Path Splicing: Reliable Connectivity with Rapid Recovery

Murtaza Motiwala, Nick Feamster and Santosh Vempala
College of Computing, Georgia Tech

ABSTRACT

We present *path splicing*, a primitive that constructs network paths from multiple independent routing processes that run over a single network topology. The routing processes compute distinct routing trees using randomly perturbed link weights. A few additional bits in packet headers give end systems access to a large number of paths. By changing these bits, nodes can redirect traffic without detailed knowledge of network paths. Assembling paths by "splicing" segments can yield up to an exponential improvement in path diversity for only a linear increase in storage and message complexity. We present randomized approaches for slice construction and failure recovery that achieve near-optimal performance and are extremely simple to configure. Our evaluation of path splicing on realistic ISP topologies demonstrates a dramatic increase in reliability that approaches the *best possible* using only a small number of slices and for only a small increase in latency.

1. Introduction

The Internet should be highly available. It should be reliable and robust to failures and changes in network conditions. Unfortunately, an Internet that is "always on" in the face of fiber cuts, power outages, blackouts, equipment failure, and operator error remains elusive [5, 12]. In the past, scalability requirements have overshadowed concerns about availability and robustness. As the Internet is increasingly becoming a medium for interactive applications that demand high-availability, users and operators alike are feeling the repercussions of this design choice. Designing routing protocols that scale well and recover quickly from failures has proven difficult. In this paper, we present a new primitive that demonstrates a surprising result: it is possible to achieve scalability *and* high availability; encouragingly, this can be achieved by composing existing routing protocols.

Today's routing protocols provide reachability while scaling to a large number of destinations. Scalability has come at the expense of availability; these protocols do not make use of the available connectivity in the underlying network. Internet routing protocols are single-path and destination-based [14, 15, 19]. In these scenarios, even a single link failure can disrupt a large amount of traffic or even disconnect end systems entirely. Even many overlay networks measure path characteristics (*e.g.*, latency, loss, or both) and select a single *best* overlay path over which to route traffic [2, 10].

Exploiting the *path diversity* that exists in the underlying network could make the network more reliable and also improve capacity. A routing protocol that achieves better path diversity by exploiting *multiple* paths through the network is more robust to failures. Unfortunately, scalably providing a large amount of path diversity has proven difficult. Schemes for improving path diversity must scalably (1) *disseminate* information about multiple disjoint paths; and (2) provide end systems a mechanism to *select* paths along which to forward traffic. In this paper, we show how multi-path routing can not only be scalable, but also how it can achieve *reliability that approaches that of the underlying graph* itself. We also show how end hosts can scalably select these paths.

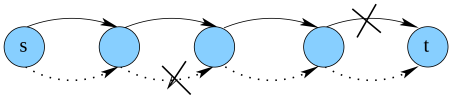


Figure 1: With two disjoint paths, *any* two failures, one on each path, disconnects s from t. With splicing, the failures must induce a graph cut to create a disconnection.

This paper presents **path splicing**, a primitive that composes route computations from existing routing protocols to (1) achieve an exponential improvement in path diversity for only a linear increase in routing complexity (in terms of routing table state, messages, etc.) and (2) provides lightweight mechanisms to allow end hosts to "deflect" traffic around faulty network elements. In path splicing, nodes run several instances of a routing protocol, but with different parameter settings (*e.g.*, multiple instances of shortest paths routing, each with different link weights). Traffic can be forwarded along *any combination of path segments* from each routing tree, and traffic can switch trees at any node.

Figure 1 shows a simple example that demonstrates the power of splicing; this simple network has two disjoint paths from s to t. *Any* two link failures with one on each path would disconnect s from t. When these two paths are spliced, disconnecting s from t requires that both links in some *cut* fail, which is a far less likely event. Path splicing thus potentially offers up to an exponential improvement in the number of available paths between each node pair.

Path splicing dramatically increases path diversity with minimal increase in path latency. Our preliminary evaluation shows that path splicing (1) achieves path diversity that approaches that of the underlying network; and (2) allows end systems and intermediate nodes to quickly recover from failures, even when used in conjunction with the most naïve recovery mechanisms. Path splicing is simple, scalable, and stable, requiring only static per-link configuration, and it does not require implementing any new routing protocol.

Although this paper uses path splicing to improve intradomain failure recovery, path splicing is a general technique for composing multiple routing protocol instances to increase the number of paths available to end systems. Thus, path splicing can apply both to other routing protocols (*e.g.*, BGP [19]) and to other settings and applications that can exploit access to multiple paths (*e.g.*, making better use of network capacity by simultaneously using many paths).

2. Design Goals

To provide high availability, a routing protocol must discover and disseminate information about a *diverse* set of paths between each pair of end systems. High diversity improves the likelihood that these hosts remain connected when links or nodes experience faults (*e.g.*, failures, congestion, packet loss). Second, the routing protocol must facilitate fast *recovery*; that is, when a failure occurs it must enable end systems to quickly discover at least one working path among the diverse set of available paths. This section discusses both diversity and recovery.

High Diversity. The routing protocol must expose a diverse set of paths between end hosts to ensure that each pair of nodes remains connected when underlying links in the graph experience faults. Many attempts to improve path diversity have operated without a clear definition of path diversity, although they have typically implicitly assumed an "operational" definition of masking path failures along paths between endpoints. We now formalize this notion.

Definition 2.1 (Reliability) *Given a graph $G = (V, E)$ and a probability of failure $p(e)$ on every edge $e \in E$, the* reliability *of the graph is the probability that upon failing each edge e independently with probability $p(e)$, the graph remains connected.*

Reliability reflects the probability that the graph becomes disconnected, given that links fail with certain probabilities. We define a related notion of a reliability curve.

Definition 2.2 (Reliability curve) *For a given graph G, and any $0 \leq p \leq 1$, let $R(p)$ denote the reliability (probability that the graph stays connected) when all edges fail independently with probability p. The reliability curve is the plot $y = R(x)$ where x ranges from 0 to 1.*

We are chiefly interested in a routing protocol's ability to best utilize the path diversity that exists in the underlying graph, *i.e.*, to minimize the *reliability shortfall*. Although end systems cannot achieve reliability that is greater than the underlying graph, the routing system should come as close as possible to achieving the reliability of the underlying network. Most routing protocols have a high reliability shortfall; in contrast, we show in Section 4 (both experimentally and theoretically) that simple path splicing configurations approach near-optimal reliability.

Fast Recovery. As part of achieving high reliability, the protocol must provide a way for end systems to *quickly discover* working backup paths when the primary path or paths fail. Thus, another important metric is the *recovery time*: how long it takes for end hosts to discover alternate working paths after a failure occurs. Recovery can be performed either by end systems or by intermediate nodes (*e.g.*, routers); in our evaluation, we investigate both recovery scenarios and show that path splicing performs well in both cases.

Small Stretch. For any pair of nodes (s, t), we also consider the ratio of the latency given by path splicing to the latency of the shortest path. We call this ratio the *path stretch* for the pair (s, t). We aim to keep this ratio small for every pair of nodes. This goal contrasts with previous work on overlay routing, path diversity, and traffic engineering algorithms [2, 7, 22], which do not consider stretch.

3. Path Splicing

This section describes *path splicing*. We first give an overview of the intuition behind path splicing and present a simple example. Figure 2 illustrates path splicing. The basic idea is simple: compute multiple shortest paths trees based on perturbations of the link weights in the original graph, and overlay these trees to create a graph over which traffic can be forwarded.

3.1 Splicing Control Plane

The path splicing control plane computes multiple routing trees based on perturbations of the underlying network topology and writes these trees to separate forwarding tables. The control plane runs multiple routing protocol instances, each with slightly different link weights.

3.1.1 *Link-weight perturbations*

To allow endpoints to discover paths other than shortest paths between any two nodes in the network, path splicing creates routing trees that are based on *random perturbations*. The perturbed link weights are based on the original weight of the link, which ensures that the length of the new shortest path is not very long compared with the original shortest path. The perturbed link weight $L'(i, j)$ is defined as:

$$L'(i,j) = L(i,j) + \text{Weight}(a, b, i, j) \cdot \text{Random}(0, L(i,j))$$

where $L(i, j)$ is the original weight of the link between node i and node j, Weight(a, b, i, j) is a function of some properties of nodes i and j (*e.g.*, the degrees of the nodes), a and b are constants, and Random$(0, L(i, j))$ is a random number chosen in the range of 0 to $L(i, j)$.

The nature of the perturbation can be changed by using different Weight$()$ and Random$()$ functions. In this work, we explored several Weight$()$ functions; although many possibilities exist, we found the following "degree-based" perturbation to work well in practice:

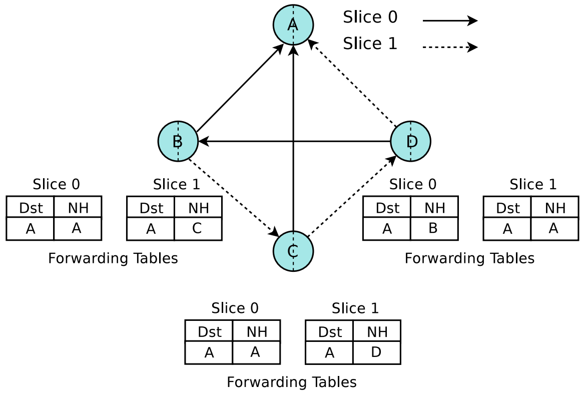
Degree-based perturbations The function $Weight(a, b, i, j)$ is a linear function of the sum of the degrees of i and j, *i.e.*

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

where f_{ab} is a linear function in $degree(i) + degree(j)$ ranging from a to b. In later sections we denote $Weight(a, b, i, j)$ as $Weight(a, b)$. The intuition behind these perturbations is that links connected to nodes with a high degree may be perturbed more than links connected to nodes with smaller degree, which reduces the likelihood of many shortest paths using the same link.

3.1.2 *Multiple routing protocol instances*

Path splicing constructs multiple shortest paths trees by running multiple routing protocol instances on the same underlying topology. Different instances of the routing proto-



Slice 0

Dst	NH
A	A

Slice 1

Dst	NH
A	C

Slice 0

Dst	NH
A	B

Slice 1

Dst	NH
A	A

Slice 0

Dst	NH
A	A

Slice 1

Dst	NH
A	D

Figure 2: Each node runs multiple routing protocol instances, each of which computes a shortest path tree. These trees are written into k ($k = 2$) forwarding tables, where each dst has a unique next hop node. The example shows two shortest paths trees rooted at A.

col compute routing trees to each destination; link-weight perturbations ensure that these routing trees are sufficiently non-overlapping. Each of these routing processes writes into a different forwarding table, as shown in Figure 2.[1] Our current design splices paths using routing trees that are computed from different configurations of the same protocol, but in theory each routing process could also run a different protocol. Cisco routers already supports for multi-topology routing [4], according to the recently standardized multiple router configuration functionality [17]. Multi-topology routing provides much of the control-plane function that would be needed to support path splicing in practice.

3.2 Splicing Data Plane

This section describes the mechanism by which slices are "spliced" together to form an end-to-end path. A sequence of bits in the packet header (placed between the network and transport layer headers)—*forwarding bits*—signal to routers which forwarding table should be used to forward traffic. Other work has proposed similar mechanisms [1, 22]. An attractive deployment scenario would involve end hosts inserting this header and changing bits; this scenario is appealing since end hosts are often the first to detect a poorly performing path. Alternatively, middleboxes, border routers, or overlay nodes could insert and manipulate these bits.

Two design features of the forwarding bits make path splicing both scalable and incrementally deployable. First, the bits are *opaque* and have no explicit semantics; end systems can change the path simply by changing the forwarding bits without needing details about the (very large number of) alternate paths. Second, routers that do not support path splicing simply forward data packets as they normally would, based on the destination IP address.

The forwarding bits need not provide explicit control over the sequence of hops in the forwarding path. The key property is that changing the bits should, with high probability, cause traffic to be diverted along a different path to the destination. Many encoding schemes have this property; a simple encoding scheme includes $\lg(k)$ bits for each network hop; each set of bits indicates which of k forwarding tables should be used to forward the traffic at that hop. As shown in Algorithm 1, each node along the path (1) reads the rightmost $\lg(k)$ bits from the splicing header to determine which of k forwarding tables to use for forwarding the packet; and (2) shifts the bitstream right by $\lg(k)$ bits to allow subsequent hops to perform the same operation. We describe in Section 5 how, for certain recovery schemes, this header can be significantly compressed.

Algorithm 1 Algorithm for forwarding packets

```
$k \Leftarrow$ number of slices
$dst \Leftarrow$ destination IP address
$src \Leftarrow$ source IP address
if fwdbits $> 0$ then
    slice $\Leftarrow$ fwdbits $\&$ $(2^k - 1)$
else
    slice $\Leftarrow Hash(src, dst)$
end if
fwdbits $>>$ $\lg(k)$
nexthop $\Leftarrow Lookup(dst, slice)$
forward packet to next-hop
```

Various strategies exist for recovery (*i.e.*, changing the forwarding bits); we show in Section 4 that even the most naïve scheme, randomly generating a new set of bits enables fast recovery. End systems could detect poorly performing paths resulting from many causes (*e.g.*, queueing, packet loss, etc.), while intermediate nodes may sometimes detect failures more quickly and could also initiate a switch to an alternate slice. Section 4 evaluates both approaches.

4. Preliminary Evaluation

This section presents the results from our preliminary evaluation. Table 1 summarizes the main results, which demonstrate, first, that splicing attains reliability that approaches that of the underlying graph; and second, that even simple schemes for selecting forwarding bits can provide very fast recovery. The rest of the section describes our experimental setup and each of these results in detail.

4.1 Experimental Setup and Method

To evaluate the reliability of path splicing under a variety of link-failure scenarios, we implemented a simulator that takes as input a "base" network topology (with link weights) and outputs the different shortest paths trees for that network using both uniform and degree-based perturbations. To simulate link failures, we remove each edge from the underlying graph with a fixed failure probability. We then determine whether a spliced path exists between each source-destination pairs in the resulting graph.

We used two "base" network topologies: (1) the GEANT backbone topology [8], which has 23 nodes and 37 links (typical for a medium-sized ISP); and (2) the Sprint backbone network topology inferred from Rocketfuel, which has 52 nodes and 84 links [20]. Due to space constraints, we present the results from the Sprint topology only.

4.2 Reliability Approaches Optimal

Path splicing increases the paths available between each pair of endpoints by running multiple versions of the routing

[1] An alternative to running separate processes would be for each node to compute the perturbed link weights in each slice based on a common pseudorandom function.

Result	*Location*
Reliability approaches optimal: The reliability achieved with random perturbations for ≤ 10 slices approaches the optimal that can be achieved by *any* routing algorithm.	§ 4.2, Fig. 3
Recovery is fast: An end host or intermediate node can typically recover in slightly more than two trials when recovering by selecting forwarding bits at random.	§ 4.3, Figs. 4, 5
Loops are rare: Using two slices, loops occur in only about 1% of all cases for path recovery. Simple modifications can reduce the likelihood of loops or eliminate them entirely.	§ 4.4

Table 1: Summary of preliminary results.

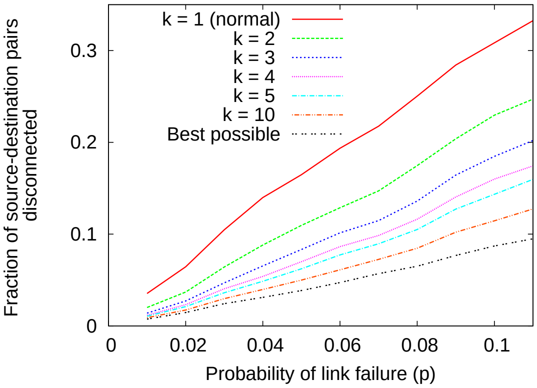


Figure 3: Reliability: Sprint topology using slices based on degree-based perturbations of link weights in the original topology.

protocol in parallel (specifically, without running a protocol that must compute an exponential number of paths). As the number of slices increases, path diversity grows exponentially, but the complexity of path splicing grows linearly (in terms of required routing state, convergence time, and the number of routing messages that are exchanged).

We computed the *reliability curves* (Definition 2.2, Section 2) for routing graphs generated using path splicing and compared this characteristic both to "conventional" shortest paths routing and to that of the original underlying graph, whose reliability reflects the *best possible* reliability that could be achieved by any routing protocol. We used the simulator to generate "spliced" graphs by taking the union of k link-perturbed shortest-path trees. The resulting graph is equivalent to what would result from writing k such trees into k forwarding tables. Each trial selects k such trees at random to generate a spliced graph. We plot the reliability curve for each resulting graph for various values of k and various perturbation strategies. The lower the fraction of pairs disconnected for a given failure probability, the more reliable the resulting graph.

To examine the effects of failure on connectivity, we remove each edge from the graph independently with a probability p. We start with $k = 1$, evaluate the reliability for the resulting graph, increase k to 2 (*i.e.*, add edges to the graph by taking the union of the two graphs) and evaluate the reliability of the resulting graph by failing the *same set of links* as we did for smaller values of k (simulating the effects of a link failure in the underlying network). We fail the same set of links for different values of k, so as to compare how adding slices improves reliability. We perform this process $1,000$ times and compute the average reliability for each value of k and failure probability.

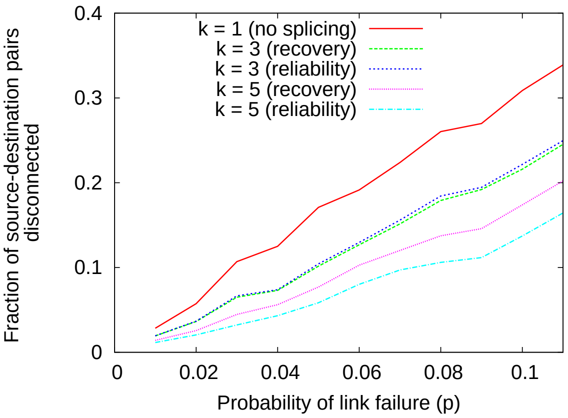


Figure 4: Recovery using *end-system recovery* and Sprint topology.

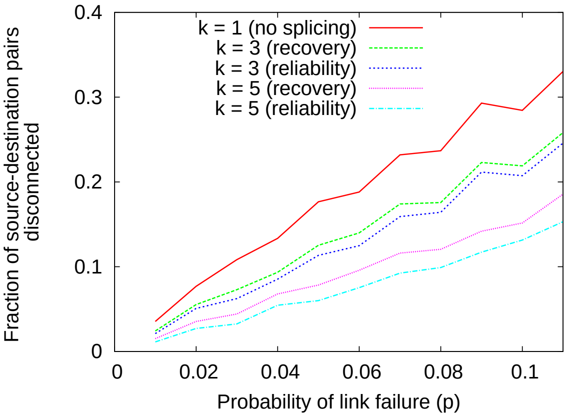


Figure 5: Recovery using *network-based recovery* and Sprint topology.

Figure 3 shows the reliability curves for Sprint using degree-based perturbations with $Weight(0, 3)$. Adding just one slice (*i.e.*, increasing k to 2) significantly improves reliability; adding more slices improves reliability further. Figure 3 demonstrates that even with just a few slices (*i.e.*, 5) and a very simple scheme for generating alternate graphs (*i.e.*, link-weight perturbations), *the reliability of path splicing approaches the reliability of the original underlying network!*

Our analysis in Appendix A shows that the number of slices required to achieve near-optimal connectivity with bounded stretch scales well with the size of the graph. Specifically, Theorem A.1 shows that the number of slices required to achieve connectivity that is close to that of the underlying graph scales as $\log n$, where n is the number of nodes in the graph.

4.3 Recovery is Fast

Routing protocols often react slowly (or not at all) to failures along end-to-end paths. Path splicing allows end hosts to signal to nodes along the network path that traffic should be forwarded along a different set of path segments. In this section, we demonstrate how the end-user and/or the network can quickly recover from failures by selecting spliced paths in the network at random.

We evaluate two approaches to recovery: *end-system recovery* is network-agnostic and relies on the end system (*e.g.*, user, proxy, edge router) to initiate recovery; *network-based recovery* assumes that the node in the network can detect a failure on an incident link and initiate recovery by diverting traffic to a different slice. We use a simulation setup similar to the one described for the reliability experiment to generate a spliced graph with failed links. For all disconnected source-destination pairs, we evaluate the effectiveness of these schemes for recovering from failure. If splicing can recover the path in five or fewer trials, we consider the path recoverable, particularly since these trials could be run in parallel.

End-system recovery. Figure 4 shows the recovery where the end system controls the specific spliced path to the destination. In our experiments, we use a header that allows 20 hops to be spliced. For a failed path, the new shim header (*i.e.*, the forwarding bits) is constructed as follows: A coin is tossed for every hop in the shim header; if the result is a head, a different slice is selected for that hop (*i.e.*, at every hop we switch slices with 0.5 probability). We check to see if a failed path can be recovered in fewer than 5 trials. The average number of trials in any case where splicing could recover from the failure was slightly more than 2. Paths were on average 1.3 times longer in delay compared to the shortest path in the "base" topology; the resulting paths typically use about 50% more hops compared to the original shortest path. In any particular slice, 99% of all paths in each tree have stretch of less than 2.6.

Network-based recovery. Figure 5 shows results for network-based recovery: When a router x receives packets destined to d with next-hop y and discovers that link (x, y) has failed, it finds in its forwarding table an alternate slice with a connected next-hop for d (if one exists). If a path between two endpoints is discovered using this recovery scheme we consider the path recoverable. All paths between connected endpoints need not be recoverable since the packet could end up in a dead-end from where there is no connected next-hop to reach the destination, due to the specific slices selected by the routers. The average stretch for this recovery scheme was 1.33 while there were 55% more hops in the recovered paths; these numbers are slightly higher compared to the end-system recovery scheme.

Our evaluation shows that path splicing may be practical: it provides fast access to a large number of diverse paths. More sophisticated recovery schemes that use some information about network failures, etc. should perform even better compared to the two simple schemes we have studied.

4.4 Forwarding Loops are Rare

Because traffic is no longer forwarded along a single routing tree, the potential for loops does exist. Forwarding loops are problematic because they increase the total length of the end-to-end path, and they also unnecessarily use network capacity. Fortunately, certain recovery strategies can avoid forwarding loops entirely. For example, in the splicing data-plane design we presented in Section 3.2 the splicing header will eventually run out of forwarding bits as each node shifts $\lg(k)$ bits from the header; at this point, the traffic will remain in its current tree en route to the destination. Additionally, paths that never switch back to a previously used slice would never contain persistent forwarding loops of any length; therefore, a recovery scheme that generates the forwarding bits so that the end-to-end path never revisits a slice would be guaranteed to not have persistent loops. Another useful strategy to limit forwarding loops is to restrict the number of switches between slices that a packet can make to a small number.

Although the potential for loops does exist in the general case, we note that these loops are still unlikely. First, exercising a forwarding loop requires switching between slices *repeatedly* and in exactly the right order to cause traffic to be forwarded along a cycle in the directed graph. The likelihood that the forwarding bits in the header will be chosen to exercise the same loop more than once decreases exponentially with each iteration of the loop.

In our experiments we observed that loops of more than two hops were extremely rare. Theorem B.1 in Appendix B proves this result. Two-hop loops occurred more frequently (about one per 100 trials for $k = 2$, and about one in ten trials for higher values of k). Using any of the schemes discussed above could eliminate loops entirely, at the cost of restricting the paths available for recovery. We are investigating the extent of these loops (*e.g.*, how many times packets traverse these loops in practice, the effects of explicitly preventing loops on reliability and recovery).

5. Discussion

In this section, we discuss several open issues and areas for future study. We first discuss alternative mechanisms for creating slices and other possible recovery mechanisms. Then, we discuss issues and various open questions related to the effect splicing may have on traffic. Finally, we discuss various deployment scenarios and applications of splicing. We intend to study these issues in the context of experiments on a network test-bed, as well as to evaluate the reliability and recovery numbers on other real ISP topologies.

Alternate slicing mechanisms. Generating slices by randomly perturbing link weights is appealing because of its simplicity: unlike other schemes [3, 9, 11, 16], splicing requires no explicit computation of backup paths, no complex configuration, no complex offline computation, and no frequent configuration tuning. Still, we expect that path splicing might perform even better if each slice were configured with some consideration of the edges in the underlying graph that were already covered by other slices. We intend to explore how other approaches to generating backup trees (*e.g.*, multi-router configuration, multi-topology routing) might be used to achieve more reliability with fewer slices.

Alternate recovery mechanisms. Our evaluation showed that even simple recovery mechanisms allow end systems and intermediate nodes to quickly find alternate working paths, but other approaches (*e.g.*, flipping bits to change the first hops in the path with higher probability) might result in even faster recovery. We are also investigating the use of alternate encoding schemes. One possibility is that the forwarding bits are simply reduced to a single number: any hop that saw a non-zero number in the splicing header could select an alternate path (perhaps deterministically, based on the number) and decrement the number.

Interactions with traffic engineering. Path splicing gives end systems control over the paths that traffic takes through the network, which may induce unpredictability and instability in offered traffic loads seen by ISPs. This unpredictability may make traffic engineering more difficult if the arrival patterns of traffic are continually varying. Still, we expect that these interactions to be manageable, if splicing can reduce the need for automatic tuning of link weights. Path splicing spreads traffic across the network even in the absence of failure if sources select their initial set of slices at random (as shown in Algorithm 1). This "automatic" load balancing might mitigate the need for various tuning that is necessary with today's routing protocols [6, 7], which can be potentially sensitive to small changes in the network topology (*e.g.*, link or node failures). We are currently comparing the traffic balance that path splicing achieves versus that which conventional link-weight optimization achieves, both in the case of failures and in steady state.

Selfish-routing effects. Path splicing gives end systems some control over the paths that traffic takes, which introduces the possibility that all end systems will react the same way upon seeing a faulty network path. If all end systems choose the same backup path when a link or node fails, the resulting traffic shifts could introduce congestion on certain links. Because each end system selects a new sequence of forwarding bits at random, we expect that traffic will disperse evenly across the network topology upon failure recovery; still, examining the effects of failures on traffic dynamics deserves further study.

Extensions to interdomain routing. Path splicing may also scalably provide access to multiple *interdomain* paths. BGP-speaking routers already have multiple routes to a destination in their routing table. The BGP decision process could be modified to select k best routes to a destination and install them in the forwarding tables. These alternate routes can be accessed with the forwarding bits. Furthermore, the forwarding bits could not only be used for selecting an alternate interdomain route but also for selecting the egress router in the network (our previous work describes this mechanism in more detail [13]). In contrast to existing multi-path routing architectures (*e.g.*, MIRO [21]), a spliced BGP would provide end systems access to multiple interdomain paths without requiring any additional communication among BGP routers to forward traffic along multiple interdomain paths.

Other applications and deployment scenarios. Path splicing is a general technique that applies to routing at other layers (and for other purposes). End hosts could set splicing bits in packets to simultaneously use disjoint paths, as opposed to simply changing splicing bits in reaction to failures, thus allowing hosts that send traffic between endpoints to achieve throughput that approaches the capacity of the underlying graph. Second, splicing could be applied to overlay routing. RON [2] uses a pairwise probing strategy to determine end-to-end paths that have low overall end-to-end latency or loss rates and then composes overlay paths based on a single link-state routing protocol that uses a single metric (*e.g.*, loss, latency, throughput). Applying path splicing to overlay routes may improve fault tolerance and capacity; it may also mitigate the oscillations that arise when independent overlays react to the observed conditions and induce overload or poor performance. [18]. It might also be used to combine overlay networks that use independent metrics (*e.g.*, splicing RON [2] with SOSR [10]).

6. Summary and Research Agenda

This paper has presented *path splicing*, a new primitive that constructs network paths from multiple independent routing protocol instances that run over a single network topology. Path splicing instantiates multiple routing protocol instances, each with its own random perturbation of link weights, to create an exponential increase in path diversity for only a linear increase in state and message complexity. Splicing is a general mechanism for composing routing protocols and can be applied to *any* routing protocol (*e.g.*, overlay routing). Path splicing is simple, scalable, and stable. It requires only static, per-link configuration, and it can be implemented by composing existing routing protocols.

Our experiments showed that splicing can be practical, but much work remains to determine the performance and behavior of path splicing in practice. We are also exploring alternate schemes for slice generation and recovery and extending the design to interdomain routing. We are building a prototype implementation to evaluate the performance of path splicing in a realistic setting. This prototype implementation and deployment study will improve our understanding of splicing's dynamic behavior and interactions with dynamic routing protocols. In particular, an important open question concerns the interactions of path splicing with the convergence of the routing protocol, which could affect forwarding-table entries at the same time as path splicing is re-routing traffic. In fact, we believe that path splicing may provide enough reliability from link and node failures to permit dynamic routing to react much more slowly to failures, and, in some settings, may even eliminate the need for dynamic routing altogether.

Acknowledgments

This work was inspired by discussions at the Georgia Tech Algorithms and Randomness Center ThinkTank (`http://arc.cc.gatech.edu/`) and funded by NSF CAREER Award CNS-0643974 and NSF Awards CNS-0626950 and CCR-0307536. We thank David Andersen, Hari Balakrishnan, Andy Bavier, Roch Guerin, Amund Kvalbein, Richard Lipton, Anirudh Ramachandran, Jennifer Rexford, Stefan Savage, Scott Shenker, Alex Snoeren, Muhammad Mukarram bin Tariq, Geoff Voelker, and Xiaowei Yang for helpful feedback and suggestions.

REFERENCES

[1] D. Andersen, N. Feamster, and J. Jannotti. Providing Mechanism and Policy-Independent Path Selection to End Systems: The Case for Path Bits. Technical Report CMU-CS-07-160, Carnegie Mellon University, 2007.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 131–145, Banff, Canada, Oct. 2001.

[3] G. Apostolopoulos. Using multiple topologies for ip-only protection against network failures: A routing performance perspective. Technical Report 377, ICS-FORTH, Apr. 2006.

[4] Cisco Multi-Topology Routing. `http://www.cisco.com/en/US/products/ps6922/products_feature_guide09186a00807c64b8.html`.

[5] D. D. Clark, S. Shenker, and A. Falk. GENI Research Plan. Technical Report GDD-06-28, GENI Planning Group, Apr. 2007.

[6] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000.

[7] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE JSAC*, 20(4):756–767, May 2002.

[8] GÉANT. `http://www.geant.net/`.

[9] S. Gjessing. Implementation of two Resilience Mechanisms using Multi Topology Routing and Stub Routers. In *International Conference on Internet and Web Applications and Services/Advanced*, Feb. 2006.

[10] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *Proc. 6th USENIX OSDI*, San Francisco, CA, Dec. 2004.

[11] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast IP Network Recovery using Multiple Routing Configurations. In *Proc. IEEE INFOCOM*, pages 23–26, Barcelona, Spain, Mar. 2006.

[12] N. McKeown and B. Girod. Clean slate design for the internet. Whitepaper, Apr. 2006.

[13] M. Motiwala, N. Feamster, and S. Vempala. Improving Interdomain Routing Security with BGP Path Splicing. In *Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*, Princeton, NJ, May 2007.

[14] J. Moy. *OSPF Version 2*, Mar. 1994. RFC 1583.

[15] D. Oran. *OSI IS-IS intra-domain routing protocol*. Internet Engineering Task Force, Feb. 1990. RFC 1142.

[16] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault. *Multi-Topology (MT) Routing in OSPF*. Internet Engineering Task Force, Jan. 2007. `http://www.ietf.org/internet-drafts/draft-ietf-ospf-mt-08.txt` Work in Progress, expires July 2007.

[17] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault. *Multi-Topology Routing in OSPF*. Internet Engineering Task Force, June 2007. RFC 4915.

[18] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker. On selfish routing in Internet-like environments. In *Proc. ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003.

[19] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, Jan. 2006. RFC 4271.

[20] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[21] W. Xu and J. Rexford. MIRO: Multi-path Interdomain ROuting. In *Proc. ACM SIGCOMM*, Pisa, Italy, Aug. 2006.

[22] X. Yang, D. Wetherall, and T. Anderson. Source selectable path diversity via routing deflections. In *Proc. ACM SIGCOMM*, Pisa, Italy, Aug. 2006.

APPENDIX

A. Reliability Analysis

Fix a maximum allowable stretch D. Then, for each pair of vertices s, t, we consider the subgraph $G(D, s, t)$ induced by paths of length at most D from s to t. Let $\chi_G(D, s, t)$ be the connectivity of this graph and $\chi_G(D) = \min_{s,t} \chi_G(D, s, t)$. We show that the connectivity of the paths used by path splicing approaches $\chi(D)$ (*i.e.*, that of the underlying graph).

Theorem A.1 *Let H denote the union of k shortest path trees to a destination t, each obtained from a graph G by independent random perturbations of the link weights uniformly in the range $(L, 2DkL)$. Then for any $k > c_0 \log n$, with high probability, the connectivity of H is at least $c_1\chi_G(D)$ where c_0, c_1 are universal constants and n is the number of nodes in the graph.*

We only give the idea of the proof here. We argue that every cut of H has $\Omega(k)$ edges. This uses two ideas: (1) there are at most $n(n-1)/2$ mincuts in an undirected graph with n vertices and at most $2^l n(n-1)/2$ cuts of with at most l times the minimum number of edges (2) An cut C with $|C|$ edges has an edge subset set of support $\Omega(|C|)$ with the property that each edge is chosen roughly uniformly in a shortest path tree with perturbed weights. Combining (1) and (2) along with a Chernoff bound gives the claimed result.

B. Stretch Analysis

In this section, we show that stretch is bounded and that, as a consequence, long forwarding loops are unlikely.

Theorem B.1 *Assume the perturbations of a link i with original weight L_i are uniform in the range $[-cL_i, cL_i]$. Consider a packet traveling from source s to destination t that has made m hops of perturbed lengths $L' = (L'_1, \ldots, L'_m)$ on a single slice and reached a node u. Let P be a shortest path from s to t. Then, for any $r > 1$,*

$$\mathsf{P}\left((1-c)d(u,t) \leq ||P||_1 - ||L'||_1 + \frac{rc}{\sqrt{3}}||P||_2\right) \geq 1 - \frac{1}{r^2}.$$

PROOF. We begin with a simple probabilistic bound on the perturbed length of any fixed path. Let X_i be the perturbed length of a traversed link with original length L_i. Then $\mathsf{E}(X_i) = L_i$. Further,

$$\begin{aligned} \mathsf{Var}(X_i) &= \mathsf{Var}(X_i - L_i))^2 \\ &= \mathsf{E}((X_i - L_i)^2) - \mathsf{E}(X_i - L_i)^2 \\ &= \mathsf{E}(Y_i^2) \end{aligned}$$

where Y_i is uniform in the interval $[-cL_i, cL_i]$. Thus,

$$\begin{aligned} \mathsf{Var}(X_i) &= \frac{\int_{-cL_i}^{cL_i} y^2\, dy}{2cL_i} \\ &= \frac{c^2}{3}L_i^2. \end{aligned}$$

Let $X = \sum_{i=1}^m X_i$. Using Chebychev's inequality, we have

$$\mathsf{P}\left(|X - L| \geq r\frac{c}{\sqrt{3}}|L|_2\right) < \frac{1}{r^2}.$$

Let $d'(.,.)$ denote the shortest path distances with perturbed weights in the current slice. Let P be a shortest path between s and t with the original weights.

$$\begin{aligned} d(u,t) &\leq \frac{1}{1-c}d'(u,t) \\ &= \frac{1}{1-c}(d'(s,t) - ||L||_1) \\ &\leq \frac{1}{1-c}(d(s,t) + \frac{rc}{\sqrt{3}}||P_{st}||_2 - ||L||_1) \end{aligned}$$

where the equality follows from the property of a shortest path and the inequality holds with high probability using the above analysis. ■