shorter backup paths while keeping the number of backup topologies about the same. MRC in [16] is a greedy algorithm. As a result, the early computed back-up topologies have more isolated nodes than the later computed back-up topologies. A large number of isolated nodes could result in long path lengths in backup topologies. We try to distribute the number of isolated (or failed) nodes evenly through all the backup topologies such that the backup paths are smaller in length.

We also employ techniques for choosing the set of isolated nodes in each backup topology carefully such that the paths in the backup topologies are not too long. Nodes adjacent to the isolated node play a key role in keeping the path lengths short in backup topologies. Since an isolated node can only receive packets along the restricted link, the node adjacent to the isolated node along this restricted link carries all of the traffic of the isolated node in the backup topology. Hence, it is important to choose this node carefully (termed *restricted node* here).

We define routing density (RD) as the number of times a neighbor node is selected as the next hop. Routing density is computed using the entries of the routing table for the the primary path.

$$RD(i, j) = \sum_{d=\{v \in V - \{i\}\}} cnt_rd^j(d) \quad (5)$$

$$\text{where } j \in N^i \text{ and } cnt_rd^j(d) = \begin{cases} 1 & \text{next-hop}(i, d) = j \\ 0 & \text{otherwise} \end{cases}$$

The node which has the lowest routing density is selected as the restricted node. It is expected that since this node is used the least number of times in the primary path, by making it the only option for routing to the isolated node in the backup topology, the set of paths used in the backup topology will be very likely different from the set of paths used in the primary topology. In order to facilitate this idea, we add to the link weights, in backup topologies, a weight proportional to the routing density as shown in (6). This particular weight function retains the restrictions on routing to isolated nodes.

$$w(i, j) = w(i, j) + \frac{(RD^j)}{\max_{k \in N^i} RD^k} W \quad (6)$$

,where W is the sum of the weights of all links.

The construction of backup topologies is given in Algorithm 1 as a pseudo code. The algorithm employs the routing density metrics and tries to evenly distribute the number of failed components in each backup topology. In Algorithm 1, $div(i)$ is the function to check if isolating node i makes the graph disconnected.

Algorithm 1 D-MRC

```

 $p \leftarrow 0$ 
 $S \leftarrow \emptyset$   $\triangleright S$  is isolated nodes in all configurations
 $R \leftarrow \emptyset$   $\triangleright R$  is isolated nodes in all configurations
while  $N(S) = N(V)$  do
   $p++$ 
   $G_p \leftarrow G$   $\triangleright G_p$  is the graph in configuration  $p$ 
   $N(S_p) \leftarrow \emptyset$   $\triangleright S_p$  is isolated nodes in configuration  $p$ 
  for all  $v_i \in V$  do
    for all  $v_j \in N^{v_i}$  do
       $w_p(v_i, v_j) \leftarrow w_p(v_i, v_j) + \frac{RD(v_i, v_j)}{\max_{k \in N^{v_i}} RD(v_i, k)} W$   $\triangleright$  weight proportional to the routing density
    end for
    if  $v_i \notin S$  then
      if  $div(v_i) = \text{FALSE}$  &  $N(S_p) < \frac{N(E)}{n}$  then
         $\triangleright n$  is the maximum number of the isolated node per configuration
         $Candidate \leftarrow \emptyset$ 
        for all  $v_j \in N^{v_i}$  do
          if  $v_j \notin R$  then
             $Candidate \leftarrow Candidate \cup v_j$ 
             $w_p(v_i, v_j) \leftarrow \infty$   $\triangleright$  isolated node
          end if
        end for
         $v_R = \arg \min_{v_k \in Candidate} RD(v_i, v_k)$ 
         $\triangleright$  select restricted node
         $R \leftarrow R \cup v_R$ 
         $w_p(v_i, v_R) \leftarrow 2W$   $\triangleright$  restricted node
         $S_p \leftarrow S_p \cup v_i$ 
      end if
    end if
  end for
   $S \leftarrow S \cup S_p$ 
end while

```

3.2.2 Forwarding rules

We employ three fields in the packet header to enable packet forwarding. Backup topology indicator (BTI) field indicates which topology is being used for forwarding this packet. If BTI is 0, the packets are forwarded to the primary next-hop. Otherwise, the packets are forwarded to the next hop of backup topology indicated by BTI. Switching number (SN) field indicates how many times backup topology is switched while this packet has been forwarded. D-MRC allows switching

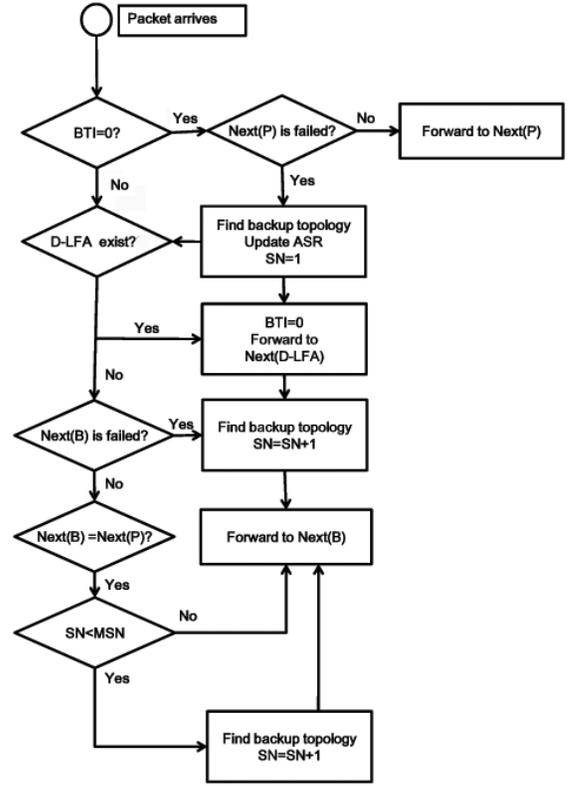


Figure 3: D-MRC forwarding

topologies multiple times to increase the chance of creating a disjoint path from the primary path. In order to avoid potential loops in routing, the number of backup topologies utilized in routing a packet is limited to MSN (Maximum SN). Avoiding subtree root (ASR) field indicates the failure-detecting node's subtree's root in the destination's sink tree.

For disjoint backup path, if the backup forwarding link is a common (used in the primary path) link, this backup forwarding link is regarded as failure. In addition, while the packet is forwarded along the backup path using D-MRC, D-LFA is applied whenever D-LFA next hop is available.

The state diagram in Figure 3 shows the steps that are taken in a node's forwarding process. In Figure 3, $Next(P)$ is the next-hop of the primary path and $Next(B)$ is the next-hop of backup path. First, packets that are not affected by the failure are forwarded to primary next hop. Special steps are only taken for packets that would be forwarded along a backup path (BTI \neq 0).

D-LFA is the first applied recovery scheme. If the node receiving a packet has D-LFA next-hop, the packet is forwarded to D-LFA next-hop and BTI changes to 0.

If there is no D-LFA next-hop, D-MRC is applied. At first, D-MRC checks whether the backup forwarding

link of the topology indicated by BPI is a common link or a failed link. If the backup forwarding link is failed as a common link or a failed link, D-MRC tries to find a new backup topology. If SN is less than MSN, BPI is updated to the backup topology where the backup link is isolated and SN is incremented by 1. The packet is forwarded to the next-hop indicated in this backup topology.

3.3 D-NotVia

We present a new forwarding scheme D-Notvia, for constructing a disjoint path using NotVia addresses. D-NotVia uses NotVia addresses multiple times until the packet reaches the destination. In D-NotVia, when the node with the NotVia address receives an encapsulated packet, it decapsulates the packet and encapsulates it again with the NotVia address of the next-hop of the primary path. This method is repeated until the destination receives the packet. As a result, D-NotVia employs the NotVia method along every hop of the primary path.

D-NotVia, however, incurs the extra cost of encapsulating and decapsulating the packets at every hop of the primary path. This cost of encapsulation and decapsulation needs to be factored into the decision of employing such an approach for constructing disjoint paths. D-NotVia can construct a fully disjoint path, but the path length can be high because the packet is derouted at every hop of the primary path. To reduce the path length, we use D-LFA with D-NotVia.

Suppose node s sends packets to node d and link (s, a) fails in Figure 4. At the node s , the packet is encapsulated and sent to the Notvia address a' , the NotVia address of node a . After the packet reaches a , in the NotVia approach, the packet is decapsulated and forwarded along the primary path $a - b - d$. In this case, the primary path $(s - a - b - d)$ and the backup path $(s - c - a'(a) - b - d)$ have two common links: $a - b$ and $b - d$. In D-NotVia forwarding, the packet is encapsulated again at node a , and sent to destination to b' , the NotVia address of b . At node b , the packet is sent to the notvia destination d' . As a result, the backup path by D-NotVia is $s - c - a'(a) - e - b'(b) - f - d'(d)$ is fully disjoint from the primary path $(s - a - b - d)$. D-NotVia achieves fully disjoint path, but the length of backup path is doubled compared to the primary path in this example. However, if node a has D-LFA next-hop, node e , the backup path becomes $s - c - a'(a) - e - d$. As a result, the the backup path is disjoint and is only one hop longer than the primary path $(s - a - b - d)$.

4. BUILDING DISJOINT RECOVERY PATHS USING HC AND SOC

D-MRC and D-NotVia with D-LFA greatly reduce the length of the backup paths and improve disjoint-

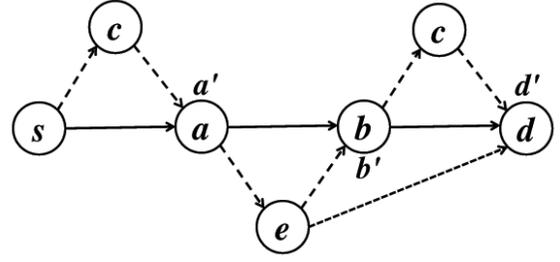


Figure 4: The example of D-NotVia's forwarding

ness from primary path, as we will show later. However, these approaches require considerable increases in additional infrastructure such as the routing table entries, space in the packet headers and so on.

The additional routing table size of D-MRC is $N(T) \times N(V)$ entries and the routing table size of D-NotVia is $2N(E) + 3N(V)$ entries, where $N(V)$ is the number of nodes, $N(E)$ is the number of links, and $N(T)$ is the number of backup topologies. The additional space in the packet headers is ASR (the size of IP address) for D-LFA, BPI (2-4 bits) and SN (2 bits) for D-MRC, NotVia address field (the size of IP address) for D-NotVia. There is the extra cost for encapsulating and decapsulating for D-NotVia.

In order to reduce the infrastructure support for backup paths, we propose HC and SOC routing. HC and SOC routing for failure recovery is employed in optical networks [8] [9] and in IP networks in [25]. However, the proposed HC and SOC routing approaches here are different from [25] and have a different aim: reducing the path length and routing overhead and providing a disjoint path from the primary path.

4.1 HC

HC routing employs a Hamiltonian circuit [6] in the given network. For failure recovery, the failure-detecting node forwards the packet along the HC which covers all nodes in the network.

The packet can be forwarded in two directions along the HC: Clockwise direction (CW) and Counter Clockwise direction (CCW). Any node in HC has two next nodes: next node in the clockwise direction (N_{CW}) and the next node in the counter clockwise direction (N_{CCW}).

The failure-detecting node should determine to which next node among N_{CW} and N_{CCW} the packet should be forwarded as a default backup next hop.

One approach is to always forward the packet along the CW direction. We call this scheme *no-pref* scheme. The advantage of this scheme is that it reduces the routing table entries for backup paths. Without regard to the packet's destination, *no-pref* has two additional

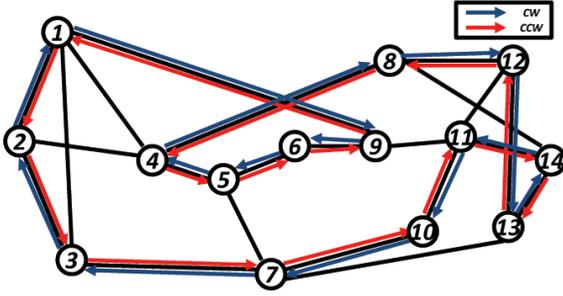


Figure 5: An example of HC routing in a network

next-hops, N_{cw} and N_{ccw} . *no_pref* needs not compute distance to every destination. However, the length of the backup path can be long. The path length of the default direction could be longer than the path length along the alternative direction.

Another approach, called as *pref* scheme, compares the distance to the destination via *CW* direction and via *CCW* directions and chooses the preferred direction. We define the next node of the shorter HC path as N_{pref} and this direction as dir_{pref} . We also define the next node of the longer HC path as $N_{alternative}$ and this direction as $dir_{alternative}$. The *pref* scheme reduces the length of the backup path. However, *pref* scheme increases the routing table size since we need to keep track of dir_{pref} along the HC at each node for each destination.

Any single link failure can be tolerated with HC. If the failed link is not on the HC, routing along the default direction will ensure that the packet reaches the destination. If the failed link is on the HC, routing along the opposite direction of the failed link by the failure-detecting node will ensure that the packet reaches the destination.

Any single node failure can be tolerated within HC. If there is a node failure, packet may reach the destination as it is forwarded along the default direction. The packet sent along the default direction could meet the failed node before the packet reaches the destination. In such a case, the forwarding direction is changed to the alternative direction and forwarding along the alternative direction ensures that the packet reaches the destination.

Figure 5 shows a Hamiltonian circuit in the NSF network topology [1]. Blue line is *CW*-directional HC and red line is *CCW*-directional HC. Suppose node 1 wants to send a packet to node 5, and link 4-5 is failed. (link-failure case) The primary path is 1-4-5. However, node 4 detects the failure of link 4-5, and tries to use HC. N_{CW} of node 4 is node 8 and N_{CCW} of node 4 is node 5. N_{CCW} is the preferred next-hop, N_{pref} . However, *CCW* meets the failed link 4-5. Thus, node 4 chooses

the path via *CW*. As a result, the backup path for link failure (4-5) is 1-4-8-12-13-14-11-10-7-3-2-1-9-6-5. Here is another example of the node failure. Consider sending a packet from node 1 to node 8 and node 4 has failed. Since node 1 cannot use the primary path 1-4-8, it will route the packet along the HC. However, along *CW*, the packet encounters the failed node 4 again. At this point, rerouting the packet in the reverse direction, *CCW* enables the packet to reach the destination.

HC routing reduces the routing table size significantly (2 in *no_pref* scheme and $2N(V)$ in *pref* scheme). However, we will show later HC routing may result in significantly longer backup paths than other approaches and that it may not always be possible to construct Hamiltonian Cycles in all the network topologies.

4.2 SOC

SOC generalizes the HC approach to failure recovery by employing a number of cycles (instead of one cycle in HC approach). SOC covers all the nodes in the network with a set of k -cycles whose maximum size is regulated by k . The cycles employed by SOC can have common nodes and links, unlike in the cycle cover problem [6]. We define *SRC_cycle* as the cycle(s) that contains the source node. *DST_cycle* is the cycle(s) which contains the destination node. If *SRC_cycle* and *DST_cycle* are the same, forwarding along the *SRC_cycle* (*DST_cycle*) ensures that the packet reaches the destination as in HC routing. Forwarding along one cycle is referred as intra-cycle routing in SOC. If *SRC_cycle* and *DST_cycle* are different, inter-cycle forwarding from *SRC_cycle* to *DST_cycle* and then intra-cycle forwarding in *DST_cycle* allows the packet to reach the destination. Inter-cycle routing is facilitated by the common (shared by multiple cycles) nodes between cycles.

For failure recovery of any single link/node failures, two adjacent cycles should have at least two common nodes and one common link. Having one common node between two cycles is a sufficient and a necessary condition for link failure recovery. If forwarding in one direction detects a link failure in intra-cycle routing, the failure recovery is achieved by changing the direction to the opposite direction for forwarding to the common node. However, one common node is not sufficient for node failure recovery. If the common node fails, packets cannot be forwarded from one cycle to another cycle. Thus, at least two common nodes are required for SOC routing. When one of the common nodes fails, packet can be forwarded in the opposite direction until it reaches the other common node. A common link between these common nodes enables a common node to monitor the failure of the other common node.

For forwarding a packet from one cycle to another cycle, a packet should be forwarded to a common node

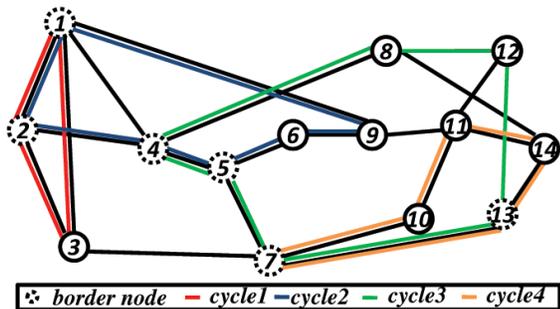


Figure 6: An example of SOC routing in a network

between the cycles. In order to reduce the load on the common links and common nodes, we minimize the number of cycles a link or node belongs to. To keep routing decisions simple and to reduce the amount of infrastructure support, we try to minimize the number of common links and common nodes between cycles.

To reduce routing table entries further, only two common nodes between any two cycles, referred as *border nodes*, have the information of other cycles, (ex: which cycle contains the destination node), even if there are two or more common nodes between the cycles. Inter-cycle routing is supported by only the border nodes.

To compute SOC, each node computes all possible k -cycles [11]. Given all k -cycles, we compute subset sum algorithm [6] with the constraint: having two or more common nodes and one or more common links. An initial cycle is constructed from a starting point (SP). After the initial cycle, multiple cycles are configured to have two or more common nodes and one or more common links with an existing cycle. Intra-cycle routing is similar to HC routing. If the destination is in the cycle, intra-cycle routing delivers packets to the destination. If the destination is not in the cycle, intra-cycle routing delivers packets to border nodes in the cycle. Inter-cycle routing is performed at the border nodes. Inter-cycle routing delivers packets from one cycle to other cycle. Inter-cycle routing determines *next-cycle* and new forwarding direction in the *next-cycle*. *next-cycle* is determined by cycle-level routing. Border nodes compute an order of cycles that will lead the packet to the destination. As mentioned above, border nodes have the distance to the border nodes in other cycles. Based on distances to the border nodes in other cycles, inter-cycle routing computes the *next-cycle*. Packet is forwarded from one cycle to the next by the border nodes until the packet reaches the *DST_cycle*. Packet is forwarded to the destination within the *DST_cycle* using intra-cycle routing. Figure 6 shows a SOC for the NSF network. SOC consists of 4 cycles covering all the nodes in the network, and any adjacent two cycles having two common nodes and a common link.

Table 2: The routing table size in different schemes

Fast recovery scheme	Additional routing table entries	Additional packet header
D-MRC	$N(T) \times N(V)$	ASR(The size of IP address) BTI (2-4bits) SN (2 bits)
D-NotVia	$2N(E) + 3N(V)$	NotVia address (The size of IP address) ASR(The size of IP address)
HC	2 (<i>no_pref</i>) $2N(V)$ (<i>pref</i>)	ASR(The size of IP address)
SOC	- At border nodes $2N(cycle)$ (<i>no_pref</i>) $2N(V)$ (<i>pref</i>) - At other nodes 2 (<i>nopref</i>) $2N(V)$ (<i>pref</i>)	BI (1 bit) ASR(The size of IP address)

The routing table size of SOC at border nodes is $2N(V)$ in *pref* scheme and $2N(cycles)$ in no *pref* scheme. The table size of SOC at other nodes is the same as HC routing.

In order to reduce the length of the backup paths computed by HC and SOC schemes, we combine them with D-LFA. As the packet is forwarded along the designated cycle (in either method), if the packet reaches a node that has D-LFA next-hop, the packet is not forwarded by HC and SOC any more, but is forwarded by D-LFA.

To implement HC and SOC, we add HC and SOC information to the routing table at nodes except border nodes: the nodes in the clock-wise (N_{CW}) and counter-clock-wise directions (N_{CCW}). The optional information are $cost(N_{CW})$, $cost(N_{CCW})$ for *pref* scheme. We add HC and SOC information to the routing table, DST_cycle , $next_cycle$, $cost(N_{CW})$, and $cost(N_{CCW})$ at the border nodes.

A bit in the packet header (BI) indicates whether the packet is being forwarded along the primary paths or the backup paths. The ASR field is needed (as explained earlier) to support the employment of D-LFA along with HC and SOC routing.

Table 2 provides a comparison of the infrastructure support needed by the different methods discussed so far. It is observed that the infrastructure support required by the HC and SOC techniques is much smaller than what is required in the other two schemes MRC and NotVia. The routing table sizes for backup paths are considerably smaller in the HC and SOC techniques. No additional addresses are needed for HC and SOC

Table 3: The portion of existence of D-LFA

	COST239	GEANT	NSF	DARPA	Tiscali
D-LFA portion	93%	33%	73%	63%	27%

Table 4: The number of backup topologies

	COST239	GEANT	NSF	DARPA	Tiscali
MRC	3	6	4	5	7
D-MRC	6	8	6	6	17

routing techniques proposed here as opposed to the requirement of additional addresses in the NotVia approach or the approach in [25]. In the next section, we evaluate all these schemes in simulations.

5. SIMULATION

We evaluate the different schemes for simultaneous failure recovery and disjoint-path routing in this section. We employ a number of network topologies in our study. The networks used for this evaluation are COST 239 (11 nodes, 26 links), GEANT (19 nodes, 30 links), NSF (14 nodes, 22 links) and DARPA (20 nodes, 32 links), Tiscali (40 nodes, 67 links) networks [1].

5.1 Computing disjoint backup path by recovery scheme

5.1.1 D-LFA

First of all, we evaluate how often we can find a D-LFA node in the networks under study. The primary path is computed by Disjtra’s algorithm. We compute paths for all the available source-destination pairs. As seen in Table 3, D-LFA next-hop does not always exist. For COST239, NSF and DARPA networks, more than half of all source-destination pairs have a D-LFA next-hop (average node degree is higher in these networks). However, GEANT and Tiscali have lower node degrees and these networks have fewer percentage of source-destination pairs with a D-LFA next-hop. The results in Table 3 also show that the networks under study exhibit diverse characteristics.

5.1.2 D-MRC

In this subsection, we evaluate the performance of D-MRC. We evaluate the number of backup topologies for D-MRC compared to MRC [16]. The maximum number of the isolated nodes in a single back-up topology is 3 for COST239, GEANT, and NSF, and 4 for DARPA and Tiscali.

As seen in Table 4, D-MRC requires more backup topologies compared to MRC. The reasons are (a)The restriction of maximum number of the isolated nodes in a single back-up topology. (b) The neighbor of the

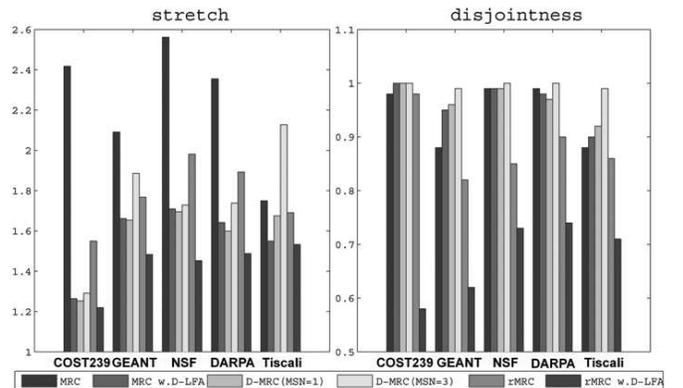
restricted node cannot be the isolated node in the same back-up topology.

With the pre-computed backup topologies, we evaluate stretch based on the average length of the backup paths and disjointness for all the source-destination pairs in the networks. Path length is measured by the number of hops from the source to destination.

The backup paths computed by

1. MRC: MRC with forwarding rule in [16]
2. MRC with D-LFA: MRC with the proposed forwarding rule
3. D-MRC with D-LFA (MSN=1): D-MRC with the proposed forwarding rule (MSN= 1)
4. D-MRC with D-LFA (MSN=3): D-MRC with the proposed forwarding rule (MSN= 3)
5. rMRC: Relaxed MRC (rMRC)[5]
6. rMRC with D-LFA: rMRC with the proposed forwarding rule

The results of creating disjoint paths using MRC are shown in Figure 7. As MRC uses D-LFA in forwarding, backup path length is reduced) and disjointness is improved. D-MRC improves performance marginally when compared to MRC (combined with D-LFA). When we allow multiple backup topologies to be used for creating a disjoint backup path, we see an improvement in disjointness of backup paths, at the cost of slightly longer paths, when we allow up to 3 backup topologies instead of using only one backup topology. It is also observed that nearly 100% of the time disjoint backup paths can be created using D-MRC in all the networks. In the networks such as COST239 and NSF network which have higher node degree, more than half of the source-destination pairs have a D-LFA node and this enables construction of efficient backup paths. Furthermore, in these networks, disjointness of the backup paths is very close to 1, even with only one backup topology. In the remaining networks, allowing multiple backup topologies to be employed in constructing the backup path improves disjointness without significantly

**Figure 7: Evaluation of D-MRC**

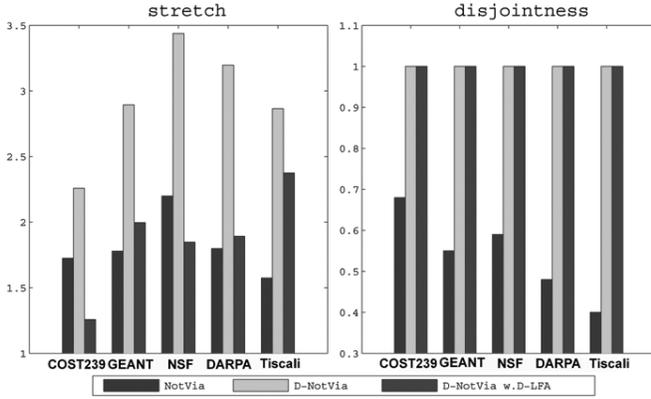


Figure 8: Evaluation of D-NotVia

improving the backup path length.

5.1.3 D-NotVia

In this subsection, we evaluate the performance of D-NotVia. To evaluate how much D-LFA reduces the backup path length of D-NotVia, we compare the performance of D-NotVia and D-NotVia without D-LFA.

D-NotVia guarantees disjoint backup paths (disjointness=1) in all networks. However, deflection at all the nodes of the primary path increases the backup path length. D-NotVia without D-LFA compared to NotVia increases the path length. D-NotVia with D-LFA greatly decreases the backup path length (Figure 8) while maintaining the disjointness of backup paths still at 100%.

5.1.4 HC and SOC

In this subsection, we evaluate the performance of HC and SOC routing.

The number of cycles needed to cover all the nodes in a network is shown in Table 5. We used a constraint of cycle length (k) of 6. While we strived to design cycles that had overlaps of 2 nodes, in some networks (GEANT and Tiscali), the designed cycles had more than 2 nodes in common. Moreover, we could not construct a Hamiltonian cycle for the GEANT and Tiscali networks. It is noticed that for most networks, we could cover the nodes with only few cycles. Based on the results in Table 5, 3-4 bits may be sufficient to identify the cycle that is being used for backup path. This number is similar to the number of bits required for indicating the backup topology in MRC techniques. Figure 9 show the average stretch and the disjointness of backup path using *pref* scheme in HC and SOC. Figure 10 show the

Table 5: The number of cycles for HC and SOC

	COST239	GEANT	NSF	DARPA	Tiscali
HC	1	NA	1	1	NA
SOC	5	5	4	7	11

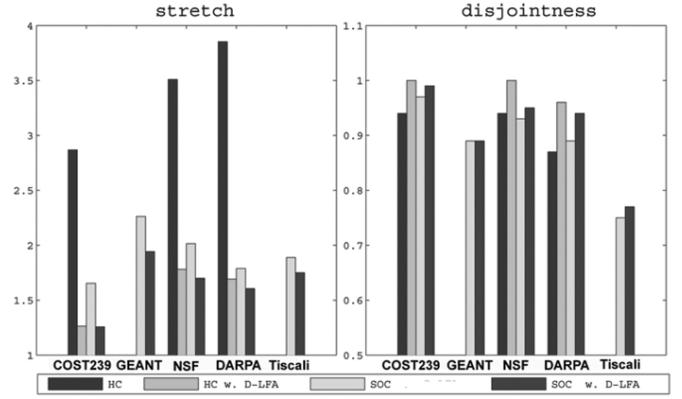


Figure 9: Evaluation of HC and SOC (*pref*)

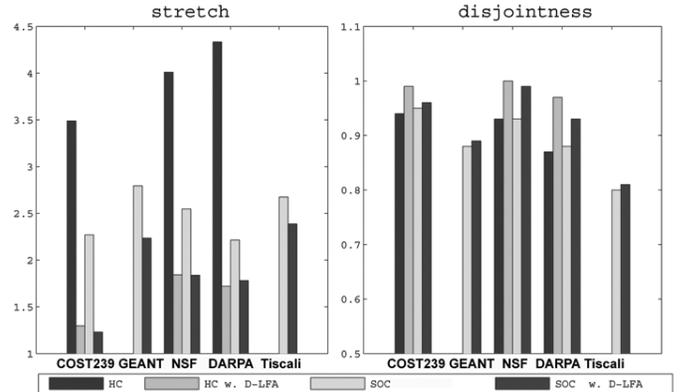


Figure 10: Evaluation of HC and SOC (*no_pref*)

average length and the disjointness of disjoint path with *no_pref* scheme in HC and SOC. In Figure 9 and Figure 10, SOC reduce the backup path length compared with the backup path length of HC. In the comparison between Figure 9 and Figure 10, *pref* scheme shows the 1-hop shorter path length than *no_pref* scheme without D-LFA in both HC and SOC. D-LFA greatly reduces the path length in HC and SOC. In addition, the difference between HC with D-LFA and SOC with D-LFA becomes less than 1-hop. The difference between *pref* scheme and *no_pref* scheme also becomes smaller as D-LFA is used. In Figure 9 and Figure 10, disjointness is not close to 1 without D-LFA. The disjointness of HC with D-LFA and SOC with D-LFA is almost close to 1 in COST and NSF networks and the disjointness of the backup paths in the other networks is still not close to 1, but D-LFA improves disjointness considerably. The disjointness of *pref* scheme and the disjointness of *no_pref* are almost the same.

Since SOC performs significantly better than HC and since it is not always possible to construct a Hamiltonian cycle in all the networks, we will only discuss SOC routing further to keep the discussion and presentation

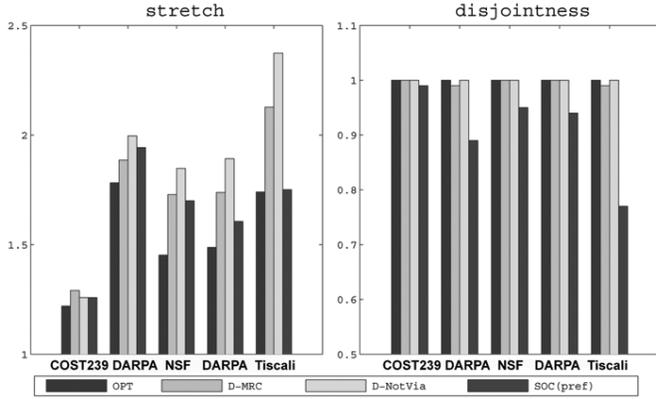


Figure 11: Evaluation of the proposed schemes

of the results clearer.

5.1.5 Comparison between failure recovery schemes

We compare all the different schemes together in Figure 11. For all the different schemes, D-LFA improved the lengths of backup paths and their disjointness. Hence, in this section, we compare the different schemes with the combination of D-LFA. We compare D-MRC (MSN=3), D-NotVia, and SOC (*pref*). We also show the results for optimal disjoint path computation OPT (computed by removing all the links of the primary path for each source-destination pair) for comparison purposes.

Stretch, and disjointness of these schemes are shown in Figure 11. D-NotVia guarantees fully disjoint paths. D-MRC schemes do not guarantee completely disjoint paths, but their disjointness is close to 1 for most of the paths. In addition, average path length is shorter than the other schemes. SOC has longer backup path lengths than D-MRC and less disjointness than that of D-MRC and D-NotVia. However, HC and SOC schemes need much smaller routing tables as seen in Table 2.

5.2 Failure recovery

In this section, we measure the average stretch of recovery paths against link/node failures. The first aim of fast recovery schemes is to recover from link/node failures. Thus, the performance of failure recovery is as important as the performance of computing disjoint backup paths.

We consider all source-destination pairs and all link/node failures. However, we count only the case where the failed link/node is used in the primary path of a source-destination pair.

The average stretch for link-failure recovery is shown in Figure 12. The average stretch for node-failure recovery is shown in Figure 13. Among the different schemes considered, D-MRC provides shorter paths after failures in most of the networks considered here. D-Notvia and SOC(pref) schemes provide longer paths after link or

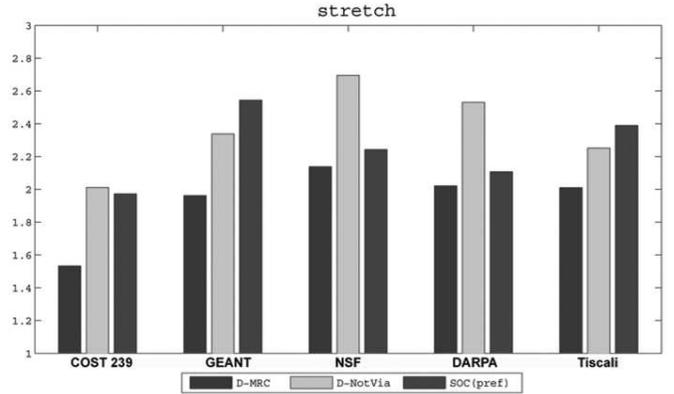


Figure 12: Stretch of the proposed schemes for link-failure recovery

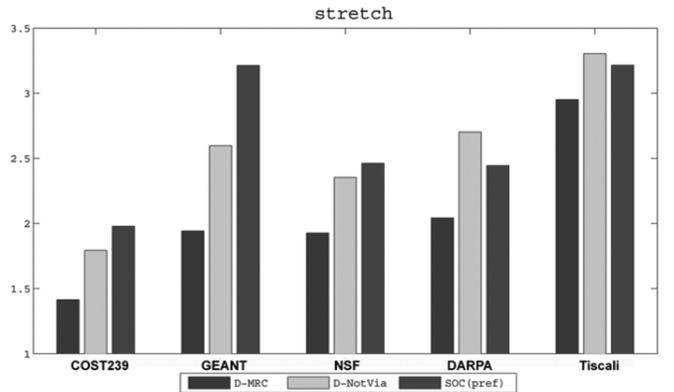


Figure 13: Stretch of the proposed schemes for node-failure recovery

node failures than D-MRC. The choice among the different schemes could depend on the available resources in terms of addresses, routing table entries etc. It is also noted that SOC requires much lower infrastructure support than the other schemes.

6. RELATED WORK

Multi-topology framework is investigated in [22, 16]. QOS routing based on multi-topology is studied in ([17]). Schemes for load-balancing after fast failure recovery are proposed in [15, 2], and a scheme for increasing the diversity of backup topologies is proposed in [5]. These approaches focus on path lengths and link loads after a failure and disjointness of paths is not considered.

Path splicing [20] provides multi-path routing through source level control of derouting a packet from the original path. The source can try to deroute packet along multiple random paths to choose among the possible choices. Not all the choices made at the source may lead to the destination due to routing loops or failures.

Red-blue tree construction [23] is proposed for dis-

joint multi-path routing. While this approach provides disjoint paths, the primary path may not be the shortest cost path and hence may result in providing higher cost even when no failures exist. Failure recovery scheme using disjoint paths by coloring trees is proposed in [13]. Failure recovery for dual link-failure using disjoint path by coloring trees is proposed in [14].

Classical algorithms for computing disjoint backup paths remove or reverse links along the primary path [19]. However, these algorithms, while useful for source routing, can be very expensive in terms of the required infrastructure support.

Pre-configured protection cycle (p-cycle) is proposed in [8] for failure recovery in optical networks [8], [9] and in IP networks using MPLS in [25]. However, p-cycle scheme in [25] is different from our HC and SOC routing. The approach in [25] employs extra addresses and encapsulation (similar to NotVia), but routes the packets after a failure along a pre-computed cycle using MPLS paths. Our approach here does not employ extra addresses or encapsulation and does not require MPLS.

7. CONCLUSION

In this paper we investigated the potential for providing disjoint paths utilizing the same infrastructure that may be provided for fast failure recovery. We proposed D-LFA, D-MRC and D-NotVia to provide backup paths with smaller path stretch and disjointness close to 1. D-MRC and D-NotVia are shown to reduce the backup path lengths and enhance the disjointness. In order to reduce the required infrastructure support, we proposed failure recovery schemes based on Hamiltonian cycles and Set of cycles. HC and SOC have much smaller infrastructure requirements. The backup path lengths in the proposed approaches are slightly longer than with D-MRC and D-Notvia and the disjointness of backup paths is close to 1. The proposed SOC scheme provides an alternative with lower infrastructure requirements, and nearly similar quality backup/alternate paths.

8. REFERENCES

- [1] Rocketfuel topology mapping. WWW <http://www.cs.washington.edu>.
- [2] G. Apostolopoulos. Using multiple topologies for IP-only protection against network failures: A routing performance perspective. In *Tech. Report 377, ICS-FORTH*, April 2006.
- [3] S. Bryant and M. Shand. A framework for loop-free convergence. *Internet draft, draft-bryant-shand-1f-conv-frmwk-03.txt*, October 2006.
- [4] S. Bryant, M. Shand, and S. Previdi. IP fast reroute using notvia addresses. *Internet draft, draft-ietf-rtgwg-ipfrr-notvia-addresses-00.txt*, Dec. 2006.
- [5] T. Cicic, A. F. Hansen, A. Kvalbein, M. Hartman, R. Martin, and M. Menth. Relaxed multiple routing configurations for IP fast reroute. In *IEEE Network Operation Management Symposium*, 2008.
- [6] T. H. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to algorithms*. Prentice Hall, 2006.
- [7] P. Francois and O. Bonaventure. An evaluation of IP-based fast reroute techniques. In *ACM CoNEXT*, 2005.
- [8] W. D. Grover and D. Stamatiadis. Cycle-oriented distributed preconfiguration: Ring like speed with mesh-like capacity for self-planning network restoration. In *ICC*, 1998.
- [9] H. Hong and J. Copeland. Hamiltonian cycle protection: a novel approach to mesh wdm opticalnetwork protection. In *IEEE Workshop on High Performance Switching and Routing*, 2001.
- [10] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyyam, and C. Diot. Analysis of link failures in an IP backbone. In *Proc. of the Internet Measurement Workshop*, 2002.
- [11] N. Immerlica, M. Mahdian, and V. S. Mirrokni. Cycle cover with short cycles. In *Proc. of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, 2005.
- [12] A. Iselt, A. Kirstdter, A. Pardigon, and T. Schwabe. Resilient routing using ECMP and MPLS. In *Proc. of HPSR*, 2004.
- [13] G. Jayavelu, S. Ramasubramanian, and O. Younis. Maintaining colored trees for disjoint multipath routing under node failures. *IEEE/ACM Transactions on Networking*, 17(1):346–359, 2009.
- [14] S. Kini, S. Ramasubramanian, A. Kvalbein, and A. F. Hansen. Fast recovery from dual link failures in IP networks. In *Proc. INFOCOM 2009*, 2009.
- [15] A. Kvalbein, T. Cicic, and S. Gjessing. Post-failure routing performance with multiple routing configurations. In *Proc. of INFOCOM*, May 2007.
- [16] A. Kvalbein, F. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast IP network recovery using multiple routing configurations. In *Proc. of INFOCOM*, April 2006.
- [17] K. W. Kwong, R. Guerin, A. Shaikh, and S. Tao. Improving service differentiation in IP networks through dual topology routing. In *Proc. of CoNEXT*, 2007.
- [18] S. Lee, Y. Yu, S. Nelakuditi, Z. Zhang, and C. Chuah. Proactive vs. reactive approaches to failure resilient routing,. In *Proc. of INFOCOM*, 2004.
- [19] M. Medard, S. Finn, R. Barry, and R. Gallager. Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs. *IEEE/ACM Transactions on Networking*, 7(5):641–652, 1999.
- [20] M. Motiwala, N. Feamster, and S. Vempala. Path splicing: Reliable connectivity with rapid recovery. In *ACM SIGCOMM*, 2008.
- [21] P. Narvaez, K. Y. Siu, and H. Y. Tzeng. Local restoration algorithms for link-state routing protocols. In *Proc. of IEEE International Conference on Computer Communications and Networks*, 1999.
- [22] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault. Mt-ospf: Multi topology MT routing in ospf. *Internet draft, draft-ietf-ospf-mt-04.txt*, Apr. 2005.
- [23] S. Ramasubramanian, H. Krishnamoorthy, and M. Krunz. Disjoint multipath routing using colored trees. *Computer Networks*, 51(8):2163 – 2180, 2007.
- [24] C. Reichert, Y. Glickmann, and T. Magedanz. Two routing algorithms for failure protection in IP networks. In *Proc. of the 10th IEEE Symposium on Computers and Communications (ISCC)*, 2005.
- [25] D. Stamatiadis and W. D. Grover. IP layer restoration and network planning based on virtual protection cycles. *IEEE Journal on Selected Areas in Communications*, 18(10):1938–1949, June 2000.
- [26] Z. Zhong, S. Nelakuditi, Y. Yu, S. Lee, J. Wang, and C. N. Chuah. Failure inferring based fast rerouting for handling transient link and node failures. In *Proc. of IEEE Global Internet*, March 2005.