Path Splicing with Guaranteed Fault Tolerance

Thomas Erlebach
Department of Computer Science
University of Leicester
University Road, Leicester LE1 7RH, UK
Email: t.erlebach@mcs.le.ac.uk

Anna Mereu
Department of Electrical and Electronic Engineering
University of Cagliari
Piazza d'Armi, 09123 Cagliari, Italy
Email: anna.mereu@diee.unica.it

***Abstract*—This paper addresses the problem of exploring the fault tolerance potential of the routing primitive called path splicing. This routing mechanism has been recently introduced in order to improve the reliability level of networks. The idea is to provide for each destination node in a network several different routing trees, called slices, by running different routing protocols simultaneously. The possibility for the traffic to switch between different slices at any hop on the way to the destination makes it possible to achieve a level of reliability that is close to the ideal level achieved by the underlying network. In this work we show that there is a method for computing just two slices that achieves fault tolerance against all single-link failures that do not disconnect the underlying network. We present an experimental evaluation of our approach, showing that for a number of realistic topologies our method of computing the slices achieves the same level of fault tolerance that is achieved by a much larger number of slices using the previously proposed method.**

***Index Terms*—Path Splicing, Reliability, Fault Tolerance.**

I. Introduction

The existence of multiple paths between two nodes can be exploited to improve the reliability of the network. One problem using traditional approaches is that in order to achieve good fault tolerance, either a large number of backup paths must be precomputed, or the time the network takes to converge to new routes after a link failure is rather long. A new routing primitive called *path splicing* has recently been introduced by Motiwala et al. [1], [2] to address these issues. The main idea is to compute several different routing trees, called *slices*, for each destination by running different routing protocols simultaneously. Furthermore, packets can switch between different slices at any node on the way to the destination, based on a sequence of bits in an additional splicing header of the packet.

The main advantages of path splicing are that a large number of different backup paths can be obtained by using only a small number of slices, and that endnodes have some control over the slices used by the packets they are sending. When a traffic source notices that the current path to a destination is no longer operational, it can randomly determine a new set of splicing bits and has a good chance of quickly discovering a new usable path to the destination.

Motiwala et al. [2] proposed to obtain each slice by running a shortest-paths based routing protocol such as OSPF, after randomly perturbing the link weights for each slice. Using simulation experiments they show that this approach works well for two examples of realistic network topologies, in the sense that the fault tolerance achieved using path splicing approaches the ideal fault tolerance of the underlying network already with a relatively small number of slices.

In this paper, we explore the potential for increasing the fault tolerance achieved with path splicing by varying the way in which the slices are computed. In particular, we prove that two slices are already enough to achieve fault tolerance against all single-link failures that do not disconnect the underlying network. The previously proposed method of computing slices by randomly perturbing link weights cannot provide any such guarantee.

In our method, the computation of the two slices is derived from ear decompositions of all biconnected components of the underlying network. Furthermore, we show that our approach can be generalized to k-edge-connected networks in the sense that k slices are sufficient to guarantee fault tolerance against any set of $k-1$ arbitrary link failures.

We present an experimental evaluation of our approach showing that for different realistic topologies, just two slices computed according to our method are sufficient to achieve a level of fault-tolerance that would require a much larger number of slices using the method proposed in [2].

The remainder of this paper is organized as follows. Section II discusses related work. Preliminaries are presented in Section III. Our proposed method of computing two slices based on an ear decomposition of the underlying network is presented in Section IV. Experimental results obtained with this method are discussed in Section V. The generalization of the method to k-edge-connected networks is discussed in Section VI. Alternative evaluation metrics and future work are discussed in Section VII and we conclude in Section VIII.

II. Previous Work

The approaches that have been traditionally adopted to address the problem of improving the reliability of a network can be mostly described as multipath routing approaches. They rely on the presence of multiple paths to a destination, i.e., when an event of failure occurs in the network, the traffic can be routed through precomputed backup paths. The main drawback of this approach is related to scalability: For large networks, it is very expensive to compute backup paths for

Thomas Erlebach would like to acknowledge that part of this work was done during a study leave granted by University of Leicester.

more than a small number of traffic demands. Moreover, multipath routing is often performed by computing k edge-disjoint paths, for a suitable value of k, for an origin-destination pair. If a link fails on each of the k paths, the node pair will be disconnected even if the network topology is still connected. On the other hand, by using the path splicing technique with k slices, it is possible to ensure that an origin-destination pair gets disconnected only if k links fail in the same cut. Such concentrated failures are unlikely to occur, as far as IP backbones are concerned, because there the main failure scenarios arise from single link failures, as shown in [3].

The concept of path splicing was introduced by Motiwala et al. [2]. The idea is to provide the network alternative paths besides the original shortest path in order to avoid a bottleneck scenario along the low-cost path. Splicing bits are added to the header of a packet when it is sent into the network, and routers inspect the splicing bits to determine which of the available slices is used to forward the packet. Motiwala et al. propose to generate different paths for different slices by modifying the link weights using random link-weight perturbations. The new routing trees to each destination are then found by computing shortest paths with the modified weights.

It is suggested in [2] that the link perturbation is performed by means of a linear function of the original weights in order to obtain new shortest paths whose length is comparable to the one of the original shortest paths. Their proposed link perturbation function is the following:

$$L'(i,j) = L(i,j) + Weight(a,b,i,j) * Random(0, L(i,j)), \quad (1)$$

where $L(i,j)$ is the original weight of the link connecting node i and node j; $Weight(a,b,i,j)$ is a weight function with parameters a and b that takes into account properties of the nodes i and j; and $Random(0, L(i,j))$ is a random value between 0 and $L(i,j)$. The concrete weight function chosen in [2] is a degree-based perturbation function that depends linearly on the degrees of the nodes that are the endpoints of the link, and whose values are in the range from a to b:

$$Weight(a,b,i,j) = f_{ab}(degree(i) + degree(j)) \quad (2)$$

where $degree(i)$ and $degree(j)$ are the degree of node i and j, respectively. The reason for using a degree-based perturbation technique is to discourage the use of links between nodes with large degree in order to introduce more path diversity.

III. Preliminaries

We model the communication network as a connected, undirected graph $G = (V, E)$. Let $\bar{G}$ denote the bidirected version of G, i.e., the directed graph obtained from G by replacing each edge $e \in E$ by a pair of directed edges with opposite directions. A *slice* is given by specifying, for each node t of G, a directed in-tree in $\bar{G}$ rooted at t. A slice specifies for each node w of G and each destination node t of G an edge (w,u) of $\bar{G}$, the interpretation being that if a packet with destination t is received by w, it gets forwarded by w to u.

If a connected, undirected graph contains a node whose removal disconnects the graph, such a node is called a *cut node*. An undirected graph is called *biconnected* if the deletion of any single node leaves the remaining graph connected. For a given graph G, any maximal biconnected subgraph with at least three nodes is called a *biconnected component* or *block*. An edge whose deletion disconnects the graph is called a *bridge*. The blocks and bridges of a connected graph form a tree structure. Formally, this tree structure is a bipartite graph whose vertices are the blocks and bridges of G and the cut nodes of G, and with an edge between a block or bridge and a cut node if that cut node is contained in the block or bridge.

Every biconnected graph G possesses an *ear decomposition*, i.e., the edge set of the graph can be partitioned into a simple cycle C, called the *base cycle*, and a number of paths P_1, P_2, ..., P_t, such that each path P_i, $1 \leq i \leq t$, has its endpoints in the subgraph $C \cup P_1 \cup \cdots \cup P_{i-1}$ and its internal nodes outside that subgraph. We refer to i as the *index* of the ear P_i. An ear decomposition of a biconnected graph can be computed in linear time (see, e.g., [4]). For every edge e of a biconnected graph, there is an ear decomposition with a base cycle that contains e.

An undirected graph G is *k-edge-connected* if it contains k edge-disjoint paths between every pair of nodes; equivalently, G is k-edge-connected if it remains connected after $k-1$ arbitrary edge deletions.

IV. Two Slices: Ear Decomposition

In this section we present a method for computing two slices such that the following strong property is satisfied for every pair of nodes v, w of the network: it is guaranteed that if an arbitrary link fails in each biconnected component on the way from v to w, the union of the in-trees with root w from the two slices still contains at least one path from v to w. This means that a packet with destination w that is sent out by v can still reach w if the splicing bits in the packet header are set appropriately. (In practice, the splicing bits can be set randomly until a feasible path to the destination is found.)

Our method is based on the concept of ear decompositions. We describe how to compute the in-trees for the two slices for one specific destination t. First, we compute the biconnected components of the graph and an ear decomposition of each biconnected component according to the algorithm described in [4]. Here, we choose the ear decomposition in such a way that in the biconnected component containing t, the base cycle of the ear decomposition contains t, and in every other biconnected component, the base cycle contains the cut node of that component that separates the component from t.

We specify for each node $v \in V \setminus \{t\}$ two outgoing edges, one for the in-tree with root t in the first slice and one for the in-tree with root t in the second slice. Consider any biconnected component. Let w be t, if t is in the component, or the cut node that separates the component from t, otherwise. The two outgoing edges for each node in the biconnected component are determined as follows:

- Let w, v_1, v_2, ..., v_ℓ, w be the base cycle C of the ear decomposition of this biconnected component. For the first slice, we pick the directed edges (v_1, w) and

(v_{i+1}, v_i) for $1 \leq i \leq \ell - 1$. For the second slice, we pick the directed edges (v_i, v_{i+1}) for $1 \leq i \leq \ell - 1$ and the edge (v_ℓ, w).

- Let v_1, v_2, …, v_ℓ be an ear. For the first slice, we pick the directed edges (v_i, v_{i+1}) for $2 \leq i \leq \ell - 1$. For the second slice, we pick the directed edges (v_i, v_{i-1}) for $2 \leq i \leq \ell - 1$.

Now consider any bridge $e = \{u, v\}$. If v is the endpoint of e that is closer to t, then we pick the directed edge (u, v) for both slices.

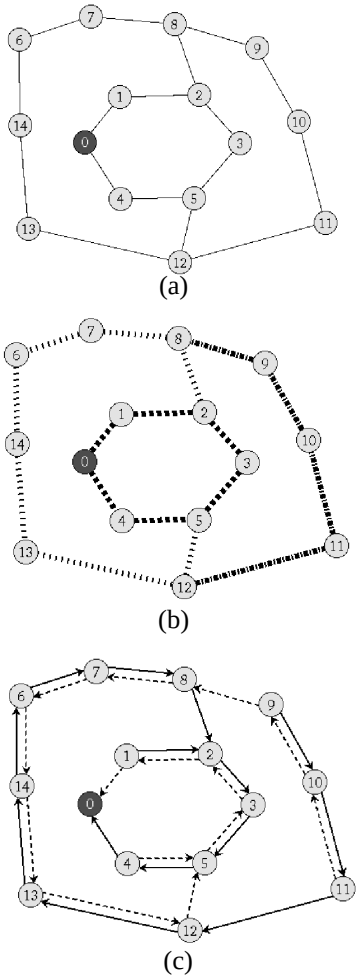

Fig. 1. (a) A biconnected network. (b) An ear decomposition. (c) The in-trees for destination 0 in the two slices.

Two examples illustrating the algorithm are shown in Figures 1 and 2. Figure 1 shows a biconnected graph, an ear decomposition, and the two resulting slices. Figure 2 shows a graph consisting of several biconnected components and one bridge, ear decompositions of all its biconnected components, and the two resulting slices.

Lemma 1: In each of the two slices, the directed edges selected by the algorithm for destination t form an in-tree with root t.

Proof: It is clear that for every destination node t, our algorithm selects one (and only one) out-edge for every node

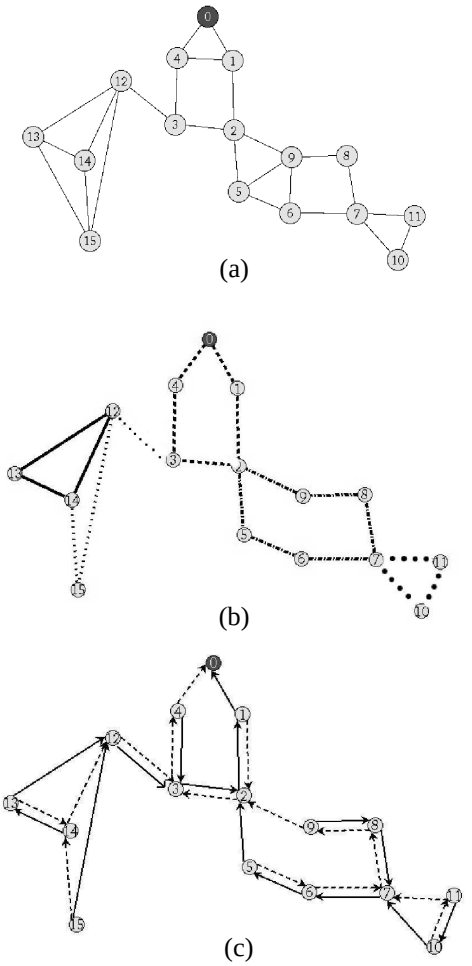

Fig. 2. (a) A network with four biconnected components and one bridge. (b) Ear decompositions of all biconnected components (single-edge ears omitted). (c) The in-trees for destination 0 in the two slices.

different from t. To show that the directed edges selected by the algorithm form an in-tree with root t, we show that every other node has a path to t consisting of only such edges.

Consider an arbitrary source node s. Any path from s to t in the underlying graph G consists of segments of the following types:

- Travelling from some node u inside a biconnected component to the unique node v in the same biconnected component that is closest to t. We call u the entry node and v the exit node of the biconnected component.
- Traversing a bridge edge from some node u to some node v. We call u the entry node and v the exit node of the bridge.

Furthermore, the sequence of segments and their entry and exit nodes are uniquely determined by s and t; different paths differ only in how they reach the exit node of a biconnected component from the entry node.

Consider the directed edges selected by the algorithm for destination t in one of the two slices. For every bridge segment of the path from s to t, the algorithm indeed selects the directed edge from the entry node to the exit node of

the bridge. Consider any other segment of the path in a biconnected component B with entry node u and exit node v. The directed edges selected by the algorithm in B form a path from u to v that can be constructed as follows: start with u as current node. While the current node is not a node of the base cycle C of B, the current node must be an internal node of a unique ear P_i. The directed edges picked by the algorithm allow to route from the current node to an endpoint of that ear. That endpoint becomes the new current node; note that if it is an internal node of an ear, that ear must have index smaller than i (thus ensuring that the path cannot enter the same ear twice). If the current node is a node on the base cycle C, the directed edges picked by the algorithm contain either the clockwise or the counterclockwise path on C to v. ■

Theorem 1: The two slices computed by the algorithm are resilient to arbitrary simultaneous single-edge failures in all biconnected components of the given network. They are even resilient to arbitrary simultaneous single-edge failures in all ears and base cycles of the ear decompositions of all biconnected components.

Proof: Consider an arbitrary source s and destination t. As observed in the proof of Lemma 1, every path from s to t consists of segments that are either bridges or travel from a unique entry node u to a unique exit node v of some biconnected component B. We need to consider only the latter case. Assume that an arbitrary edge has failed in each ear and in the base cycle of B. We need to show that the union of the two in-trees computed by the algorithm for destination t contains a path from u to v that avoids all failed edges. That path can be constructed as follows: Start with u as current node. While the current node is not a node of the base cycle C of B, the current node must be an internal node of a unique ear P_i. The directed edges picked by the algorithm allow to route from the current node to one endpoint of that ear in the first slice, and to the other endpoint of that ear in the second slice. No matter which edge has failed, we can still route to one endpoint of the ear. That endpoint becomes the new current node. If the current node is a node on the base cycle C, the directed edges picked by the algorithm contain the clockwise path on C to v in one slice and the counterclockwise path on C to v in the other slice, and one of the two paths avoids the failed edge on C. ■

V. Experimental Results

This section shows the results obtained with an implementation of our slice construction method from the previous section. We have already shown in Theorem 1 that our method guarantees the reliability of the network in case of simultaneous single link failures in all biconnected components of the network. In the following we show that we obtain a level of reliability that is better than the one that can be achieved with the slice generation method of [2] also for the failure scenario considered in [2], where each link of the network is assumed to fail independently with a certain probability.

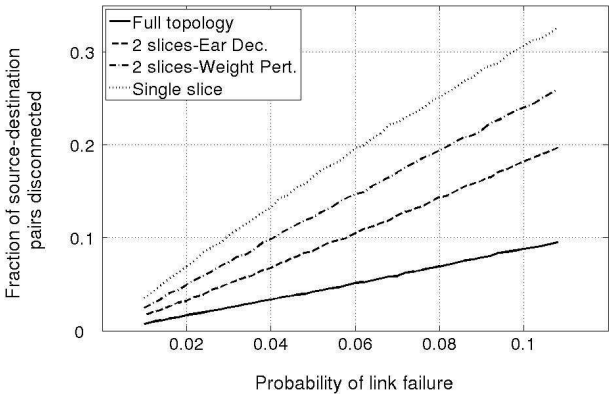

Fig. 3. Reliability curves for the Sprint network

In order to compare the performance of our method with that obtained by the random weight perturbation method, we have implemented the latter method and considered the same simulation setting as [2]. The parameter measured in the evaluation is the fraction of origin-destination pairs that are disconnected when each link of the network fails independently with a fixed probability p. Reliability curves have been produced by varying the failure probability of the links from 0.01 to 0.11 in steps of 0.002. The reliability achieved by path splicing with two slices is also compared to the ideal reliability achieved by the underlying network.

To analyze the reliability achieved for a network topology, each link is deleted independently with a fixed probability p, and then the fraction of disconnected node pairs is computed. This process is repeated 8,000 times for each value of the link failure probability, and the average values are plotted. With this procedure we have constructed the reliability curve for the underlying network, for the routing based on a single shortest-path tree for each destination (which is the most common approach in standard IP routing), for path splicing with two (or more) slices generated as proposed in [2], and for path splicing with two slices generated according to our new method.

The first network that we analyze is the Sprint backbone network topology. This topology has been inferred with the software Rocketfuel [5], and we consider the resulting PoP level topology: it is composed of 52 nodes and 84 links. This is the same network that is considered in the performance evaluation of [2]. The results are shown in Figure 3. In the figure, the solid line represents the reliability curve for the full topology, which represents the ideal curve that we would like to approach using as few slices as possible. The dotted curve is the one obtained for a single slice that is obtained by computing shortest paths according to the actual link weights. Between these two curves we have the two curves that represent path splicing with $k = 2$ slices. As can be seen from Figure 3, the dashed curve that represents the reliability of the network achieved with the two slices obtained with our method based on ear decompositions performs very well in comparison to path splicing with slices constructed using

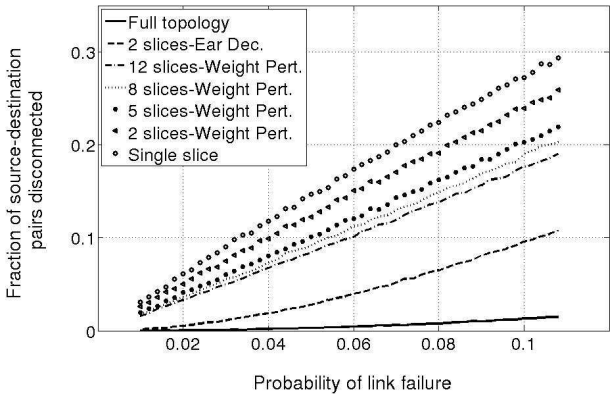

Fig. 4. Reliability curves for the Géant network

the random weight perturbation method of [2] (dash-dotted line). In particular, we can see that for $p = 0.1$ the fraction of disconnected pairs decreases by almost 25% using our method.

The second network that we analyze is the Géant Network, the European network that connects the national research networks from the different countries to each other. It is composed of 23 nodes and 38 links. The reliability curves for the Géant network are plotted in Figure 4. In this case we compare path splicing with two slices constructed from the ear decomposition to path splicing with k slices constructed using the random link weight perturbation method of [2], where k ranges from 2 to 12. It is demonstrated that our construction method allows a significant improvement not only if we compare it with $k = 2$ random weight perturbation slices but also if we compare it with up to $k = 12$ such slices. It is remarkable that for a link failure probability of 0.1, the fraction of disconnected pairs for just two slices built with the ear decomposition method is almost 44% lower than that for 12 slices computed with the random link weight perturbation method.

These results show that only two slices built with the ear decomposition method are sufficient to guarantee better fault tolerance even if compared to a considerably larger number of slices computed with the random link weight perturbation method. Comparing Figure 3 and Figure 4, we can infer that this behavior is particularly apparent for networks whose nodes have quite homogeneous node degrees, i.e., networks where the difference between the maximum and minimum node degree is small: For the Sprint network, this difference is equal to 26 (since the maximum degree is 28 and the minimum is 2), whereas for the Geánt network the maximum degree is 12 and the minimum degree is 4.

VI. k Slices: Edge-Disjoint In-Trees

If G is a biconnected graph, it is not difficult to see that the two in-trees that are constructed by our algorithm of Section IV for each destination node t are in fact edge-disjoint, in the sense that the set of directed edges of $\bar{G}$ that is used by one in-tree is disjoint from that used by the other in-tree. It is possible that one in-tree uses a directed edge (u, v) and the other uses a directed edge (v, u), however, so the undirected trees underlying the two in-trees are not edge-disjoint in general. In fact, it is easy to see that there are biconnected graphs (e.g., graphs consisting of a single cycle) that do not contain two edge-disjoint spanning trees.

In this section, we present a generalization of our approach to the construction of k slices in a k-edge-connected network for arbitrary values $k \geq 2$. A classical result of Edmonds [6] states that if a directed graph contains k edge-disjoint paths from some node r to every other node of the graph, then the graph contains k edge-disjoint out-trees (branchings) rooted at r. Furthermore, these k edge-disjoint out-trees can be computed efficiently in time $O(kmn + k^3n^2)$ for graphs with m edges and n nodes [7].

It is easy to see that if an undirected graph G is k-edge-connected, then $\bar{G}$ contains k edge-disjoint directed paths between each pair of nodes. Using this observation and the fact that the result by Edmonds can be adapted from out-trees to in-trees in a straightforward way, we have the following.

Proposition 1: Let G be an undirected graph. If G is k-edge-connected, then for every node t it holds that $\bar{G}$ contains k edge-disjoint in-trees rooted at t, and these in-trees can be computed efficiently.

A natural idea is then to compute k slices for a k-edge-connected network by choosing, for each destination t, k edge-disjoint in-trees with root t, one for each slice. We refer to slices computed in this way as *slices based on edge-disjoint in-trees*. We are interested in the level of fault tolerance that can be guaranteed by this method of computing slices. In particular, we would like to see whether the k slices computed in this way can tolerate $k - 1$ arbitrary edge failures of the underlying undirected graph. This is not obvious, because the failure of a single edge $\{u, v\}$ in the underlying undirected graph G corresponds to simultaneous failures of the two directed edges (u, v) and (v, u) in $\bar{G}$; $k - 1$ edge failures in G therefore correspond to $2(k - 1)$ failures of directed edges in $\bar{G}$. Nevertheless, we can prove that the desired level of fault tolerance is indeed achieved by this method of computing slices.

Theorem 2: Let $G = (V, E)$ be a k-edge-connected undirected graph. Path splicing with k slices based on edge-disjoint in-trees is resilient to $k - 1$ arbitrary edge failures.

Proof: Assume for a contradiction that there is a set F containing $k-1$ edges of G with the property that if all edges of F fail, then some node t is no longer reachable from some node s using the k slices based on edge-disjoint in-trees. Note that t is still reachable from s in the underlying graph G, because G is k-edge-connected. Let $\bar{F}$ be the $2k - 2$ directed edges of $\bar{G}$ that correspond to the undirected edges in F, i.e., for every edge $\{u, v\}$ in F, $\bar{F}$ contains the two directed edges (u, v) and (v, u).

Each of the k slices contains an in-tree rooted at t, and these k in-trees are edge-disjoint. Let D be the directed graph that is the union of these k in-trees. It is clear that D contains k edge-disjoint directed paths from s to t (namely the paths from s to t in the k edge-disjoint in-trees rooted at t). As the

failures of the edges in F disconnect t from s, we must have that $D \setminus \bar{F}$, the graph obtained from D by deleting all edges that are in $\bar{F}$, does not contain a directed path from s to t. This implies that there must be a cut (S, T) of D, where $S \subset V$ contains s and $T = V \setminus S$ contains t, such that $\bar{F}$ contains all edges with tail in S and head in T. Since D contains k edge-disjoint paths from s to t, we know that D contains at least k different directed edges with tail in S and head in T. Furthermore, no two of these directed edges correspond to the same undirected edge, because if (u, v) is an edge with tail in S and head in T, then (v, u) does not have this property. As the directed edges in $\bar{F}$ correspond to only $k - 1$ undirected edges, there must be one edge with tail in S and head in T that is not contained in $\bar{F}$, giving the desired contradiction. ■

VII. Discussion and Future Work

Our analysis has focussed on the reliability achieved by the computed set of slices. Other metrics that have been considered in [2] include stretch (i.e., how much longer the paths in the slices are compared to shortest paths) and recovery time (i.e., how long it takes the source node, after a path fails, to find a a set of splicing bits that yields a new path avoiding the failed links). It would be interesting to investigate these metrics for the slices constructed by our method as well.

The stretch metric is highly related to the concept of novelty (i.e., path diversity). In particular, stretch and novelty are conflicting goals. One wants to achieve that the alternative paths are not much longer than the original shortest paths (low stretch), and, at the same time, that the new paths exploit path diversity (high novelty). We expect that in our method the stretch of the paths is larger but the novelty is also larger than in the random link weight perturbation method. A feasible method to limit the length of the paths could be using normal shortest-path routing in the absence of failure scenarios and switching to path splicing with two slices computed by our method when link failures affect the shortest path.

As far as the recovery time is concerned, in [2] this is analyzed in terms of the number of recovery attempts before a working path is found. We expect the recovery performance with the implementation of our method to be very similar to that of the original splicing procedure since we modify the original approach only in the way the alternative paths are computed and we do not introduce any change in the recovery schemes.

A problematic scenario that can occur with path splicing is the possibility of forwarding loops. This is because traffic is not routed along a single routing tree. This scenario seems to be more likely to happen with our path splicing methodology. A simple method to eliminate forwarding loops would be to allow packets to switch slices only when they enter an ear or base cycle of the ear decomposition, but not while they traverse an ear or base cycle. Moreover, in [2] different strategies are suggested in order to avoid forwarding loops, such as the restriction of the number of switches between slices that the packets can perform, and the same methods should be effective in our case.

It could be interesting to adopt a combination of both methods (computing some slices using our method based on ear decompositions or edge-disjoint in-trees, and some slices using the random link weight perturbation method) in order to obtain a good compromise between the different metrics in practice.

Moreover, from a theoretical point of view, an interesting question could be: How can we compute, for a given graph, the minimum number of slices that is sufficient so that the reliability that can be achieved with path splicing is the same as the ideal reliability of the underlying network? For example, it is easy to see that one slice suffices if the network is a tree and two slices suffice if the network is a ring, but we do not yet know how to answer this question for arbitrary networks.

VIII. Conclusion

In this paper we have explored how the slices used for path splicing can be computed in such a way that the resilience to edge failures improves. We have presented a method based on ear decompositions that allows us to compute two slices that can tolerate arbitrary single-link failures that do not disconnect the underlying network; in fact, the slices can tolerate simultaneous single-link failures in all biconnected components. Furthermore, we have shown experimentally that in a failure scenario where each link fails independently with some probability p, our method of slice computation achieves with only two slices a level of reliability for which the previously proposed random link weight perturbation method requires a substantially larger number of slices. Finally, we have shown that our approach can be generalized to the setting of k slices in k-edge-connected networks by computing slices based on edge-disjoint in-trees.

We remark that ear decompositions can be computed efficiently in parallel [4], indicating that it would be feasible to incorporate our method of slice computation into a routing protocol that is executed in a distributed fashion by the nodes of the network.

Finally, future directions aiming at obtaining a good overall performance of the path splicing technology have been discussed.

References

[1] M. Motiwala, N. Feamster, and S. Vempala. Path splicing: Reliable connectivity with rapid recovery. In *Proc. 6th ACM Workshop on Hot Topics in Networks (Hotnets-VI)*, Atlanta, GA, November 2007.

[2] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path splicing. In *Proc. ACM SIGCOMM*, pages 27–38, Seattle, WA, 2008. ACM.

[3] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, and C. Diot. Characterization of failures in an IP backbone. In *Proc. INFOCOM 04*, Hong Kong, China,, March 2004.

[4] V. Ramachandran. Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity. In J.H. Reif, editor, *Synthesis of Parallel Algorithms*, pages 275–340. Morgan-Kaufmann, 1993.

[5] N.T. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. In *Proc. ACM SIGCOMM*, pages 133–145, Pittsburgh, PA, 2002.

[6] J. Edmonds. Edge-disjoint branchings. In R. Rustin, editor, *Combinatorial Algorithms*, pages 91–96. Algorithmics Press, 1972.

[7] P. Tong and E.L. Lawler. A faster algorithm for finding edge-disjoint branchings. *Information Processing Letters*, 17:73–76, 1983.