Fast Recovery from Dual Link Failures in IP Networks

Shrinivasa Kini[†] Srinivasan Ramasubramanian[†] Amund Kvalbein[‡] Audun F. Hansen[‡⋆]
[†]Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, USA
[‡]Simula Research Laboratory, Oslo, Norway
[⋆]Telenor R&I, Fornebu, Norway
skini@ece.arizona.edu, srini@ece.arizona.edu, amundk@simula.no, audunh@simula.no

***Abstract*—This paper develops a novel mechanism for recovering from dual-link failures in IP networks. The highlight of the developed routing approach is that a node re-routes a packet around the next-hop failed link without the knowledge of the second link failure. The proposed technique requires three protection addresses for every node, in addition to the normal address. Associated with every protection address of a node is a protection graph, in which some of the links connected to the node are removed. Every protection graph is guaranteed to be two-edge connected, hence can recover from one failure. The network recovers from the first failure by tunneling with a protection address; and the tunneled packet is routed over the corresponding protection graph. We prove that it is sufficient to provide up to three protection graphs per node to tolerate any arbitrary two link failures in a three-edge connected graph. We evaluate the effectiveness of the proposed technique over several network topologies.**

I. Introduction

The Internet is increasingly being used as a platform for applications with strict demands on robustness and availability, like trading systems, online games, telephony, and video conferencing. For these applications, even short service disruptions caused by routing convergence can lead to intolerable performance degradations. As a response, several mechanisms have been proposed to give fast recovery from failures at the Internet Protocol (IP) layer [1], [2], [3], [4], [5]. In these schemes, backup next-hops are prepared before a failure occurs, and the discovering router handles a component failure locally, without signalling to the rest of the network. The advantage of such solutions is that they allow an almost instantaneous response to a failure, without the instability that follows a convergence. Often, proactive recovery schemes are thought of as a first line of defense against component failures. They are used to maintain valid routing paths between the nodes in the network, until the routing protocol converges on a new global view of the topology. Such a strategy is particularly germane when facing transient failures, which are common in IP networks today [6].

The goal of this paper is to enhance the robustness of the network to dual link failures. To this end, we develop a technique that combines the positive aspects of the various single-link failure recovery techniques. In the developed approach, every node is assigned up to four addresses – one normal address and up to three protection addresses. The network recovers from the first failure using IP-in-IP tunneling [RFC2003] using one of the "protection addresses" of the next node in the path. Packets destined to the protection address of a node are routed over a protection graph where the failed link is not present. Every protection graph is guaranteed to be two-edge connected by construction, hence is guaranteed to tolerate another link failure. We develop an elegant technique to compute the protection graphs at a node such that each link connected to the node is removed in at least one of protection graphs, and every protection graph is two-edge connected. The highlight of our approach is that we prove that every node requires at most three protection graphs, hence three protection addresses.

The paper is organized as follows: Section II surveys the techniques developed for fast recovery from single link failure. Section III describes the network model. Section IV describes our approach for dual link failure recovery, proves the requirement of up to three protection addresses per node, and discusses two different approaches to route using colored trees in the protection (auxiliary) graphs. We evaluate the effectiveness of the proposed approach on several networks and present our results in Section V. Section VI concludes the paper.

II. Fast Recovery From Single Link Failures

Traditional routing in IP networks involves computing a forwarding link for each destination, referred to as the *primary (preferred) forwarding link*. When a packet is received at a node, it is forwarded along the primary forwarding link corresponding to the destination address in the packet. To recover from the failure of the forwarding link, a node must re-route 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.

Equal cost multi-path (ECMP) [RFC 2991, RFC 2992] is a technique employed in IP networks today that computes multiple forwarding links for a specific destination as long as the cost of the paths through each forwarding link is the same as the shortest path cost to the destination. Every packet, whether forwarded along the primary or backup forwarding link, will be forwarded to a node with a lower cost to the

destination than the current node. This monotonicity property of the multiple paths keeps the routing algorithm simple, where a packet need not be identified whether it was a rerouted packet or not. In addition, the failure of a link need not be advertised in the network. However, the drawback of the ECMP approach is that not all nodes in the network may have equal-cost multiple (shortest) paths to a destination. A trivial example is a ring network with odd number of nodes, where no node has ECMP paths. For every destination node in a ring network with even number of nodes, there exists only one node in the ring with two ECMPs to the destination.

In [7], Iselt et al. establish virtual links in the network using Multi-Protocol Label Switching (MPLS) with a specific cost that would enable every node in the network to have equal-cost multi-paths to a destination node. Narvaez et al. [8] develop a method that relies on multi-hop repair paths to route around a failed link. This approach requires message exchanges among nodes within a local neighborhood around the failed link, in order to avoid looping and achieve local re-convergence of routing table. In [9], a similar approach that considers dynamic traffic engineering is developed. Reichert et al. [10] propose a routing scheme named O2, where all routers have two or more valid loop-free next hops to any destination. However, the technique does not guarantee single link failure recovery in any two-edge connected network.

The IETF community is also showing interest in a solution for fast rerouting in IP networks. Shand and Bryant [11] present a framework for IP fast reroute, where they mention three candidate solutions for IP fast reroute that all have gained considerable attention. These are multiple routing configurations (MRC) [2], failure insensitive routing (FIR) [3], [12], and tunneling using Not-via addresses (Not-via) [1]. The common feature of all these approaches is that they employ multiple routing tables. However, they differ in the mechanisms employed to identify which routing table to use for an incoming packet.

The MRC approach 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 port fails, a packet is routed over the auxiliary where the primary link was removed. The routing table to use (or equivalently the auxiliary graph over which the packet is forwarded) is carried in the header of every packet. The drawback of this approach is that it does not bound the number of auxiliary graphs employed. For example, a ring network with n nodes would require n auxiliary graphs, thus requiring $\log n$ bits to specify the routing table to use. The MRC approach has been extended to handle multiple failures [13]. The auxiliary graphs are then constructed so that for any combination of two component failures, there exists an auxiliary graph that does not use the two failed components. With this approach, the number of auxiliary graphs is not bound to a maximum. In [13], medium-sized networks require as much as 12 auxiliary graphs to guarantee recovery from two link failures.

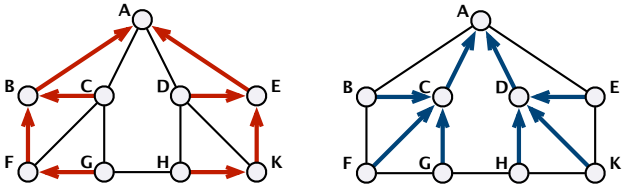
The number of routing tables to be maintained at a node may be reduced by observing that several auxiliary graphs may have the same forwarding node. The idea behind the FIR approach is to use the incoming link over which the packet was received at a node to compute the forwarding link. Therefore, every node will maintain as many routing table entries as the number of links incident at the node. The advantage of this approach is that there is no additional information carried in the packet header. In [14], the authors improve the multi-failure tolerance of FIR; however, no guarantees can be given. To the best of our knowledge, there are no FIR-based approaches that guarantees recovery from dual link failures.

In the Not-via approach, the network is divided into L auxiliary graphs, where L is the number of links in the network, such that in each auxiliary graph only one link is removed. In the auxiliary graph where link ℓ is removed, nodes x and y that are connected by link ℓ are assigned "not-via" addresses, referred to as x_ℓ and y_ℓ. Every node computes the route to nodes x and y in the auxiliary graph. When the primary forwarding link ℓ fails, node x *tunnels* the packet to node y using the not-via address y_ℓ. Tunneling may be implemented using any standard encapsulation protocol, such as IP-in-IP [RFC2003], GRE [RFC1701] or L2TPv3 [RFC3931]. Once the packet arrives at node y, the packet continues along its original path. Observe that the number of not-via addresses required for a node will be the same as the degree of the node, and the network employs as many addresses as the number of links in the network. The idea of tunneling is elegant as routing in the auxiliary graphs is independent of the routing in the original graph. However, the requirement of a not-via address for every link at a node and that different nodes may have different number of addresses assigned to them does not scale. The scalability issue is even more pronounced when multiple links may fail as a not-via address would be required for every possible failure scenario.

A. Colored trees

An efficient approach to route packets along link- or node-disjoint paths in packet-switched networks with minimum routing table overhead and lookup time is to employ colored trees (CTs) [15], [16]. In this approach, two trees, namely red and $blue$, are constructed rooted at a drain such that the paths from any node to the drain on the two trees are link- or node-disjoint. Figure 1 shows an example network with red and blue trees rooted at node A. It is necessary and sufficient for a network to be two-edge (vertex) connected to compute colored trees such that the paths from a node to the root on the two trees are link-disjoint (node-disjoint).

The colored trees approach provides two forwarding links (red and blue) at every node for a destination, thus falls into the class of techniques that employs multiple routing tables. While it resembles MRC, the colored tree approach employs only two routing tables, thus requiring one overhead bit to be carried in the packet header. This overhead bit may be eliminated computing the forwarding link based on input link.



(a) Red Tree rooted at A (b) Blue tree rooted at A

Fig. 1. Example network with colored trees rooted at node A

The packets received on a red (blue) link may be forwarded to the red (blue) neighbors. The packets received over links that are not on either tree may be forwarded on any of the outgoing links. The colored trees may also be employed for tunneling, where if the preferred forwarding link fails, the packet is tunneled to the next node. If the failed forwarding link is present on the red (blue) tree, then the packet is tunneled using blue (red) tree. If the failed forwarding link is not present on any of the trees, the packet may be tunneled to the next node on either tree. However, with colored trees, the packet may be redirected directly to the destination, while still employing any desired routing algorithm when there are no failures. Under this approach, every packet carries a one-bit overhead that specifies if the packet has seen a failure or not. If this bit is set to 0, the packet is forwarded based on the destination address only. If this bit is set to 1, the packet is routed based on the destination address and incoming link.

III. Network Model

Consider a network represented as a graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ denotes the set of nodes and $\mathcal{L}$ denotes the set of links in the network. The links are assumed to be bidirectional. An *edge* $i \rightarrow j$ represents a directed link from node i to node j. A link failure is assumed to affect the edges on both directions. The network is assumed to have at most two link failures at any given time. The link failures are known only to nodes connected to the failed link and the information is not propagated to the rest of the network. We assume that the network employs link-state protocol by which every node is aware of the network topology.

A network must be three-edge connected in order to be resilient to two arbitrary link failures, irrespective of the recovery strategy employed. We assume that the given network is three-edge-connected. Verification of three-edge connectivity and determination of three-vertex connected components have been extensively studied [17], [18], [19], and the complexities of verification and decomposition algorithms are $O(|\mathcal{L}|)$.

IV. Our Approach

In order to recover from arbitrary dual-link failures, we assign up to four addresses per node – one normal address and up to three protection addresses. In IPv4 address space, the two additional bits may be derived from the reserved bits. As IPv6 provides a much larger range of address space, every node may be assigned up to four addresses where the least two significant bits will have specific meaning as below. The default (normal) address of a node $u \in \mathcal{N}$ is denoted by u_0. This acts as the primary address for the routing protocol. In addition, there are three backup addresses denoted by u_1, u_2, and u_3 which are employed whenever a link failure is encountered.

The links connected node to u are divided into three protection groups, denoted by $\mathcal{L}_{u1}$, $\mathcal{L}_{u2}$, and $\mathcal{L}_{u3}$. Node u is associated with three protection (auxiliary) graphs – $\mathcal{G}_{ui}(\mathcal{N}, \mathcal{L} \backslash \mathcal{L}_{ui})$, where $i = 1, 2, 3$. The protection graph $\mathcal{G}_{ui}$ is obtained by removing the links in $\mathcal{L}_{ui}$ from the original graph $\mathcal{G}$. The highlight of our approach is that each of the three protection graphs is two-edge connected by construction. We prove in Section IV-A that such a construction is guaranteed in any three-edge connected graph. Let $\mathcal{S}_{ug} = \{v \mid u\text{–}v \in \mathcal{L}_{ug}\}$ denote those nodes that are connected to u through a link that belongs to $\mathcal{L}_{ug}$. As we will restrict the nodes that can generate traffic on the protection graph $\mathcal{G}_{ug}$ to $\mathcal{S}_{ug}$, we refer to the nodes in $\mathcal{S}_{ug}$ as source nodes in the protection graphs.

A. Computing Protection Graphs

The decomposition of the graph into three protection graphs for every node $u \in \mathcal{G}$ is achieved temporarily removing node u and obtaining the connected components in the resultant network. For each connected component, we observe its connectivity with node u in the original graph and distribute the links connecting node u and the component considered into the required groups. We are given a three-edge connected network. If in addition, the network is also two-vertex connected (3E-2V), then removal of a node will keep the remaining network connected. However, if the network is three-edge connected but only one-vertex-connected (3E-1V), removal of node u may split the network into multiple connected components. In such a scenario, we consider every connected component individually and then combine the corresponding protection graphs obtained from multiple connected components.

Theorem 1: Given a 3-edge connected graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$, there exists three protection graphs for every node u such that each protection graph is two-edge connected and every link connected to u is not present in at least one of the protection graphs.

Proof: We prove the theorem by construction. Consider an arbitrary node $u \in \mathcal{N}$. Let $\mathcal{L}_u$ denote the set of links at node u. The steps for computing the protection graphs from $\mathcal{G}$ are:

1. Remove node u and all the links connected to node u. The remnant graph will consist of one or more connected components. Let $\mathcal{C}$ denote the set of connected components.
2. For every connected component $c \in \mathcal{C}$, we denote the set of links connecting node u and nodes in c in $\mathcal{G}$ by $\mathcal{L}_{uc}$. For component c, perform the following steps:
   - 2.a) Decompose the connected component c into two-edge-connected components. Let $\mathcal{D}_c$ denote the set of two-edge components.
   - 2.b) Reintroduce node u and its links to component c, while retaining the two-edge-connected components. We denote this new subgraph of $\mathcal{G}$ by $\mathcal{G}_{uc}$.

Also, we denote the link protection groups associated with this component by $\mathcal{L}_{uic}$ $(i = 1, 2, 3)$.

2.c) If the number of two-edge connected components in c is exactly 1, i.e., $|\mathcal{D}_c| = 1$, then

2.c.i) If $|\mathcal{L}_{uc}|$ = 3, i.e., there are exactly three links from node u connecting to nodes in the component, then assign one link each to the three groups $\mathcal{L}_{u1c}$, $\mathcal{L}_{u2c}$ and $\mathcal{L}_{u3c}$.

2.c.ii) If $|\mathcal{L}_{uc}| > 3$, of all the edges from node u in $\mathcal{G}_{uc}$, assign at least two edges to group $\mathcal{L}_{u1c}$ and the remaining edges to group $\mathcal{L}_{u2c}$. The third group does not have any links associated with it.

2.d) If $|\mathcal{D}_c| > 1$, then

2.d.i) As $\mathcal{G}$ is three-edge connected, every two-edge-connected component with degree 1 in $\mathcal{D}_c$ has at least two links connecting to node u from the nodes in that component. Therefore, for every two-edge connected component in $\mathcal{D}_c$ which has degree 1, assign at least one link connecting to u in $\mathcal{G}_{uc}$ to groups $\mathcal{L}_{u1c}$ and $\mathcal{L}_{u2c}$ each.

2.d.ii) For every link connected to u in $\mathcal{G}_{uc}$ that is not considered in step 2.d.i, assign it randomly to either $\mathcal{L}_{u1c}$ or $\mathcal{L}_{u2c}$.

3. Combine the corresponding groups obtained across different connected components to obtain the final protection groups.

$$\mathcal{L}_{ui} = \bigcup_{c \in \mathcal{C}} \mathcal{L}_{uic}$$

We now show that each protection graph obtained with protection groups $\mathcal{L}_{ui}$ is two-edge connected. Note that we split the graph $\mathcal{G}$ in step 1 and merge the link groups obtained from the different connected components in step 3. It is sufficient to prove the two-edge connectivity for protection graphs obtained for a single component subgraph $\mathcal{G}_{uc}$ because if every protection graph for $\mathcal{G}_{uc}$ is two-edge-connected, the union with corresponding protection graphs across all components also results in two-edge connected graphs. Therefore, we consider a single connected component c and its subgraph $\mathcal{G}_{uc}$ to demonstrate the two-edge connectivity of its protection graphs. Steps 2.c.i, 2.c.ii and 2.d are the three cases which handle the distribution of links from u in $\mathcal{G}_{uc}$.

Let us first consider the three protection graphs obtained by links distributed for case 2.c.i. Since $\mathcal{G}$ is three-edge connected, the removal of any single link will result in a graph that is at least two-edge connected. And so, each of the three protection graphs obtained is clearly two-edge connected.

Consider the second case of 2.c.ii where the component c consists of one single two-edge connected component and the groups $\mathcal{L}_{u1c}$ and $\mathcal{L}_{u2c}$ have at least two links each. Since c is two-edge connected, addition of node u and links in $\mathcal{L}_{u2c}$ to form $\mathcal{G}_{u1c}$ maintains the two-edge connectivity for $\mathcal{G}_{u1c}$. The same is true for $\mathcal{G}_{u2c}$.

Finally, we consider the case of step 2.d. For any node $v \in \mathcal{G}_{u1}$, we observe its connectivity in the protection graph from its location in its two-edge connected component. Each two-edge connected component with degree 1 in $\mathcal{G}_{uc}$ has at least one link connected to u in the protection graph. Considering links in the set $\mathcal{L}_{uc}$, each component has degree greater than or equal to two in the protection graph. Since the original graph was three-edge connected and the component is two-edge connected, we can compute two link-disjoint paths from v to u, either through links connected to u in the same component or by switching over to other components. Thus, $\mathcal{G}_{u1c}$ is two-edge connected. Identically, $\mathcal{G}_{u2c}$ is also two-edge connected. ■

Consider the example network in Figure 1. The network is three-edge and two-vertex connected. To obtain the protection graphs for node A, we remove A and obtain the decomposition of the network into its connected components. In this case, the connected components themselves are two-edge connected and so no further decomposition is required. Figure 2 shows the two-edge connected components identified for the network. Based on the step 2d of our scheme, we obtain the protection groups as $\mathcal{L}_{A1} = \{$A–B, A–D$\}$, $\mathcal{L}_{A2} = \{$A–E, A–C$\}$, and $\mathcal{L}_{A3} = \phi$. Observe that the network remains two-edge connected after the removal of each $\mathcal{L}_{A1}$, $\mathcal{L}_{A2}$, and $\mathcal{L}_{A3}$.

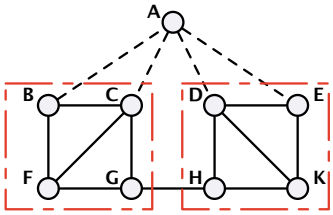
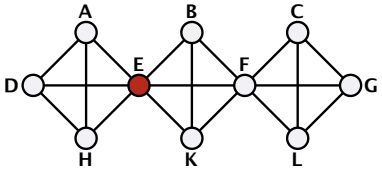


Fig. 2. Example network showing the two-edge connected components when obtaining the protection groups for node A.

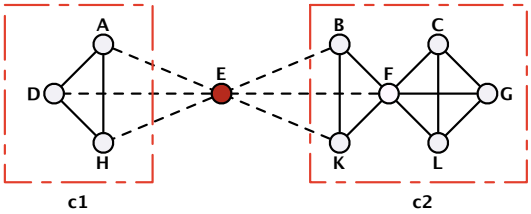
Now, consider the three-edge and one-vertex connected network in Figure 3(a). As earlier, in order to obtain the protection groups at a node, say E, we remove node E and obtain the connected components. We further compute the two-edge connected components in each of these components, as shown in Figure 3(b). In this case, as both components have exactly three links to E, both components will have three protection groups. The final protection groups are obtained by combining corresponding groups from the two component subgraphs and one possible result could be $\mathcal{L}_{E1} = \{$E–A, E–B$\}$, $\mathcal{L}_{E2} = \{$E–D, E–F$\}$ and $\mathcal{L}_{E3} = \{$E–H, E–K$\}$.

B. Packet Forwarding

By default, all packets are transmitted to the default address of the destination. A packet destined to d is transmitted with address d_0, and is routed on graph $\mathcal{G}$. The network is assumed to employ any desired routing algorithm under no failure scenario. Every node is assumed to route the packet based on the destination address and the interface (incoming link) over which the packet was received. For every destination-interface pair, the routing table at a node specifies the interface (outgoing link) over which the packet has to be forwarded. Note that if the network employs shortest path routing, the



(a) A 3-edge and 1-vertex connected network.



(b) 2-edge connected components in the split components obtained after removing node E.

Fig. 3. An example 3-edge connected network and its decomposition into two-edge connected components for node E.

outgoing link for default destination address for a node would be the same, irrespective of the incoming interface.

Consider a packet destined to node d with address d_0 has the default forwarding link as x–y at node x. Let link x–y belong to group g ($\in \{1, 2, 3\}$) at node y. In the event that link x–y is not available, node x stacks a new header to the packet with destination address as y_g. The packet is now transmitted on the protection graph $\mathcal{G}_{yg}$, where it may encounter at most one additional link failure. Given that the protection graph is two-edge connected, we employ the *colored tree* technique to route the packet. Under the colored tree approach, in every protection graph $\mathcal{G}_{yg}$, we construct two trees, namely red and blue, rooted at y_g such that the path from every node to y_g are link-disjoint. Observe that an incoming link in the protection graph may either be red or blue. Therefore, the tree on which a packet is routed is identified based on the incoming link. Thus, it is not necessary to explicitly specify the tree in the packet header. Without loss of generality assume that the packet is routed on the red tree. Given that the packet experiences a failure in the protection graph, it is simply forwarded along the blue tree. Once the packet reaches the desired node y_g, the top header is removed, and the packet continues on its original graph in $\mathcal{G}$. It is worth noting that only the neighbors of y whose link to y are removed in $\mathcal{G}_{yg}$ are the only nodes that will transmit packets to the alias address y_g.

C. Forwarding Tree Selection in a Protection Graph

Consider a packet destined to node d with address d_0 encounters a failure at node x, where the default forwarding link x–y. Node x stacks a new header to the packet with the destination address as y_g. The packet may now be transferred either along the red or blue tree. There are two approaches to select the default tree over which the packet is routed.

The first approach is referred to as the *red tree first* (RTF), where every packet is forwarded along the red tree. Upon failure of a red forwarding link in the protection graph, the packet will be forwarded along the blue tree. When a blue forwarding link fails, the packet is simply dropped as it indicates that the packet has already experienced two link failures[1].

The second approach is referred to as the *shortest tree first* (STF), where a packet is forwarded along that tree which provides the shortest path to the root of the tree. As the packets are first forwarded on the shortest tree, the packets experience lower delays under single link failure scenarios. While the red tree may offer the shortest path for node x in the protection graph $\mathcal{G}_{yg}$, the blue tree may offer the shortest path for another node x' in the same protection graph, where $x, x' \in \mathcal{N}_{yg}$. A packet that is forwarded on the red (blue) tree will be re-routed to the blue (red) tree upon a red (blue) forwarding link failure. The limitation of this approach is that it may result in perennial looping if more than two links fail in the network. Unlike the RTF approach, where a packet to be forwarded on the blue link implies that it has already experienced two link failures, the STF approach does not provide any implicit indication on the number of failures experienced by the packet. We will employ an additional bit that denotes the number of failures the packet has encountered in a protection graph. When forwarded on the shortest-path tree, the bit is set to 0. Upon the failure of the forwarding link on the first tree, the packet is forwarded on the other tree with the bit set to 1. Upon failure of a forwarding link in the protection graph, a packet is dropped if the bit is set to 1.

D. Example

Figure 4 shows the normal path for a packet in our example network without any failures. To recover from a possible failure of link B–A in that path (and sustain one more link failure in the backup path) we obtain the protection graph of the network after removing the protection group $\mathcal{L}_{A1}$ (which contains link B–A) and construct the red-blue trees. Figure 5 shows the protection graph with the red and blue trees rooted at node A. As earlier, note the link disjointedness in the red and blue paths from any node to the root A in the graph. As an example, if the network employs STF, node B chooses the blue tree as the backup path for B–A and then a packet arriving at node B will be tunneled on the path B $\rightarrow$ C $\rightarrow$ A to the appropriate protection address of A.

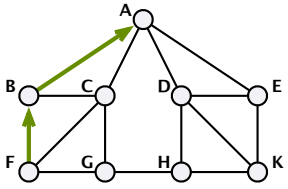
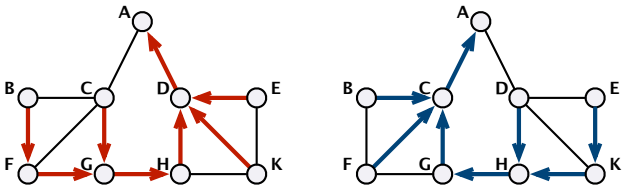


Fig. 4. Network where the normal path from F to A follows F–B–A.

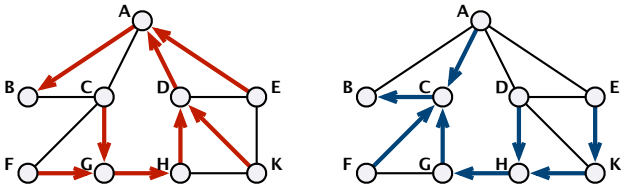
Notice that the packet may experience a second failure along the path, which will then be handled in exactly the same way as the first failure. The maximum number of deflections a

[1]The fact that the packet is destined to the alias address of a node indicates the first link failure, while the reception of the packet along the blue tree indicates that the packet has experienced the second failure.



(a) Red Tree with root A (b) Blue tree with root A

Fig. 5. Example network with colored trees in protection graph after removing link B–A and rooted at node A



(a) Red Tree with root B (b) Blue Tree with root B

Fig. 6. Example network with colored trees in protection graph after removing link F–B and rooted at node B

packet may experience is bounded by four, which happens when the network has two failed links both of which are present in the normal path of a packet. In addition, each failed link is present on the red tree of the other link's protection graph. This can be illustrated by considering another protection graph of our example network by removing link F–B and obtaining the red-blue trees rooted at node B as shown in Figure 6. Note the presence of A → B in the red tree in this protection graph and link B → F in the protection graph of Figure 5. Now for our packet from Figure 4, if both links F–B and B–A have failed in the network, then the path traversed by the tunneled packet using the RTF approach will be F → G → H → D → A → C → B on the protection graph of Figure 6 and B → C → A on the graph in Figure 5. Note that the tunneled packet is switched from the red tree to the blue tree at node A in the recovery path after starting from node F. After arriving at node B, the stacked IP header is removed and the packet is forwarded based on the original header. As the next forwarding link B–A has also failed, the packet is tunneled to node A. The tunneled packet experiences another link failure on the red tree, hence finally arrives at node A over the blue tree.

E. Populating the routing tables

Every node is aware of the network topology obtained using the link-state protocol employed in the network. Every node is assumed to follow the same deterministic procedure, hence the decisions made by every node will be consistent, assuming a consistent view of the network topology. The steps taken by node u to compute its routing table entries are shown in Figure 7.

The decomposition of a graph into two-edge connected components is achieved by employing DFS numbering rooted at an arbitrary node and computing lowpoint for every node. A network is two-edge connected if the lowpoint of every

Steps to compute routing table entries at node u

1. Decompose the network into a set of two-vertex connected components, $\mathcal{C}$.
2. For every node v and every component c, compute the three protection graphs, $\mathcal{G}_{vgc}$ where $g = \{1, 2, 3\}$.
3. For every node v and every component $c \in \mathcal{C}_v$, compute the red and blue trees rooted at node v, referred to as $\mathcal{R}_{vgc}$ and $\mathcal{B}_{vgc}$.
4. If node $u \in \mathcal{S}_{vgc}$ or node u is an intermediate node for any source $s \in \mathcal{S}_{vgc}$ in $\mathcal{R}_{vgc}$ and/or $\mathcal{B}_{vgc}$, then a routing table entry for node v_g and the corresponding incoming link(s) is added to the routing table at u.

Fig. 7. Steps to compute the routing table entries at node u.

node is *less than or equal to* the DFS-index of the parent. A node which does not have a lowpoint less than or equal to the DFS-parent forms the boundary of another component. The link connecting such a node and its parent adds to the degree of both components. The network may be divided into two components by considering the node and the successors along that node as one component (along with the articulation node) and the rest of the nodes as the second component. This procedure is repeated successively to obtain the two-edge connected components of the graph. Once the decomposition is complete, the lowpoint of every node in a component (except the root node of the component) will be less than or equal to the DFS-index of the parent. The DFS numbering and lowpoint computation requires $O(|\mathcal{L}|)$ time, hence the decomposition requires the same time as well. Computing the protection groups for all the nodes, therefore, requires $O(|\mathcal{N}||\mathcal{L}|)$.

The computation of colored trees requires $(|\mathcal{L}|)$ time for specific node as root. Thus, the computation of colored trees for a maximum of $3|\mathcal{N}|$ protection graphs requires $O(|\mathcal{N}||\mathcal{L}|)$.

Finally, the routing table entries at a node may be derived from each colored tree in $O(|\mathcal{N}|)$ time. Therefore, the complexity for computing the routing table entries from every colored tree requires $O(|\mathcal{N}|^2)$.

The total complexity of the algorithm is $O(|\mathcal{N}||\mathcal{L}|)$, determined by steps 2 and 3.

F. Application to networks that are not three-edge-connected

Several real-life networks may not be three-edge-connected, the requirement to tolerate any arbitrary two link failures. However, the network may have enough redundant links to tolerate most dual link failures. In such cases, we may still employ the above developed technique. If the removal of link ℓ connecting nodes u and v leaves the graph one-edge-connected, then we will not be able to construct colored trees to nodes u and v in the protection graph. However, we may divide the protection graph into two-edge connected components and obtain colored trees in each component. A link ℓ' in the protection graph whose removal will disconnect the protection graph will be used as both the red and blue

TABLE I
BITS INVOLVED IN AN IP PACKET UNDER THE PROPOSED SCHEME.

Bit	Purpose	Alternative
Color	Distinguish Red-Blue trees	Use input link
STF	Failure indication in first backup path when using STF	-
Address (Two bits)	Identify protection address IPv4: Reserved bits in header IPv6: Least Significant Bits in address space	-

forwarding links in the colored trees. Therefore, except for failure those links that disconnect the protection graph, any other single link failure may be tolerated.

Table I lists all the bits involved in an IP packet to support our scheme.

G. Highlights of the proposed approach

Here, we list the salient features of the developed approach:

- **Local and proactive recovery.** Our method allows nodes to recover traffic locally. The recovery is pre-planned, hence the failure recovery time is greatly reduced.
- **No explicit failure notification messages.** The proposed technique recovers from the first failure using tunneling with protection address corresponding to the protection graph in which the failed link does not exist. The protection address provides information on the first failed link, thus eliminating the need for employing explicit failure notification mechanisms in the network.
- **Arbitrary routing in the failure-free case.** Our method does no impose any restrictions on the routing in the failure-free scenario. This gives the flexibility to optimize routing/link weights to meet the selected traffic engineering goal. Specifically, our method works in networks employing ECMP and asymmetric link weights.
- **Three protection addresses per node.** Recovery from any two arbitrary link failures in the network is achieved by using three protection addresses and three protection graphs per node. This makes our scheme highly scalable with the size of the network.
- **Repair first failure to next hop.** Our protection scheme tunnels recovered traffic around the protected link to the next node on the original path. When the traffic exits from the tunnel, it will follow its normal path to the destination. This works in much the same way as in not-via routing, hence the proposed scheme can be easily implemented in the current routers.
- **Red/blue trees to protect against second failure.** We employ colored trees in the protection graphs as it is the only single link failure scheme that allows the use of both the trees at the same time. The ability to use both the trees at any instant helps achieve load balancing, particularly when the network has experienced a single link failure.
- **Minimal overhead.** The scheme incurs a minimal overhead of two bits to distinguish the three protection addresses from the normal address of a node for a tunneled packet. If the STF approach is employed to route along the shorter of the two trees, then an additional one bit will be used to indicate the number of failures seen by the tunneled packet and prevent perennial looping.

V. PERFORMANCE EVALUATION

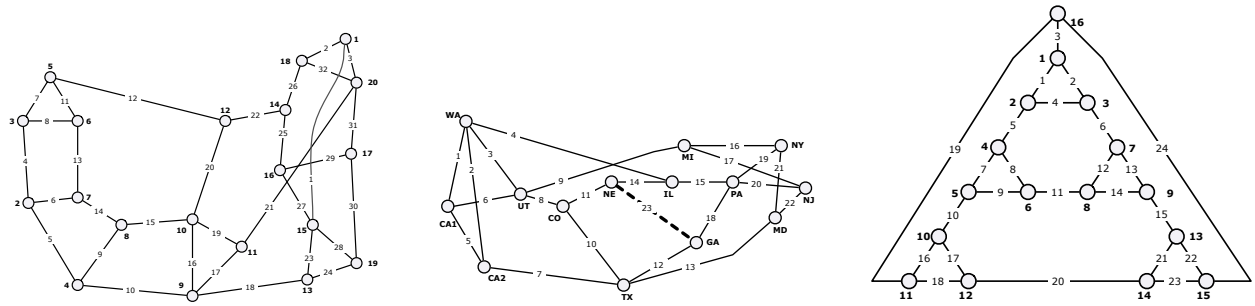
We evaluate the performance of our proposed routing scheme through simulations by applying the algorithm to five networks, as shown in Figure 8: (a) ARPANET; (b) NSFNET[2]; (c) Node-16; (d) Node-28; and (e) Mesh-4x4. The Node-16 and Node-28 networks are hypothetical *minimally 3-connected* networks such that all nodes have exactly three links connected to them.

The performance metrics that we use for evaluation are: (1) average length of the default path (on the protection graph) for every removed link in the protection graph; (2) maximum length of the default path (on the protection graph) across all protection graphs; (3) average length of the backup path under a single link failure in the protection graph averaged over every link failure that affects the default path; and (4) maximum length of the backup path under a single link failure in the protection graph over all protection graphs. We assume that the failure of every link in the network is equally probable.

Consider a link ℓ that connects nodes u and v. When there are no failures, the path length from u to v is 1 hop. When link ℓ fails, both edges $u \rightarrow v$ and $v \rightarrow u$ fail. Consider the edge $u \rightarrow v$. Let $\mathcal{G}_{vg}$ denote the protection graph at node v in which link ℓ was removed. Let $\mathcal{P}_{vg,uv}$ denote path from u to v on the default path in the protection graph $\mathcal{G}_{vg}$. Note that, this path denotes the path on the red tree in the RTF approach, while it will denote the path with the minimum path length among the two trees in the STF approach. If we denote $|\mathcal{P}_i|$ as the path length of a path $\mathcal{P}_i$, we compute the average backup path length between a node pair when the link connected between them has failed as:

$$A_1 = \frac{1}{2|\mathcal{L}|} \sum_{\ell \in \mathcal{L}} \left(|\mathcal{P}_{vg,uv}| + |\mathcal{P}_{ug,vu}| \right)$$

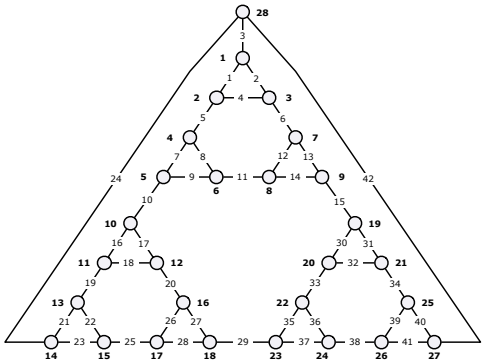
[2]The NSFNET network considered here has been modified from the original network with the addition of link numbered 23 to keep the network three-edge-connected.



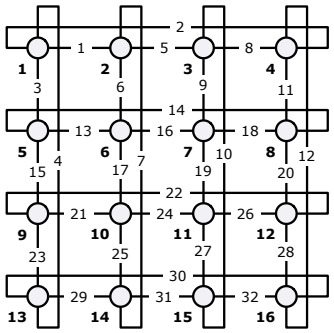
(a) ARPANET
(20 nodes, 32 links)

(b) NSFNET
(14 nodes, 23 links)

Node-16
(16 nodes, 24 links)



(d) Node-28
(28 nodes, 42 links)



(e) Mesh-4×4
(16 nodes, 32 links)

Fig. 8. Networks considered for performance evaluation.

TABLE II
AVERAGE PATH BACKUP PATH LENGTH FOR A LINK UNDER SINGLE- AND DUAL-LINK FAILURES USING THE RTF APPROACH.

Metric	ARPANET	NSFNET	Node-16	Node-28	Mesh-4x4
Average backup path length (single link failure)	6.250	4.630	5.375	8.274	4.062
Maximum backup path length (single link failure)	16	11	14	24	11
Average backup path length (dual link failure)	7.352	5.836	7.942	12.015	5.304
Maximum backup path length (dual link failure)	21	15	20	37	15

TABLE III
AVERAGE BACKUP PATH LENGTH FOR A LINK UNDER SINGLE- AND DUAL-LINK FAILURES USING THE STF APPROACH.

Metric	ARPANET	NSFNET	Node-16	Node-28	Mesh-4x4
Average backup path length (single link failure)	2.641	2.609	2.083	2.274	2.812
Maximum backup path length (single link failure)	8	6	6	8	5
Average backup path length (dual link failure)	8.203	6.492	8.269	12.384	6.508
Maximum backup path length (dual link failure)	22	14	14	24	17

The maximum backup path length under single link failure scenario is obtained as:

$$M_1 = \max_{\ell \in \mathcal{L}} [\ \max(|\mathcal{P}_{vg,uv}|, |\mathcal{P}_{ug,vu}|)\ ]$$

We compute the path length from u to v under two link failures, assuming that link u–v has failed; and that the second failure affects the default path in the protection graph. Assume that the second failure occurs at node $x \in \mathcal{P}_{vg,uv}$. Let $\mathcal{P}'_{vg,xv}$ denote the path from x to v in the tree that is not the default tree on the protection graph $\mathcal{G}_{vg}$. In case of RTF, $\mathcal{P}'_{vg,xv}$ denotes the path from x to v on the blue tree. In case of STF, it denotes the path with the maximum length of the two paths in the protection graph. Let $\mathcal{P}_{vg,ux}$ denote the path from u to x on the default tree in the protection graph. The complete backup path, denoted by $\ddot{\mathcal{P}}_{u,v,x}$, has length equal to the sum of the hops on the two paths and given as:

$$|\ddot{\mathcal{P}}_{u,v,x}| = (|\mathcal{P}_{vg,ux}| + |\mathcal{P}'_{vg,xv}|)$$

The average (maximum) path length from u to v under a link failure in the default path is computed as:

$$H_{uv} = \frac{1}{|\mathcal{P}_{vg,uv}|} \sum_{x \in \mathcal{P}_{vg,uv}, x \neq v} |\ddot{\mathcal{P}}_{u,v,x}|$$

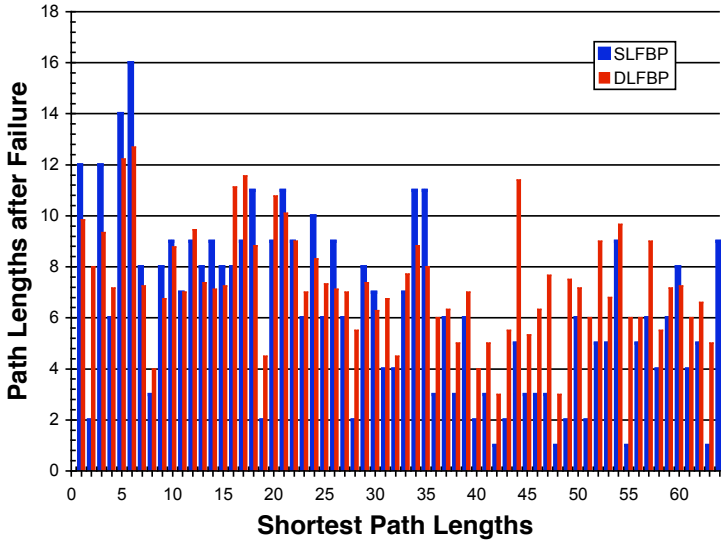
$$M_{uv} = \max_{x \in \mathcal{P}_{vg,uv}} \ddot{\mathcal{P}}_{u,v,x}$$

The average (maximum) path length between two nodes that were connected by a failed link and that the second failure affects the default path in the protection graph is computed as the average of H_{uv} (maximum of M_{uv}) over all u, v pairs that have a link between them, denoted by A_2 (M_2).

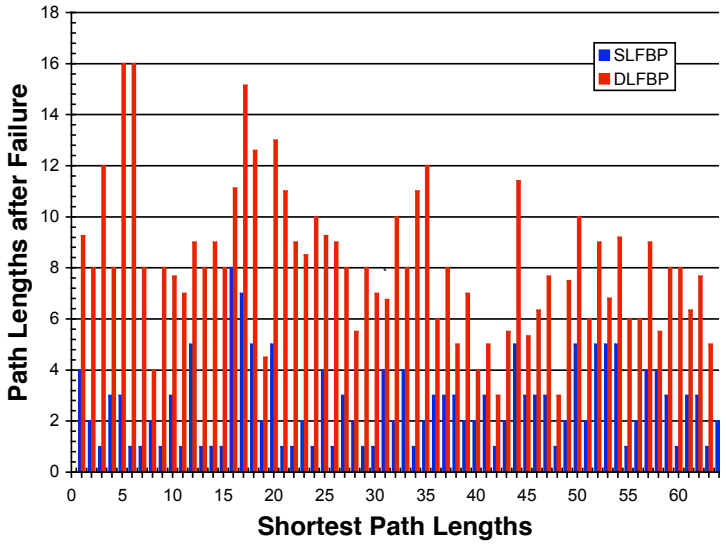
$$A_2 = \frac{1}{2|\mathcal{L}|} \sum_{\ell \in \mathcal{L}} (H_{uv} + H_{vu})$$

$$M_2 = \max_{\ell \in \mathcal{L}} [\ \max(M_{uv}, M_{vu})\ ]$$

Tables II and III show the results for the five networks to be resilient to any arbitrary two link failures with RTF and STF approaches respectively. Figure 9 shows the distribution of backup paths for links in ARPANET using both RTF and STF. The Single Link Failure Backup Path (SLFBP) for a link u–v is the first backup path for a packet arriving at u when u–v is absent. The Dual Link Failure Backup Path (DLFBP) length is the average path length from u–v given u–v is unavailable and another link fails in SLFBP. As expected, STF performs much better than RTF in terms of the backup path lengths under single link failures. However, the advantage of choosing the shortest path after the first failure may be offset by the second failure producing longer paths during the recovery, as seen in the case of NSFNET and Mesh-4x4 networks in the tables. Note that in all cases, the path lengths for the single failure is less than half the total number of links in the network. During recovery from the second failure, a particular node may be re-visited, but the maximum number of links traversed is still much lesser than the total number of links in the network. Because we employ the SimCT algorithm from [16] for the construction of the red-blue trees, we reap the benefits of that algorithm in our scheme as well[3].



(a) RTF



(b) STF

Fig. 9. Distribution of Single Link Failure Backup Path (SLFBP) length and Dual Link Failure Backup Path (DLFBP) (average) length in ARPANET employing RTF and STF schemes.

Finally, we compare our single link failure backup paths with the shortest paths in the network. For every pair of nodes in the network, we obtain the shortest path using Dijkstra's algorithm and compute the modified path, obtained by RTF or STF, considering any single link failure on the shortest

[3]The paper employs the concept of "generalized lowpoint" and "preferred ancestor" techniques to achieve shorter path lengths on the trees. a A discussion of these concepts is beyond the scope of this paper, and the readers are referred to [16].

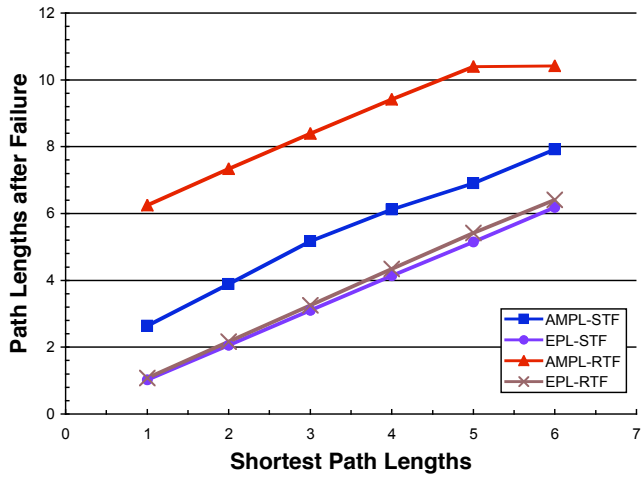


Fig. 10. Plot of Average Modified Path Lengths (AMPL) and Expected Path Lengths (EPL) against Shortest path lengths for single link failures in ARPANET employing RTF and STF schemes.

path. We also consider the probabilistic expected path length considering an arbitrary link failure in the network which may or may not affect the shortest path. Figure 10 shows the plot of the average modified path lengths and expected path lengths under single link failures in ARPANET, employing both RTF and STF. We observe that the expected path lengths are only slightly greater than the shortest path lengths indicating that more often than not an arbitrary single link failure may not affect a packet.

VI. Conclusion

The paper develops a novel scheme to provide dual link failure resiliency in IP networks using IP-in-IP encapsulation based tunneling. The fast recovery from the first failure is handled in a protection domain around the failed link, which can inherently sustain a second link failure within itself. The paper develops the necessary theory to prove that the links connected to a node may be grouped such that at most three protection graphs are needed per node. All backup routes are constructed apriori using three protection addresses per node, in addition to the normal address, making the scheme scalable with the size of the network with minimal overhead. The paper uses aspects from established schemes as intermediate steps and does not put restrictions on the routing protocol handling the normal failure free case. The paper discusses two approaches, RTF and STF, to forward the tunneled packet in the protection graph, describing the benefit of shorter paths in STF at the cost of an extra overhead bit. The performance of the scheme is evaluated by applying the algorithm to five networks and comparing the path lengths obtained with the two approaches.

Acknowledgment

The research developed in this paper is supported by National Science Foundation under grants CNS-0325979 and Cisco Collaborative Research Initiative.

References

[1] S. Bryant, M. Shand, and S. Previdi, "IP fast reroute using not-via addresses," Internet Draft, Feb 2008, draft-ietf-rtgwg-ipfrr-notvia-addresses-02.txt.

[2] A. Kvalbein, A. F. Hansen, T. Čičić, S. Gjessing, and O. Lysne, "Fast IP network recovery using multiple routing configurations," in *IEEE INFOCOM*, Apr. 2006.

[3] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah, "Proactive vs. reactive approaches to failure resilient routing," in *IEEE INFOCOM*, Mar. 2004.

[4] S. Ramasubramanian, H. Krishnamoorthy, and M. Krunz, "Disjoint multipath routing using colored trees," University of Arizona, Technical Report, 2005.

[5] G. Schollmeier, J. Charzinski, A. Kirstädter, C. Reichert, K. J. Schrodi, Y. Glickman, and C. Winkler, "Improving the resilience in IP networks," in *Proceedings of HPSR*, Torino, Italy, Jun. 2003, pp. 91–96.

[6] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an IP backbone network," in *Proceedings INFOCOM*, Mar. 2004.

[7] A. Iselt, A. Kirstadter, A. Pardigon, and T. Schwabe, "Resilient routing using ECMP and MPLS," in *Proceedings of HPSR*, Phoenix, Arizona, USA, Apr 2004.

[8] P. Narvaez and K. Y. Siu, "Efficient algorithms for multi-path link state routing," in *Proceedings of ISCOM*, 1999.

[9] R. Rabbat and K.-Y. Siu, "Restoration methods for traffic engineered networks for loop-free routing guarantees," in *Proceedings of ICC*, Helsinki, Finland, Jun. 2001.

[10] C. Reichert, Y. Glickmann, and T. Magedanz, "Two routing algorithms for failure protection in IP networks," in *Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC)*, Jun. 2005, pp. 97–102.

[11] M. Shand and S. Bryant, "IP Fast Reroute Framework," IETF Internet Draft, Feb. 2008, draft-ietf-rtgwg-ipfrr-framework-08.txt.

[12] S. Nelakuditi *et al.*, "Failure insensitive routing for ensuring service availability," in *IWQoS'03 Lecture Notes in Computer Science 2707*, Jun. 2003.

[13] A. F. Hansen, O. Lysne, T. Čičić, and S. Gjessing, "Fast Proactive Recovery from Concurrent Failures," in *ICC 2007*, June 2007.

[14] J. Wang, Z. Zhong, and S. Nelakuditi, "Cam05-4: Handling multiple network failures through interface specific forwarding," *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pp. 1–6, Nov. 2006.

[15] S. Ramasubramanian, M. Harkara, and M. Krunz, "Distributed linear time construction of colored trees for disjoint multlipath routing," in *Proceedings of IFIP Networking*, May 2006.

[16] G. Jayavelu, S. Ramasubramanian, and O. Younis, "Maintaining colored trees for disjoint multipath routing under node failures," *to appear in IEEE/ACM Transactions on Networking*, 2008.

[17] J. Hopcroft and R. E. Tarjan, "Dividing a graph into triconnected components," in *SIAM Journal of Computing*, vol. 2, no. 3, 1973, pp. 135–158.

[18] S. M. Lane, "A structural characterization of planar combinatorial graphs," *Duke Math Journal*, vol. 3, no. 3, pp. 460–472, 1937.

[19] Z. Galil and G. F. Italiano, "Maintaining the 3-edge-connected components of a graph on-line," *SIAM Journal of Computing*, vol. 22, no. 1, pp. 11–28, 1993.