IP Fast Rerouting for Multi-Link Failures

Theodore Elhourani, Abishek Gopalan, Srinivasan Ramasubramanian
Department of Electrical and Computer Engineering, University of Arizona, USA
{telhoura, abishek, srini}@ece.arizona.edu

***Abstract*—The introduction of coherent optics and wavelength division multiplexing (WDM) in telecommunication networks has led to unprecedented gains in backbone capacity. The increase in optical layer capacity inadvertently exacerbates the problem of traffic loss due to optical component failures. IP networks are designed over optical backbone networks, where each IP link traverses a multihop optical lightpath. Therefore, the failure of optical components often lead to multiple link failures in the IP network. In this paper, we develop an IP fast reroute mechanism using rooted arc-disjoint spanning trees that guarantees recovery from $(k-1)$ link failures in a k-edge-connected network. As arc-disjoint spanning trees may be constructed in sub-quadratic time in the size of the network, our approach offers excellent scalability. Through experimental results, we show that employing arc-disjoint spanning trees to recover from multiple failures reduces path stretch in comparison with previously known techniques.**

I. Introduction

Recent breakthroughs in optical transport technologies promise considerable gains in optical link capacity [1]. Coherent detection with dense wavelength division multiplexing (DWDM) will allow for an increase from the current 1.6 Tb/s, to a projected 8 Tb/s in link capacity. This development comes right on time as demand for bandwidth multiplies due to the wide adoption of multimedia applications, mobile devices and the cloud computing model. Unfortunately, fiber cuts, channel impairments, and other photonic device malfunction are not uncommon. With increased optical link capacity, such failures can cause the loss of large amounts of traffic.

A network is said to be k-edge-connected if the failure of any arbitrary $(k-1)$ links does not disconnect the network. Physical networks often provide modest connectivity, such as the ability to tolerate any single link or node failure, and some two link or node failures. Rarely does a physical backbone network remain connected following the failure of three or more arbitrary links or nodes. However, IP networks that are built on top of these physical networks are likely to be more densely connected. For example, in Figure 1 the optical transport network is a ring, therefore is 2-edge connected. The IP network built on top of it, a fully-connected mesh of four nodes, is 3-edge connected. In such overlay networks, the failure of a single component in the optical domain will likely translate into the failure of several IP links.

To illustrate this, consider Figure 1 where IP link 1–3 is routed over path A–D–C, and link 2–4 is routed over path B–C–D. The failure of link C–D in the optical layer would result in the failure of three links at the IP layer. Nonetheless, the IP network remains connected, even after the three link failures.

Several optical domain failover mechanisms [2], [3] have been developed in the past to recover from failed links and nodes at the optical layer. Typically, such mechanisms attempt to hide the failure of physical links/nodes from the layers above. Alternatively, the optical layer may protect some connections based on priority. In such cases, some connections are restored in the optical domain, while other connections are not. The unrestored connections would still manifest at the IP layer as multiple link failures. Despite the optical layer recovery, operators favor IP layer recovery as it allows them to flexibly reconfigure traffic based on quality-of-service requirements.

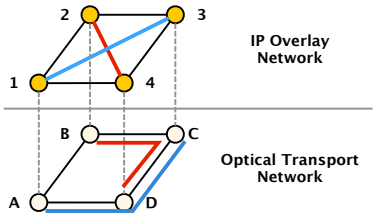


Fig. 1. An example network design involving an optical transport ring of 4 nodes and a fully meshed IP overlay network. The failure of link C–D would result in the failure of links 3–4, 2–4 and 1–3 in the IP layer.

Layer 3 protocols are designed to recompute paths in response to topology changes. However, it takes a few seconds for link information to be disseminated in the network with OSPF-like routing protocols. Meanwhile, a maximum recovery time of 50 ms has become the standard for network fault recovery. Researchers argued this limit on recovery time to be sufficient for maintaining SLAs and acceptable quality for end-users [4]. From the time the link failure occurs to the time the routing table entries are recomputed, packets are dropped because the forwarding entries computed prior to the failure are no longer valid. It is preferred that packets are rerouted along alternate paths while minimizing, if not avoiding, packet drop.

A. Reactive failure recovery

An ideal approach to packet forwarding would be to always attempt to route the packet along the shortest path at every node, assuming that every node in the network has the *accurate and instant* knowledge of all link state updates. If nodes do not have up-to-date information, then rerouting packets may result in loops as intermediate nodes do not know the set of link failures already encountered by the packet. To guarantee loop-freeness, a packet's header can carry the list of the failed links the packet has encountered. The task of any node in the network will then be to forward the packet along the shortest path to the destination, taking into account the link failures

978-14799-3360-0/14/$31.00 ©2014 IEEE

encountered by the packet thus far, as inscribed in the packet's header. Such an approach has three main issues. First, the packet has to carry the set of link failures it has encountered as *per-packet overhead*. Second, an intermediate node has a *compute overhead* of identifying the shortest path forwarding entry for a destination, given a set of link failures in the packet. Once computed, the decision is stored in the forwarding table[1]. Third, the number of forwarding entries per destination d stored in the forwarding table at node x will correspond to the number of different failure scenarios encountered by packets destined to d traversing node x. Therefore in the worst case, if the network has f failures then the number of forwarding entries for d at node x is 2^f.

The above-mentioned overheads may be optimized to some extent in practice. For example, the Failure-Carrying Packets (FCP) technique [5] reduces the per-packet overhead by using labels to indicate the set of failures encountered in the packet. The labels are specific to a directed link, and therefore are meaningful only between neighboring nodes. For example, node n_1 piggybacks label h in order to indicate to node n_2 that the packet has seen failure scenario f involving one or more link failures. The label h will have a different meaning elsewhere in the network. Encoding the ids of k failed links would require $k \log L$ bits in a network with L links, however by using labels the number of bits can be reduced to k, which would allow encoding all possible failure scenarios involving k links. Adjacent nodes exchange control messages to indicate the mapping of labels to failure scenarios.

The computation of the shortest path forwarding entry for a destination under a given failure scenario is performed when a new failure scenario is learned by the router. When a new failure occurs in the network, several packets addressed to different destinations will be rerouted. Depending on the set of failures already encountered by the packets, an intermediate node may see a sudden influx of packets with multiple new failure scenarios. For example, if the network has suffered $(k-1)$ link failures, the worst-case number of new failure scenarios in the network is 2^{k-1}. While the forwarding entries are being calculated, the router would have to buffer packets or drop them. The computation of forwarding entries for all destinations for a failure scenario may be computed in a few (tens of) microseconds. However, once computed, the decisions must be stored in the forwarding table so that packet forwarding delay can be minimized without having to recompute this decision on a per-packet basis. Although each packet may see only a small number of failures, the number of failure scenarios for which an intermediate router has to store a forwarding entry is much larger. Whereas the worst-case number is exponential in the number of failures in the network, in reality, the failure scenarios encountered per destination at a router may be fewer—in the order of a few multiples in the number of failures in the network.

[1]The storing of forwarding entry in the forwarding table is also referred to as *caching*, which helps in reducing the packet forwarding delay at the node. If caching is not employed, then the router has to recompute the forwarding entry for every packet, thus increasing the packet forwarding time.

While the computational overhead of the reactive approach can be reduced by increasing the compute power of a node, the per-packet overhead and forwarding table overhead still remain. In addition, the reactive approach also requires that every router have the capability and the necessary information to recompute forwarding entries, thus requiring a distributed control plane in the network. Although this reactive approach becomes more attractive as technology advances, there are several instances where a *proactive* approach is still preferable.

B. Proactive failure recovery

In proactive failure recovery, also referred to as IP fast rerouting (IPFRR), a router maintains multiple forwarding entries per destination in the forwarding table. One of these forwarding entries may be the default or preferred forwarding entry, typically corresponding to the shortest path. If a failure renders the default interface unavailable, then the router uses an alternative interface to reroute around the failure. The decision of which alternate path to use is based on a combination of parameters, such as destination address, interface on which packet was received, and additional information carried in the packet. Different techniques for fast recovery differ in what parameters are used to select alternate routes and what additional information is required to be carried in the packet. By storing multiple forwarding entries in the forwarding table, proactive techniques attempt to increase the number of failure scenarios that could be handled in the fast datapath of the routers, thus reducing the number of times reactive approaches are invoked.

Our focus in this paper is on proactive failure recovery, which is motivated due to the following reasons. First, even though the time to recompute the forwarding entries under a failure scenario is minimal, it is not small enough to guarantee absolutely no packet dropping. Typically, one alternate forwarding entry per destination is employed when the preferred forwarding link is not available. This mechanism is even employed in the FCP technique [5]. There are numerous techniques in the literature for IP fast re-routing[2]. Some techniques compute the primary and alternate forwarding in specific ways to guarantee recovery from any arbitrary single link failure in the network [6], [7], [8], [9], [10], [11]. Second, software-defined networking (SDN) argues for eliminating the distributed control plane at the switches and routers and replace it with a centralized controller. In such networks, the nodes in the network may neither have the computational capability nor the required information to recompute alternate forwarding entries. Clearly, it is not possible to store forwarding entries for a destination under all possible failure scenarios at the router. To decrease the load on the controller, it is preferred that each router be equipped with a small number of alternate forwarding entries so that the switches continue to forward packets under multiple failures, while

[2]Most IP fast rerouting techniques proposed thus far in the literature are for single-link failure recovery. We do not discuss these works in detail as it is outside the scope of this paper. We will specifically discuss those that address fast recovery from multi-link failures.

the central controller can do route recompilation when the number of failures goes beyond a threshold. Finally, even from a theoretical perspective, it is of interest to evaluate the scalability of proactive failure recovery techniques. For example, what is the maximum number of link failures that one can recover from if every router is allowed to maintain k forwarding entries per destination?

C. Prior work on IP fast rerouting for multi-link failures

To the best of our knowledge, there are only two works [12] and [13] that address IP fast rerouting for multiple link failures. In particular, these works guarantee fast recovery from dual-link failures in three edge connected networks. The approach in [12] employs tunneling to recover from the first failure and routing the tunneled packet over an auxiliary graph that doesn't contain the first link and is guaranteed to be two-edge connected. Every node maintains seven forwarding entries per destination and requires two or three bits of overhead per packet depending on how the tunneled packets are forwarded in the auxiliary graphs. The approach in [13] employs three edge-independent trees rooted at each destination to recover from dual link failures. Every node has three forwarding entries per destination. In addition, if shortest path routing is employed under failure-free scenarios, then a fourth forwarding entry is used. This approach requires two overhead bits per packet.

D. Contributions and organization of this paper

The contribution of this paper is to demonstrate that scalable fast recovery is achievable in IP networks for up to a certain number of link failures determined by the link connectivity of the network, k. The highlights of our approach are: (1) Packets are forwarded on a shortest path tree when there are no failures; (2) Every router has exactly $(k+1)$ forwarding entries per destination; (3) Every packet carries a $(k+1)$-bit overhead; (4) The computational complexity of obtaining the $(k+1)$ forwarding neighbors per destination is sub-quadratic in the size of the network; and (5) Forwarding decisions are made with only local information and information carried in the packet. Our approach is scalable—with only $k+1$ forwarding entries per destination, we guarantee delivery as long as a packet does not encounter more than $(k-1)$ link failures.

Organization: The rest of the paper is organized as follows. In section II, we describe the network model and the rationale for using arc-disjoint spanning trees. Section III describes our fast rerouting approach with arc-disjoint trees along with the correctness proofs and complexity analysis. We evaluate the performance of our fast rerouting technique based on arc-disjoint trees, comparing it with both *link-independent trees*-based IPFRR and the FCP mechanism, in Section IV. Section V concludes the paper.

II. Rationale for Arc-Disjoint Trees

Network model and assumptions. We model the network as a graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$, $\mathcal{N}$ being the set of nodes representing routers and $\mathcal{L}$ the set of links. The links are assumed to be bidirectional. A link x–y is represented by two unidirectional arcs: $x \rightarrow y$ and $y \rightarrow x$. When link x–y fails, we consider both arcs $x \rightarrow y$ and $y \rightarrow x$ to have failed. We refer to undirected links as *links* or *edges*. Directed links are referred to as *arcs*. We assume that the given network is k-edge-connected. Therefore, the removal of up to $(k-1)$ arbitrary links leaves the network connected.

We consider three approaches based on spanning trees for fast re-routing and provide a rationale for employing arc-disjoint.

Link-disjoint spanning trees: In this approach, k link-disjoint spanning trees are constructed per destination. If arc $x \rightarrow y$ is present in one tree, then any other tree cannot contain arcs $x \rightarrow y$ or $y \rightarrow x$. With k link-disjoint trees all nodes remain connected to root d upon $(k-1)$ link failures. By definition, the removal of an arbitrary set of $(k-1)$ links leaves at least one of the spanning trees intact. As every link is present on only one tree, the tree to be employed for routing may be inferred from the input link on which the packet was received. Moreover, if we number the trees from 1 through k, then we may simply start from the tree with the lowest id and reroute the packet on a tree with a higher id when a failure is encountered, until no such tree can be found. This approach does not require any per-packet overhead.

The complexity for computing k link-disjoint spanning trees is $O(k^2|\mathcal{N}|^2)$, assuming the network topology has sufficient connectivity [14]. However, to guarantee the existence of k link-disjoint spanning trees, the network must be $2k$-edge-connected [15]. Note that any network that is at least k-edge-connected remains connected after $(k-1)$ link failures. Using link-disjoint trees requires double the network connectivity as compared to keeping the network connected under $(k-1)$ link failures. Therefore, employing link-disjoint trees for recovering from a large number of failures is not practical due to its high connectivity requirement.

Link-independent trees (IT): A second approach to achieve fast recovery is to employ *link-independent spanning trees*. In addition to being arc-disjoint, any two rooted independent trees have the property that a path from any node x to the root on the two trees are mutually link-disjoint. It is conjectured that a k-edge-connected network contains k link-independent trees [16]. However, to-date, there is no known algorithm for constructing more than three link-independent trees [17], [18]. Moreover, the computational complexity of constructing independent trees is likely to be $O(|\mathcal{N}|^{k-1})$, given the prior works [18], [19], [20]. Hence, employing link-independent trees for recovering from a large number of failures is not practical due to its computational complexity.

Arc-disjoint spanning trees (ADST): Our approach in this paper is to employ k arc-disjoint spanning trees rooted at a destination. If an arc $x \rightarrow y$ is present on one tree, then any other tree cannot contain the same arc. However, arc $y \rightarrow x$ may belong to another tree. In a k-edge-connected network, it is guaranteed that k arc-disjoint trees may be constructed rooted at any node d, referred to as d-rooted trees [21]. Note that the connectivity requirement for the existence of

k arc-disjoint spanning trees is significantly lower than that for constructing k link-disjoint spanning trees. In addition, the running time complexity of constructing k arc-disjoint spanning trees is $O(k^3|\mathcal{N}|\log^2|\mathcal{N}|+|\mathcal{L}|)$ [22]. Therefore, arc-disjoint spanning trees offer excellent scalability compared to the other techniques discussed above.

Note that the failure of a link results in the failure of at most two arcs. Therefore, the failure of $(k-1)$ links can translate into $2(k-1)$ arc failures. One may think it is necessary to compute at least $2(k-1)+1 = 2k-1$ arc-disjoint spanning trees, consequently the network should be at least $(2k-1)$-edge connected. Nonetheless, we will show that it is sufficient to construct just k arc-disjoint spanning trees. Note that $2(k-1)$ arc failures can potentially disconnect each of the k arc-disjoint trees. Interestingly, in [23], the authors note that if we merge all the arcs of the k arc-disjoint trees rooted at a destination, the resulting directed graph will remain connected even after $(k-1)$ links are removed. Therefore, we may reach the destination from any node in the network by traversing segments of multiple trees. The challenge, however, is to identify the right tree to reroute a packet on at an intermediate node with only local information and information carried in the packet, such that additional failures can still be tolerated.

III. Our Approach

We construct a set of k arc-disjoint spanning trees for each destination d, denoted by $\mathcal{T}_{d1}$ through $\mathcal{T}_{dk}$. In addition to these trees, we consider the shortest path tree rooted at d, denoted by $\mathcal{T}_{d0}$. The shortest path tree is the default forwarding tree for all nodes. Thus, for a given destination, every node in the network has $(k+1)$ forwarding entries. Any given arc $i \rightarrow j$ appears on at most one of the $\mathcal{T}_{d1}$ through $\mathcal{T}_{dk}$ trees. Accordingly, at node j, the tree to be employed for routing may be uniquely identified from the incoming arc on which the packet was received.

Every packet carries with it a $(k+1)$-bit vector, referred to as the *failure vector* denoted by f. If the value of the bit in position i $(0 \leq i \leq k)$ is 1, then this indicates that the packet has already been routed along tree $\mathcal{T}_{di}$. Which implies that the packet must not be rerouted along $\mathcal{T}_{di}$ thereafter. Before a packet is forwarded at its source, the value of f is set to 0, indicating that the packet must be routed along the shortest path tree. A non-zero value of f indicates that the packet has encountered at least one failure. If all the bits of f are 1, then the packet cannot be rerouted as all the trees have been exhausted. The packet is then dropped.

Figure 2 shows the procedure employed at a router for forwarding packets. Consider a packet destined to an IP address d, whose prefix is advertised by some router r in the network. As shown in **Step 1**, if the current router is r, the packet is simply forwarded to the corresponding port on which the IP address d can be reached. If the current router has advertised the prefix corresponding to the source IP address, s, then as shown in **Step 2**, the failure vector, f, is reset to 0 before forwarding the packet.

Procedure for forwarding a packet

Given: Source IP address s, Destination IP address d, incoming arc ℓ, failure vector f

1) If the packet is destined to one of router's advertised IP-prefix, then the packet is forwarded to that port. Go to Step 7.
2) If the packet originates from one of the router's advertised IP-prefix, then set the failure vector f to 0.
3) Get the tree $\mathcal{T}_{di}$ to be employed.
   a) If $f = 0$, then $i = 0$.
   b) If $f \neq 0$, $i = \{t \mid \ell \in \mathcal{T}_t\}$.
4) Forwarding.
   a) Let ℓ_1 be the forwarding link on $\mathcal{T}_{di}$.
   b) If ℓ_1 is available, then forward the packet along ℓ_1. Go to Step 7.
   c) Set the i-th bit in the failure vector to 1. $f_i \leftarrow 1$.
   d) If $f = 2^{k+1}-1$, then drop the packet. Go to Step 7.
5) Reverse forwarding.
   a) Assume that the packet arrived along the arc that is reverse of ℓ_1, referred to as ℓ_2.
   b) Let j be the tree corresponding to ℓ_2. $j = \{t \mid \ell_2 \in \mathcal{T}_t\}$.
   c) If $f_j = 1$, then this tree cannot be used. Go to Step 6.
   d) Let ℓ_3 be the forwarding link on $\mathcal{T}_{dj}$.
   e) If ℓ_3 is available, forward packet along ℓ_3. Go to Step 7.
   f) Set j-th bit in the failure vector to 1. $f_j \leftarrow 1$.
   g) If $f = 2^{k+1}-1$, then drop the packet. Go to Step 7.
6) Picking an available tree.
   a) Among all the trees that are still available, assign i to be the tree with shortest path to destination from the current node.
   b) Go to Step 4.
7) Stop.

Fig. 2. Forwarding procedure for routing packets using of arc-disjoint trees.

We now describe the forwarding procedure employed at any router, x, that receives this packet. To facilitate explaining the packet forwarding procedure we use Figure 3.

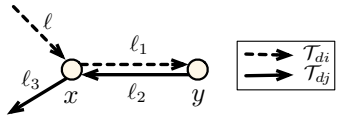
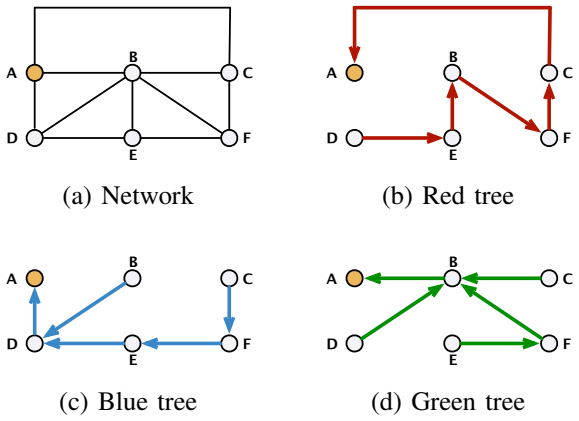


Fig. 3. Illustration of packet forwarding at an intermediate router x that receives a packet on arc ℓ destined to IP address d.

First, as shown in **Step 3**, the tree to use for forwarding is selected based on the incoming arc ℓ and failure vector f. If the failure vector, f is zero, this implies that the packet has not seen any link failures thus far, and hence the shortest path tree rooted at the destination, namely $\mathcal{T}_{d0}$, must be used. If on the other hand f is non-zero, then the tree t on which the incoming arc ℓ is present is selected for a forwarding attempt $(\mathcal{T}_{di})$.

Step 4 describes the steps taken to forward the packet along the selected tree, $\mathcal{T}_{di}$. If the link of the forwarding arc on tree $\mathcal{T}_{di}$, ℓ_1, is available, then the packet is forwarded along $\mathcal{T}_{di}$. Otherwise, the tree is marked unusable ($f_i \leftarrow 1$), and a check for possible re-routing is made. In Step 4(d)) the router verifies that some tree is still unused. In case no tree is available, the



(a) Network (b) Red tree

(c) Blue tree (d) Green tree

Fig. 4. Example network and the arc-disjoint trees on it (rooted at B) to illustrate the re-routing procedure. (a) Network, (b) Red tree, (c) Blue tree and (d) Green tree.

packet is dropped.

Step 5 is the key step describing the procedure for selecting the next tree, upon encountering a forwarding link $(x - y)$ failure. As shown in Step 5(b), the tree $\mathcal{T}_{dj}$ corresponding to the reverse arc (ℓ_2) on the failed link is attempted first. As we will demonstrate in Section III-B, this choice of the reverse arc is imperative for guaranteeing recovery. If this tree, $\mathcal{T}_{dj}$, is not available (marked by $f_j = 1$), then Step 6 is used to pick some other available tree. Otherwise, an attempt to forward along $\mathcal{T}_{dj}$ is made and Steps 5(d)–(g) are similar to the forwarding step 4. The only difference being that now the packet is assumed to have arrived on ℓ_2.

Step 6 is used to select a new tree when both trees $\mathcal{T}_{di}$ and $\mathcal{T}_{dj}$, corresponding to the forward and reverse arcs, ℓ_1 and ℓ_2 are deemed unusable. Among the set of available trees, we select the one that offers the shortest path to the destination from the current node.

A. An example

The network in Figure 4 is used to illustrate the forwarding mechanism shown in Figure 2. Consider a destination IP that is advertised via node A. Since the example network is three-edge connected and the packet is destined to router A, we consider three arc-disjoint spanning trees rooted at A as shown in Figures 4(b), (c) and (d).

Assume links E–D and F–B have failed. With these two link failures, the reader can verify that all three forwarding trees are affected. Consider a source IP reachable via node E. Assuming E–D is on the shortest path tree, E sets the packet's f vector to 0 and forwards on E–D. Because E–D has failed, E sets the blue tree's bit in f, and must choose the tree which uses the opposite arc, the red tree in this case, for forwarding. The packet then reaches B on the red tree where it encounters failed link B–F. B sets the red tree's bit in f and chooses the tree of the opposite arc on B–F. The green tree is selected for recovery, and the packet finally reaches its destination using link B–A.

The recovery was successful even though the two failed links have disconnected paths on all three trees. In general, any set of $(k-1)$ link failures in a k-link-connected network could disconnect k distinct trees. It is not immediately obvious why recovery would still be possible when all k trees are affected. However, as we prove later, using the procedure in Figure 2, we guarantee recovery under any arbitrary set of $(k-1)$ simultaneous failures, even when all k colored trees are affected.

In this example, a packet starting at E takes a recovery path $E \rightarrow B \rightarrow A$ to reach its destination. Observe that even if additional links were to simultaneously fail the packet would still reach A. In fact, in this particular example, all links except links B–A and B–E can fail and our recovery procedure would still successfully deliver a packet from E to A. Thus, in a k-link-connected network, even under more than $(k-1)$ link failures recovery can succeed.

Here we highlight the importance of choosing the tree of the reverse arc on the failed link for recovery. In the same network example of Figure 4, consider a source IP reachable via node F, and the simultaneous failure of links E–D and F–B. Given that F–B is on the shortest path tree, F fails to forward the packet, setting the green tree's bit in the header. According to the forwarding procedure, F must now use the red tree. For the sake of demonstration, let's assume F chose instead to forward on the blue tree. Observe that at this point there is no guarantee the packet will not encounter the same failed link if it gets routed on the red tree at an ulterior point. The packet reaches E on the blue tree, and node E sets the blue bit in the packet and forwards on the red tree (the only available tree). At B the packet encounters failed link F–B for the second time and the packet is dropped as all trees have been exhausted. Thus, the packet is dropped even though there are only two link failures in a three-link-connected network. This shows that a packet runs the risk of visiting the same failed link more than once (and over-counting the failures), if the tree of the opposite arc on a failed link is not exhausted first.

B. Correctness

We now prove that packets will always reach the destination under up to $(k-1)$ simultaneous link failures, when our fast reroute procedure of Figure 2 is used in conjunction with k arc-disjoint spanning trees (ADSTs) in a k-edge-connected network. The k ADSTs are rooted at a destination node d. This proof demonstrates why the reverse arc choice made in Figure 2 is required in order to guarantee recovery.

Lemma 1: A packet attempts to use a failed link at most once.

Proof: By the definition of arc-disjoint trees, we observe that at most two of the trees can be present on any given link. Consider any failed link, $x - y$ as shown in Figure 3 that a packet destined to d attempts to use by arrival at node x on tree $\mathcal{T}_{di}$ along arc l. Thus, clearly arc ℓ_1 is on $\mathcal{T}_{di}$.

Since this link has failed, the bit f_i is set to 1 in the forwarding procedure in Figure 2 and hence tree $\mathcal{T}_{di}$ is never used henceforth. Thus, we are guaranteed that the packet will never attempt to re-use arc $x \rightarrow y$ on $\mathcal{T}_{di}$.

Now, if the reverse arc l_2 is not present on any of the arc-disjoint spanning trees, then the proof is complete as node y will not have x as a forwarding node on any of the trees $\mathcal{T}_{d1} \ldots \mathcal{T}_{dk}$. We now consider the case when the reverse arc is present on another tree, say $\mathcal{T}_{dj}$ as shown in Figure 3. According to our recovery procedure, we always attempt to use tree $\mathcal{T}_{dj}$ at node x. If it is already unavailable, then the packet will never use $\mathcal{T}_{dj}$ and hence never attempt to cross link $y \rightarrow x$ from node y. If this tree, $\mathcal{T}_{dj}$ is available, then it is used at node x. Now, the only possibility for the packet to attempt re-use of the failed link is by attempting to use $\mathcal{T}_{dj}$ from node y. This cannot happen since $\mathcal{T}_{dj}$ will be used until (1) the destination is reached (or) (2) another failure is encountered. In case (1), since node x is upstream of node y on $\mathcal{T}_{dj}$, we are guaranteed that node y is not present on the path from node x to the destination d. In case (2), f_j will be set to 1 and hence $\mathcal{T}_{dj}$ will never be used again. Thus, in either scenario we are guaranteed that a packet will not attempt to cross a failed link $x - y$ more than once. ■

Theorem 1: IP fast reroute over k d-rooted ADSTs recovers from any arbitrary set of $i \leq (k-1)$ failed links.

Proof: Consider an arbitrary set of $i \leq (k-1)$ failed links. Assume that a packet is sourced at some node $s \neq d$. By Lemma 1, the packet visits any given failed link at most once. Hence, the packet could have seen at most i link failures. Let the link failures that it has visited be in the order $l_1, l_2, ..., l_j$; $j \leq i$.

In the routing procedure in Figure 2, each visit to a failed link rules out exactly one of the k d-rooted ADSTs. Therefore, after visiting $j \leq i$ failed links, at least $(k-i)$ d-rooted ADSTs remain available. Since i is at most $(k-1)$, it is guaranteed that there is at least one tree on which d can be reached. ■

C. Complexity

The computational complexity of the construction of the arc-disjoint spanning trees dominates the running time of our proposed re-routing mechanism. Several algorithms have been developed for constructing ADSTs since Edmonds' theorem [21] was first published. The recent algorithm due to Bhalgat et. al [22] has the best known time complexity of $O(k^3|\mathcal{N}|\log^2|\mathcal{N}| + |\mathcal{L}|)$, where k is the edge-connectivity of the network.

IV. Performance Evaluation

We perform our evaluations on the largest eight topologies of the Rocketfuel[3] Internet mapping project [25]. We evaluate the performance of our proposed re-routing procedure, ADST, by comparing it to the approach employing independent trees (IT) [13] and to FCP [5]. We do not consider the link-disjoint spanning trees approach due to its high network connectivity requirement. Even for recovering from single link failures, the network needs to have two link-disjoint spanning trees. Most Rocketfuel topologies cannot be decomposed into two link-disjoint spanning trees.

The IT approach guarantees recovery from up to two simultaneous link failures. For both ADST and IT, we study three different routing alternatives explained below. The *first* alternative is based on the assumption that the default routing under no failures is on a *Shortest Path First* tree (SPF) rooted at the destination. It may be desirable to use SPF in order to retain the shortest path behavior of legacy link-state protocols, such as OSPF, under failure-free conditions. The *second* alternative is motivated by the following observation. Since we already employ k spanning trees to recover from link failures, it is *possible* that one or more of these trees contains the shortest path. Thus, we may be able to route our traffic on the shortest path in failure free scenarios, without using a separate shortest path tree. This alternative, where the shortest tree among the k trees is used when selecting trees, is referred to as *Shortest Tree First* (STF). Note that SPF will behave as STF after the first failure. Figure 5 shows the average source-destination path length in the Rocketfuel networks using the SPF and STF approaches, when the network does not have any failures. Observe that STF's average path length is very close to that of SPF. Finally, the *third* alternative is motivated by a potential disadvantage of the SPF and STF. Under SPF and STF, a node is forced to always take the same output link for a given destination. For the purpose of load balancing the traffic, it may be desirable, whenever possible, to select a random tree. We call this alternative *Any Tree First* (ATF).

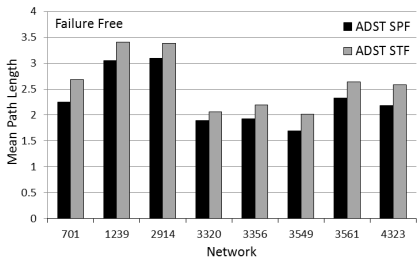


Fig. 5. Mean path lengths of the shortest ADST trees compared to the shortest paths. In failure-free scenarios the SPF paths are the shortest paths.

Hence, we have a total of (ADST, IT) $\times$ (SPF, STF, ATF); six combinations that we wish to evaluate. For both ADST as well as IT, re-routing upon link failure is still first dictated by the tree on which the reverse arc is present. If the reverse arc is either (1) not present on any of the k trees or (2) is present on a tree that has already been used up by a previous link failure, then we have to choose one of the available trees as the next tree. Only in these scenarios, we select the new tree based on the respective alternative (SPF, STF or ATF) we wish to evaluate.

As discussed earlier, FCP [5] involves reactive path computations and considerably larger packet header overhead. However, since it computes the shortest path toward the destination from every point of failure, FCP provides a good baseline to evaluate path stretch. Hence, we use the FCP mechanism as

[3]In this section, we refer to the networks by their autonomous system number as defined in the Rocketfuel project. The exact network topologies are available at [24].

the baseline in all of our performance evaluations.

A. Performance metrics

We define *path stretch*, denoted by ρ, as the ratio of the path length obtained using either the ADST or IT approach to that of the path length obtained using FCP. Formally, we can express path stretch as follows. Let ψ denote a failure scenario in the network. Let $P_{sd\psi}^{ADST}$, $P_{sd\psi}^{IT}$, and $P_{sd\psi}^{FCP}$ denote the path length from s to d under ADST, IT, and FCP. The path stretch from s to d for the ADST and IT approaches is computed as:

$$\rho_{sd\psi}^{ADST} = \frac{P_{sd\psi}^{ADST}}{P_{sd\psi}^{FCP}} \qquad \rho_{sd\psi}^{IT} = \frac{P_{sd\psi}^{IT}}{P_{sd\psi}^{FCP}} \tag{1}$$

We compute the mean path stretch for a destination node d, and for a set of failure scenarios Ψ as:

$$\rho_{d}^{ADST} = \frac{1}{(|\mathcal{N}|-1)\,|\Psi|} \sum_{\psi\in\Psi} \sum_{s\in\mathcal{N}, s\neq d} \rho_{sd\psi}^{ADST} \tag{2}$$

$$\rho_{d}^{IT} = \frac{1}{(|\mathcal{N}|-1)\,|\Psi|} \sum_{\psi\in\Psi} \sum_{s\in\mathcal{N}, s\neq d} \rho_{sd\psi}^{IT} \tag{3}$$

We consider all arbitrary one or two link failures. Correspondingly, Ψ denotes the set of all one or two link failures. Note that in computing recovery path statistics, we consider only those failure scenarios that affect the default routing path. For example, if a failure scenario does not affect the default path, then it is not included when computing the mean path stretch. Thus, the path stretch factor compares path lengths only for failure scenarios in which the packets were rerouted.

B. Performance results

We first obtain the path stretch factor with respect to FCP for both ADST and IT recovery scenarios. To evaluate the general performance of ADST and IT recovery, we tally up path stretch values for all one and two link failure scenarios that affect the default routing path. Here we consider the three-link-connected versions of the eight Rocketfuel networks. Figure 6 shows the CDF of path stretch, for both ADST and IT, in network 3549. The CDFs are for two link failure recovery scenarios under ATF, STF and SPF. As the three figures show, ADST routing consistently outperforms IT in terms of path stretch. For instance, in Figure 6(c) under SPF, in more than 90% of the failure scenarios ADST's path stretch is 1.4 or less. In contrast, IT's path stretch is less than 1.8 in fewer than 60% of the failure scenarios. The figure shows as well that, for all failure scenarios, ADST's path stretch under SPF was less than 2. The path stretch CDFs for ADST and IT are qualitatively identical in the other networks and under both single and two link failures. We have observed that under ADST SPF routing, the path stretch in 90% or more of the failure scenarios was consistently less than 2 in all of the eight topologies.

In addition, we have observed SPF's and ATF's path stretch performance to improve in larger networks, which are more densely connected, even though the global link connectivity remains unchanged. Figure 7 compares the CDFs of path stretch in two different networks under ADST using all three routing alternatives. The CDFs to the left are of path stretch in the Rocketfuel 1239 network under two link failures. To the right we show the CDF of path stretch in network 3549 under two link failures as well. These two networks are topologically different in that network 1239 has the least number of nodes and links of all eight networks (33 nodes and 71 links), and 3549 having the largest number of links, 479, connecting a total of 51 nodes. As the figures demonstrate, the path stretch performance has improved in the larger network under ATF and SPF. In particular, when randomly choosing a tree to route the traffic, as is done in ATF, it is more likely to reach the destination faster in more densely connected networks. In the case of SPF, the availability of additional links may allow for shorter paths, particularly in the initial failure free routing phase. Note that this improvement occurs despite the fact that global network link connectivity has remained the same.

Mean path lengths are presented in Tables I and II for failure scenarios involving one and two link failures, respectively. In these tables all the networks are 3-link-connected. As predicted, under all failure scenarios, the mean path length of the SPF approach is the best, followed by STF and then ATF. The difference in the path lengths between the SPF and STF alternatives using ADSTs is marginal for all scenarios. Thus a shortest path tree for default routing may not be necessary.

In all networks, STF and SPF mean path lengths increase as the number of failed links increases from one to two. This is not always true for ATF, as is case for networks 3356 and 3561. This counter-intuitive result may be explained as follows. ATF randomly selects the first tree. It, as well, randomly selects the recovery tree upon encountering a failure, and when the reverse arc tree is not available. Thus routing on more than one tree will on average lead to shorter paths.

To improve our understanding of ADST recovery, we next evaluate path stretch performance in versions of the networks with higher global link connectivity.

In k-link-connected (k-LC) networks, with $k \geq 4$, it is possible to construct three or more arc-disjoint spanning trees for every destination. As link connectivity increases and additional trees are constructed, more path options for reaching the destination node become available. Figures 8 and 9 show the CDF of path stretch for respectively one link failure in the 1239 network and for two link failures in the 3549 network. Under ATF, path stretch increases as more trees are used in the 1239 network. This is true across all networks with lower link density. This behavior is not observed in higher link density networks, as demonstrated by network 3549 in Figure 9(a). Under both SPF and STF, increasing the number of trees in higher link connectivity networks has no significant effect on path stretch.

V. Conclusion

In this work, we developed a mechanism for guaranteed recovery from up to $(k-1)$ simultaneous link failures in arbitrary k-edge-connected networks. We demonstrated how k arc-disjoint spanning trees, rooted at every destination, can be used to recover from up to $(k-1)$ simultaneous link

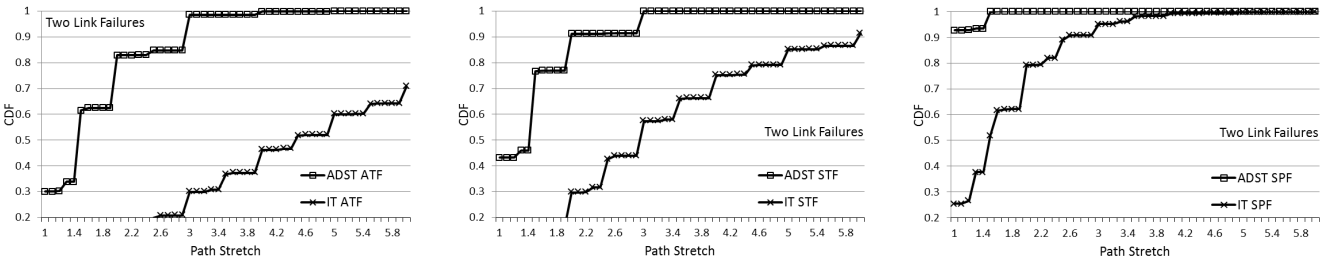


Fig. 6. CDF of path stretch with respect to FCP recovery of *failure scenarios that affect the default routing paths* under (a) ATF, (b) STF, and (c) SPF, for both ADST and IT in the 3549 *3-link-connected* network. This network is the largest in terms of number of links among the eight Rocketfuel topologies we utilize in this study.

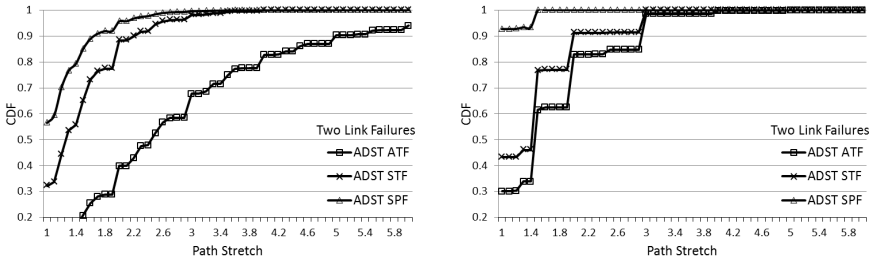


Fig. 7. CDFs of path stretch for *failure scenarios that affect default routing paths* in the (a) 1239 and (b) 3549 *3-link-connected* networks for ADST.

TABLE I
MEAN PATH LENGTH FOR SINGLE LINK FAILURE SCENARIOS

	701			**1239**			**2914**			**3320**		
	ATF	**STF**	**SPF**	**ATF**	**STF**	**SPF**	**ATF**	**STF**	**SPF**	**ATF**	**STF**	**SPF**
ADST	6.2894	4.0416	3.7258	6.7554	5.2036	4.9692	8.2247	5.7796	5.5703	2.9146	2.7313	2.6101
IT	9.5769	7.0288	4.9405	8.2319	6.4941	5.6787	9.6137	7.9300	6.4777	7.3283	5.1829	3.9332
FCP		3.1988			4.0542			4.2959			2.5514	
	3356			**3549**			**3561**			**4323**		
ADST	3.2247	3.0144	2.8788	2.8952	2.7200	2.5581	6.0279	4.0530	3.8279	4.7510	3.6597	3.4232
IT	6.8942	5.2772	4.0627	8.1788	5.8069	4.2248	9.5074	6.9941	5.1704	7.5136	5.7817	4.4489
FCP		2.7066			2.4753			3.2887			3.0881	

TABLE II
MEAN PATH LENGTH FOR TWO LINK FAILURE SCENARIOS

	701			**1239**			**2914**			**3320**		
	ATF	**STF**	**SPF**	**ATF**	**STF**	**SPF**	**ATF**	**STF**	**SPF**	**ATF**	**STF**	**SPF**
ADST	7.2719	4.0670	3.7549	7.5714	5.3009	5.0707	9.2352	5.8874	5.6784	2.9243	2.7345	2.6134
IT	9.6128	7.1098	5.0156	8.3340	6.6834	5.8651	9.7053	8.1081	6.6539	7.3243	5.2124	3.9593
FCP		3.2082			4.1007			4.3307			2.5539	
	3356			**3549**			**3561**			**4323**		
ADST	3.2233	3.0185	2.8835	2.8733	2.7221	2.5604	6.0277	4.0664	3.8425	5.3985	3.6767	3.4414
IT	6.8982	5.3052	4.0890	8.1902	5.8316	4.2463	9.5103	7.0453	5.2173	7.5154	5.8444	4.5103
FCP		2.7097			2.4773			3.2939			3.0984	

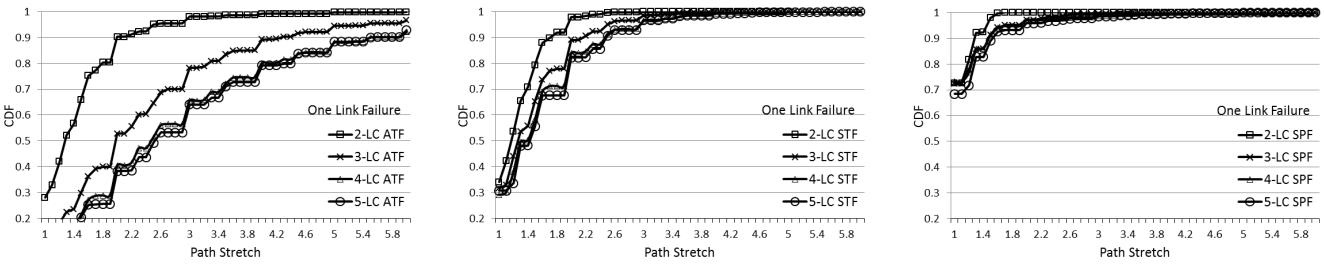
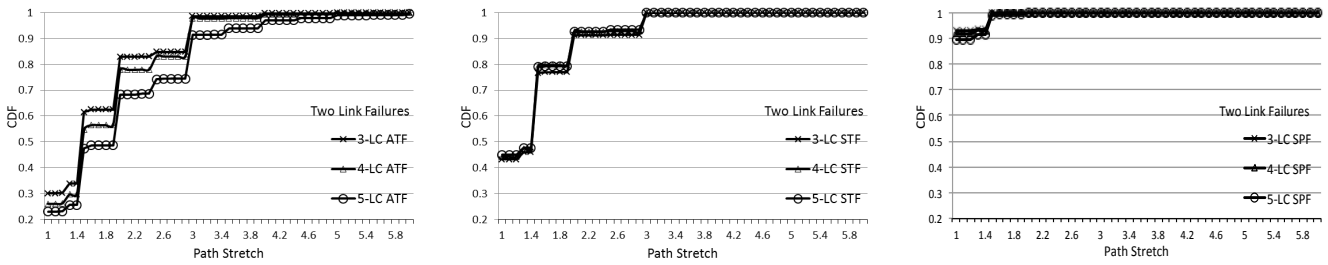


Fig. 8. CDF of ADST's path stretch with respect to FCP recovery in *2-LC, 3-LC, 4-LC and 5-LC* versions of the 1239 network for *one link failure*.



(a) (b) (c)

Fig. 9. CDF of ADST's path stretch with respect to FCP recovery in *3-LC, 4-LC and 5-LC* versions of the 3549 network for *two link failures*.

failures. In particular, we showed that using the incoming arc information and k bits in the packet header, it is always possible to reach the destination using our proposed recovery mechanism. Moreover, with one extra bit in the header, it is possible to use our recovery method in conjunction with any default unicast routing, such as the shortest path. We then evaluated our approach and compared it to a similar method that utilizes link-independent spanning trees. When employed with shortest path routing, recovery over ADSTs outperforms recovery over ITs in terms of path length. Furthermore, our mechanism exhibits low path stretch factor with respect to the previously proposed, path length optimal, albeit impractical, failure-carrying packets mechanism.

ACKNOWLEDGMENT

The research work presented in this paper is supported by the National Science Foundation under grant CNS-1117274.

REFERENCES

[1] "Integrated OTN switching virtualizes optical networks," White Paper, Infonetics Research, Inc, Jun. 2012.

[2] R. Asthana, Y. N. Singh, and W. D. Grover, "p-cycles: An overview," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 1, pp. 97–111, 2010.

[3] W. D. Grover, *Mesh-based Survivable Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking*. Upper Saddle River, NJ: Prentice Hall PTR, 2004.

[4] G. Bernstein, B. Rajagopalan, and D. Saha, *Optical Network Control: Architecture, Protocols, and Standards*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[5] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving convergence-free routing using failure-carrying packets," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2007, pp. 241–252.

[6] G. Rétvári, J. Tapolcai, G. Enyedi, and A. Császár, "IP Fast ReRoute: Loop Free Alternates revisited," in *Proceedings of IEEE INFOCOM*, 2011, pp. 2948–2956.

[7] M. Shand, S. Bryant, and S. Previdi, "IP fast reroute using not-via addresses," Internet Draft, Mar. 2010, draft-ietf-rtgwg-ipfrr-notvia-addresses-05.

[8] K. Xi, H. Chao, and C. Guo, "Recovery from shared risk link group failures using IP fast reroute," in *Proceedings of 19th International Conference on Computer Communications and Networks (ICCCN)*, 2010.

[9] S. Cho, T. Elhourani, and S. Ramasubramanian, "Independent directed acyclic graphs for resilient multipath routing," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 153–162, Feb. 2012.

[10] T. Elhourani, S. Ramasubramanian, and A. Kvalbein, "Enhancing shortest path routing for resilience and load balancing," in *Proceedings of IEEE ICC*, 2011.

[11] K. Xi and J. Chao, "IP fast rerouting for single-link/node failure recovery," in *Proceedings of BROADNETS - Internet Technologies Symposium*, Sep 2007, pp. 142–151.

[12] S. Kini, S. Ramasubramanian, A. Kvalbein, and A. Hansen, "Fast recovery from dual link failures in IP networks using tunneling," *IEEE/ACM Transactions on Networking*, vol. 18, no. 6, pp. 1988–1999, Dec. 2010.

[13] A. Gopalan and S. Ramasubramanian, "Multipath routing and dual link failure recovery in IP networks using three link-independent trees," in *Proceedings of the Advanced Networks and Telecommunication Systems (ANTS) Conference*, Bengaluru, India, Dec 2011.

[14] J. Roskind and R. E. Tarjan, "A note on finding minimum-cost edge-disjoint spanning trees," *Mathematics of Operations Research*, vol. 10, no. 4, pp. 701–708, 1985.

[15] C. S. A. Nash-Williams, "Edge-disjoint spanning trees of finite graphs," *Journal of the London Mathematical Society*, vol. s1-36, no. 1, pp. 445–450, 1961.

[16] A. Zehavi and A. Itai, "Three tree-paths," *Journal of Graph Theory*, vol. 13, no. 2, pp. 175–188, 1989.

[17] A. Gopalan and S. Ramasubramanian, "A counterexample for the proof of implication conjecture on independent spanning trees," *Elsevier Information Processing Letters*, vol. 113, no. 14 - 16, pp. 522–526, Jul-Aug 2013, http://srini.ca/p/3trees-counterexample.pdf.

[18] ——, "On constructing three edge independent spanning trees," *Technical Report, The University of Arizona*, 2011. [Online]. Available: http://srini.ca/p/3trees.pdf

[19] J. Cheriyan and S. N. Maheshwari, "Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs," *Journal of Algorithms*, vol. 9, no. 4, pp. 507–537, 1988.

[20] S. Curran, O. Lee, and X. Yu, "Finding four independent trees," *SIAM Journal on Computing*, vol. 35, no. 5, pp. 1023–1058, 2006.

[21] J. Edmonds, "Optimum branchings," *Journal of Research of the National Bureau of Standards*, vol. 71B, no. 4, pp. 233–240, 1967.

[22] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi, "Fast edge splitting and edmonds' arborescence construction for unweighted graphs," in *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '08. Society for Industrial and Applied Mathematics, 2008, pp. 455–464.

[23] T. Erlebach and A. Mereu, "Path splicing with guaranteed fault tolerance," in *Proceedings of the 28th IEEE conference on Global telecommunications Series (GLOBECOM)*, 2009, pp. 623–628.

[24] (2012, Jan.) Network topologies. [Online]. Available: https://github.com/network-research/topologies.git

[25] N. Spring, R. Mahajan, and T. Anderson, "The causes of path inflation," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2003, pp. 113–124.