Source Selectable Path Diversity via Routing Deflections

Xiaowei Yang
University of California, Irvine
xwy@ics.uci.edu

David Wetherall
University of Washington
djw@cs.washington.edu

ABSTRACT

We present the design of a routing system in which end-systems set tags to select non-shortest path routes as an alternative to explicit source routes. Routers collectively generate these routes by using tags as hints to independently deflect packets to neighbors that lie off the shortest-path. We show how this can be done simply, by local extensions of the shortest path machinery, and safely, so that loops are provably not formed. The result is to provide end-systems with a high-level of path diversity that allows them to bypass undesirable locations within the network. Unlike explicit source routing, our scheme is inherently scalable and compatible with ISP policies because it derives from the deployed Internet routing. We also suggest an encoding that is compatible with common IP usage, making our scheme incrementally deployable at the granularity of individual routers.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.2 [**Computer- Communication Networks**]: Network Protocols—*Routing Protocols*

General Terms

Design, Algorithms

Keywords

Routing deflections, path diversity, source routing

1. INTRODUCTION

Source routing, in which end-systems partially or fully specify the paths taken by their packets, is the basis of a variety of schemes to improve the reliability and performance of networks. For example, the Detour study [17] and RON overlay [1] show that "loose source route" style concatenations of default Internet paths may possess lower latency or greater available bandwidth. Similarly, SOSR [7] demonstrates that routing via a random point of indirection can mask many Internet failures. And Perlman's work on sabotage-proof routing [13, 14] depends at its core on the ability of sources to select their own routes to find one that works correctly. Source routing is a fundamental means of improving communications because it provides path diversity that reduces the dependence on a single network path with undesirable characteristics.

Despite these advantages, source routes are not in mainstream use in the Internet today, perhaps due to several associated problems. They do not scale to permit widespread use (except in trivial applications) because each end-system needs some map of the overall network to formulate its preferred routes. Yet detailed, up-to-date maps do not readily exist, and even simple lists of locations for indirection are complicated by the need to maintain availability and spread load. By letting users specify paths, source routes do not fit the Internet model in which ISPs set routing policy based primarily on destination addresses. And in some forms, such as the IP loose source route option, they pose a security threat and as a result are often disabled.

In this paper, we revisit source-controlled routes. Our goal is to find a design that provides much of the benefit of explicit source routes but addresses the problems we have identified so as to remain practical. Our insight is that, to be useful, it is not necessary for the end-system to specify which of the exponentially many possible routes to take. Instead, it is sufficient to provide a small set of diverse paths and let the end-system select from them. This is because many benefits of source routing stem from path diversity. An end-system can then test different paths without knowing the routes to which they correspond; even if the end-system did know the path it would often need to test it for reliability or bandwidth. And, as work on source routing for reliability has shown [7], simple tests are sufficient to solve problems that depend on avoiding a bad path rather than finding the optimal path. Thus, while this small set of diverse paths is less flexible than arbitrary source routes, we are willing to adopt it in exchange for a practical scheme.

Our approach to construct these diverse paths draws on deflection routing and hot-potato routing, in which routers forward packets off the shortest path when it is not available [11, 2]. We develop *routing deflection* rules that enable routers to independently deflect packets and thereby collectively construct a diverse set of paths. Our rules exercise the latitude routers have to forward packets off the shortest path yet maintain loop-free connectivity. For instance, a well-known rule (on which we will improve) is that any router can safely deflect packets to a neighbor that has a smaller cost to reach the destination. Then, sources access this path diversity by supplying a hint that affects the choice of deflection. Because routing deflections build on the shortest path machinery and do not alter its character, they scale well and fit the Internet model of routing that is based on destination addresses and ISP policy. They are also incrementally deployable at routers within and across ISPs because different routers do not need to coordinate their deflection decisions.

SIGCOMM'06, September 11–15, 2006, Pisa, Italy.

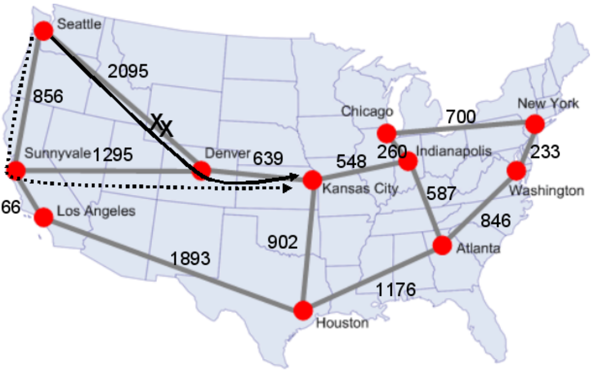


Figure 1: The Abilene backbone network. Numbers give the link weights, which are symmetric. The solid line between Seattle and Kansas City shows the lowest-cost route. The dotted line via Sunnyvale shows a deflected route, which avoids the Seattle-Denver link. The map and weights were taken from the Abilene Observatory (http://abilene.internet2.edu/observatory/) on Dec 5, 2005.

In the body of this paper, we present a design that provides sources access to path diversity via routing deflections. We then evaluate our design on real, measured and random network topologies. We find that, by using it, sources are very likely to have enough diversity to avoid an undesirable node, link or peering point. We make two contributions. The first is architectural: the use of end-system tags to select path diversity as an alternative to explicit source routes. Our tags are compact (10-bit in our design) and do not have global meanings. We show how they can be encoded in a way that is compatible with common IP usage as well as carried more cleanly in a shim protocol layer. Routers can use whatever mechanism is preferred to bind these tags to diverse paths, e.g., MPLS tunnels or routing deflections. The second contribution is the design of routing deflections, and in particular two new rules for constructing diverse paths that we prove to be loop-free despite independent choices at different routers. These rules are similar to local route repair mechanisms [25, 19, 8] but more general in the sense that concurrent deflections can be made, e.g., to bypass multiple failures, in arbitrary topologies and without the danger of loops. That is, routing deflections are akin to multi-path routing schemes.

The rest of this paper is organized as follows. In Section 2 we motivate routing deflections and path selector tags with an example. In Section 3 we present deflection rules that generate alternate paths within ISPs; we prove them to be loop-free in the appendix. In Section 4 we describe how path selector tags are used for routing. In Section 5, we extend our rules to paths across multiple ISPs. In Section 6, we evaluate the combination of path selector tags and deflection rules on various network topologies. We then discuss related work and conclude.

2. DIVERSITY VIA DEFLECTIONS

The key idea of this paper is that a diverse set of end-to-end paths may be constructed by allowing the routers of a network to individually "deflect" packets by forwarding them off the known shortest path; end-systems can then select from the available paths by labeling their packets with hints. The design we present here realizes this idea with two components: 1) deflection rules that determine which neighbors of a router can be used to forward a packet; and 2) a signaling mechanism that lets end systems control which of the available paths routers use for a given packet. In this section, we use a motivating example to explain the concept of deflections within a single ISP. We describe the components in the following sections, working up to deflections that change the selection of peering points across multiple ISPs.

2.1 Example

Figure 1 shows the backbone of Abilene, a US-based research and education network, complete with link weights. We omit the intra-POP details for simplicity. Consider packets sent from Seattle to Kansas City. The lowest-cost route (solid line) begins by sending the packets to Denver. However, observe that as an alternative (dotted line) it is possible for the Seattle router to instead forward the packets to Sunnyvale. This is because the remaining cost to reach the destination falls, and so the packets will still arrive at Kansas City without the possibility of a loop. Moreover, multiple routers along the path can safely deflect to neighbors in this manner; the cost to reach the destination will fall at every step and so the destination must be reached eventually.

The example demonstrates a routing deflection rule that works for all topologies: each router can deflect to any neighbor with a lower cost to the destination than itself and the result will be a loop-free path to the destination. In our example, the alternate path might be useful to avoid the Seattle-Denver link if it were congested, had a relatively high error rate, etc. To allow end-systems (rather than routers) to choose between the available paths, we tag packets with a path selection hint. For instance, a tag of 0 may indicate the lowest-cost path, and a tag of 1 an alternate path. The source does not need to name any intermediate router to use these paths.

The above rule is well-known and works for our simple scenario. However, it may not work when intra-POP structure is considered, as small uphill hops may be needed to switch egresses within a POP [8]. And it is not sufficient to construct many desirable alternate paths. Suppose, for instance, that we wished to avoid the Denver node entirely. Then it would be necessary to reach Kansas City on the lower route via Los Angeles and Houston. However, Sunnyvale cannot safely deflect to Los Angeles because it will loop the packet back. In the next section we give stronger routing deflection rules that can, for example, forward along this lower route to avoid Denver entirely.

2.2 Advantages

Routing deflections are conceptually simple, yet they generalize shortest-path routing. With shortest-path routing, a packet may be forwarded to any one of multiple equal lowest-cost neighbors. Similarly, with routing deflections, a packet may be forwarded to any neighbor in a larger *deflection set* that is computed according to the specific deflection rule. We restrict our attention in this paper to sets that are computed via the shortest-path machinery and that include the lowest-cost neighbors, though other kinds of deflection would be possible. With shortest-path routing, each router may independently decide which packets to forward along which of the equal-cost routes as a local matter, without causing loops. Correspondingly, each router may decide its own deflection as a purely local matter without loops. Loop-freedom is important in our context because we allow end-systems to select paths even when there are no transient repair events. So if deflections were to cause loops, they would be persistent steady-state loops that may disrupt connectivity.

Deflections have several other desirable properties by design. First, deployment is trivial because deflection choices are compatible with lowest-cost routing: individual routers can be upgraded across multiple ISPs with no need for coordination. Moreover, end systems need not know about the network topologies in order to explore alternative paths. This leads us to suggest how to carry tags in a manner that is compatible with common IP usage.

Second, deflections scale to real-world usage. This is because they are simple extensions of the shortest-path routing machinery that is already deployed at ISPs. They do not require additional messages in the manner of other source routing schemes [27, 15, 28, 6, 4, 3]. And they require no more than a constant factor of added computation. For example, our rule above only requires information about its neighbors' costs to compute its own deflection set. These costs are either already signaled in a distance-vector routing protocol or can be easily computed in a link-state protocol.

Third, deflections are highly robust because they inherit the failure tolerance of distributed routing algorithms. No centralized process is used to compute deflection routes, and hence they work as well as shortest-path routing when the network is partitioned.

2.3 Applications

Our focus in this paper is on how to provide hosts with access to a diverse set of Internet paths. However, deflections are likely to be useful in other contexts such as local route repair, in which a failure is masked while new global routes converge. For instance, deflection paths could be selected by routers, rather than end-systems, to locally bypass faults while news of their existence is globally suppressed. By construction, this would allow multiple faults to be bypassed without loops, whereas most local repair schemes [25, 8, 19] target the common case of a single failure and may form transient loops in other cases. However, deflections are not a complete solution as they do not address transient loops due to inconsistent forwarding tables. Also, the price for their guarantee of loop-freedom is that they may not be able to bypass as many single faults as schemes with weaker guarantees. Section 7 compares deflections with specific schemes for local route repair.

3. DEFLECTION RULES

In this section, we present our deflection rules from the viewpoint of a single ISP network. We describe how deflections are extended across multiple ISPs in Section 5.

Each rule generates a *deflection set* of neighbors that a router can use to reach particular destinations. Routers can then independently select any neighbor in their deflection sets to use for forwarding; we describe how hosts can tag packets to influence this selection in the next section. We define our rules in terms of shortest-path costs. Routers can compute the various shortest path costs as an extension of whatever routing protocol they run to provide base routing, be it OSPF, ISIS or a distance vector style of protocol such as RIPv2 or EIGRP.

For each rule, the key issues we must consider are the correctness of its deflections and how effective they are at providing diversity. By correct we mean that paths are loop-free (a safety condition) and reach the destination (a liveness condition). We prove the correctness in the appendix for arbitrary topologies with multiple equal-cost paths and asymmetric link costs. We study effectiveness via simulations as part of our evaluation in Section 6.

For all rules, we let n_i for $i \geq 0$ be the sequence of nodes along a path, and let $cost(n_i)$ be the shortest path cost to reach a given destination from node n_i, by whichever neighbors are on the shortest path. We omit the destination in the cost function, since it does not change.

3.1 Rule 1 (One Hop Down)

Our first rule was used to motivate deflections, and serves as a strawman for assessing the strength of our other rules: a router can send to any neighbor provided that the neighbor has a lower cost to reach the destination. More formally:

Rule 1 (One Hop Down): The deflection set for a node n_i is those neighbors n_{i+1} for which $cost(n_{i+1}) < cost(n_i)$.

Intuitively, Rule 1 is loop-free to destinations because the cost to a destination at each node is strictly decreasing, and will eventually become zero. Lowest-cost forwarding or Equal Cost Multiple Path (ECMP) forwarding is a special case of Rule 1. We prove the correctness of Rule 1 in Appendix A.

Rule 1 is simple to implement at routers. To run Rule 1, each node needs to obtain costs for its neighbors as well as itself. With a distance vector protocol the cost information is already signaled between neighbors. With a link-state protocol, it requires multiple shortest-path computations, but does not require additional routing messages. These computations may be run in the background since shortest-path routes already provide basic connectivity.

Rule 1 is also trivial to deploy in an ISP network on a per router basis: observe that Rule 1 generalizes shortest path routing by including the shortest path neighbors in its deflection sets. As a result, any mixture of routers following either Rule 1 or shortest path routing are loop-free and reach the destination.

3.2 Rule 2 (Two Hops Down)

The first rule provides greater diversity than shortest-path routing, but it is limited because sometimes there will be very few choices that cause cost to decrease. Our next rule provides greater flexibility. It includes all choices allowed by Rule 1 plus that it allows the cost to a destination to increase temporarily provided that the cost decreases sufficiently on the next hop.

Rule 2 (Two Hops Down): The deflection set for a node n_i is those neighbors n_{i+1} for which either of these conditions apply, subject to the two caveats that follow:

1. $cost(n_{i+1}) < cost(n_i)$ [downhill]
2. $cost(n_{i+1}) < cost(n_{i-1})$ [two-hop]

We remove the incoming interface n_{i-1} from the deflection set unless the set would otherwise be empty, and we expand the deflection set when n_i is the initial node by treating $cost(n_{i-1})$ as infinity. Both are optimizations. The former case prunes the uninteresting deflection, in which a packet needlessly returns to a neighbor only to take a different deflection. The latter case exploits a situation that permits all neighbors to be in the deflection set.

To see the power of this rule, reconsider our example. Recall that in sending from Seattle to Kansas City, Rule 1 could not take a long round route via Los Angeles and avoid Denver entirely. But Rule 2 can. Los Angeles is a valid deflection, even though it normally sends via Sunnyvale, because of the two-hop clause: its cost does not rise as high as Seattle, which is two hops back. Continuing on, Los Angeles can then forward to Houston using the downhill clause, as the cost falls after crossing an expensive link.

In Appendix B, we prove Rule 2 to be loop-free in the sense that a directional link can be crossed at most once in a deflection path. Intuitively, on the forwarding path, the cost to a destination must strictly decrease at every two hops. No two-node sequence can repeat. Hence, no link-level loop exists. Note that it is possible for a node to be visited more than once with this rule. We do not consider this to be a problem because a packet will not be queued twice for the same interface, and it is interfaces that are the key underlying resource. Rule 2 also satisfies the liveness property because the deflection set always contains the shortest path neighbor. Therefore, a packet will eventually reach its destination.

Rule 2 has a slightly higher implementation cost than Rule 1. Similar to Rule 1, each node must obtain costs for neighbors. In addition, forwarding decisions depend on the incoming link (or previous router) as well as the destination. This is similar to the way in which routers use source addresses or incoming links to forward along equal-cost paths and multicast paths [22].

As before, mixtures of nodes that follow Rule 2 or shortest-path routes provide loop-free routes to their destinations. This follows because the shortest path neighbor satisfies the deflection rule.

3.3 Rule 3 (Two Hops Forward)

With Rule 2, it is possible that a node will send uphill to a neighbor that has no alternative but to return the packet. We now construct a new rule that always provides an alternative to immediate backtracking in the hope that it will increase diversity.

To state Rule 3, we define the following terms. In the forwarding path, we let l_i denote both directions of the incoming link used to reach n_i, i.e., the link connecting n_{i-1} and n_i. Now we let G be the overall network graph and $G \backslash l_i$ be the same graph with the link l_i removed. These other graphs with incoming links removed are the key to our rule. To use them, we also need to extend the cost function to include the graph to which it applies, i.e., $cost(G \backslash l_i, n_i)$ is the shortest path cost from the node n_i to a given destination in the graph $G \backslash l_i$.

Rule 3 (Two Hops Forward): The deflection set for a node n_i is comprised of the neighbors n_{i+1} for which $n_{i+1} \neq n_{i-1}$ and either:

1. $cost(G \backslash l_{i+1}, n_{i+1}) < cost(G \backslash l_i, n_i)$ [downhill]

2. $cost(G \backslash l_{i+1}, n_{i+1}) < cost(G, n_{i-1})$ [two-hop]

Rule 3 eliminates the need of immediate backtracking, because if n_{i+1} receives a packet from n_i, then $cost(G \backslash l_{i+1}, n_{i+1})$ must not be infinity. It implies that n_{i+1} must have a path to reach the destination without using the backtracking link l_{i+1}. Therefore, we can safely remove the backtracking node n_{i-1} from the deflection set for all n_is.

The first clause compares $cost(G \backslash l_{i+1}, n_{i+1})$ with $cost(G \backslash l_i, n_i)$ rather than $cost(G, n_i)$, which might be simpler. This is because $cost(G \backslash l_i, n_i)$ is the larger quantity and leads to a larger deflection set; if we used $cost(G, n_i)$ then Rule 3 would actually have been a subset of Rule 2. With our rule as stated, neither Rule 2 nor Rule 3 are subsets of each other, but Rule 3 does allow paths that Rule 2 does not. In Figure 1, for example, the path Indianapolis, Kansas City, Huston, Atlanta, is a valid deflection path via Rule 3, but not via Rule 2. This is because excluding the backtracking link forces the shortest path to follow a more roundabout path to the destination.

The cost of this increased flexibility of Rule 3 is a slight increase in the implementation complexity over Rule 2. Specifically, a node must now compute costs for its neighbors in related graphs rather than the same graph. This might be done incrementally in a link-state implementation. And curiously, as before, we observe that distance vector protocols can already signal the required information. This is because costs from all neighbors but one can be used to derive costs in the graph with the link to the one neighbor removed.

Implementation considerations also led us to the asymmetry in Rule 3. We deliberately do not use the graph $G \backslash l_{i-1}$ in the two-hop clause, as might be expected, even though it would result in correct paths. This is because the current node will not in general know the incoming link of the previous node unless it is signaled with the packet.

As a variant of Rule 3, we observe that we can define an analogous rule by removing the incoming node (rather than link). This places stronger connectivity demands on the underlying topology: it ensures that there is a path to the destination that does not return to the previous node, rather than one that does not re-cross the previous link. We have found that it produces otherwise similar results, and so omit it from our evaluation due to space limitations.

The correctness proof of the link version of Rule 3 is given in Appendix C. It is similar to that of Rule 2, and the node version follows by analogy. We also show that Rule 3 is compatible with shortest-path routing and hence trivially deployable.

4. TAG ARCHITECTURE

In this section, we describe a tag routing architecture that provides end-systems with path diversity. Each packet carries a tag that determines the path it takes through the network from the present location to the destination[1]. Thus tags act as selectors across a set of network paths. They are an alternative to explicit source routes selected by hosts and label-switched paths selected by ISPs. To describe our architecture, we begin with tags themselves, and then present two design variants: a shim protocol that fits between IP and higher protocol layers and cleanly signals tags; and an encoding of tags into IP packets that is compatible with common IP usage.

4.1 Tag Properties

We require that tags have several properties to render them useful and practical for path selection. First, tags must be consistent in their path selections to the same extent as existing Internet routes. This allows end-systems to systematically explore the tag space and avoids adverse interactions with existing transports, e.g., packet reordering slows TCP.

Second, tags are opaque and lack global meaning except that we require a value of zero to correspond to the default Internet path. For other tag values, each ISP selects a path through its network, without the requirement that it communicate the choice of paths to end-systems or other ISPs. This means that tag routes are policy compliant in the same manner as default routes, since each ISP will apply its policies by definition. It also means that end-systems must send packets along tagged routes to discover them.

Third, different tags should select a diverse set of network paths. By this we mean that union of all paths that can be selected between a source and destination covers a region of the network that is significantly larger than the default Internet path (were tags not used). This implies that it will usually be possible to avoid an undesirable portion of the default Internet path. However, we do not require that different tags select different paths. This makes it easier to construct tag paths. It also enables incremental deployment because ISPs that have not been upgraded can be viewed as trivially mapping all tag values to the default path.

Given the above properties, tag routes are useful for applications that benefit from diverse paths, such as routing around the location of a fault that might be a lossy link, point of congestion, Byzantine failure, low capacity link, or high delay link. For this kind of application, it matters little that the route is not known a priori. This is because finding a good route will typically involve testing an alternative path to check that it does not suffer from loss, delay or bandwidth problems, etc., regardless of whether the route is explic-

[1] Clearly, tags could be defined for a connection-oriented network too. Here we focus on extending IP.

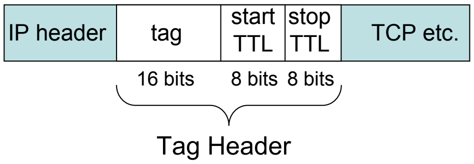


Figure 2: Shim Encoding. A shim header is inserted between IP and the next higher layer to carry the tag and TTL range for which deflections are enabled.

itly known. In these applications there tend to be many paths that are acceptable, e.g., any path that avoids the fault, so that trying paths is a reasonable strategy. Conversely, tag routes are less suitable for sending packets along particular routes to satisfy security or QoS policies, since the route must be found by trying tag values.

4.2 Shim Layer Tag Encoding

Figure 2 shows how packets are tagged using a shim protocol layer that sits between IP and the next higher layer. Tagging is a simple insertion of two pieces of compact, fixed-length information. First, the tag itself is carried in the clear. Ten bits (selecting among 1024 paths) are sufficient, and we round this up to 16 bits for convenience. Note that this tag size is much smaller than the (exponentially large) number of possible source routes. However, there is no compelling reason to make the tag size large since, given the properties of tags, it must be searched by trial and error. Second, range information is carried in the form of a start TTL and stop TTL values. This range signifies the portion of the path for which tag selection is to be used. It enables the end-system to narrow the region of faults, e.g., if a fault can be bypassed with tags operating on the last half of a path then it must lie in the last half of the path.

Given this encoding, tags are used at routers as follows. First, the current TTL is checked to see if it lies within the range. If not, or if the tag is zero, the default route is used. We also use the default route for IP fragments because only the initial fragment will contain the shim header. Otherwise, the tag is used to select a possibly alternative path. This begs the question of how ISPs map tag values to paths. We give a procedure for doing so later in this section, for the case in which the diverse paths are provided by deflections. However, ISPs could use any method they prefer, e.g., to map tags to MPLS [16] paths if they are available.

4.3 IP Tag Encoding

We observe that it is possible to carry tag information on IP packets themselves by overloading IP header fields in a manner that is compatible with common IP usage. The information is then used at routers in the same manner as for the shim protocol. A sample encoding is shown in Figure 3.

We use 10 of the IP identifier bits to encode the tag, setting the remaining 6 bits to a well-known flag pattern. We then use TTL values to define the tag selection region, carrying it implicitly by setting the initial TTL instead of separate start and stop TTL values. To do this, we define a rarely used portion of the TTL space to indicate that tag selection should be used. Common initial TTL values include 30, 32, 60, 64, 128, and 255, and Internet paths rarely exceed 40 hops [26]. This means that TTL values between 128 and 215 are rarely seen in practice. We define the range 160 to 200 to indicate that tag selection should be used. Hosts can then set their initial TTL value such that tag selection is applied to the entire path (by using 200), only the end of the path (by using values > 200) or only the beginning of the path (by using values > 160 and < 200). Other initial TTL values, including those in common use, will cause tag selection to be turned off and the default path to be followed. Note that this scheme is not as flexible as the shim protocol, where an arbitrary subpath can be used for tag selection, but we will see that it is sufficient to provide useful diversity.

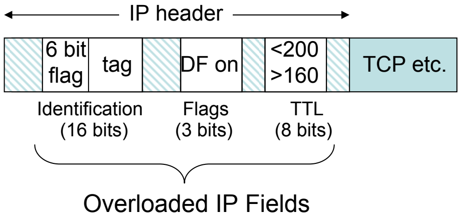


Figure 3: IP Encoding. The tag is carried in a subset of the Identification field, deflections are switched on for a range of TTL values, and fragments are not deflected.

The advantage of this overloading is that it enables true incremental deployment. Individual hosts and ISP networks can be upgraded to use tag selection independent of all other parties; the deflection rules we consider in the next section allow routers within an ISP to be individually upgraded too. An upgraded host can then use tags for path diversity even when communicating with a host that has not been upgraded.

The disadvantage of this method is that, like all such schemes, no overloading of IP is entirely backwards-compatible. In our case, the small fraction of hosts that do use TTLs within the tag selection range will have a small fraction of their packets re-routed (when the IP identifier contains the flag value). This will not cause a loss of connectivity, but may degrade performance. A further issue is that traceroute cannot be used to trace tag paths because the TTL has been overloaded. Finally, note that there are other proposals to overload the IP identifier field for incremental deployment (e.g., CSFQ [21] and IP traceback [18]) that, if adopted, would not be compatible with our usage. Nonetheless, while we do not claim it is the best that can be found, this encoding shows it is possible to provide tag routing with a high degree of backwards-compatibility.

4.4 Mapping Tags to Deflections

When deflection routing is in use, we use the following procedure to map tags carried on packets to choices in the deflection set at a router.

Tag Mapping Procedure: Let the deflection set at a router given by a rule contain K members. Number these members pseudo-randomly, starting with zero for the default shortest cost neighbor. Let the router also pseudo-randomly choose a small prime number P from the first few primes (e.g., the first 10) greater than or equal to K. Given a tag value of T on a packet, the router should forward to the member of the deflection set identified by number $N = (T \ mod \ P) \ mod \ K$.

This rule uses modulo arithmetic to pick from the deflection set. The outer mod operation produces a number in the right range. The randomization is used to avoid correlated choices at different routers. The purpose of the inner mod operation is to produce a further degree of freedom. It converts the input tag into an effective tag value that is different for routers that chose different primes. In this manner, the same tag values can be found to make different choices at different routers, even when the routers have deflection sets of the same size. We find the inner mod operation to be valuable in terms of path diversity as part of our evaluation (Section 6).

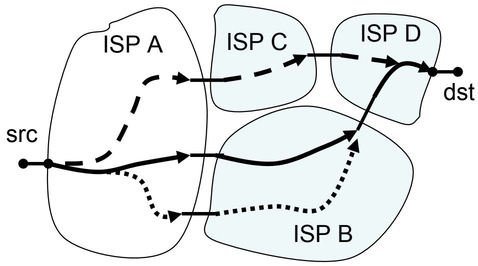


Figure 4: **Inter-domain deflections can switch peering points and ASPATHs. The solid line from source to destination shows the default path via ISPs A-B-D. The dotted line shows a deflection that changes the peering point but not the ASPATH. The dashed line shows a deflection that changes both peering point and ASPATH to ISPs A-C-D.**

5. INTER-DOMAIN RULES

We now show how to extend deflections across multiple ISPs to provide peering point diversity. It is straightforward for each ISP to independently use deflections to route to external IP prefixes advertised by BGP via one or more peering points. Deflections may then change the ISP egress point (and hence next ingress point) compared to default routes.

For inter-domain routing, we consider each ISP in isolation. We assume that all routers in the ISP run BGP (with some policy chosen by the ISP) as well as an IGP protocol such as OSPF or ISIS. Each router then forwards packets on the shortest path to the IP nexthop of the path selected by its BGP decision process. This will often result in different routers in the ISP sending to different egress points for a given destination because IGP cost metric is included in the BGP decision process, e.g., early-exit routing. Thus the complication for our deflection rules is that the default egress point and hence cost metric for a destination may change unexpectedly when the packet is deflected.

We can handle this complication to avoid intra-ISP loops by extending the cost function. Revealing the destination parameter that has been implicit, the cost metric so far has the form $cost(n, dst)$. To capture the BGP decision process, we simply extend it to have the form $cost(n, nexthop(n, dst))$. Here, $nexthop()$ models the BGP decision process that selects the IP nexthop to a destination IP address dst. This decision process is part of BGP implementation and should be the same for all routers in one ISP to prevent routing inconsistencies. A node n can compute the $nexthop()$ of any neighbor as the inputs to the $nexthop()$ function: the BGP route advertisements to the destination dst and the IGP costs to the candidate nexthops in the BGP advertisements, are both known to n. The former is known from iBGP relay sessions and the latter from IGP. With this nexthop information, a node n_i can compute the costs $cost(n_{i\pm1}, nexthop(n_{i\pm1}, dst))$. They are all a node needs to know to compute a deflection set (Section 3). With this extension of cost, all proofs in the appendix go through unchanged.

Inter-domain deflections provide two benefits in terms of path diversity. Both are shown in Figure 4. First, the peering point used between two ISPs may change; deflections are not limited in their diversity to intra-ISP changes. This can occur because a packet heading for a given peering point may deflect to a router that prefers a different peering point. In fact, the presence of multiple peering points will tend to increase the deflection potential.

Second, and more generally, the next ISP may be changed. This is because the BGP decision process that is run at each router chooses the ASPATH and nexthop jointly; BGP does not bind an entire ISP to an ASPATH before choosing nexthops. An important consideration in this case, given that deflected paths are loop-free within individual ISPs, is that they remain loop-free across multiple ISPs. This will continue to be the case when ISPs use "prefer-customer" and "valley-free" routing policies, as is the common case. These policies mean that any router of an ISP will only choose an egress point that advertises the most preferred ASPATH, barring inter-ISP loops as long as there are no customer-provider loops. Interestingly, it is possible that the deflected ASPATH is one that was not advertised downstream, but is nonetheless policy compliant by its construction. For instance, in Figure 4, the advertised ASPATH to dst by ISP A is ISPs A-B-D, but the deflected one may be A-C-D.

6. EVALUATION

In this section we simulate our tag architecture and deflection rules to characterize the kinds of path diversity that they provide. A high degree of path diversity is desirable to increase the ability of a source to avoid faulty links or nodes on their default paths. We characterize path diversity in three respects: the deflection paths that exist between particular source and destination nodes (6.2); the ability to route around particular nodes or links deemed faulty (6.3); and the ability to switch peering points (6.4).

6.1 Methodology

We implemented a custom simulator to explore deflection routes and evaluate tag-based deflections.

Input Topologies: We study a wide range of topologies because deflection routes are a property of the network on which they are computed:

1. Real networks for which we can obtain topologies and link weights. These are Abilene and GEANT, research and educational networks based in the US and Europe, respectively. These networks have relatively large capacities but relatively few nodes and links.

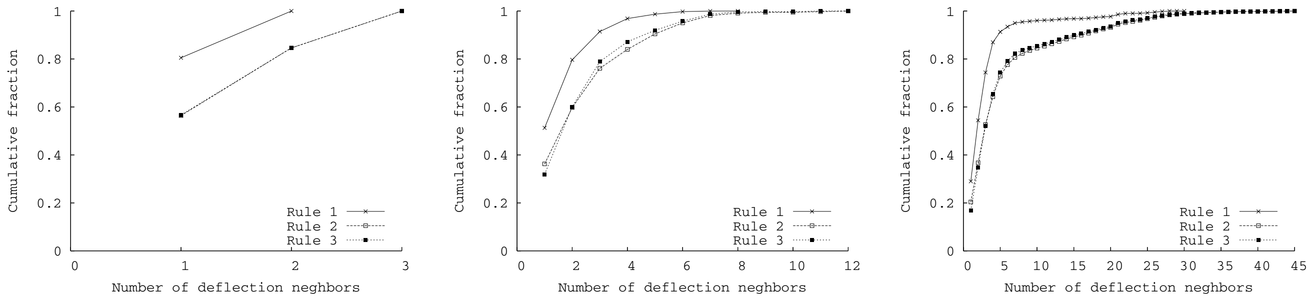
2. Measured ISP topologies from Rocketfuel [20]. We use five topologies (Sprint, Ebone, Tiscali, Exodus and Abovenet) complete with link-weights that are inferred to match observed routing patterns. (We exclude Telstra because the mapping is of low quality.) These networks are substantially larger than Abilene and GEANT.

3. Topologies randomly generated with Brite [12]. We used two different models: Barabasi Albert (BA) and the Waxman model (Waxman). The BA model generates graphs with a power-law degree distribution, and the Waxman model generates graphs with a uniform degree distribution. For each model, we use link delay as the cost metric for routing and generate low and high degree graphs.

The size and degree of all simulated networks are given in the first column of Table 1. Networks within the same category are ordered by average node degrees. Rule 3 refers to the link-version of Rule 3 (Section 3). The node-version has similar results and is omitted to save space. For Abovenet, we use and give figures for the maximally connected component, since the network is not connected otherwise.

Output Metrics: We compute several metrics for each network and each deflection rule. They are summarized below, with further detail where the corresponding results are presented.

To measure the number of usable deflection paths, we compute the number of neighbors in the deflection set at each router. This captures the number of opportunities there are to deflect off the shortest path. We next compute the number of different deflection paths between a source and a destination. This shows how deflection opportunities at nodes are translated into deflection paths through the network. Finally, we find the largest fraction of the shortest path between a source and a destination that can be by-



(a) Abilene

(b) Exodus

(c) Sprint

Figure 5: **The number of deflection neighbors a router has per incoming interface per destination. Rule 2 and 3 produce more deflection neighbors than Rule 1. (Note that the lines for these rules overlap in the left graph.) Larger networks produce more deflection neighbors in which the majority of routers have a choice of neighbor and some routers have a large number of deflection choices.**

passed by deflections. This looks at how diverse the different deflection paths are in terms of their component links and nodes.

To measure the ability to route around faults, we consider individual links and nodes instead of source-destination pairs. We compute the fraction of shortest paths that can be re-routed to bypass a faulty link or node. We then consider how many of these paths will be found when the source uses simple strategies to set tag values.

To measure the ability to switch peering points, we pick sets of nodes to represent egress points, and compute how often a source can arrive at an egress that is not its lowest-cost exit. In this setting, the lowest-cost path is the shortest path to any egress point.

For each metric, larger is better. Some of the results also differ across runs since they depend on the pseudo-random ordering of the deflection neighbors. When this is the case, we present the average of 10 runs. We omit the deviation across runs unless otherwise noted because it is generally too small to be visible.

The sections below describe our results. We summarize the average metrics for all topologies in Table 1 and present the distribution of the metrics for Abilene, Exodus and Sprint. These three networks have 11, 79 and 315 nodes, respectively. They provide a sample of the results that allow us to see how deflections change with the scale of the network. We also find that the results for randomly generated networks are consistent with those for designed ISP networks. This suggests that deflections are reasonably robust to variations in topology.

6.2 Deflection Paths

The first metric we consider is the number of neighbors in the deflection set, K. This number is the function of the router, the deflection rule, the destination, and the incoming interface of a packet. We compute the value K for all legitimate combinations of router interfaces and destinations. (Rule 1 and Rule 3 do not allow all combinations, since they will not use certain incoming interfaces for a given destination.) Average results for all topologies are summarized in the *Deflection Nbr* column of Table 1. Figure 5 shows the cumulative distribution of K for Abilene, Exodus, and Sprint.

We make several observations. First, Rule 2 and Rule 3 are more flexible than Rule 1. They produce more deflection choices in all simulated networks, usually by a substantial margin. Second, the larger networks provide more opportunities to deflect, as measured by the size of the deflection set. Third, a large fraction of the routers can deflect off the shortest path with Rules 2 and Rule 3. More than 40% of routers have $K > 1$ in all simulated topologies, and the fraction is considerably higher for larger networks.

Next, we measure the number of different deflection paths a packet can take between any two nodes in a network. Roughly, this shows how tags convert deflection opportunities at individual routers into complete deflection paths. This measure also depends on how a router maps a tag into a deflection neighbor and tends to be larger for longer default paths. Again, average results for all topologies are summarized in the *Deflection Path* column of Table 1 and distributions are given for Abilene, Exodus and Sprint in Figure 6.

As before, we see that Rule 2 and Rule 3 outperform Rule 1 by a wide margin. In this case they have more deflection paths. Even for a small network such as Abilene, more than 80% of node pairs have a deflection path that differs from the default shortest path with Rule 2 or Rule 3. For larger networks, nearly all node pairs have a deflection path. Moreover, in the case of Sprint we see that many node pairs have close to the maximum number of deflection paths, which is $2^{10} - 1$ for our ten bit tag. This suggests that our tag mapping rule does a good job of mapping different tags to different routes. We also simulated a different tag mapping rule that does not use a pseudo-random modulo operation (Section 4.4). The number of deflection paths (averaged over all rules and all topologies) is nearly four times less than that produced by our tag mapping rule. We also compared our deflection rules with equal-cost multi-path (ECMP) routing. The average number of alternative paths produced by ECMP on our input topologies ranges from 0 to 1.4, much smaller than that produced by our deflection rules (Table 1).

Finally, we measure how much the deflection paths differ from the default shortest path. The more they differ, the more likely it is that a source can bypass faulty nodes or links that lie on the default routing path. We compute differences as the largest fraction of the shortest path nodes and links, respectively, that can be bypassed with a single deflection. Suppose $P_s = (A, N_1, N_2, ..., N_n, B)$ is the default shortest path routing between node A and B, $P_d = (A, M_1, M_2, ..., M_m, B)$ is a deflection path between A and B. If N_i does not appear in P_d, then we count it as a node difference. If there are a total of x node differences, the fraction of node differences is computed as x/n. For each node pair, we record the maximum node difference among all deflection paths. This corresponds to the largest portion of the path that can be avoided. Similarly, we also computed link difference. We omit results for link differences to save space, since node differences provide the stricter test: a one node difference requires at least two link differences.

We present summary results for each topology in the *Node Difference* column of Table 1 and distributions of node difference for

	Network	Rule	Deflection Nbr		Deflection Path		Node Difference		Node Bypassed		Link Bypassed		Peering Bypassed			
													$\|P\|$ = 2		$\|P\|$ = 5	
			Mean	> 1	Mean	Median	Mean	Median	10 tries	All	10 tries	All	10 tries	All	10 tries	All
Real	Abilene	1	1.2	19%	1	1	30%	0%	62%	64%	54%	64%	35%	37%	68%	69%
	Nodes: 11	2	1.6	43%	5	4	68%	100%	90%	95%	93%	95%	77%	81%	98%	98%
	Degree: 2.55	3	1.6	43%	4	3	69%	100%	90%	95%	97%	98%	77%	82%	98%	99%
	Geant	1	1.4	33%	2	1	51%	50%	70%	72%	67%	72%	43%	48%	76%	77%
	Nodes: 23	2	2.1	53%	24	20	76%	100%	89%	94%	95%	97%	84%	93%	99%	99%
	Degree: 3.22	3	2.1	55%	18	16	76%	100%	90%	95%	96%	97%	84%	94%	99%	99%
Measured	Ebone	1	1.8	46%	11	3	43%	40%	60%	61%	64%	61%	37%	40%	55%	57%
	Nodes: 87	2	2.6	62%	311	258	70%	80%	77%	81%	87%	89%	69%	81%	84%	87%
	Degree: 3.70	3	2.5	66%	167	112	72%	83%	78%	82%	88%	89%	70%	83%	86%	90%
	Exodus	1	1.8	48%	26	6	53%	60%	68%	70%	65%	70%	41%	46%	55%	57%
	Nodes: 79	2	2.6	63%	415	405	79%	100%	87%	90%	92%	93%	67%	78%	84%	89%
	Degree: 3.72	3	2.6	68%	300	253	81%	100%	88%	91%	93%	93%	69%	79%	85%	90%
	Tiscali	1	2.9	57%	74	22	60%	66%	68%	69%	69%	69%	43%	51%	65%	67%
	Nodes: 161	2	4.0	67%	653	761	76%	85%	78%	80%	81%	83%	67%	81%	85%	88%
	Degree: 4.07	3	3.8	71%	488	506	76%	87%	78%	81%	81%	83%	69%	82%	86%	89%
	Abovenet	1	2.7	70%	101	29	73%	100%	85%	89%	80%	89%	42%	53%	64%	69%
	Nodes: 138	2	4.0	81%	734	867	89%	100%	94%	97%	95%	97%	71%	88%	90%	95%
	Degree: 5.39	3	3.9	85%	629	711	89%	100%	94%	97%	96%	97%	70%	88%	90%	96%
	Sprint	1	3.3	71%	61	25	73%	100%	75%	77%	78%	77%	48%	56%	71%	74%
	Nodes: 315	2	5.9	79%	849	984	89%	100%	86%	90%	95%	96%	68%	87%	89%	95%
	Degree: 6.17	3	5.7	83%	808	952	90%	100%	87%	91%	95%	96%	68%	87%	90%	95%
Random	BA-1	1	2.2	53%	14	6	62%	100%	81%	83%	76%	83%	65%	73%	82%	83%
	Nodes: 100	2	4.1	67%	488	516	92%	100%	97%	99%	97%	99%	90%	98%	98%	99%
	Degree: 3.94	3	3.5	69%	240	230	93%	100%	98%	99%	98%	99%	91%	99%	99%	99%
	Waxman-1	1	1.9	55%	14	6	66%	100%	79%	81%	78%	81%	57%	65%	78%	79%
	Nodes: 100	2	3.0	69%	357	363	94%	100%	97%	99%	97%	99%	86%	97%	98%	99%
	Degree: 4.00	3	2.8	72%	218	208	94%	100%	97%	99%	98%	99%	89%	98%	99%	99%
	BA-2	1	3.2	76%	38	19	75%	100%	89%	90%	86%	90%	74%	82%	88%	89%
	Nodes: 100	2	5.0	87%	606	642	93%	100%	98%	99%	99%	99%	94%	99%	99%	100%
	Degree: 5.88	3	4.8	91%	470	488	93%	100%	99%	99%	99%	99%	94%	99%	99%	100%
	Waxman-2	1	3.0	77%	46	19	77%	100%	89%	90%	89%	90%	71%	78%	85%	85%
	Nodes: 100	2	4.4	87%	554	584	93%	100%	99%	99%	99%	99%	93%	99%	99%	100%
	Degree: 6.00	3	4.4	90%	467	475	93%	100%	99%	99%	99%	99%	94%	99%	99%	100%

Table 1: Summary of results for all simulated networks for all rules. Metrics other than medians and > 1 are averaged over all source-destination pairs, nodes, links or peering trials, as appropriate from left to right. The node, link and peering bypass percentages are intended to convey the chance that a node, link or peering point could be avoided with deflections.

our three example topologies in Figure 7. The value 100% corresponds to deflection paths that are node-disjoint with the shortest path (other than at source and destination). An ideal result would hug the x-axis then jump to one at 100%, meaning that every node pair had a node-disjoint deflection path. For the networks we show here, we see positive results. At least 60% of all node pairs have a node-disjoint deflection path under Rule 2 or Rule 3, with larger networks having near node-disjoint deflections even more often.

6.3 Fault Tolerance

The results above show that deflections can provide significant path diversity between source-destination pairs. We now consider how well sources are able to harness this diversity by using tags to avoid faults.

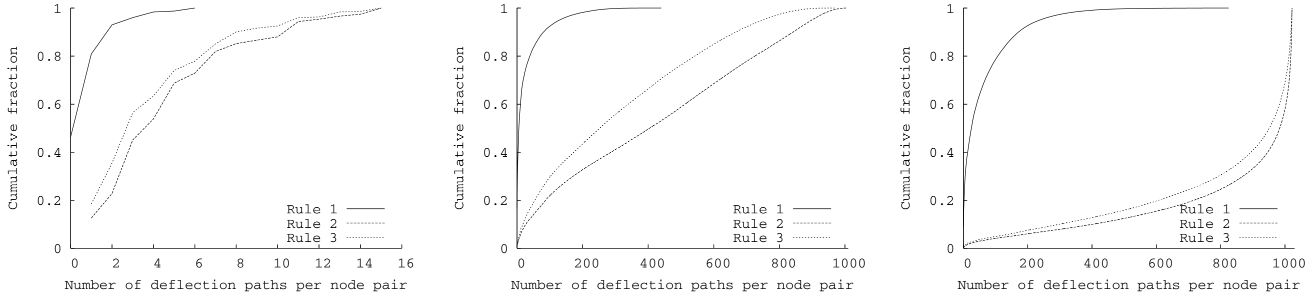
Specifically, we construct an experiment as follows. We pick a random link or node to be faulty. This fault will lie on the default routing path of a set of (perhaps many) source-destination pairs. For each pair, we simulate the source as it tries to bypass the fault by selecting different tag values. To stress the tag mechanism, we assume that a source tries at most ten tags before it assumes it cannot bypass the fault, since there is a cost to sending packets to try tags. For each fault and node pair, we record whether the source can bypass the failure and the number of tries it takes.

We use a simple strategy to select tag values. In the first five tries, the source chooses tags 1 through 5. This instructs each router to try five pseudo-random deflections, if that many exist. (Recall that tag 0 is reserved to indicate the default routing path.) In the last five tries, the source randomly picks a tag value in the remaining tag space [6,1023]. The intent here is to try to decouple effective tag choices at each router, since the tag is likely to be mapped to different values with different primes at different routers. We have not studied tag search strategies in detail, and better ones are likely to exist. However, they will only improve our results.

The results of this experiment are shown in the *Node Bypassed* and *Link Bypassed* columns of Table 1 and Figures 8 and 9. The summary results show that for nearly all topologies the vast majority of node pairs (often exceeding 90%) can bypass single node and link faults with deflections and that these deflections can be found by trying a small number of tags. The figures provide detail on the distributions for our sample networks.

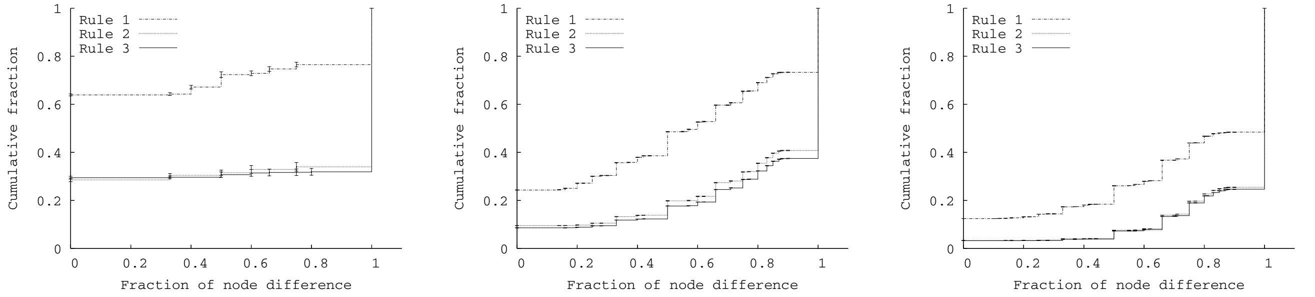
Figure 8 shows the distribution of the number of node pairs that can avoid a faulty node after 10 tries. The x-axis specifies faulty nodes that are on default routing paths. We rank these nodes according to the number of source-destination pairs that use them for default routes, R. We start with nodes that are on the most paths because they are the most important ones to be able to bypass. The vertical lines show the R values, and the points on each line show how many node pairs can avoid the faulty node after trying 10 tags. In the ideal situation, all points should stay on top of the vertical lines, indicating all node pairs that are affected by the faulty node can avoid it. We see that in all three networks the black squares that represent Rule 3 stay close to the top of the lines. This shows that a large fraction of node pairs affected by a faulty node can avoid it. We also ran simulations for faulty links using the same methodology. These results are generally even better, since it is harder to avoid a faulty node than a faulty link. We omit them to save space.

Figure 9 reports on the distribution of the number of tries that were made (with different tag values) to avoid the faults. It shows the fraction of the node pairs that could avoid a fault with a given number of tries, averaged over the different possible faults. This ap-



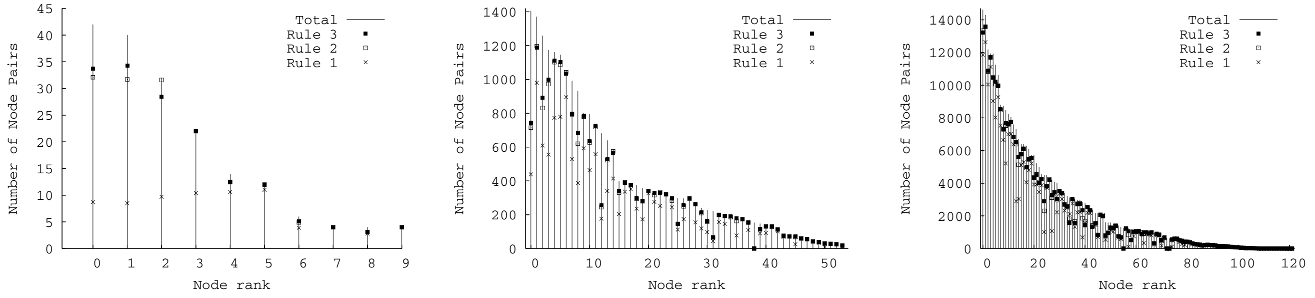
(a) Abilene (b) Exodus (c) Sprint

Figure 6: The number of deflection paths between two nodes. An ideal result would hug the x-axis until 2^{10} and then rise vertically, such that all node pairs have the maximum number of deflection paths. We see that most paths have deflections and larger networks have more deflection paths.



(a) Abilene (b) Exodus (c) Sprint

Figure 7: The largest fraction of node differences between a deflection path and the default shortest path for every node pair. Short vertical lines with horizontal bars show the (very tight) standard deviations across runs. An ideal result would hug the x-axis then jump to one at 100%, meaning that every node pair had a node-disjoint deflection path. We see that most node pairs can deflect a large fraction of the shortest path nodes, with larger networks being able to deflect more of the path more often.



(a) Abilene (b) Exodus (c) Sprint

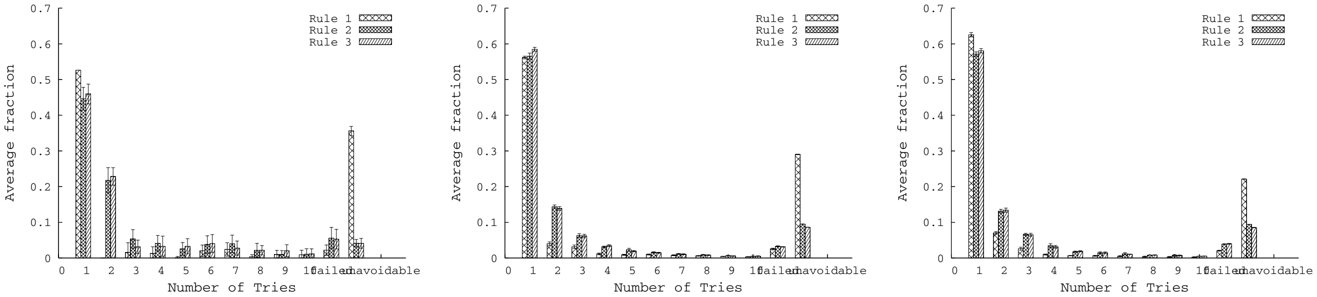
Figure 8: The number of node pairs that can avoid a faulty node after 10 tries. The x-axis shows the index of the faulty node. The vertical bars show the total number of node pairs that have the faulty node on their default routing paths. The points show how many node pairs successfully avoid the faculty nodes. The closer the points are to the top of the lines, the better. Rule 3 is consistently able to mostly or completely avoid faults.

proximates the probability with which a source can avoid a faulty node after a given number of tries. Not all faults can be bypassed. The column labeled “failed” shows the fraction of node pairs that needed more than 10 tries. The column labeled “unavoidable” shows the fraction of node pairs that cannot avoid a faulty node even if all tag values are tried. We see that, in all three networks, a significant fraction of node pairs can avoid faulty nodes in 10 tries, especially for Rules 2 and 3. Moreover, most successes happen quickly, such that only one or two tags need to be tried in most cases. The difference between trying ten tags and all tags is also insignificant. This suggests that a source can quickly find an alternative path to avoid a particular node (or link).

6.4 Inter-domain Deflections

It is difficult to assess the path diversity that deflections will provide in the Internet. This is because it depends on inter-domain routing policies and peering patterns as well as ISP topologies, and precious little data are publicly available. Instead, to gain a basic understanding of deflections with multiple ISPs, we look at how often they can change the peering points used between pairs of ISPs.

We construct a simple experiment to do this as follows. For each



(a) Abilene (b) Exodus (c) Sprint

Figure 9: The number of tags needed to bypass a fault. The y-axis gives the fraction of node pairs that can avoid a faulty node on their shortest path after the number of tries on the x-axis. The column labeled "failed" shows the fraction of node pairs that needed more than 10 tries to bypass the fault. The column labeled "unavoidable" shows the fraction of node pairs that cannot avoid a faulty node even if all tag values are tried. The short vertical bars show the standard deviation across different node faults. We see that most faults can be bypassed with one or two tag choices, and that the difference between trying 10 tags and all tags is insignificant. Note the y-axis stops at 0.7 to show more detail.

network, we randomly choose P nodes to be peering points, where $|P| = 2, 3, 4, 5$. We assume that BGP policies have chosen this set as the egress or peering points to an adjacent ISP. Each node n in the network will have a default peering point $p \in P$ for which n has the lowest IGP cost. This simulates the shortest path routing mechanism inside the ISP.

We then run simulations to measure the fraction of nodes that can switch their peering points away from their defaults. We assume a node only tries ten tags to stress the design. The fraction we compute is the likelihood that a node can bypass its default peering point if it considers the default faulty. It depends on both the number of peering points and the specific points we choose. To obtain an overall estimate, for each simulation, we fix the number of peering points $|P|$, and choose 100 random sets (or as many combinations as exist, if that is smaller). We record the fraction of nodes that can change their peering points for each peering set P, and average the results over all peering sets. As before, the final results are averaged over 10 simulation runs to reduce the effect of the pseudo-random ordering of deflection neighbors.

The *Peering Bypassed* column of Table 1 summarizes the results. We only show the results when the number of peering points is 2 and 5, respectively. The sub-column *All* shows the fraction of nodes that can bypass a peering point if all tags are tried. Figure 10 shows the results for 2, 3, 4, and 5 peering points for our example networks. The x-axis shows the number of peering points. The y-axis shows the fraction of nodes that can use a different peering points after 10 tag tries.

Rule 2 and 3 consistently provide more peering choices than Rule 1. With them, a significant fraction of nodes can change their peering points after only 10 tag tries. When the number of peering points is larger, this fraction is higher. When there are only two peering points, more than 67% of nodes for all simulated topologies can use the alternative peering point. When there are five peering points, most topologies have more than 90% of nodes that can choose a different peering point. We also note that trying all tags helps to bypass a peering point somewhat more than to bypass a faulty node or link that lies on the default routing path.

6.5 Summary

Overall, our results show that it is possible to construct deflection rules that provide good path diversity. In particular, our Rules 2 and 3 are significantly better than the straightforward Rule 1. Greater levels of diversity are available in designed networks that are larger in size as well as random networks that have higher average degree. Path diversity via deflections allows a peering point or a faulty node (or link) to be avoided most of the time, even in small networks. Moreover, tags are effective to access path diversity. A deflection that bypasses one fault can be found by trying a single tag most of the time, with a small number of faults requiring more tries.

7. RELATED WORK

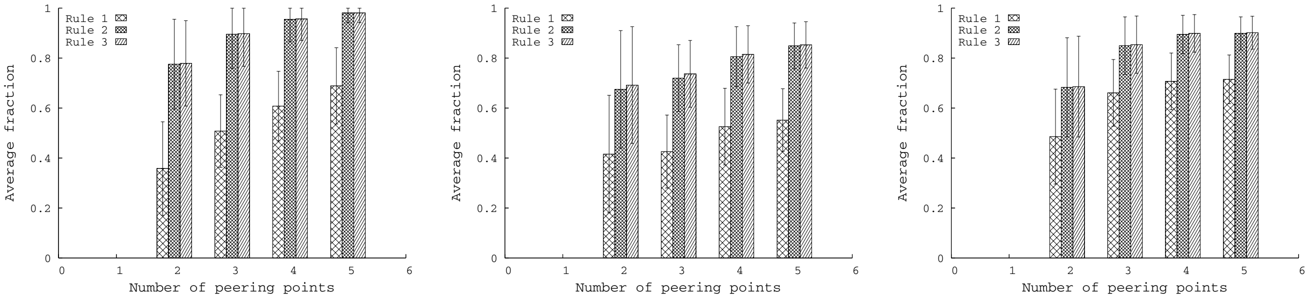
Our work is motivated by results that show variants of source routing to be beneficial. RON [1], Detour [17], and SOSR [7] show that overlay routing can improve end-to-end reliability, throughput, latency, and loss rate. In early work, Perlman used source routing as an essential means to avoid Byzantine failures [13]. Clark et al. [5] argue that end user control over provider-level routes has the potential to create a competitive ISP marketplace.

Much work addresses the difficulties of implementing source routes. In particular, to handle scaling issues, many schemes use a link-state like routing protocol to provide end systems with a map of the network [27, 15, 28, 6, 4, 3]. In contrast, we build on the existing shortest-path machinery to capture ISP policy and eliminate the need for sources to obtain any map. Further, we do not require end-systems to forward packets for each other, as do overlays, nor install path-specific state at routers, as do schemes such as packet reflection [9]. As a tradeoff, the region of the network over which a source can deflect its packets is restricted. Nevertheless, our results show most node and link failures can be bypassed in practice.

Other routing and forwarding schemes use short, fixed-length labels to represent multiple paths. MPLS [16] is widely used by backbone ISPs to split traffic along multiple paths. Bananas [10] uses a 32-bit hash of an AS path as a label. NIRA [27] uses hierarchically allocated IPv6 addresses to represent provider-level routes. Unlike deflections, all these mechanisms map a label to a unique path and so require additional signaling messages to establish the mapping.

Similar to deflections, some multi-path routing formulations allow a router to choose among multiple next hops to reach a destination without looping. OSPF and ISIS permit multi-path routing among next hops with equal cost to a destination [22]. Vutukury et al. propose a multi-path scheme similar to Rule 1 in which a router can choose any neighbor with a cost less than itself as the next hop [23, 24]. Our rules construct larger sets of paths.

Finally, work on local route repair explores the use of alternate next hops to bypass faults before new routes have converged. Ongoing work in the IETF [19] studies the well-known Rule 1 and



(a) Abilene (b) Exodus (c) Sprint

Figure 10: The fraction of nodes that can switch peering points after 10 tag tries. The x-axis shows the number of peering points, which are randomly chosen. The short vertical bars show the standard deviation across different peering sets. We see that Rule 2 and Rule 3 consistently provide more peering choices.

variants that are similar to our Rule 2. The main distinction is that this work targets a single fault and may result in loops if there are multiple faults, whereas our deflections can be used safely at multiple locations. This IETF work is similar to earlier work by Wang and Crowcroft [25]. More recently, Iyer [8] studied re-routings that are equivalent to Rule 1 and similar to Rule 2. However, that work places restrictions on the intra- versus inter-POP weights to avoid loops while we do not.

8. CONCLUSIONS

We have presented a practical system that provides the benefits of source-controlled routes in the Internet without the problems associated with explicit source routes. It is a tag-based routing architecture that uses routing deflections to provide path diversity. Users tag packets with hints, rather than explicit source routes, and ISPs use these hints to select among alternative paths. These tags can be encoded in a way that is compatible with common IP usage. ISPs generate the underlying path diversity with the routing deflections that we have introduced. This mechanism is scalable, compatible with ISP policies and easily incrementally deployable. To evaluate the overall system, we performed simulations with real, measured and random network topologies. We found that deflections provide a high-level of path diversity and tags make effective use of this diversity. With our rules, a source can avoid most single node or link faults by trying only a handful of tags, with better results for larger networks.

We consider the routing deflections rules we have defined to be the most interesting aspect of our work. We were surprised to realize that such a large set of non-shortest path neighbors could be used to reach the destination without the danger of loops, and that this could be done robustly without any coordination between neighboring routers. It is likely that there exist other, perhaps more powerful, deflection rules, since we have not yet systematically explored the design space. We have also restricted our attention to deflection rules that are incrementally deployable with the existing shortest path routers. Easing this restriction, say by signaling path information on packets, would permit other deflections. We are also interested in exploring the use of deflections in other settings, the most immediate of which is to locally repair routes and minimize transient loops during routing convergence.

9. ACKNOWLEDGEMENTS

Wetherall gratefully acknowledges the support of a Sloan Research Fellowship. We thank Minas Gjoka for converting the input topologies to the format needed by our simulator, and the anonymous reviewers, our shepherd Bruce Davie, Nick Feamster, Jinyang Li, Xin Liu, and Junfeng Yang for providing useful feedback.

10. REFERENCES

[1] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP*, Oct. 2001.

[2] P. Baran. On distributed communications, vol. i. RAND Technical Report RM-3420-PR, Aug. 1964.

[3] I. Castineyra, N. Chiappa, and M. Steenstrup. The Nimrod Routing Architecture. IETF RFC 1992, Aug. 1996.

[4] D. Clark. Policy Routing in Internetworks. *Internetworking: Research and Experience*, 1, 1990.

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

[6] D. Estrin, Y. Rekhter, and S. Hotz. Scalable Inter-Domain Routing Architecture. In *ACM SIGCOMM*, 1992.

[7] 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 *OSDI*, Dec. 2004.

[8] S. Iyer, S. Bhattacharyya, N. Taft, and C. Diot. An approach to alleviate link overload as observed on an IP backbone. In *INFOCOM*, 2003.

[9] J. Jannotti. Network layer support for overlay networks. In *IEEE OPENARCH*, June 2002.

[10] H. T. Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi. Bananas: An evolutionary framework for explicit and multipath routing in the Internet. In *ACM SIGCOMM FDNA workshop*, Aug. 2003.

[11] N. Maxemchuk. Routing in the manhattan street network. *IEEE Trans. on Communication*, COM-35(5), May 1987.

[12] A. Medina, I. Matta, and J. Byers. BRITE: A flexible generator of Internet toplogies. Technical Report BU-CS-TR-2000-005, Boston University, 2000.

[13] R. Perlman. Network layer protocols with byzantine robustness. Technical report, MIT, Oct. 1988. MIT-LCS-TR-429.

[14] R. Perlman. Routing with byzantine robustness. Technical report, Sun Labs, Aug. 2005. TR-2005-146.

[15] B. Raghavan and A. C. Snoeren. A System for Authenticated Policy-Compliant Routing. In *ACM SIGCOMM*, 2004.

[16] E. C. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. IETF RFC3031, Jan. 2001.

[17] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of Internet path selection. In *SIGCOMM*, Aug. 1999.

[18] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *SIGCOMM*, Aug. 2000.

[19] M. Shand and S. Bryant. IP Fast Reroute Framework. IETF Routing Working Group, work in progress, Mar. 2006.

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

[21] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high speed networks. In *SIGCOMM*, 1998.

[22] D. Thaler and C. Hopps. Multipath issues in unicast and multicast next-hop selection. IETF RFC 2991, Nov. 2000.

[23] S. Vutukury and J. Garcia-Luna-Aceves. MDVA: A distance-vector multipath routing protocol. In *IEEE Infocom*, 2001.

[24] S. Vutukury and J. Garcia-Luna-Aceves. Mpath: a loop-free multipath routing algorithm. *Journal of Microprocessors and Microsystems*, 2001.

[25] Z. Wang and J. Crowcroft. Shortest path first with emergency exits. In *SIGCOMM*, 1990.

[26] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against ddos attacks. In *IEEE Symposium on Security and Privacy*, 2003.

[27] X. Yang. NIRA: A new Internet routing architecture. In *ACM SIGCOMM FDNA workshop*, 2003.

[28] D. Zhu, M. Gritter, and D. R. Cheriton. Feedback Based Routing. In *Proc. of HotNets-I*, 2002.

APPENDIX

We prove that each rule provides paths that are loop-free and reach their destinations, even when shortest-path routers are present in the network.

A. RULE 1 (ONE HOP DOWN)

Let the sequence of nodes on the forwarding path be n_i for $i \geq 0$. Consider the sequence $cost(n_i)$ for $i \geq 0$. By Rule 1 it strictly decreases. Hence each node in the sequence must correspond to a different node so that the path is loop-free. To reach the destination, it suffices to show that the deflection set is not empty. This is so because shortest-path neighbors are always valid choices because they have lower cost than the current node by the definition. This further implies that shortest-path routers make valid deflections and can be freely mixed with Rule 1 routers. □

B. RULE 2 (TWO HOPS DOWN)

To show loop-freedom, we prove that no directional link will repeat in the forwarding path. Define the cost of a directional link u_i that connects n_i and n_{i+1} to be the maximum cost of its endpoints. We now show that the cost of adjacent links is non-increasing. To do this we state link cost and substitute Rule 2 expressed in succinct form as a maximum operator that combines its two clauses:

$$\begin{aligned} cost(u_{i+1}) &= max(cost(n_{i+1}), cost(n_{i+2})) \\ &\leq max(cost(n_{i+1}), max(cost(n_{i+1}), cost(n_i))) \\ &= max(cost(n_i), cost(n_{i+1})) \\ &= cost(u_i) \end{aligned}$$

Next we show that the cost of every other link along a path is strictly decreasing:

$$\begin{aligned} cost(n_{i+3}) &< max(cost(n_{i+2}), cost(n_{i+1})) \\ &\leq max(max(cost(n_{i+1}), cost(n_i)), cost(n_{i+1})) \\ &= max(cost(n_i), cost(n_{i+1})) \\ &= cost(u_i) \end{aligned} \qquad (1)$$

By the definition of Rule 2 and link costs we also have:

$$\begin{aligned} cost(n_{i+2}) &< max(cost(n_i), cost(n_{i+1})) \\ &= cost(u_i) \end{aligned} \qquad (2)$$

Both $cost(n_{i+2})$ and $cost(n_{i+3})$ are less than $cost(u_i)$. Hence by definition, $cost(u_{i+2}) < cost(u_i)$. And from (1) and (2) it follows that on the forwarding path, the cost of any link u_{i+k} for $k > 1$ is strictly less than u_i. Therefore, any link u_{i+k} with $k > 1$ cannot be the same as link u_i. It remains to show that the adjacent link u_{i+1} cannot be the same as u_i. This is true because these two links start at different nodes. Thus, no directional link can be re-visited on the forwarding path; there are no link-level loops.

To see compatibility with shortest-path routers, observe that the shortest path neighbor is always valid deflection choice because it satisfies the downhill clause. Liveness follows from this too, as the deflection set is non-empty, and a packet will eventually reach its destination. □

C. RULE 3 (TWO HOPS FORWARD)

Observe that removing links from the graph can only increase the cost of paths that would otherwise use it, i.e.:

$$cost(G, n_i) \leq cost(G \backslash l_j, n_i) \quad \forall \text{ nodes } n_i, \text{ links } l_j \qquad (3)$$

We now restate Rule 3 compactly and substitute (3) for the bidirectional incoming link l_i (that connects n_{i-1} and n_i):

$$\begin{aligned} cost(G \backslash l_{i+1}, n_{i+1}) &< max(cost(G \backslash l_i, n_i), cost(G, n_{i-1})) \\ &\leq max(cost(G \backslash l_i, n_i), cost(G \backslash l_{i-1}, n_{i-1})) \end{aligned}$$

This has the same form we saw in Rule 2 when we consider the sequence $cost(G \backslash l_i, n_i)$ instead of $cost(n_i)$. Hence we can show loop-freedom in precisely the same manner, by defining an analogous directional link cost and showing that no directional link is repeated. (We omit this to avoid repetition.)

To show that Rule 3 reaches destinations, it suffices to show that the deflection set is not empty before the destination is reached. To do this, we will show that the shortest-path neighbor in $G \backslash l_i$ is always an allowed choice. Let this shortest-path neighbor be node n_{i+1} so that by definition we have $cost(G \backslash l_i, n_{i+1}) < cost(G \backslash l_i, n_i)$. Next we show that $cost(G \backslash l_{i+1}, n_{i+1}) < cost(G \backslash l_i, n_{i+1})$. This is because n_{i+1} is n_i's shortest path neighbor. Its shortest path in the graph $G \backslash l_i$ cannot go back across the last incoming link l_{i+1}. Hence, the cost of n_{i+1} in the graph that excludes both links l_i and l_{i+1}, i.e., $G \backslash (l_i, l_{i+1})$, is the same as $cost(G \backslash l_i, n_{i+1})$. Then by using inequality (3), we can upper bound $cost(G \backslash l_{i+1}, n_{i+1})$. With these steps we have:

$$\begin{aligned} cost(G \backslash l_{i+1}, n_{i+1}) &\leq cost(G \backslash (l_i, l_{i+1}), n_{i+1}) \\ &= cost(G \backslash l_i, n_{i+1}) \\ &< cost(G \backslash l_i, n_i) \end{aligned} \qquad (4)$$

This inequality (4) satisfies Rule 3 because it is simply the downhill clause. So the shortest-path neighbor in $G \backslash l$ is in the deflection set as required.

Finally, to show compatibility with shortest-path routes, it suffices to show that the shortest path neighbor of n_i in G is an allowable deflection choice. By definition of the shortest neighbor n_{i+1}, $cost(G, n_{i+1}) < cost(G, n_i)$. In addition, the shortest path from n_{i+1} will not go back across the incoming link, l_{i+1}. Therefore, the cost of n_{i+1} in the graph $G \backslash l_{i+1}$ is the same as in G. Combining these facts we have $cost(G \backslash l_{i+1}, n_{i+1}) < cost(G, n_i)$. Applying (3) we obtain $cost(G \backslash l_{i+1}, n_{i+1}) < cost(G \backslash l_i, n_i)$. This is simply the downhill clause of Rule 3. Thus shortest path forwarding satisfies Rule 3, as required. □