Slick Packets

Giang T. K. Nguyen
University of Illinois
at Urbana-Champaign, USA
nguyen59@illinois.edu

Rachit Agarwal
University of Illinois
at Urbana-Champaign, USA
agarwa16@illinois.edu

Junda Liu
University of California
at Berkeley, USA
liujd@cs.berkeley.edu

Matthew Caesar
University of Illinois
at Urbana-Champaign, USA
caesar@illinois.edu

P. Brighten Godfrey
University of Illinois
at Urbana-Champaign, USA
pbg@illinois.edu

Scott Shenker
University of California
at Berkeley, USA
shenker@cs.berkeley.edu

Abstract

Source-controlled routing has been proposed as a way to improve flexibility of future network architectures, as well as simplifying the data plane. However, if a packet specifies its path, this precludes fast local re-routing within the network. We propose SLICKPACKETS, a novel solution that allows packets to slip around failures by specifying alternate paths in their headers, in the form of compactly-encoded directed acyclic graphs. We show that this can be accomplished with reasonably small packet headers for real network topologies, and results in responsiveness to failures that is competitive with past approaches that require much more state within the network. Our approach thus enables fast failure response while preserving the benefits of source-controlled routing.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Packet-switching networks; C.2.2 [**Network Protocols**]: Routing protocols; C.2.6 [**Internetworking**]: Routers

General Terms

Algorithms, Design, Performance, Reliability

Keywords

Reliability, failures, routing, source routing, forwarding

1. INTRODUCTION

Traditional routing protocols are **network-controlled**: routes are computed within the network, with each router picking, from among its neighbors, the next-hop to each destination. Examples include BGP for interdomain routing, and OSPF for intradomain routing. An alternate paradigm, **source-controlled routing** (SCR), improves the flexibility of the network architecture. Rather than computing all routes within the network, SCR architectures [10, 20, 29–31] reserve some choice of routes for the *sources*[1] to select on a per-packet basis. The uses of SCR's routing flexibility are quite diverse. Sources can observe end-to-end reliability problems and switch to a working path within a few round-trip times (RTTs); pick better-performing routes based on observed performance [5, 11, 25]; improve load balance since path selection is finer-grained [24]; encourage competition among network providers [7]; improve security [28]; or optimize for other application-specific objectives. SCR is thus a promising approach to improve the flexibility of the network layer in future Internet architectures.

However, one remaining problem is that of *fast failure reaction*. This problem arose in early network-controlled routing (NCR) protocols, which suffered from unreliability during network dynamics: during the distributed convergence process, packets could enter "black holes" or loops, resulting in tens of seconds or minutes of downtime in Internet end-to-end paths [14, 27]. Treating these basic protocols as a baseline, two high-level approaches have been proposed to improve failure reaction.

The first approach works within the NCR paradigm by computing an alternate path to each destination (or IP prefix or AS); a router can locally switch to the alternate path without waiting for a control-plane convergence process. Packets can thus be delivered continuously, except for the minimal time it takes for a router to detect failure of one of its directly connected links and locally switch to an alternate path. Examples include MPLS Fast Reroute [23], SafeGuard [18], and FCP [17] for intradomain routing, and R-BGP [16] for interdomain routing. However, this approach lacks the routing flexibility of SCR.

A second approach to improve failure reaction is to leverage SCR's routing flexibility: a source can switch routes without waiting for the Internet's control plane to reconverge. While this improves failure reaction time relative to the baseline above, the source still must wait to receive notice of the failure. Regardless of the means of notification, this will take at least on the order of one RTT, which at Internet scales would be much slower than the first approach of using NCR with alternate paths.

SIGMETRICS'11, June 7–11, 2011, San Jose, California, USA.

---

[1] In this paper, we use "source" to refer either to end-hosts or to edge routers acting on their behalf.

And in the SCR proposals that provide the most flexibility [10,30], sources specify in the packet header an explicit route (perhaps at the level of autonomous systems) rather than a destination, so the NCR and SCR techniques cannot be immediately combined.

The goal of this paper is to achieve the best of two worlds: the fast failure reaction of alternate routes embedded within the network, and the flexibility of routes chosen by sources at the edge of the network. To meet this goal, we work within the SCR paradigm, but with a twist. Instead of specifying a single path to the destination, the packet header contains a directed acyclic graph that we call the *forwarding subgraph* (FS). Each router along the packet's path may choose to forward it along any of the outgoing links at that router's node in the FS (optionally preferring a path marked as the primary), with no danger of causing a forwarding loop. This approach, which we call SlickPackets, allows packets to "slip" around failures in-flight while retaining the flexibility of source route control. Moreover, SlickPackets provides a scalability benefit over NCR with alternate paths: rather than requiring multiple routes to every destination in every router's forwarding table, SlickPackets routers need only local information.

Of course, our approach also presents several challenges. Chief among these is how to encode an FS with sufficient path diversity into the small space afforded by a packet header. We introduce techniques through which the FS can be encoded compactly enough for our mechanism to be feasible. For example, an FS providing an alternate path at every hop along the primary occupies less than 26 bytes for 99% of evaluated source-destination pairs in an AS-level Internet map, and no higher than 50 bytes in all evaluated cases. Thus, the technique incurs manageable overhead for applications that send packets of moderate to large size. We also demonstrate through a simulation-based performance evaluation that SlickPackets achieves failure reaction performance that is comparable to the best of NCR architectures [18].

The rest of this paper proceeds as follows. In §2, we present an overview of SlickPackets and its principal design challenges. §3 gives a detailed presentation of the SlickPackets design. We evaluate the performance of our design in terms of header size and failure reaction in §4. We discuss extensions of SlickPackets in §5 and related work in §6, and conclude in §7.

2. OVERVIEW

In this section, we provide an overview of SlickPackets, and discuss several critical design challenges.

SlickPackets is a failure reaction mechanism for SCR protocols. In contrast to traditional SCR protocols that specify a single path in the packet header, SlickPackets enables fast recovery within the network by allowing the source to embed the rerouting information within the packet header in the form of a **forwarding subgraph (FS)**. The FS specifies a set of paths that intermediate routers can use to reroute packets in case of failures. The source, if it desires, can designate one of these paths as the **primary path** to be used in the absence of failure; the rest of the paths are then treated as **alternate paths** that can be used if the primary path is not available. In order to avoid forwarding loops, SlickPackets requires that the FS be a directed acyclic graph (DAG).

Performing forwarding in this way has two main benefits. First, since the source specifies the FS, it has full control of not only the primary path, but also how the network forwards the packet when the primary path is not available. Second, since alternate path information is embedded directly in the packet header, the network can react immediately without requiring involvement of the source, which reduces the reaction time in presence of link failures. In addition to these two benefits, the task of a router becomes simpler: a router requires only local knowledge of its neighbors, rather than needing an alternate path for every destination (which may require information such as the multi-homing locations of each host). In summary, SlickPackets achieves key benefits of SCR architectures (flexibility in route selection and scalability of network routing state) while simultaneously attaining failure reaction performance that is comparable to that of NCR architectures with backup paths.

Fig. 1 shows an example to illustrate the design of SlickPackets. Suppose the source s wishes to send a packet to a destination d. The source has acquired, by some mechanism to be discussed later, a map of the network. It selects the FS as shown in Fig. 1 and designates (s, R_1, R_2, R_5, d) to be the primary path. Note that the FS provides each node on the primary path with sufficient alternatives so that if a link on the primary path fails, the packet can be rerouted to the destination. Next, s constructs a data packet with the subgraph embedded in the packet header, and forwards it on to the first-hop R_1. At R_1, the packet is forwarded to the next-hop on the primary path (R_2). Now suppose that at R_2 the primary path's next hop R_5 lies across a failed link (R_2, R_5); then R_2 forwards the packet to R_4, the next-hop on its alternate path in the FS, after which the packet continues to R_5 and finally d.

Realizing the high level idea of source-controlled routing along an FS, however, involves several key challenges. We outline these challenges and our solutions here.

Obtaining the map. Like other SCR architectures in which sources construct end-to-end paths, our sources require a map of the available links. When deploying SlickPackets as an interdomain routing protocol, this immediately raises questions of scalability and policy compliance. Is it feasible to push a map of the Internet, at some level of granularity, to every source or at least every edge router? Is there an acceptable way to balance control of network resources between the senders and the network owners? Fortunately, we can adopt the solutions developed by past work, in particular NIRA [30] and pathlet routing [10], which have shown how maps of policy-compliant transit service can be constructed and disseminated in ways that can be much *more* scalable than traditional NCR protocols like BGP.

Packet header overhead. The next challenge is to design an efficient encoding mechanism that embeds the FS into the packet header with minimal overhead. By using link labels with only local significance and allocating every bit carefully, we are able to achieve acceptable packet header sizes on realistic network topologies.

Fast data-plane operations. Another challenge is to

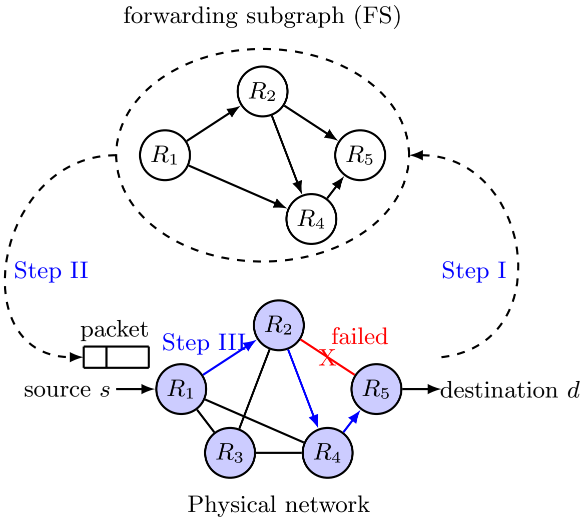


Figure 1: Overview of the SlickPackets design. Step I: the source selects a forwarding subgraph (FS) based on the topology of the physical network; Step II: the source encodes and embeds the FS in the packet header to inform routers how to route around encountered failures; and Step III: routers forward the packet based on the FS contained in the packet header.

design an efficient data plane forwarding algorithm: the encoding and forwarding mechanisms in SlickPackets should minimize next-hop lookup time without substantially increasing header processing cost, forwarding delay and/or design complexity of modern router forwarding planes. Fortunately, forwarding along an FS requires only lookup and pointer-increment operations, as in standard SCR protocols, and can be efficiently implemented in practice.

The next section discusses our design in more detail, including our solutions to these challenges.

3. SLICKPACKETS DESIGN

In this section, we present in more detail the four main components of SlickPackets: definition and dissemination of the network map (§3.1); selection of a forwarding subgraph (FS) at the source (§3.2); encoding of the FS into the packet header (§3.3); and the data plane forwarding mechanism at routers (§3.4).

The SlickPackets approach could be applied in multiple contexts. We describe here how the design can be applied to interdomain and intradomain routing. The differences principally lie in map dissemination and data plane forwarding, with the core approach taking the same form in both contexts.

3.1 Map format and dissemination

As in other SCR protocols in which the source composes end-to-end paths [10, 30], in SlickPackets, the source must obtain a network "map" (topology) from which it can construct paths. This map is an abstract directed graph in which each directed directed link (u, v) at node node u is annotated with a *label*. The label is a compact, variable-length bitstring, which the source will use when encoding the FS (§ 3.3) to tell node u that it wants u to use the link (u, v). Similar to an MPLS label, the label identifies a link only locally at u, not globally. Thus, u will generally announce labels of length $\lceil \log_2 \delta(u) \rceil$ bits where $\delta(u)$ is the degree of u.

What this map corresponds to in the physical network and how the map is disseminated depend on the deployment scenario. In an intradomain environment, the map would correspond to the physical topology of routers and links and could be distributed via a protocol like OSPF or through a centralized coordinator as in [17].

In an interdomain environment, we have to deal with the significant challenges of scalability and network owners' transit policies. In order to overcome these challenges, we build on solutions developed in past work and briefly describe them here for completeness.

Basic approach. Both NIRA [30] and pathlet routing [10] provide sources with a policy-compliant map of the Internet, roughly at the autonomous system (AS) level. NIRA's map assumes common customer-provider-peer relationships between ASes and allows a subset of *valley-free* routes: that is, packets travel up a chain of providers, potentially across a peering link, and down a chain of customers to the destination. Pathlet routing represents this map explicitly as an arbitrary virtual topology, whose edges (pathlets) represent policy-compliant transit service.

Scalability. NIRA, while dependent on the existence of a typical AS business hierarchy, offers the opportunity of vastly *improving* BGP's control plane scalability. Rather than learning an Internet-wide topology, each node learns its "up-graph" of routes through providers, stopping at the "core" of the Internet. The up-graph requires fewer than 20 entries for 90% of domains [30], many orders of magnitude less than the roughly 300,000 prefixes that BGP propagates today. Each destination stores its up-graph in a global DNS-like database; to route to a destination, a source queries the database and combines its own up-graph with the destination's up-graph. Though the resulting map is a small fraction of the Internet, it includes all policy-compliant (valley-free) routes. Pathlet routing could use a NIRA-style approach for disseminating the pathlet topology, or it can be disseminated via a BGP-like mechanism with slightly more messaging and control state ($\leq 1.7\times$) than traditional BGP.

SlickPackets can take advantage of either the pathlet or NIRA approach for interdomain map dissemination. Thus, SlickPackets does not require a source to have complete topological knowledge of the network, but rather only enough to construct a path and alternate paths to the destination.

We also note that SlickPackets, like other SCR and multipath routing architectures, can benefit from significantly reduced rate of control plane updates [6] compared with basic single-path NCR architectures. This is because short-lived failures need not be disseminated through the control plane, since failure reaction will happen anyway via forwarding along alternate paths without waiting for control-plane updates.

Link labels. Along with the map itself, SlickPackets requires labels on the links. Routers (or ASes for interdomain; for convenience we'll use "routers" in what follows) can piggyback this information with the link advertisements [10]. To change a label, a router readvertises the link. While readvertisements increase control traffic, we expect that changing a router's link labels will be fairly rare, for two reasons. First, the operator could change a single label from one bit sequence to another; however, there should be little need for such changes because the labels are arbitrary identifiers with no significance. Second, the operator may need to increase the number of links exiting the router. This *may* increase the label length and require readvertisements of all of the router's link labels, creating a period of inconsistency from when the router changes its label length to when sources receive the updated announcement. However, label lengths change only once every time the number of outgoing links doubles (or halves) in size, which is expected to be a very rare event.

An alternate approach is to make labels self-describing: their first few bits encode the label length [10]. This avoids the need to readvertise links after a length change and the resulting inconsistency, but labels become slightly longer. Since compactness is important for SLICKPACKETS, we do not evaluate this approach in this paper.

Map consistency. A natural question is whether all sources and the network must have an entirely consistent view of the map at all times. Fortunately, this difficult task is unnecessary. There are three possible types of inconsistency.

First, if a source uses a non-existent label (e.g., the link has been removed or its label changed), this is equivalent to a link failure and the packet can be re-routed along an alternate path. To avoid even this minor disturbance, routers can insert a short delay between announcing a label deletion and its removal from forwarding tables.

Second, if a source uses a label that has changed to identify a different link, then the packet will follow an incorrect path and will be unlikely to reach its intended destination. This is similar to inconsistency problems in basic NCR protocols. (Unlike in basic NCR protocols, however, the packet cannot get into a loop of any significant length because one link in the DAG will be consumed at each hop.) To avoid label-change inconsistency, routers can simply use new labels rather than reusing ones that have recently had a different meaning.

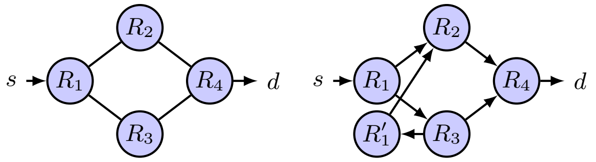
Third, a source might be unaware of some valid labels. This simply results in a slightly restricted set of options until it receives the relevant control plane advertisement, as in essentially any other distributed routing protocol.

Thus, in all cases, inconsistency issues can be mitigated.

3.2 Selection of the forwarding subgraph

Once a source has obtained the network map, it selects a *forwarding subgraph* (FS) along which it desires the packet to be routed in the network. The FS is a DAG corresponding to a subset of nodes and links in the network map. The directed edges inform routers of the packet's allowed next-hops, and acyclicity ensures there are no forwarding loops. Additionally, for each node in the FS, the source may mark one outgoing link as the preferred primary.

Sources have a great deal of flexibility in how they choose an FS. For instance, the source may select an FS that avoids any single link failure along a low-latency primary path, avoids node failures, optimizes for other metrics like bandwidth, or picks alternate paths that avoid shared risk link groups. We discuss some of these uses in §5. For concreteness, we describe here and evaluate in §4 how the source can pick an FS that will minimize primary-path latency and provide alternate paths to avoid any single link failure. As noted below, accommodating shared risk link groups is similar.



(a) Network map (b) Forwarding subgraph

Figure 2: An FS may have multiple representations of a network map node, to allow "backtracking" without introducing cycles in the FS.

A source s, for a given destination d, constructs a single-failure-avoiding FS as follows. First, s computes a primary path P to d by running a shortest path algorithm over the network map. Next, s visits each link along P, and computes the alternate path P_i it would prefer the packet to be routed along if that link were to fail. In particular, for each node v_i on the primary path, we (a) remove v_i's outgoing edge corresponding to its next hop along the primary path; (b) compute a shortest path from v_i to d, not using the removed outgoing edge; and (c) restore the removed edge. In case of a node having multiple shortest paths to the destination, the source may arbitrarily select one of these shortest paths. Finally, the primary and the alternate paths are assembled into the FS. Note that the above algorithm requires $|P|$ runs of Dijkstra's algorithm. Surprisingly, it is possible to construct a primary path and all the alternate paths in a *single* run of a shortest-path algorithm; see [12].

Beyond single-link-failure protection, a source may want to protect against failures of shared risk link groups (i.e., sets of links that are likely to have correlated failures, such as multiple logical links allocated to a single physical fiber). Assuming it has knowledge of these groups, it can do this by removing all links in the group in substep (a) above, and restoring them all in (c).

Note that there is a subtlety in how the the primary and the alternate paths are "assembled" into the FS: if we simply take the union of all these links and edges, we might create a loop, violating the acyclicity requirement. Consider the network map in Fig. 2(a). Assume that s desires to use (s, R_1, R_3, R_4, d) as the primary path. Then to escape a failure of the link (R_3, R_4), a packet located at R_3 must follow the path (R_3, R_1, R_2, R_4, d). Taking the union of these primary and alternate paths would result in a loop $R_1 \rightarrow R_3 \rightarrow R_1$. Due to symmetry, the problem persists if (s, R_1, R_2, R_4, d) is the primary path.

In order to avoid such loops, when adding an alternate path edge (u, v) to the FS, we first check to see if this would cause a loop. If so, we create a second FS representation v' of the physical node v, and add the edge (u, v'). This can be seen as "tunneling" the packet back along an alternate path. In the example of Fig. 2, before adding the second alternate path, we create a new copy R_1' corresponding to the node R_1. The alternate path then follows (R_3, R_1', R_2, R_4, d), resulting in a acyclic representation of the FS as shown in Fig. 2(b).

3.3 Encoding the forwarding subgraph

After choosing an FS, the source must encode the FS into a sequence of bits and place it in the packet header. SlickPackets is agnostic to the particular location this header appears in the packet (for example, it may reside in a "shim" header between the IP and MAC layers, in an IP option, or in a novel header format in a next-generation Internet protocol). There are two key goals in designing an encoding format: (a) minimizing the size of the resulting encoding; and (b) ensuring data plane forwarding operations are simple. We designed and evaluated several encoding formats to achieve these goals.

In this paper we present two encoding formats, called Direct and Default. Each may result in a smaller encoding in certain scenarios as discussed below. But the latter resulted in smaller encoding sizes in the network topologies we evaluated using the single-failure-avoiding FS selection (§3.2), so it is our default.

Direct format. The Direct format encodes the FS directly, in the sense that the FS's DAG data structure in memory is essentially directly serialized into a DAG data structure in the packet header. The header contains a sequence of node representations, each containing one or more outgoing link representations; each link representation contains its corresponding label and a pointer to another node within the header, corresponding to the node at the other end of the link. We describe the bit-layout of this format in detail in [22].

Default format. One source of overhead in the Direct format is the use of pointers within the header. Our Default format avoids some of that overhead, by grouping together sequences of labels corresponding to alternate paths, without needing an explicit representation of each node along the alternate path. The disadvantage of this grouping is that it involves duplicating link representations, similar to how a depth-first traversal of all paths in the DAG could visit links multiple times.

In fact, there exist DAGs that have exponentially large numbers of possible traversals (thus specifying exponentially large numbers of ways the packet could be forwarded through the network). Consequently, the Direct format can be exponentially more efficient than the approach of Default in the most extreme case. In general, we expect Direct will be more compact for situations in which the alternate paths often share nodes with one another or with the primary. However, in this paper we focus on the particular application of choosing single-failure-avoiding FSes. For that application, we found that the savings from avoiding pointers outweighed the duplication of link representations, so that Default was somewhat more compact in several realistic networks (§4). We therefore choose the Default format as our default and describe it in more detail now.

In the Default format, the FS is represented as a sequence of **segments**, one for each router on the primary path. For instance, in Fig. 3, the primary path consists of k hops and $S_1, S_2, \ldots, S_k$ are the segments corresponding to those k hops. The segment corresponding to a router v on the primary path contains three pieces of information (see Fig. 3): (a) v's next-hop on the primary path; (b) the bit-length of the encoding of v's alternate path; and (c) v's alternate path, as a sequence of next-hop labels. By "v's alternate path" we mean the alternate path beginning at v that avoids the primary next-hop from v. (We assume here that the FS has the format of one alternate path for each link on the primary.[2])

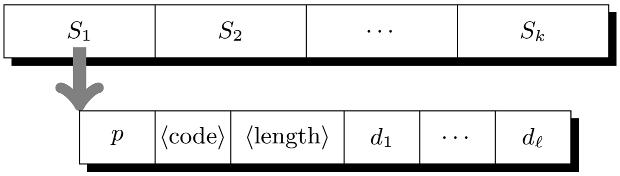


Figure 3: Default encoding format layout. S_i is the segment corresponding to node v_i on the primary path. It encodes the node's primary next hop p and alternate path $(d_1, d_2, \ldots, d_\ell)$. ⟨length⟩ specifies the bit-length of the alternate path, and ⟨code⟩ specifies the bit-length of the ⟨length⟩ field.

For (a), we need to include the router's label (§3.1) for the given outgoing edge, and similarly for (c) we include a sequence of labels. Recall that these labels are only locally unique to each node, which is critical to achieving a compact encoding, because the average number of neighbors of a router in a real-world network is typically vastly smaller than the total number of routers in the network [3,13]. By exploiting the structure of the real-world graphs, we are able to reduce the size of the encoding significantly compared with globally-unique labels.

For (b), we use the two fields: ⟨code⟩ and ⟨length⟩. Here, ⟨length⟩ specifies the total bit-lengths of all the labels $d_1, \ldots, d_\ell$ of the alternate path. Based on our evaluation, alternate paths are shorter than 32 bits in most cases and always shorter than 128 bits; in cases a node has no alternate path, the alternate path bit-length is 0. Thus, for greater compactness, we make the bit-length of the ⟨length⟩ field be variable and store it in the ⟨code⟩ field using a prefix-free code, with the ⟨code⟩ bit sequences 0, 10, and 110 mapping to values of 5, 7, and 0, respectively.

The header contains two additional pieces of information. First, the SlickPackets header begins with a two byte field, specifying its *header length*. Second, a one-bit field ON-ALTERNATE? specifies whether the packet is traversing along the primary path or an alternate path, and is initially false. We discuss next how routers use this information to forward packets.

3.4 Forwarding

We now describe the forwarding mechanism used by SlickPackets routers for the Default format. The input to this mechanism is the SlickPackets header described in §3.3, and the output is the interface out which the packet will be forwarded.

---

[2] While the Default format could be generalized to have multiple alternates at a router, or segments within segments to provide alternates for routers along an alternate path, we do not explore that generalization here; in any case, such applications can use the Direct format.

Upon receiving a packet, the router first checks the value of the SlickPackets header length. If this is 0, this router is the destination for the packet. If not, the router checks the on-alternate? bit to see whether it is on the primary path or on an alternate path. We describe the forwarding operations for the two cases separately.

Router on the primary path. The router reads the first segment in the header, which corresponds to itself, and inspects the primary next-hop label p. If the corresponding link available, the router deletes this first segment corresponding to itself. It also updates the header length by subtracting the length of its segment. The packet is then forwarded to the next-hop on the primary path with the new header.

If the primary next-hop link is not available, and the alternate path length is 0, the packet is dropped. Otherwise, the router reads its next-hop label d_1 on the alternate path. If the link corresponding to d_1 is not available, the packet is dropped.[3] If the link is available, the router removes *all* segments in the header, replacing them by its remaining alternate path labels $(d_2, \ldots, d_\ell)$. It also updates the header length appropriately and sets the on-alternate? bit. The packet is then forwarded to the next-hop via label d_1.

Router on an alternate path. The router reads its next-hop label. If the corresponding link is not available, the packet is dropped (or, as earlier, some other failure reaction mechanism is employed). If the link is available, the router deletes its label from the header, updates the header length, and forwards the packet to the next-hop.

Simplifying forwarding operations. The above description involved removing a prefix of the header, and in the case of moving to an alternate path, a suffix as well. In some data plane implementations, these operations may be costly. In this case, we can simply add *start* and *end* pointers at the front of the header, indicating the extent of the remaining header. In an extra 3 bytes, we can fit two pointers that can point to individual bits in a 512-byte header (which is far larger than we need).

Interdomain vs. intradomain issues. In an intradomain deployment, we may assume that each router runs SlickPackets and forwards packets as described above. However, in an interdomain deployment the forwarding subgraph roughly represents AS-level paths (as discussed in more detail in 3.1). When the packet is forwarded though an intermediate domain, that domain must forward the packet on to the next AS-level hop. Network operators may independently choose from a variety of ways to do this, for example by tunneling the packet with MPLS, or perhaps running SlickPackets internally as well as interdomain.

4. EVALUATION

SlickPackets advocates the idea of embedding a forwarding subgraph (FS) in the packet header, giving routers multiple forwarding options in order to provide the source with some property that it desires. While SlickPackets can support flexible FS selections that provide different guarantees, for concreteness, this section evaluates the FS selection exemplified in §3.2, which targets fast reaction in the presence of single-link failures. The source constructs a DAG comprised of the shortest primary path, and the shortest alternate path for each node on the primary path in case that node's outgoing link along the primary path fails. In terms of performance, three metrics are important: (a) encoding size, (b) failure reaction effectiveness, and (c) router complexity and packet forwarding rates. We present results for (a) and (b) in this section and discuss (c) in §7.

Topologies. We use three network topologies in our evaluation: the latency-annotated topology from Sprint ISP 1239 [2], with 315 nodes and 972 links; an AS-level map of the Internet [13], with $33,508$ nodes and $75,001$ links; and the largest component, with $190,914$ nodes and $607,610$ links, of a router-level map of the Internet [1]. The latter two topologies lack latency information; we take all links to have equal length. While using SlickPackets directly on a router-level map of the Internet is not a likely deployment scenario (due to privacy and scaling issues, ASes do not propagate internal topologies globally in today's Internet), we consider this extreme design point to investigate scaling issues of our design.

4.1 Encoding size

Since we encode the FS into the packet header, the **encoding size** determines the bandwidth overhead. We evaluate the resulting encoding sizes of the Direct and Default encoding formats presented in §3.3, for FSes constructed using the algorithm presented in §3.2.

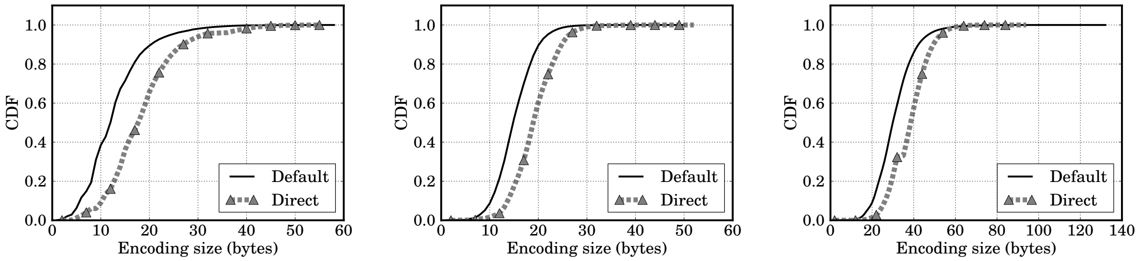
Furthermore, regardless of the encoding format used, the **FS size**—the number of edges—is a factor influencing the encoding size. We are thus also interested in comparing the sizes of FSes constructed by the algorithm described in §3.2 to **lower bounds** on the sizes of FSes returned by any algorithm that provides shortest path latencies and single-link failure protection. These lower bounds impose a fundamental limit on the encoding size; intuitively, for a given encoding format that already uses optimized label lengths, it is hard to reduce the encoding size significantly without reducing the FS size. We describe in [22] an algorithm that yields a lower bound on the size of the FS for a given primary path hopcount.

Methodology. We evaluate all 98,910 possible ordered source-destination pairs of the Sprint topology. For the AS- and router-level topologies, we randomly sample ten million unique ordered source-destination pairs. For each pair, we record these values: the Default and Direct encoding sizes, the size of the FS constructed using our algorithm, and the lower bound on FS sizes.

Results. Fig. 4 shows the encoding size results. We see that Default has somewhat smaller size almost always; Direct performs noticeably better only in the extreme tail of the router-level topology. We therefore discuss Default in what follows. For the intradomain Sprint topology, the maximum encoding size is 58 bytes. The plot has a long tail with 90% and 99% of the source-destination pairs requiring less than 21 bytes and 34 bytes of encoding, respectively. For the interdomain AS-level map of the Internet, the maximum encoding size is 50 bytes. As with the Sprint topology, the plot has a long tail, with 90% of the source-destination pairs resulting in encodings of less

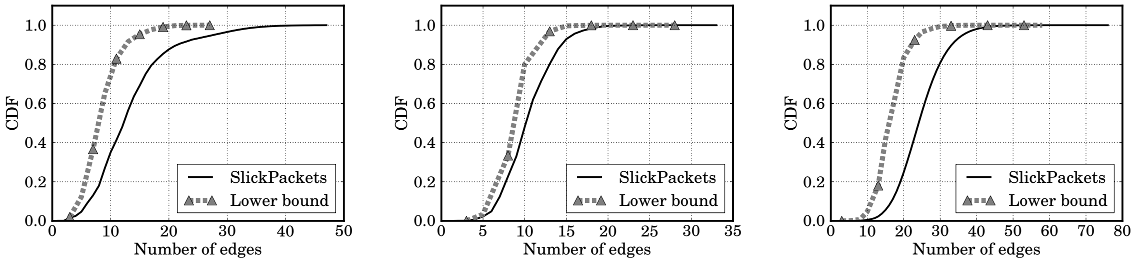
---

[3] Or any other failure reaction mechanism can be applied.



(a) Sprint Topology (b) AS-level Topology (c) Router-level Topology

Figure 4: CDF of SlickPackets encoding size in bytes for the Direct and Default encoding formats, for handling single-link failures.



(a) Sprint Topology (b) AS-level Topology (c) Router-level Topology

Figure 5: CDF of SlickPackets FS size and the lower bound in number of edges for handling single-link failures.

than 21 bytes; 99% of the source-destination pairs result in less than 26 bytes.

For the extreme case of router-level topology, 90% of the source-destination pairs result in encodings of less than 43 bytes; 99% less than 60 bytes. The remaining less than 1% of the source-destination pairs constitute the long tail, with maximum encoding size of 132 bytes. Although the router-level realization of SlickPackets may be impractical, the above results demonstrate that SlickPackets can scale on graphs as large as $200,000$ nodes with moderate increase in the packet header sizes. If desired, this overhead may be amortized over more data (e.g., by leveraging IPv6 jumbo frames) or using SlickPackets only for application data that is most sensitive to failures.

Fig. 5 shows the FS size (in number of links) and lower bound. For the AS-level and router-level topologies, our FS size is very close to the lower bound; for the Sprint topology, the difference is somewhat larger. Overall, the results suggest that, for handling single-link failures, our simple FS selection algorithm is relatively close to optimal in terms of minimizing the number of links in the FS.

For the Sprint topology, there is also a long tail in both our FS sizes and the lower bounds. The reason is that there are a few source-destination pairs that have long primary paths, requiring alternate paths for a large number of nodes, resulting in larger number of edges.

4.2 Failure reaction effectiveness

One metric to evaluate the effectiveness of a failure reaction mechanism is the packet **stretch**, the ratio of the length of a packet's path to the length of the shortest possible path. Previous works calculate stretch based on packets' traversed path costs or transit times. However, for a delay-sensitive application, we are interested in the time a packet is *live* from the application's perspective—from the time the packet is generated by the source application to the time it is received by the destination. Thus, we define the stretch for a packet that does not fully traverse the original shortest path, to be the ratio of the time the packet is *live* to the post-link-failure shortest path latency; for other packets—those that traverse the original shortest path—the stretch is 1. For brevity of the ensuing discussion, l_0 denotes the failed link on the primary path from source s to destination d; r_0 denotes the router that is adjacent to and upstream from l_0 on the primary path; and t_0 denotes the time of failure of l_0.

Modeling delay at network devices. A router in the network, upon a link failure, has to perform a number of tasks before it has new valid default next hops for affected destinations. The four major tasks are: (1) detecting a failed link (if the router is adjacent to the failed link) and generating a control plane message; (2) processing of received control packets; (3) computing the new shortest path tree (SPT); and (4) updating the forwarding information base (FIB). We assume that the delay in detecting a failed link is zero since irrespective of the underlying routing architecture, all packets during this period are lost;[4] this does not make a difference in our performance comparison results. We consider the three other major contributors.

---

[4] Unless packets are duplicated along multiple paths—a design point that may be reasonable for certain kinds of traffic, but which we do not consider in this paper.

Let d_r be the time spent by a router in processing a control packet (i.e., the time between the router's receipt and forwarding of the packet). d_r (along with link latencies) dictates the propagation rate of control packets through the network. Let d_p be the delay between a router's learning of the link failure and starting a new SPT computation; d_c be the time taken to compute the new SPT; and d_u be the time taken to update the FIB. Note that, upon receiving a control packet, a router necessarily spends $D = (d_p + d_c + d_u)$ time before having new valid default next hops for affected destinations. The values of d_c and d_u depend on the router architecture, algorithms in use, the topology, and the router's location. Lacking a good model, we set these values to 0 in our simulations. However, we use $D = d_p = 50$ms [18] and $d_r = 2$ms [8, 9] for the Sprint topology. For the AS-level and router-level topologies, we use $D = d_r = 0$.

4.2.1 Failure reaction schemes

The performance of source routing protocols also depends on the *control plane* mechanism: the technique used to inform sources about the failures in the network. We describe three variants of SlickPackets design with different control plane mechanisms. We also describe three protocols—one from the SCR paradigm and two from the NCR paradigm—that we compare with SlickPackets.

Flooded-SlickPackets. Upon detecting the link failure, r_0 floods the network with a link state advertisement (LSA). This is similar to running an SCR protocol with an OSPF [21] style control plane mechanism.

Fast-SlickPackets. When r_0 receives a packet whose primary next-hop traverses l_0, it informs s about the link failure by directly sending an ICMP-style notification message to s. The rationale is that, to reduce control overhead, only sources that use l_0 in their primary paths need to be notified. Intuitively, this significantly reduces the control plane packets sent into the network.

e2e-SlickPackets. The router r_0 piggybacks the link-failure information on the packet being forwarded on the alternate path towards d, which, upon receiving this information, may inform s of the link failure. Thus, failure information is sent to the source in an end-to-end manner.

All SlickPackets schemes use the same FS selection algorithm (§3.2) and incur the delay D between learning of the failure and switching to new primary paths.

Vanilla source routing (VSR). For purposes of comparison with SlickPackets, we evaluate a simple "vanilla" source routing protocol. In VSR, each source s specifies a single shortest primary path to its destination d in the packet header. For the control plane mechanism, we use the "fast" version, where r_0 directly notifies s. After receiving the notification, s incurs the delay D before computing a new primary path. Without a valid path, packets generated during this time are queued. Packets that use l_0 in their paths will be dropped by r_0 after the link failure. However, once s has computed a new path, it resends the packets that would have been dropped, i.e., those that it sent in the time interval $[t - R, t)$ where t is the time s learned of the failure, and R is the RTT between s and r_0. Note that for some of these resent packets, there are two concurrent live copies: the resent copy that will be delivered along the new primary path, and the original copy that will be dropped when it reaches r_0. This scheme may be difficult or undesirable to implement in practice, but as an idealized VSR, it is a useful comparison.

Ideal-SafeGuard. We simulated an idealized version of SafeGuard [18], a network-controlled routing protocol that achieves fast failure reaction. SafeGuard uses the standard OSPF as the control plane substrate. In SafeGuard, r_0 immediately uses pre-computed shortest alternate paths to quickly redirect packets that it would otherwise forward along l_0. Other routers recognize redirected ("escort mode") packets and forward them along their intended alternate paths; however, until they have updated their FIBs (after delay D after receiving the LSA), these routers continue to forward "normal mode" packets along their sub-optimal paths towards l_0. In practice, the "alternative path databases," which are found to be 2 to 8 times larger than a router's intradomain FIB [18], might increase lookup latencies or be an impractical memory requirement. However, our ideal version of SafeGuard ignores these issues.

Ideal-NCR. This represents an ideal (and unachievable) NCR scheme, in which each router learns of a link failure in exactly the propagation delay along the shortest path from the point of failure to the router; and the router instantly begins forwarding packets along the shortest alternate path. Ideal-NCR is equivalent to a special case of Ideal-SafeGuard where all delays, except propagation delay, are zero (i.e, $D = d_r = 0$).

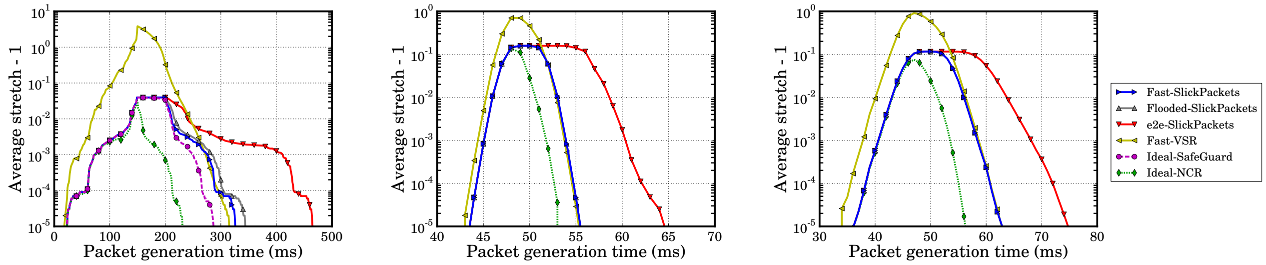
4.2.2 Methodology

We wrote a static simulator for our evaluation purposes. The simulator uses the packet stretch computations described in [22]. Since we are evaluating the reaction to single-link failures, we evaluate only (l_0, s, d) triples where the primary path from s to d uses l_0, and s and d remain connected after the failure of l_0, so that at least one alternate path to d exists for each router upstream from l_0. For the Sprint topology, we evaluate all 424,569 possible such triples. For each of the AS- and router-level topologies, we sample 1,000 random links and use a sampling algorithm (described in [22] due to space constraints) to obtain over 750,000 and 890,000 such triples, respectively.

In our simulations, the application at the source generates packets every 1ms, starting at time $t = 0$ms. For the time of link failure t_0, however, recall that in Ideal-SafeGuard, Ideal-NCR, and Flooded-SlickPackets, r_0 floods the LSA when it detects the link failure, not when it receives sources' packets. For these schemes, the sooner the link fails, the sooner intermediate routers and the source learn of the failure and use better paths. So, for a fair comparison with non-flooding schemes, we consider two extreme points: when t_0 is greater than the network diameter in terms of *link latencies* and when $t_0 = 0$. The former case ensures that by the time t_0, all sources in all evaluated (l_0, s, d) triples have had packets reaching r_0. For the Sprint topology, with a diameter of 139ms, we use $t_0 = 150$. For the AS- and router-level topologies, we assume all links have latencies 1ms and use $t_0 = 50$.

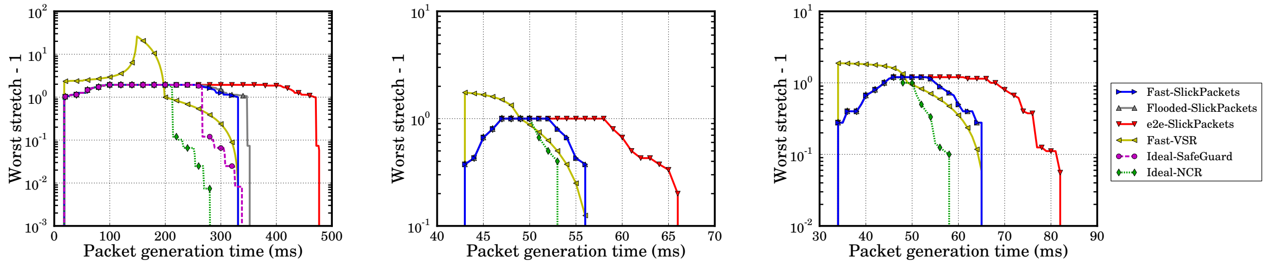
4.2.3 Results

The high-level results reveal that SlickPackets schemes (particularly the Fast and Flooded variants) achieve packet stretch comparable to that of NCR scheme Ideal-SafeGuard.



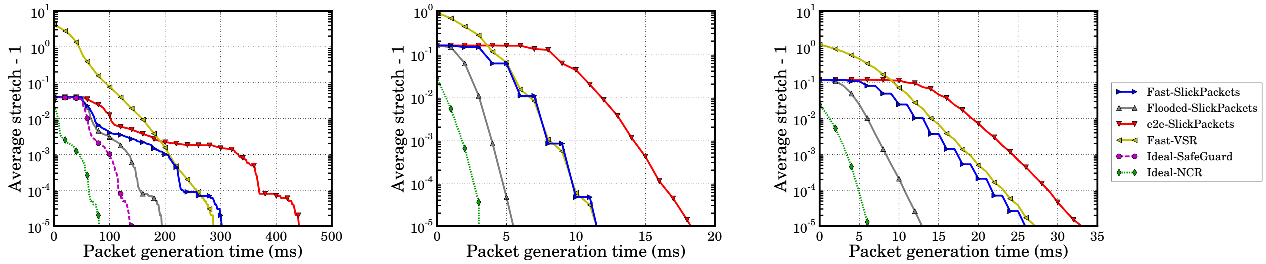
(a) Sprint Topology (b) AS-level Topology (c) Router-level Topology

Figure 6: Average packet stretch - 1 vs. packet generation time when t_0 is greater than the network diameter. The y-axes are on log scales. For the Sprint topology, $t_0 = 150, D = 50, d_r = 2$. For the AS- and router-level topologies, $t_0 = 50, D = d_r = 0$.



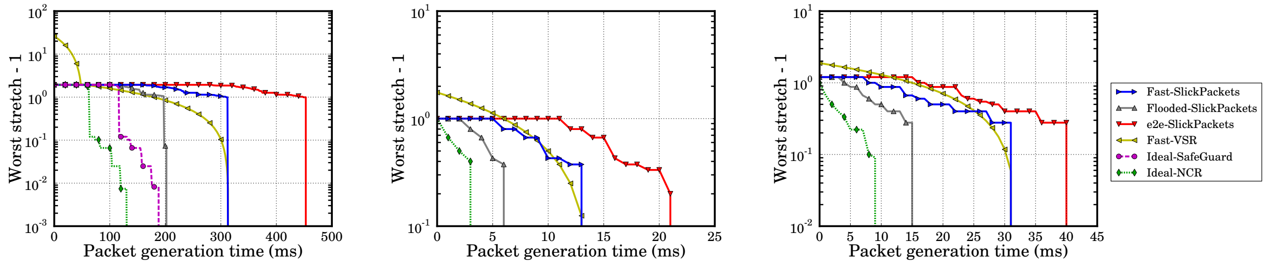
(a) Sprint Topology (b) AS-level Topology (c) Router-level Topology

Figure 7: Worst packet stretch vs. packet generation time when t_0 is greater than the network diameter. The y-axes are on log scales. For the Sprint topology, $t_0 = 150, D = 50, d_r = 2$. For the AS- and router-level topologies, $t_0 = 50, D = d_r = 0$.



(a) Sprint Topology (b) AS-level Topology (c) Router-level Topology

Figure 8: Average packet stretch - 1 vs. packet generation time when $t_0 = 0$. The y-axes are on log scales. For the Sprint topology, $D = 50, d_r = 2$. For the AS- and router-level topologies, $D = d_r = 0$.



(a) Sprint Topology (b) AS-level Topology (c) Router-level Topology

Figure 9: Worst packet stretch vs. packet generation time when $t_0 = 0$. The y-axes are on log scales. For the Sprint topology, $D = 50, d_r = 2$. For the AS- and router-level topologies, $D = d_r = 0$.

Although SLICKPACKETS schemes take slightly longer to converge compared to SafeGuard, they avoid the high packet stretch of Fast-VSR.

Average stretch. Fig. 6 shows the packet stretch averaged over all evaluated (l_0, s, d) triples when t_0 is greater than the network diameter. We first consider features common to all schemes. For a given scheme, all packets generated early in the simulation have stretch 1. Gradually, as packets generated closer to t_0, as well as more triples where s is closer to l_0, are affected by the failure, the average stretch increases. Additionally, for any triple, all packets generated after t_0 have stretch no higher than those generated at t_0; this is reflected in the average stretch over all triples.

We now compare NCR and SLICKPACKETS schemes. In NCR schemes, routers upstream from l_0, once they receive the LSA and update their FIBs, can redirect packets before they reach l_0; while in SLICKPACKETS schemes, packets have to reach l_0 before being redirected. This difference gives NCR schemes only a small advantage for early packets, especially for the Sprint topology in Fig. 6(a), because upstream routers still incur the delay D between receiving the LSA and updating their FIBs. For later packets, this advantage becomes more significant as more upstream routers update their FIBs. As expected, Ideal-NCR is the best performing scheme in all three topologies: it converges 57ms before Ideal-SafeGuard for the Sprint topology (due to $D = 50$ and $d_r = 2$) and is equivalent to Ideal-SafeGuard (not shown) in the other two topologies, where $D = d_r = 0$.

Consider the SLICKPACKETS schemes in Fig. 6(a). We see that for packets generated between $t_0 = 150$ and $t_0 + D = 200$, the average packet stretch is (1) constant within the same scheme and (2) identical across all schemes. Recall that all SLICKPACKETS schemes use the same FS selection algorithm and incur the same delay D between learning of the failure and switching to new primary paths. Thus, the only factor affecting their relative performances is the time s learns of the failure, which is determined by the relative distances among l_0, s, and d for different triples in the same scheme, and the different control schemes given the same triple. So, regardless of the (l_0, s, d) triple or the control scheme, there is a minimum window of D time where s uses the same (old) primary path. After this window, we can see that Fast-SLICKPACKETS converges slightly faster than Flooded-SLICKPACKETS because the LSAs in Flooded-SLICKPACKETS incur delay d_r at intermediate routers; in Fig. 6(b) and (c), where $d_r = 0$, Fast- and Flooded-SLICKPACKETS are identical. And both of them converge significantly faster than e2e-SLICKPACKETS as expected.

Finally, we see that in Fast-VSR, early packets experience higher stretch than in other schemes. This is because these packets are dropped and have to be resent by s. They experience on average a delay of one half the RTT between s and r_0, plus the delay D before being sent along the new path, resulting in a high stretch. However, Fast-VSR can catch up to and overtake Fast-SLICKPACKETS for two reasons. First, consider the packet sent 1ms before s learns of the failure: in Fast-VSR, it is delayed $(1 + D)$ms before being resent along the new path; while in Fast-SLICKPACKETS, the amount of time this packet traverses the original primary path only to be redirected backwards can be larger than $(1 + D)$, especially if both the primary path and alternate path contain a very high latency link. Second, consider the packet generated 1ms before s has a new primary path: in Fast-VSR, it is delayed (queued) only 1ms before being sent on the new optimal path; while in Fast-SLICKPACKETS, this packet will be sent along the original primary path and will be redirected, experiencing a higher stretch than its Fast-VSR counterpart. These two effects enable Fast-VSR to noticeably overtake Fast-SLICKPACKETS in Fig. 6(a), but in Fig. 6(b) and (c), where $D = 0$ and all links have latencies 1ms, these two effects are less pronounced.

Worst stretch. Fig. 7 shows the worst stretch of packets given their generation time, among all evaluated (l_0, s, d) triples, when t_0 is greater than the network diameter. Note that the simulation-wide worst stretches for all schemes except Fast-VSR are equal, which are 2.93, 2.0, and 2.2 in Fig. 7(a), (b), and (c), respectively. This is because all these schemes do not drop packets, so the worst stretch is that of packets that r_0 redirects, which is the same for all these schemes. We also note that for schemes that do not drop packets, the worst stretch occurs when a packet traverses the maximum possible distance along the original shortest path without reaching d, is redirected back to s, and traverses the shortest alternate path. Thus, 3 is the upper-bound stretch because the shortest alternate path cannot be shorter than the original shortest path.

For the Sprint topology in Fig. 7(a), the simulation-wide worst stretch for Fast-VSR is 27. This happens to packets sent right before $t_0 = 150$ in triples where s is close to d, so that the time duration D that these packets are delayed dominates the latencies of the original and post-link-failure shortest paths. In the AS- and router-level topologies, where $D = 0$, the simulation-wide worst stretch of Fast-VSR are 2.75 and 2.88 respectively.

When $t_0 = 0$. Fig. 8 and 9 show the results for when $t_0 = 0$. The overall behavior of each individual scheme exhibits similar patterns to when t_0 is greater than the network diameter. The differences are that the peak stretches occur for packets generated at $t_0 = 0$. Furthermore, as expected, flooding schemes benefit from the earlier time of failure: for example, for the Sprint topology in Fig. 8(a), Ideal-NCR and Ideal-SafeGuard converge further ahead of Fast-SLICKPACKETS compared to Fig. 6(a), and even Flooded-SLICKPACKETS now converges ahead of Fast-SLICKPACKETS (similarly for the AS- and router-level topologies).

In terms of simulation-wide worst stretch, those of non-flooding schemes (Fast- and e2e-SLICKPACKETS as well as Fast-VSR) are the same as when t_0 is greater than the network diameter. This is as expected because for these schemes, it is still r_0 that redirects packets and/or triggers the notification of sources. For flooding schemes, however, it can be expected that simulation-wide worst stretch would be lower compared to when t_0 is greater than the network diameter. Nevertheless, the Sprint topology contains triples where an upstream link that is close to r_0 has very high latency compared to the distance between s and r_0, so that s's first packet does not benefit from the flooded LSA: it still has to reach r_0 before being redirected. This results in the simulation-wide worst stretch of 2.93 in Fig. 8(a).

5. DISCUSSION: FORWARDING SUBGRAPH SELECTION

The SlickPackets design is agnostic to how the source selects the forwarding subgraph (FS). For example, the FS selection may be guided by demands of the application running at the source (for example, if the source is an end host) or the performance goals of a network operator (for example, if the source is an edge router). In this paper, we presented and evaluated one such FS selection algorithm: where the FS allows re-routing of packets within the network in case of single-link failures. We now discuss alternative FS selection strategies.

Handling node failures. For the FS to handle node failures, we need only a simple modification to the link-failure-avoiding FS selection of §3.2. A source s, for a given destination d, constructs the FS in three steps. First, s computes a primary path P to d by running an instance of the shortest path algorithm. Next, to protect against single *node* failures, s visits each node along P, and computes the alternate path P_i it would prefer the packet to be routed along if that node were to fail. In particular, for each node v_i on the primary path with node v_{i+1} as the next hop along the primary path, we (a) remove v_{i+1}; (b) compute a shortest path from v_i to d; and, (c) restore v_{i+1}.

Handling multiple link failures. A source may desire to construct an FS that protects against multiple link failures. This may be done by extending the scheme from §3 to construct an FS that protects from multiple edge-failures. For example, it may be sufficient to have two strategically chosen alternate paths for all nodes on the primary path. The idea is that the source can choose alternate paths that are not failure-correlated with the primary path. This may allow a much larger amount of resiliency; although the performance evaluation of such a scheme is subject to future work.

Congestion avoidance. Our focus in this work so far has been on dealing with failures. However, alternate paths in the FS may also be used to react to congestion in the network. For example, intermediate routers along the path may choose to forward the packet along an alternate path if the primary path is congested (e.g., if the interface queue for the corresponding link is filled beyond a particular threshold). Using a FS also enables the source to optionally provide control over load balancing, by providing feedback on which set of paths are tolerable for the load balancing process.

6. RELATED WORK

Our goals are related to two key areas of related work:

Failure reaction in network-controlled routing protocols. There has been much work on coping with failures in IP networks. We focus on the most closely related work: protocols that guarantee packet delivery in the presence of one or more link failures. R-BGP [16] constructs interdomain backup paths to handle single link failures, given some assumptions about routing policies. SafeGuard [18] uses a remaining path cost field in a packet as a heuristic to determine whether the path expected by the previous hop is different than the path available to the current hop. In this way, it can decide when to reroute packets along pre-computed backup paths. FCP [17] takes a different approach to determining when packets should be rerouted: each packet carries a list of the failed links it has encountered. The best backup paths are computed on the fly at routers, thus allowing FCP to be robust to multiple link failures, but requiring fairly heavyweight graph processing in the data plane. MPLS Fast Reroute [23] relies on precomputation of backup paths. In its local repair variant, an additional path is constructed to avoid each neighboring link or node, which can inflate storage requirements and will not result in lowest-stretch backup paths. As discussed in the introduction, all of the above approaches are NCR protocols, which do not permit source control of primary or backup paths. In addition, backup paths are computed or stored at every router within the network, so that there is a dependency between each router's forwarding table and the topology of the entire network.

One way to get a small amount of route control at the source within an NCR architecture is to use multihoming: the source can then select between several providers [4]. This could be used to enable some source control, while still applying the NCR resilience techniques described above. However, this provides only a very limited amount of control to the source, and does not yield the full benefits of source control described in the introduction. Moreover, if many sources are multihomed, this vastly increases routing state within the network, since each router would be required to know about every point of multihoming attachment if we desire to provide alternate paths that avoid a failure of one of these links.

Our use of routing along FSes was inspired by [19], which argues that a directed acyclic graph is a better forwarding architecture than the more traditional shortest-path tree. While [19] focuses on improving NCR schemes, we target achieving the benefits of both network- and source-controlled routing. Additionally, while [19] will deliver every packet even during link failures, it does not guarantee the latency that these packets will have. SlickPackets can guarantee that for single-link failures, packets will follow the shortest alternate path from the point of failure to the destination.

Source routing. There is also a large body of work on source controlled routing, ranging from dynamic source routing in wireless networks [15] to future interdomain routing architectures [10, 20, 30, 31]. Two of these, Routing Deflections [31] and Path Splicing [20], target fast re-routing within the network. Both use path label bits set by the source to pseudorandomly select a next hop at each router or AS. In [20], pseudorandom forwarding can lead to forwarding loops. In [31] routers follow certain rules that ensure loop-freedom, but reduce path diversity.

There are three important differences between [20, 31] and SlickPackets. First, [20, 31] do not fully support source control over primary or backup routes; although sources can select among some set of paths, they cannot tell which paths they are selecting. Second, although packets can be rerouted quickly within the network after a link failure, this is not guaranteed (packets may be dropped), and the backup paths are not guaranteed to have optimal latency. Third, [20, 31] are similar to traditional NCR schemes in terms of the state in the network; indeed, [20] increases forwarding table size because each router stores multiple next-hops for each destination. In contrast, SLICKPACKETS enables source control, can guarantee resilience[5] to single-link failures with packets sent along the shortest alternate path from the point of failure to the destination, and requires only local state at routers.

Giving sources control over constructing end-to-end paths introduces a number of practical questions, for example in terms of policy compliance, security, and scalability of disseminating topological state. For these questions, we rely on previous work (e.g., [10,30], and citations within), which provide solutions to these problems.

7. CONCLUSION

In this paper, we presented SLICKPACKETS, an approach to routing that attains failure reaction, while simultaneously retaining the benefits of source routing. SLICKPACKETS works by compactly encoding a set of alternate paths into data packet headers as a *directed acyclic graph*. Towards this goal, we provide simple algorithms for computing efficient graphs, and for encoding them into packets in a manner that can be processed by intermediate routers in an efficient manner.

One major area left for future work is to evaluate the complexity of implementing SLICKPACKETS in production routers, and achievable packet forwarding rates; a key challenge here is dealing with increased header size. A promising avenue for evaluation is the Supercharged PlanetLab Platform [26], a network processor-based platform on which John DeHart has implemented a prototype version of SLICKPACKETS.

This work was supported by National Science Foundation grant CNS 10-40396.

8. REFERENCES

[1] CAIDA's router-level topology measurements. `http://www.caida.org/tools/measurement/skitter/router_topology/`.

[2] Rocketfuel: An ISP topology mapping engine. `http://www.cs.washington.edu/research/networking/rocketfuel/`.

[3] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proc. ACM WWW'07*, pages 835–844, May 2007.

[4] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh. A comparison of overlay routing and multihoming route control. *ACM SIGCOMM*, 34(4):93–106, 2004.

[5] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. 18th ACM SOSP*, October 2001.

[6] M. Caesar, M. Casado, T. Koponen, J. Rexford, and S. Shenker. Dynamic route computation considered harmful. *ACM SIGCOMM Computer Communication Review*, 2010.

[7] D. Clark, J. Wroclawski, K. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow's Internet. In *SIGCOMM*, 2002.

[8] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving sub-second IGP convergence in large IP networks. *SIGCOMM Computer Communications Review*, 35:35–44, 2005.

[9] J. Fu, P. Sjodin, and G. Karlsson. Intra-domain routing convergence with centralized control. *Computer Networks*, 53, 2009.

[10] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *ACM SIGCOMM*, 2009.

[11] 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. OSDI*, 2004.

[12] J. Hershberger and S. Suri. Vickery prices and shortest paths: what is an edge worth. In *IEEE FOCS*, 2001.

[13] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, M. Luckie, kc claffy, and C. Shannon. The ipv4 routed /24 as links dataset, November 2010. `http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml`.

[14] G. Iannaccone, C.-N. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot. Analysis of link failures in an IP backbone. In *IMC*, 2002.

[15] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile computing*, pages 153–181, 1996.

[16] N. Kushman, S. Kandula, D. Katabi, and B. Maggs. R-BGP: Staying connected in a connected world. In *NSDI*, 2007.

[17] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. *SIGCOMM Comput. Commun. Rev.*, 37(4):241–252, 2007.

[18] A. Li, X. Yang, and D. Wetherall. Safeguard: Safe forwarding during route changes. In *Proc. ACM CoNext*, December 2009.

[19] J. Liu, J. Rexford, M. Schapira, S. Shenker, and J. Naous. Routing along DAGs, 2010. http://www.cs.berkeley.edu/∼liujd/RAD.pdf.

[20] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path splicing. In *ACM SIGCOMM*, 2008.

[21] J. Moy. *OSPF: Anatomy of an Internet Routing Protocol*. 1998.

[22] G. T. K. Nguyen, R. Agarwal, J. Liu, M. Caesar, P. B. Godfrey, and S. Shenker. Slick packets. Technical Report, UIUC, April 2011.

[23] P. Pan, G. Swallow, and A. Atlas. Fast reroute extensions to RSVP-TE for LSP tunnels. In *RFC4090*, May 2005.

[24] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker. On selfish routing in Internet-like environments. In *Proc. ACM SIGCOMM*, pages 151–162, 2003.

[25] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: Informed Internet routing and transport. In *IEEE Micro*, January 1999.

[26] J. Turner, P. Crowley, J. DeHart, A. Freestone, B. Heller, F. Kuhns, S. Kumar, J. Lockwood, J. Lu, M. Wilson, C. Wiesman, and D. Zar. Supercharging planetlab: a high performance, multi-application, overlay network platform. *ACM SIGCOMM*, 2007.

[27] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush. A measurement study on the impact of routing events on end-to-end internet path performance. *SIGCOMM Comput. Commun. Rev.*, 36(4):375–386, 2006.

[28] D. Wendlandt, I. Avramopoulos, D. Andersen, and J. Rexford. Don't secure routing protocols, secure data delivery. In *HOTNETS*, 2006.

[29] W. Xu and J. Rexford. MIRO: Multi-path Interdomain ROuting. In *SIGCOMM*, 2006.

[30] X. Yang, D. Clark, and A. Berger. NIRA: a new inter-domain routing architecture. *IEEE/ACM Transactions on Networking*, 15(4):775–788, 2007.

[31] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *ACM SIGCOMM*, 2006.

---

[5] Unless, of course, no alternate path exists.