LIFEGUARD: Practical Repair of Persistent Route Failures

Ethan Katz-Bassett
Univ. of Southern California
Univ. of Washington

Colin Scott
UC Berkeley

David R. Choffnes
Univ. of Washington

Ítalo Cunha
UFMG, Brazil

Vytautas Valancius
Georgia Tech

Nick Feamster
Georgia Tech

Harsha V. Madhyastha
UC Riverside

Thomas Anderson
Univ. of Washington

Arvind Krishnamurthy
Univ. of Washington

ABSTRACT

The Internet was designed to always find a route if there is a policy-compliant path. However, in many cases, connectivity is disrupted despite the existence of an underlying valid path. The research community has focused on short-term outages that occur during route convergence. There has been less progress on addressing avoidable long-lasting outages. Our measurements show that long-lasting events contribute significantly to overall unavailability.

To address these problems, we develop *LIFEGUARD*, a system for automatic failure localization and remediation. *LIFEGUARD* uses active measurements and a historical path atlas to locate faults, even in the presence of asymmetric paths and failures. Given the ability to locate faults, we argue that the Internet protocols should allow edge ISPs to steer traffic to them around failures, without requiring the involvement of the network causing the failure. Although the Internet does not explicitly support this functionality today, we show how to approximate it using carefully crafted BGP messages. *LIFEGUARD* employs a set of techniques to reroute around failures with low impact on working routes. Deploying *LIFEGUARD* on the Internet, we find that it can effectively route traffic around an AS without causing widespread disruption.

Categories and Subject Descriptors

C.2.2 [**Communication Networks**]: Network protocols

Keywords

Availability, BGP, Measurement, Outages, Repair

1. INTRODUCTION

With the proliferation of interactive Web apps, always-connected mobile devices and data storage in the cloud, we expect the Internet to be available anytime, from anywhere.

However, even well-provisioned cloud data centers experience frequent problems routing to destinations around the Internet. Existing research provides promising approaches to dealing with the transient unavailability that occurs during routing protocol convergence [18, 22–24], so we focus on events that persist over longer timescales that are less likely to be convergence-related. Monitoring paths from Amazon's EC2 cloud service, we found that, for outages lasting at least 90 seconds, 84% of the unavailability came from those that lasted over ten minutes.

We focus on disruptions to connectivity in which a working policy-compliant path exists, but networks instead route along a different path that fails to deliver packets. In theory this should never happen – if working paths exist, the Internet protocols are designed to find them, even in the face of failures. In practice, routers can fail to detect or reroute around a failed link, causing silent failures [35].

When an outage occurs, each affected network would like to restore connectivity. However, the failure may be caused by a problem outside the network, and available protocols and tools give operators little visibility into or control over routing outside their local networks. Operators struggle to obtain the topology and routing information necessary to locate the source of an outage, since measurement tools like traceroute and reverse traceroute [19] require connectivity to complete their measurements.

Even knowing the failure location, operators have limited means to address the problem. Traditional techniques for route control give the operators' network direct influence only over routes between it and its immediate neighbors, which may not be enough to avoid a problem in a transit network farther away. Having multiple providers still may not suffice, as the operators have little control over the routes other ASes select to it.

To substantially improve Internet availability, we need a way to combat long-lived failures. We believe that Internet availability would improve if data centers and other well-provisioned edge networks were given the ability to repair persistent routing problems, regardless of which network along the path is responsible for the outage. If some alternate working policy-compliant path can deliver traffic during an outage, the data center or edge network should be able to cause the Internet to use it.

We propose achieving this goal by enabling an edge network to disable routes that traverse a misbehaving network, triggering route exploration to find new paths. While accomplishing this goal might seem to require a redesign of the Internet's protocols, our objective is a system that works today with existing protocols, even if it cannot address all outages. We present the design and implementation of a system that enables rerouting around many long-lasting failures while being deployable on today's Internet. We call our system *LIFEGUARD*, for *Locating Internet Failures Effectively and Generating Usable Alternate Routes Dynamically*. *LIFEGUARD* aims to automatically repair partial outages in minutes, replacing the manual process that can take hours. Existing approaches often enable an edge AS to avoid problems on its forward paths to destinations but provide little control over the paths back to the AS. *LIFEGUARD* provides reverse path control by having the edge AS O insert the problem network A into path advertisements for O's ad-

SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.

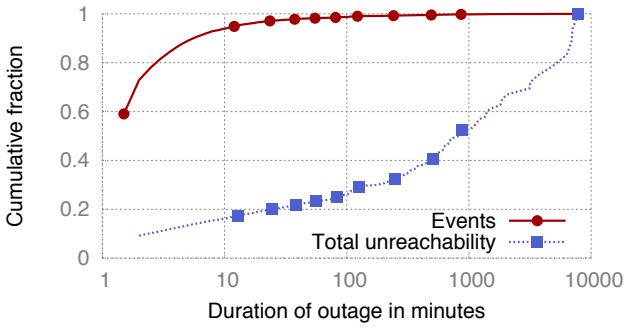

Figure 1: For partial outages observed from EC2, the fraction of outages of at most a given duration (solid) and their corresponding fraction of total unreachability (dotted). The x-axis is on a log-scale. More than 90% of the outages lasted at most 10 minutes, but 84% of the total unavailability was due to outages longer than 10 minutes.

dresses, so that it appears that *A* has already been visited. When the announcements reach *A*, BGP's loop-prevention mechanisms will drop the announcement. Networks that would have routed through *A* will only learn of other paths, and will avoid *A*. Using the BGP-Mux testbed [5] to announce paths to the Internet, we show *LIFEGUARD*'s rerouting technique finds alternate paths 76% of the time.

While this BGP *poisoning* provides a means to trigger rerouting, we must address a number of challenges to provide a practical solution. *LIFEGUARD* combines this basic poisoning mechanism with a number of techniques. *LIFEGUARD* has a subsystem to locate failures, even in the presence of asymmetric routing and unidirectional failures. We validate our failure isolation approach and present experiments suggesting that the commonly used traceroute technique for failure location gave incorrect information 40% of the time. We address how to decide whether to poison; will routing protocols automatically resolve the problem, or do we need to trigger route exploration? We show empirically that triggering route exploration could eliminate up to 80% of the observed unavailability. When it reroutes failing paths to remediate partial outages, *LIFEGUARD* carefully crafts BGP announcements to speed route convergence and minimize the disruption to working routes. Our experimental results show that 94% of working routes reconverge instantly and experience minimal ($\leq 2\%$) packet loss. After rerouting, *LIFEGUARD* maintains a *sentinel prefix* on the original path to detect when the failure has resolved, even though live traffic will be routing over an alternate path. When *LIFEGUARD*'s test traffic reaches the sentinel, *LIFEGUARD* removes the poisoned announcement.

2. BACKGROUND AND MOTIVATION

2.1 Quantifying Unreachability from EC2

To test the prevalence of outages, we conducted a measurement study using Amazon EC2, a major cloud provider. EC2 presumably has the resources, business incentive, and best practices available for combating Internet outages. We show that even EC2 data centers experience many long-term network connectivity problems.

We rented EC2 instances in the four available AWS regions from July 20, 2010 to August 29, 2010. Each vantage point issued a pair of pings every 30 seconds to 250 targets – five routers each from the 50 highest-degree ASes [33]. We selected the routers randomly from the iPlane topology [17], such that each router was from a distinct BGP prefix. We focus on paths to routers in major networks, which should be more reliable than paths to end-hosts. We define an outage as four or more consecutive dropped pairs of pings from a single vantage point to a destination. This methodology means that the minimum outage duration we consider is 90 seconds.

In 79% of the outages in our study, some vantage points had connectivity with the target (one of the routers), while others did not. Fig. 1 shows the durations of these 10,308 partial outages. By comparison, an earlier study found that 90% of outages lasted less than 15 minutes [13]. We also find that most outages are relatively short; more than 90% lasted less than 10 minutes (solid line). However, these short outages account for only 16% of the total unavailability (dotted line). The relatively small number of long-lasting problems account for much of the overall unavailability. Delayed protocol convergence does not explain long outages [23].

In fact, many long-lasting outages occur with few or no accompanying routing updates [13, 20]. With routing protocols failing to react, networks continue to send traffic along a path that fails to deliver packets. Such problems can occur, for example, when a router fails to detect an internal fault (e.g., corrupted memory on a line card causing traffic to be black-holed [35]) or when cross-layer interactions cause an MPLS tunnel to fail to deliver packets even though the underlying IP network is operational [21].

During partial outages, some hosts are unable to find the working routes to the destination, either due to a physical partition, due to a routing policy that restricts the export of the working path, or due to a router that is announcing a route that fails to deliver traffic. The techniques we present later in this paper rely on the underlying network being physically connected and on the existence of policy-compliant routes around the failure, and so we need to establish that there are long-lasting outages that are not just physical failures or the result of routing export policies.

We can rule out physical partitions as the cause in our EC2 study. All EC2 instances maintained connectivity with a controller at our institution throughout the study. So, physical connectivity existed from the destination to some EC2 instance, from that instance to the controller, then from the controller to the instance that could not reach the destination.

Since the problems are not due to physical partitions, either routing policies are eliminating all working paths, or routers are advertising paths that do not work. By detouring through our institution, the paths we demonstrated around EC2 failures violate the valley-free routing policy [15] – in not making those paths available, routers are properly enacting routing policy. However, if working policy-compliant paths also exist, it might be possible to switch traffic onto them.

2.2 Assessing Policy-Compliant Alternate Paths

Earlier systems demonstrated that overlays can route around many failures [2, 6, 16]. However, overlay paths tend to violate BGP export policies. We build on this previous work by showing that alternate policy-compliant paths appear to exist during many failures. Generally, the longer a problem lasted, the more likely it was that alternative routes existed.

Previous work found that many long-lasting failures occur outside of the edge networks [13, 20], and we focus on these types of problems. Every ten minutes for a week starting September 5, 2011, we issued traceroutes between all PlanetLab sites. This setting allowed us to issue traceroutes from both ends of every path, and the probes to other PlanetLab sites give a rich view of other paths that might combine to form alternate routes. We considered as outages all instances in which a pair of hosts were up and had previously been able to send traceroutes between each other, but all traceroutes in both directions failed to reach the destination AS for at least three consecutive rounds, before working again. This yielded nearly 15,000 outages.

We checked if the traceroutes included working policy-compliant routes around the failures. For each round of a failure, we tried to

find a path from the source that intersected (at the IP-level) a path to the destination, such that the spliced path did not traverse the AS in which the failed traceroute terminated. We only considered a spliced path as valid if it would be available under observable export policies. To check export policies, when splicing together a potential path, we only accepted it if the AS subpath of length three centered at the splice point appeared in at least one traceroute during the week [17, 25]. This check suffices to encode the common valley-free export policy [15].

Our methodology may fail to identify some valid paths that exist. PlanetLab has somewhat homogeneous routing. We also required that spliced paths intersect at a shared IP address. Two traceroutes might intersect at a router or a PoP without sharing an IP address. We would not consider this intersection when trying to splice paths.

We found that, for 49% of outages, alternate paths existed for the duration of the failure. Considering only long outages that lasted at least an hour, we found alternate routes in 83% of failures. For 98% of the outages in which an alternate path existed during the first round of the failure, the path persisted for the duration.

2.3 Current Approaches to Address Failures

Lacking better options, operators rely on insufficient techniques to try to locate and resolve long-lasting outages, especially if the failure is in a network outside the operators' control. Asymmetric paths leave operators with a limited view even when paths work. Tools like traceroute require bidirectional connectivity to function properly, and so failures restrict their view further. Public traceroute servers and route collectors [26,31] extend the view somewhat, but only a small percentage of networks make them available. In fact, these challenges mean that operators frequently resort to asking others to issue traceroutes on their behalf to help confirm and isolate a problem [28].

If operators successfully identify a failure outside their own networks, they have little ability to effect repair:

Forward path failures: The source network's operators can select an alternative egress in an attempt to avoid the problem. When choosing, they can see the full BGP paths provided by their neighbors. Each of the source's providers announces its preferred path, and the source is free to choose among them. If the network's providers offer sufficiently diverse paths, the failure may be avoided. For example, we inspected BGP routes from five universities (University of Washington, University of Wisconsin, Georgia Tech, Princeton, and Clemson) [5] to prefixes in 114 ASes. If these universities were our providers, the routes are sufficiently diverse that, if the last AS link before the destination on one of the routes failed silently, we could route around it to reach the destination in 90% of cases by routing via a different provider. In §5.2, we present an equivalent experiment demonstrating that our techniques would allow us to avoid 73% of these links on reverse paths back from the 114 ASes, without disturbing routes that did not use that link. [1]

Reverse path failures: Using traditional techniques, however, having multiple providers may not offer much reverse path diversity. Under BGP, the operators can only change how they announce a prefix to neighbors, perhaps announcing it differently to different neighbors. They have no other direct influence over the paths other networks select. A major limitation of existing techniques for announcement-based route control is that they generally act on the next hop AS, rather than allowing a network to target whichever AS is causing a problem. We discuss the techniques below:

[1]The 114 ASes were all those that both announce prefixes visible at the universities, needed for the forward path study, and peer with a route collector [1, 26, 29, 31], needed for the reverse study.

Multi-Exit Discriminator (MEDs): An AS that connects to another AS at multiple points can use MEDs to express to the neighbor on which peering point it prefers to receive traffic. However, MEDs have meaning only within the context of that single neighbor, so they generally are effective only if the problem is in the immediate upstream neighbor.

Selective Advertising: An origin AS with multiple providers can advertise a prefix through only some providers. In variations on this approach, the origin can advertise more-specific prefixes through some providers and only less-specifics through others. Or, since many ASes use path length as a tiebreaker when making routing decisions, networks sometimes prepend routes they announce with multiple copies of their AS, to make that path longer and hence less preferred than shorter ones. With all these approaches, the origin can shift traffic away from providers it wants to avoid. If the problem is not in the immediate provider, these techniques may be deficient because (1) all working routes that had previously gone through that provider will change; and (2), even if all sources with failing paths had routed through a particular provider before selective advertising, forcing them to route via a different provider may not change the portion of the path containing the failure.

BGP communities: Communities are a promising direction for future experiments in failure avoidance but do not currently provide a complete solution. An AS can define communities that other networks can tag onto routes they announce to the AS. Communities instruct the AS on how to handle the routes. For example, SAVVIS offers communities to specify that a route should not be exported to a peer. However, communities are not standardized, and some ASes give limited control over how they disseminate routes. Further, many ASes do not propagate community values they receive [30], and so communities are not a feasible way to notify arbitrary ASes of routing problems. We announced experimental prefixes with communities attached and found that, for example, any AS that used a Tier-1 to reach our prefixes did not have the communities on our announcements.

Changes to BGP announcements and to local configuration may be unable to repair outages. In such cases, operators often must resort to phone calls or e-mails asking operators at other networks for support. These slow interactions contribute to the duration of outages. We now show how our approach enables an operator to avoid reverse path failures.

3. ENABLING FAILURE AVOIDANCE

Suppose an AS O wants to communicate with another AS Q but cannot because of some problem on the path between them. If the problem is within either O or Q, operators at that network have complete visibility into and control over their local networks, and so they can take appropriate steps to remedy the problem. Instead, consider a case in which the problem occurs somewhere outside of the edge ASes, either on the forward path to Q or on the reverse path back to O. Further suppose that O is able to locate the failure and to determine that an alternate route likely exists.[2]

O would like to restore connectivity regardless of where the problem is, but its ability to do so currently depends largely on where the problem is located. If the problem is on the forward path and O's providers offer suitable path diversity, O can choose a path that avoids the problem. By carefully selecting where to locate its PoPs and which providers to contract with, O should be able to achieve decent resiliency to forward path failures. However, having a diversity of providers may not help for reverse path failures, as O has

[2]We discuss how *LIFEGUARD* does this in §4.

little control over the routes other ASes select to reach it. As explained in §2.3, route control mechanisms like MEDs and selective advertising only let *O* control the PoP or provider through which traffic enters *O*. However, these BGP mechanisms give *O* essentially no control over how other ASes reach the provider it selects.

O needs a way to notify ASes using the path that the path is not successfully forwarding traffic, thereby encouraging them to choose alternate routes that restore connectivity. As a hint as to which paths they should avoid, *O* would like to inform them of the failure location. AS-level failure locations are the proper granularity for these hypothetical notifications, because BGP uses AS-level topology abstractions. In particular, when one of the notified ASes chooses an alternate path, it will be selecting from AS paths announced to it by its neighbors. Therefore, *O* needs to inform other ASes of which AS or AS link to avoid, depending on whether the failure is within a single AS or at an AS boundary.

Ideally, we would like a mechanism to let the origin AS *O* of a prefix *P* specify this information explicitly with a signed announcement we will call AVOID_PROBLEM(X,P). Depending on the nature of the problem, *X* could either be a single AS (AVOID_PROBLEM(A,P)) or an AS link $A - B$ (AVOID_PROBLEM(A-B,P)). Note that AS *O* is only able to directly observe the problem with prefix *P*; it cannot determine if the issue is more widespread. Announcing this hypothetical primitive would have three effects:

- *Avoidance Property*: Any AS that knew of a route to *P* that avoided *X* would select such a route.
- *Backup Property*: Any AS that only knew of a route through *X* would be free to attempt to use it. Similarly, *A* would be able to attempt to route to *O* via its preferred path (through *B* in the case when *X* is the link *A*-*B*).
- *Notification Property*: *A* (and *B*, for link problems) would be notified of the problem, alerting its operators to fix it.

3.1 LIFEGUARD's Failure Remediation

Deploying AVOID_PROBLEM(X,P) might seem to require changes to every BGP router in the Internet. Instead, we use mechanisms already available in BGP to perform the notifications, in order to arrive at a solution that is usable today, even if the solution is not complete. A usable approach can improve availability today while simultaneously helping the community understand how we might improve availability further with future BGP changes. We call our approach *LIFEGUARD*, for *Locating Internet Failures Effectively and Generating Usable Alternate Routes Dynamically*.

To approximate AVOID_PROBLEM(A,P) on today's Internet, *LIFEGUARD* uses BGP's built-in loop prevention to "poison" a problem AS that is announcing routes but not forwarding packets. To poison an AS *A*, the origin announces the prefix with *A* as part of the path, causing *A* to reject the path (to avoid a loop) and withdraw its path from its neighbors [8, 10]. This causes ASes that previously routed via *A* to explore alternatives. Importantly, the poison affects only traffic to *O*'s prefix experiencing the problem. By allowing an AS to poison only prefixes it originates, our approach is consistent with the goals of work toward authenticating the origin of BGP announcements [27]. Proposals to verify the entire path [3] are consistent with the future goal for our approach, in which AVOID_PROBLEM(X,P) would be a validated hint from the origin AS to the rest of the network that a particular AS is not correctly routing its traffic. By the time such proposals are deployed, it may be feasible to develop new routing primitives or standardized communities to accomplish what we currently do with poisoning.

Although BGP loop prevention was not intended to give *O* control over routes in other ASes, it lets us experiment with failure

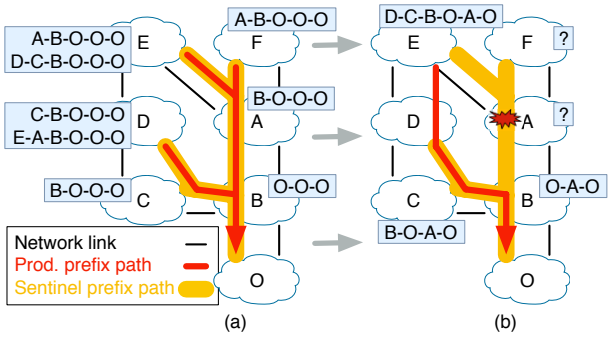

Figure 2: Routes and routing tables (a) before and (b) after *O* poisons *A* to avoid a problem. Each table shows *only paths to the production prefix*, with the in-use, most-preferred route at the top. Poisoning *A* for the production prefix causes it to withdraw its route from *E* and *F*, forcing *E* to use its less-preferred route through *D* and leaving *F* with only the sentinel. Routes to the sentinel prefix do not change, allowing *O* to check when the problem has resolved.

avoidance. In effect, poisoning *A* implements the *Avoidance Property* of AVOID_PROBLEM(A,P), giving *O* the means to control routes to it. *A*'s border routers will receive the poisoned announcement and detect the poison, a form of the *Notification Property*.

On its own, poisoning is a blunt, disruptive instrument, a limitation that *LIFEGUARD* must overcome. Poisoning inserts *A* into all routes, so even ASes that were not routing through *A* may undergo route exploration before reconverging to their original route, leading to unnecessary packet loss [23]. Instead of providing the *Backup Property*, poisoning cuts off ASes that lack a route around *A*. Poisoning disables all paths through *A*, even if some work.

In the following sections, we show how *LIFEGUARD* overcomes what initially seem like limitations of poisoning in order to better approximate AVOID_PROBLEM(X,P).

3.1.1 Minimizing Disruption of Working Routes

Inserting an AS to poison an announcement increases AS path length. Suppose that an origin AS *O* decides to poison *A* for *O*'s prefix *P*. The poisoned path cannot be *A*-*O*, because *O*'s neighbors need to route to *O* as their next hop, not to *A*. So, the path must start with *O*. It cannot be *O*-*A*, because routing registries list *O* as the origin for *P*, and so a path that shows *A* as the origin looks suspicious. Therefore, *O* announces *O*-*A*-*O*. Experiments found that BGP normally takes multiple minutes to converge when switching to longer paths, with accompanying packet loss to the prefix during this period [23]. This loss would even affect networks with working paths to the prefix.

To poison in a way that shortens and smooths this convergence period, *LIFEGUARD* crafts steady-state unpoisoned announcements in a way that "prepares" all ASes for the possibility that some AS may later be poisoned. Fig. 2 provides an example of an origin AS *O* with a *production prefix P* which carries real traffic. Fig. 2(a) depicts the state before the problem, and Fig. 2(b) depicts the state following a failure, after *O* has reacted to repair routing.

LIFEGUARD speeds up convergence and reduces path exploration by prepending to the production prefix *P*'s announcement, announcing *O*-*O*-*O* as the baseline path. If *O* detects that some networks (ASes *E* and *F* in Fig. 2) cannot reach *P* due to a problem in *A*, *O* updates the announcement to *O*-*A*-*O*. These two announcements are the same length and have the same next hop, and so, under default BGP, they are equally preferred. If an AS is using a route that starts with *O*-*O*-*O* and does not go through *A*, then receives an update that changes that route to start with *O*-*A*-*O* instead, it will likely switch to using the new route without exploring other

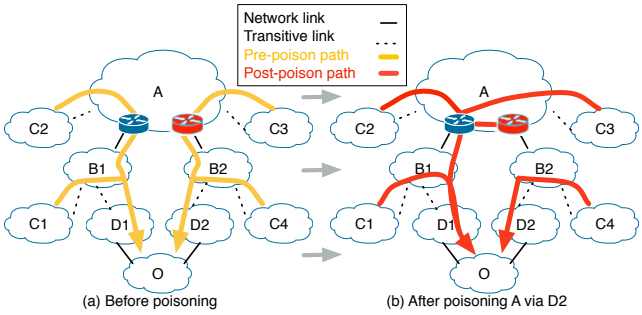

Figure 3: A case in which *LIFEGUARD* can use *selective poisoning*. By selectively poisoning *A* on announcements to *D2* and not on announcements to *D1*, *O* can cause traffic to avoid the link from *A* to *B2*, without disrupting how *C3* routes to *A* or how *C[1,2,4]* route to *O*.

options, converging instantly. We will show in a measurement study in §5.2 that this prepending smooths convergence, helping ease concerns that an automated response to outages might introduce needless routing instability. This approach is orthogonal to efforts to reduce convergence effects [18, 22, 24], which *LIFEGUARD* would benefit from.

3.1.2 Partially Poisoning ASes

LIFEGUARD tries to avoid cutting off an entire AS *A* and all ASes that lack routes that avoid *A*. We have three goals: (1) ASes cut off by poisoning should be able to use routes through *A* to reach *O* as soon as they work again; (2) if some paths through *A* work while others have failed, ASes using the working routes should be able to continue to if they lack alternatives; and (3) when possible, we should steer traffic from failed to working paths within *A*.

Advertising a less-specific sentinel prefix. While *O* is poisoning *A*, ASes like *F* that are "captive" behind *A* will lack a route [7]. To ensure that *F* and *A* have a route that covers *P*, *LIFEGUARD* announces a less-specific *sentinel prefix* that contains *P* (and can also contain other production prefixes). When *P* experiences problems, the system continues to advertise the sentinel with the baseline (unpoisoned) path. As seen in Fig. 2(b), ASes that do not learn of the poisoned path, because they are "captive" behind *A*, will receive the less specific prefix and can continue to try routing to the production prefix on it, through *A*, instead of being cut off. This effect is the *Backup Property* desired from AVOID_PROBLEM(A,P) and helps achieve goals (1) and (2).

Selectively poisoning to avoid AS links. Although most failures in a previous study were confined to a single AS, 38% occurred on an inter-AS link [13]. We use a technique we call *selective poisoning* to allow *LIFEGUARD*, under certain circumstances, to implement AVOID_PROBLEM(A-B,P). Poisoning does not provide a general solution to AS link avoidance, but, given certain topologies, selective poisoning can shift traffic within *A* onto working routes.

Specifically, under certain circumstances, *O* may be able to steer traffic away from a particular AS link without forcing it completely away from the ASes that form the link. Suppose *O* has multiple providers that connect to *A* via disjoint AS paths. Then *O* can poison *A* in advertisements to one provider, but announce an unpoisoned path through the other provider. Because the paths are disjoint, *A* will receive the poisoned path from one of its neighbors and the unpoisoned path from another, and it will only accept the unpoisoned path. So, *A* will route all traffic to *O*'s prefix to egress via the neighbor with the unpoisoned path. This *selective poisoning* shifts routes away from *A*'s link to the other neighbor, as well as possibly affecting which links and PoPs are used inside *A*.

Fig. 3 illustrates the idea. Assume *O* discovers a problem on the link between *A* and *B2*. This failure affects *C3*, but *C2* still has a working route through *A*, and *C4* still has a working route through *B2*. *O* would like to shift traffic away from the failing link, without forcing any networks except *A* to change which neighbor they select to route through. In other words, *O* would like to announce AVOID_PROBLEM(A-B2,P). If *O* only uses selective advertising without poisoning, announcing its prefix via *D1* and not *D2*, *C4*'s route will have to change. If *O* poisons *A* via both *D1* and *D2*, *C2* and *C3* will have to find routes that avoid *A*, and *A* will lack a route entirely (except via a less-specific prefix). However, by selectively poisoning *A* via *D2* and not via *D1*, *O* can shift *A* and *C3*'s routes away from the failing link, while allowing *C3* to still route along working paths in *A* and without disturbing any other routes. Selective poisoning functions like targeted prepending – prepending requires that *A* use path length to make routing decisions and potentially causes other ASes to shift from using routes through *D2*, whereas selective poisoning forces only *A* to change. In §5.2 we find that selective poisoning lets *LIFEGUARD* avoid 73% of the links we test.

4. APPLYING FAILURE AVOIDANCE

In the previous section, we described how *LIFEGUARD* uses BGP poisoning to approximate AVOID_PROBLEM(X,P). This allows us to experiment with failure avoidance today, and it gives ASes a way to deploy failure avoidance unilaterally. In this section, we describe how *LIFEGUARD* decides when to poison and which AS to poison, as well as how it decides when to stop poisoning an AS.

4.1 Locating a Failure

An important step towards fixing a reachability problem is to identify the network or router responsible for it. To be widely applicable and effective, we require our fault isolation technique to:

- be effective even if the system has control over only one of the endpoints experiencing a reachability problem.
- be accurate even if measurement probes to reachable targets appear to fail due to rate-limiting, chronic unresponsiveness, and/or being dropped on the reverse direction.
- integrate information from multiple measurement nodes, each with only partial visibility into routing behavior.

We assume that a routing failure between a pair of endpoints can be explained by a single problem. While addressing multiple failures is an interesting direction for future work, this paper focuses on single failures.

4.1.1 Overview of Failure Isolation

The failure isolation component of *LIFEGUARD* is a distributed system, using geographically distributed PlanetLab hosts to make data plane measurements to a set of monitored destinations. Because many outages are partial, *LIFEGUARD* uses vantage points with working routes to send and receive probes on behalf of those with failing paths. Because many failures are unidirectional [20], it adapts techniques from reverse traceroute [19] to provide reverse path visibility. In the current deployment, vantage points send pings to monitor destinations, and a vantage point triggers failure isolation when it experiences repeated failed pings to a destination.

LIFEGUARD uses historical measurements to identify candidates that could potentially be causing a failure, then systematically prunes the candidate set with additional measurements. We outline these steps first before describing them in greater detail.

1. **Maintain background atlas:** *LIFEGUARD* maintains an atlas of the round-trip paths between its sources and the monitored

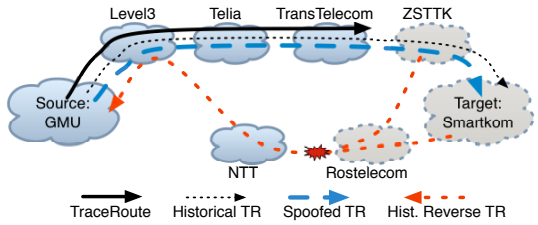

Figure 4: Isolation measurements conducted for an actual outage. With traceroute alone, the problem appears to be between TransTelecom and ZSTTK. Using spoofed traceroute, reverse traceroute, and historical path information, *LIFEGUARD* determines that the forward path is fine, but that Rostelecom no longer has a working path back to GMU.

targets to discern changes during failures and generate candidates for failure locations.

2. **Isolate direction of failure and measure working direction:** After detecting a failure, *LIFEGUARD* isolates the direction of failure to identify what measurements to use for isolation. Further, if the failure is unidirectional, it measures the path in the working direction using one of two measurement techniques.
3. **Test atlas paths in failing direction:** *LIFEGUARD* tests which subpaths still work in the failing direction by probing routers on historical atlas paths between the source and destination. It then remeasures the paths for responsive hops, thereby identifying other working routers.
4. **Prune candidate failure locations:** Routers with working paths in the previous step are eliminated. Then, *LIFEGUARD* blames routers that border the "horizon of reachability." This horizon divides routers that have connectivity to the source from those that lack that connectivity.

4.1.2 *Description of Fault Isolation*

We illustrate the steps that *LIFEGUARD* uses to isolate failures using an actual example of a failure diagnosed on February 24, 2011. At the left of Fig. 4 is one of *LIFEGUARD*'s vantage points, a PlanetLab host at George Mason University (labeled *GMU*). The destination, belonging to Smartkom in Russia, at the right, became unreachable from the source. For simplicity, the figure depicts hops at the AS granularity.

Maintain background atlas: In the steady state, *LIFEGUARD* uses traceroute and reverse traceroute to regularly map the forward and reverse paths between its vantage points and the destinations it is monitoring. During failures, this path atlas yields both a view of what recent paths looked like before the failure, as well as a historical view of path changes over time. These paths provide likely candidates for failure locations and serve as the basis for some of the isolation measurements we discuss below. Because some routers are configured to ignore ICMP pings, *LIFEGUARD* also maintains a database of historical ping responsiveness, allowing it to later distinguish between connectivity problems and routers configured to not respond to ICMP probes.

The figure depicts historical forward and reverse traceroutes with the dotted black and red lines, respectively. The thick, solid black line in Fig. 4 depicts the traceroute from GMU during the failure. Traceroute can provide misleading information in the presence of failures. In this case, the last hop is a TransTelecom router, suggesting that the failure may be adjacent to this hop, between TransTelecom and ZSTTK. However, without further information, operators cannot be sure, since the probes may have been dropped on the reverse paths back from hops beyond TransTelecom.

Isolate direction of failure and measure working direction: *LIFEGUARD* tries to isolate the direction of the failure using spoofed pings [20]. In the example, spoofed probes sent from GMU to Smartkom reached other vantage points, but no probes reached GMU, implying a reverse path failure. When the failure is unidirectional, *LIFEGUARD* measures the complete path in the working direction. Extending the spoofed ping technique, *LIFEGUARD* sends spoofed probes to identify hops in the working direction while avoiding the failing direction. For a reverse failure, *LIFEGUARD* finds a vantage point with a working path back from *D*, then has *S* send a spoofed traceroute to *D*, spoofing as the working vantage point. In the example, GMU issued a spoofed traceroute, and a vantage point received responses from ZSTTK and the destination. The blue dashed edges in Fig. 4 show the spoofed traceroute.

If the failure had been on the forward path, the system instead would have measured the working reverse path with a spoofed reverse traceroute from *D* back to *S*.

It is useful to measure the working direction of the path for two reasons. First, since the path is likely a valid policy-compliant path in the failing direction, it may provide a working alternative for avoiding the failure. Second, knowledge of the path in the working direction can guide isolation of the problem in the failing direction, as we discuss below.

Test atlas paths in failing direction: Once it has measured the working path, *LIFEGUARD* measures the responsive portion of the path. For forward and bidirectional failures, the source can simply issue a traceroute towards the destination.

For reverse failures, *LIFEGUARD* cannot measure a reverse traceroute from *D*, as such a measurement requires a response from *D* to determine the initial hops. Instead, *LIFEGUARD* has its vantage points, including *S*, ping: (1) all hops on the forward path from *S* to *D* and (2) all hops on historical forward and reverse paths between *S* and *D* in its path atlas. These probes test which locations can reach *S*, which cannot reach *S* but respond to other vantage points, and which are completely unreachable. *LIFEGUARD* uses its atlas to exclude hops configured never to respond. For all hops still pingable from *S*, *LIFEGUARD* measures a reverse traceroute to *S*.

In the example, *LIFEGUARD* found that NTT still used the same path towards GMU that it had before the failure and that Rostelecom no longer had a working path. We omit the reverse paths from most forward hops to simplify the figure. In this case, *LIFEGUARD* found that all hops before Rostelecom were reachable (denoted with blue clouds with solid boundaries), while all in Rostelecom or beyond were not (denoted with light-gray clouds with dashed boundaries), although they had responded to pings in the past.

Prune candidate failure locations: Finally, *LIFEGUARD* removes any reachable hops from the suspect set and applies heuristics to identify the responsible hop within the remaining suspects. For forward outages, the failure is likely between the last responsive hop in a traceroute and the next hop along the path towards the destination. *LIFEGUARD*'s historical atlas often contains paths through the last hop, providing hints about where it is trying to route.

For a reverse failure, *LIFEGUARD* considers reverse paths from *D* back to *S* that are in its atlas prior to the failure. For the most recent path, it determines the farthest hop *H* along that path that can still reach *S*, as well as the first hop *H'* past *H* that cannot. Given that *H'* no longer has a working path to *S*, contacting the AS containing *H'* or rerouting around it may resolve the problem.

If the failure is not in *H'*, one explanation is that, because *H'* lacked a route, *D* switched to another path which also did not work. In these cases, *LIFEGUARD* performs similar analysis on older historical paths from D, expanding the initial suspect set and repeating

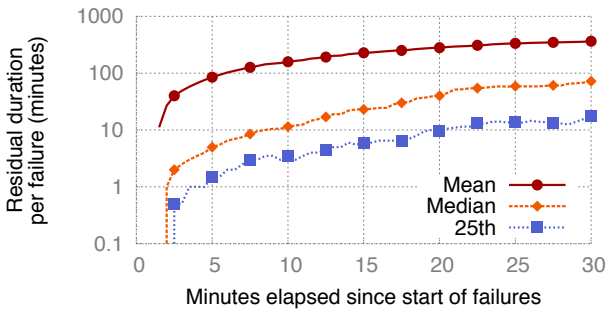

Figure 5: For our EC2 dataset, residual duration after outages have persisted for X minutes. The graph shows that, once a problem has persisted for a few minutes, it will most likely persist for at least a few more minutes unless we take corrective action.

the pruning. Since Internet paths are generally stable [37], the failing path will often have been used historically, and there will often be few historical paths between the source and destination.

Because both historical reverse paths from unresponsive forward hops traversed Rostelecom, it seems highly likely this is the point of failure. This conclusion is further supported by the fact that all unreachable hops except Rostelecom responded to pings from other vantage points, indicating that their other outgoing paths still worked. We provide details on this and other examples at `http://lifeguard.cs.washington.edu`, and we provide details on *LIFEGUARD*'s failure isolation in a tech report [32].

4.2 Deciding to Start and Stop Poisoning

Deciding whether to poison: As seen in Fig. 1, most outages resolve quickly. For the system to work effectively, it would be helpful to differentiate between outages that will clear up quickly and those that will persist. If routing protocols will quickly resolve a problem, then it would be better to wait, to avoid causing further routing changes. If the protocols will not restore connectivity quickly on their own, then poisoning may be a better approach.

The following analysis of our EC2 study (§2.1) shows that it is possible to differentiate between these cases with high likelihood. Fig. 5 shows the residual duration of these outages, given that they have already lasted for X minutes. The median duration of an outage in the study was only 90 seconds (the minimum possible given the methodology). However, of the 12% of problems that persisted for at least 5 minutes, 51% lasted at least another 5 minutes. Further, of the problems that lasted 10 minutes, 68% persisted for at least 5 minutes past that. *LIFEGUARD* triggers isolation after multiple rounds of failed pings, and it takes an average of 140 seconds to isolate a reverse path outage. If a problem persists through both those stages, then the results suggest that the problem is likely to persist long enough to justify using poisoning to fix it. We will show in §5.2 that poisoned routes converge within a few minutes in almost all cases, with little loss during the convergence period. So, if there are alternative paths that avoid the problem *LIFEGUARD* locates, our system should quickly restore connectivity.

Long-lasting problems account for much of the total unavailability. As a result, even if *LIFEGUARD* takes five minutes to identify and locate a failure before poisoning, and it then takes two minutes for routes to converge, we can still potentially avoid 80% of the total unavailability in our EC2 study. In §5.1, we will show that it is possible to determine (with high likelihood) whether alternate policy compliant paths will exist before deciding to poison an AS. If no paths exist, *LIFEGUARD* does not attempt to poison the AS.

Deciding when to unpoison: Once *LIFEGUARD* accurately identifies the AS *A* responsible for a problem, BGP poisoning can target it and cause other ASes to route around *A*. However, *A* will eventually resolve the underlying issue, at which point we would like to be able to revert to the unpoisoned path, allowing ASes to use paths through *A*, if preferred. When the poisoned announcement is in place, however, *A* will not have a path to the prefix in question.

LIFEGUARD uses a sentinel prefix to test reachability. Concerns such as aggregation and address availability influence the choice of sentinel. In our current deployment, the sentinel is a less specific prefix containing both the production prefix and a prefix that is not otherwise used. Responses to pings from the unused portion of the sentinel will route via the sentinel prefix, regardless of whether the hops also have the poisoned more-specific prefix. By sending active ping measurements from this prefix to destinations that had been unable to reach the production prefix prior to poisoning (e.g., *E* in Fig. 2), the system can detect when to unpoison the production prefix. If the sentinel is a less-specific without any unused prefixes, *LIFEGUARD* can instead ping the destinations within the poisoned AS (e.g., *A*) or within captives of the poisoned AS (e.g., *F*).

5. LIFEGUARD EVALUATION

To preview the results of our evaluation, we find that *LIFEGUARD*'s poisoning finds routes around the vast majority of potential problems, its approach is minimally disruptive to paths that are not experiencing problems, and its failure isolation can correctly identify the AS needing to be poisoned. Table 1 summarizes our key results; the following sections provide more details.

We deployed *LIFEGUARD*'s path poisoning using the BGP-Mux testbed [5], using its AS number and prefixes. *LIFEGUARD* connected to a BGP-Mux instance at Georgia Tech, which served as the Internet provider for the BGP-Mux AS (and hence for *LIFEGUARD*). For the poisoning experiments in this section, *LIFEGUARD* announced prefixes via Georgia Tech into the commercial Internet.

We assessed the effects of poisoning in the absence of failures. To obtain ASes to poison, we announced a prefix and harvested all ASes on BGP paths towards the prefix from route collectors [26, 31]. We excluded all Tier-1 networks, as well as Cogent, as it is Georgia Tech's main provider. In §5.2 and §7.1, we evaluate techniques to poison even these large ASes. For each of the remaining harvested ASes,[3] we first went from a baseline of *O* to a poisoned announcement *O-A-O*, then repeated the experiment starting from a baseline of *O-O-O*. We kept each announcement in place for 90 minutes to allow convergence and to avoid flap dampening effects. For the duration of the experiments, we also announced an unpoisoned prefix to use for comparisons.

5.1 Efficacy

Do ASes find new routes that avoid a poisoned AS? We monitored BGP updates from public BGP route collectors to determine how many ASes found an alternate path after we poisoned an AS on their preferred path. There were 132 cases in which an AS peering with a route collector was using a path through one of the ASes we poisoned. In 102 of the 132 cases (77%), the route collector peer found a new path that avoided the poisoned AS. Two-thirds of the cases in which the peer could not find a path were instances in which we had poisoned the only provider of a stub AS.

Do alternate policy-compliant routes exist in general? We analyzed a large AS topology to show that, in the common case, alternate paths exist around poisoned ASes. The topology, along

[3]We announced our experiments on the NANOG mailing list and allowed operators to opt out their ASes. None did. A handful opted out of an earlier Georgia Tech study, and we honored their list.

Criteria	Summary	Experimental Result
Effectiveness (§5.1)	Most edge networks have routes that avoid poisoned ASes, and we can calculate which do *a priori*	77% of poisons from BGP-Mux 90% of poisons in large-scale simulation
Disruptiveness (§5.2)	Working routes that already avoid the problem AS reconverge quickly after poisoning	95% of paths converge instantly
	Minimal loss occurs during convergence	Less than 2% packet loss in 98% of cases
Accuracy (§5.3)	Locates failures as if it had traceroutes from both ends	Consistent results for 93% of inter-PlanetLab failures
	Isolates problems that traceroute alone misdiagnoses	40% of cases differ from traceroute
Scalability (§5.4)	Quickly isolates problems with reasonable overhead	140 s for poisoning candidates, 280 probes per failure
	Reasonably small additional update load for addressing most of observed unavailability	$<$ 1% if 1% of ISPs use *LIFEGUARD* $<$ 10% if 50% of ISPs use *LIFEGUARD*

Table 1: Key results of our *LIFEGUARD* evaluation, demonstrating its viability for addressing long-duration outages.

with AS relationships, is from a dataset combining public BGP feeds with more than 5 million AS paths between BitTorrent (BT) peers [9]. To simulate poisoning an AS A on a path from a source S to an origin O, we remove all of A's links from the topology. We then check if S can restore connectivity while avoiding A (i.e., a path exists between S and O that obeys export policies).

We check policies using the three-tuple test [25], as in §2.2. This approach may miss alternate paths if the valley-free assumption is too strict,[4] and rarely used backup paths may not be in our topology. Conversely, it may identify valley-free paths not used in practice.

To establish the validity of this methodology, we simulated the Georgia Tech poisonings. In 92.5% of cases, the simulation found an alternate path if and only if the AS found such a path following our actual poisoning. In the remaining 7.5% of cases, the simulation found an alternate path, but in practice the AS failed to find one. In these cases, the source was an academic network connected to both a commercial provider and an academic provider (which only provides routes to academic destinations). These connections made the AS multi-homed in our simulation. In practice, however, the AS seems to reject routes from its commercial provider if the route contained an academic AS, and we had poisoned one.

Having established that our simulation closely predicts the results of actual poisonings, we simulated poisoning ASes on the BT and BGP feed paths. For each AS path with length greater than 3 (i.e., traversing at least one transit AS in addition to the destination's provider), we iterated over all the transit ASes in the path except the destination's immediate provider and simulated poisoning them one at a time.[5] An alternate path existed in 90% of the 10M cases. We then simulated poisoning the failures isolated by *LIFEGUARD* in June 2011. Alternate paths existed in 94% of them.

5.2 Disruptiveness

How quickly do paths converge after poisoning? We used updates from the BGP collectors to measure convergence delay after poisoning. We will show that, in most cases, if an AS was not routing through the AS we poisoned, it re-converges essentially instantly to its original path, requiring only a single update to pass on the poison. Global convergence usually takes at most a few minutes.

First, we assess how quickly ASes that were using the poisoned AS settle on a new route that avoids it. We also assess how quickly routes converge for ASes that were not using the poisoned AS. As explained above, we poisoned each harvested AS twice each using different pre-poisoning baseline paths. After each announcement, for each AS that peers with a route collector, we measured the delay from when the AS first announced an update to the route collector

[4]It is well known that not all paths are valley-free in practice, and we observed violations in the BitTorrent traceroutes.

[5]A multi-homed destination can use selective advertising to avoid a particular provider. A single-homed destination can never avoid having paths traverse its provider.

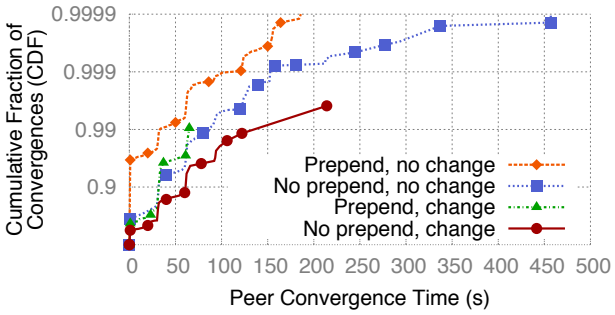

Figure 6: Convergence time for route collector peer ASes after poisoned announcements. A data point captures the convergence for one peer AS after one poisoned announcement. *Change* vs. *no change* indicates if the peer had to change its path because it had been routing through the poisoned AS. For *Prepend*, the baseline announcement before poisoning was *O-O-O*, whereas for *No prepend* it was *O*. The poisoned announcement was always *O-A-O*. Prepending reduces path exploration by keeping announcement length consistent.

to when it announced its stable post-poisoning route. We leave out (route collector peer, poisoned AS) pairs if the peer AS did not have a path to the *LIFEGUARD* prefix following the poisoning.

As seen in Fig. 6, a baseline announcement of *O-O-O* greatly speeds convergence. More than 95% of the time, ASes that were not using the poisoned AS converged essentially instantly upon receiving the poisoned update, as they did not need to change their path, and 99% of them converged within 50 seconds (*Prepend, No Change* line). In comparison, if we simply announce O as the baseline before poisoning, less than 70% of the unaffected ASes converged instantly, and 94% converged within 50 seconds (*No Prepend, No Change* line). Similarly, using *O-O-O* as the baseline helps ASes that had been routing via A settle on a new route faster: 96% converged within 50 seconds, compared to only 86% if we use O as the baseline. Prepending keeps the announcements a consistent length, which reduces path exploration for ASes not routing through A. In fact, with prepending, 97% of unaffected ASes made only a single update, informing neighbors only of the insertion of the poisoned AS A into the route. Without prepending, only 64% of these ASes made only one update. The other 36% explored alternatives before reverting back to their original path.

These graphs capture per-AS convergence. Because announcements need to propagate across the Internet and are subject to protocol timers as they do so, different ASes receive a poisoned announcement at different times.

We also assessed how long global convergence took, from the time the first router collector receives an update for our prefix until all route collector peer ASes had converged to stable routes. With prepending, global convergence took at most 91 seconds in the median case, at most two minutes for 75% of poisonings, and at most 200s in 90% of poisonings. In contrast, without prepending, the

50th, 75th, and 90th percentiles are 133s, 189s, and 226s. Compared to the delay following a poisoning, it generally takes slightly less time for routes to converge globally when we remove the poison and revert to the baseline announcement. Because most ASes that were not using the poisoned AS reconverge to their original path without exploring other options, we would not expect them to experience transient loss during the global convergence period.

How much loss accompanies convergence? Our results indicate the packet loss during convergence is minimal. We calculated loss rate during convergence following our poisonings that used a baseline of *O-O-O*. Every ten seconds for the duration of the experiment, we issued pings from the poisoned *LIFEGUARD* prefixes to all 308 working PlanetLab sites. In general, many of the sites were not routing via any particular poisoned AS, and so this experiment lets us evaluate how much we disrupt working paths. We filtered out cases in which loss was clearly due to problems unrelated to poisoning, and we excluded a PlanetLab site if it was completely cut off by a particular poisoning. Following 60% of poisonings, the overall loss rate during the convergence period was less than 1%, and 98% of poisonings had loss rates under 2%. Some convergence periods experienced brief spikes in loss, but only 2% of poisonings had any 10 second round with a loss rate above 10%.

Can poisoning shift routes off an AS link without completely disabling either AS? We demonstrate selective poisoning using two BGP-Mux sites, University of Washington (UWash) and University of Wisconsin (UWisc). Paths from most PlanetLab nodes to UWash pass through the Internet2 (I2) Seattle PoP, then to Pacific Northwest Gigapop, and to UWash. Most PlanetLab paths to UWisc pass through I2's Chicago PoP, then through WiscNet, before reaching UWisc. So, the BGP-Mux AS has UWash and UWisc as providers, and they connect via disjoint paths to different I2 PoPs.

We tested if we could shift traffic away from the I2 Chicago PoP, supposing the link to WiscNet experienced a silent failure. We advertised the same two prefixes from UWash and UWisc. We announced the first prefix unpoisoned from both. We poisoned I2 in UWisc's announcements of the second prefix, but had UWash announce it unpoisoned.

First, we show that paths that were not using I2 would not be disrupted. We looked at paths to the two prefixes from 36 RIPE RIS BGP peers. For 33 of them, the paths to the two prefixes were identical and did not use I2. The other three ASes routed to the unpoisoned prefix via I2 and WiscNet. For the selectively poisoned prefix, they instead routed via I2 and PNW Gigapop, as expected.

We then compare traceroutes from PlanetLab hosts to the two prefixes, to show that paths through I2 avoid the "problem." For the unpoisoned prefix, over 100 PlanetLab sites routed through I2 to WiscNet. Focusing on just these PlanetLab sites, we assessed how they routed towards the other prefix, which we selectively poisoned for I2 from UWisc. For that prefix, all the sites avoided the link from I2 to WiscNet. All but three of the sites routed via PNW Gigapop and UWash, as we intended. The remaining three – two sites in Poland and one in Brazil – used Hurricane Electric to reach UWisc. Excepting these three sites, selective poisoning allowed us to shift paths within a targeted network without changing how networks other than the target routed.

Are Internet paths diverse enough for selective poisoning to be effective? This technique may provide a means to partially poison large networks, and services with distributed data centers may have the type of connectivity required to enable it. In our second selective poisoning experiment, we approximated this type of deployment, and we assess how many ASes we could selectively poison. To make this assessment, we need to identify whether ASes select from disjoint paths, which would allow us to selectively poison them. It is hard to perform this assessment in general, because we need to know both which path an AS is using and which paths it might use if that path became unavailable.

Route collectors and BGP-Mux let us experiment in a setting in which we have access to these paths. We announced a prefix simultaneously via BGP-Muxes at UWash, UWisc, Georgia Tech, Princeton, and Clemson. This setup functions as an AS with five PoPs and one provider per PoP. We iterated through 114 ASes that provide BGP feeds. For each pair of AS *A* and BGP-Mux *M* in turn, we poisoned *A* from all BGP-Muxes except *M* and observed how *A*'s route to our prefix varied with *M*. We found that selective poisoning allowed us to avoid 73% of the first hop AS links used by these peers, while still leaving the peers with a route to our prefix. In §2.3, we found that these five university providers allowed us to avoid 90% of these links on forward paths to these same ASes.

5.3 Accuracy

Having demonstrated that *LIFEGUARD* can often use BGP poisoning to route traffic around an AS or AS link without causing widespread disruption, we assess *LIFEGUARD*'s accuracy in locating failures. We show that *LIFEGUARD* seems to correctly identify the failed AS in most cases, including many that would be not be correctly found using only traceroutes. Our tech report provides further analysis of *LIFEGUARD*'s failure isolation [32].

Are LIFEGUARD*'s results consistent with what it would find with control of both ends of a path?* In general, obtaining ground truth for wide-area network faults is difficult: emulations of failures are not realistic, and few networks post outage reports publicly. Due to these challenges, we evaluate the accuracy of *LIFEGUARD*'s failure isolation on paths between a set of PlanetLab hosts used as *LIFEGUARD* vantage points and a disjoint set of PlanetLab hosts used as targets.[6] Every five minutes, we issued traceroutes from all vantage points to all targets and vice versa. In isolating the location of a failure between a vantage point and a target, we only gave *LIFEGUARD* access to measurements from its vantage points. We checked if *LIFEGUARD*'s conclusion was consistent with traceroutes from the target, "behind" the failure. *LIFEGUARD*'s location was consistent if and only if (1) the traceroute in the failing direction terminated in the AS *A* blamed by *LIFEGUARD*, and (2) the traceroute in the opposite direction did not contradict *LIFEGUARD*'s conclusion. The traceroute contradicted the conclusion if it included responses from *A* but terminated in a different AS. The responses from *A* indicate that some paths back from *A* worked. We lack sufficient information to explain these cases, and so we consider *LIFEGUARD*'s result to be inconsistent.

We examined 182 unidirectional isolated failures from August and September, 2011. For 169 of the failures, *LIFEGUARD*'s results were consistent with traceroutes from the targets. The remaining 13 cases were all forward failures. In each of these cases, *LIFEGUARD* blamed the last network on the forward traceroute (just as an operator with traceroute alone might). However, the destination's traceroute went through that network, demonstrating a working path from the network back to the destination.

Does LIFEGUARD *locate failures of interest to operators?* We searched the Outages.org mailing list [28] for outages that intersected *LIFEGUARD*'s monitored paths [32]) and found two interesting examples. On May 25th, 2011, three of *LIFEGUARD*'s vantage points detected a forward path outage to three distinct locations. The system isolated all of these outages to a router in Level3's Chicago PoP. Later that night, a network operator posted the following message to the mailing list: "Saw issues routing through

[6]We cannot issue spoofed packets or make BGP announcements from EC2, and so we cannot use it to evaluate our system.

Chicago via Level3 starting around 11:05 pm, cleared up about 11:43 pm." Several network operators corroborated this report.

In the second example, a vantage point in Albany and one in Delaware observed simultaneous outages to three destinations: a router at an edge AS in Ohio and routers in XO's Texas and Chicago PoPs. *LIFEGUARD* identified all Albany outages and some of the Delaware ones as reverse path failures, with the remaining Delaware one flagged as bidirectional. All reverse path failures were isolated to an XO router in Dallas, and the bidirectional failure was isolated to an XO router in Virginia. Several operators subsequently posted to the Outages.org mailing list reporting problems with XO in multiple locations, which is likely what *LIFEGUARD* observed.

Does LIFEGUARD *provide benefit beyond traceroute?* We now quantify how often *LIFEGUARD* finds that failures would be incorrectly isolated using only traceroute, thus motivating the need for the system's more advanced techniques. In the example shown in Fig. 4, traceroutes from GMU seem to implicate a problem forwarding from TransTelecom, whereas our system located the failure as being along the reverse path, in Rostelecom. For the purposes of this study, we consider outages that meet criteria that make them candidates for rerouting and repair: (1) multiple sources must be unable to reach the destination, and these sources must be able to reach at least 10% of all destinations at the time, reducing the chance that it is a source-specific issue; (2) the failing traceroutes must not reach the destination AS, and the outage must be partial, together suggesting that alternate AS paths exist; (3) and the problem must not resolve during the isolation process, thereby excluding transient problems. During June 2011, *LIFEGUARD* identified 320 outages that met these criteria [32]. In 40% of cases, the system identified a different suspected failure location than what one would assume using traceroute alone. Further, even in the other 60% of cases, an operator would not currently know whether or not the traceroute was identifying the proper location.

5.4 Scalability

How efficiently does LIFEGUARD *refresh its path atlas? LIFEGUARD* regularly refreshes the forward and reverse paths it monitors. Existing approaches efficiently maintain forward path atlases based on the observations that paths converge as they approach the source/destination [12] and that most paths are stable [11]. Based on these observations, we implemented a reverse path atlas that caches probes for short periods, reuses measurements across converging paths, and usually refreshes a stale path using fewer probes than would be required to measure from scratch. In combination, these optimizations enable us to refresh paths at an average (peak) rate of 225 (502) reverse paths per minute. We use an amortized average per path of 10 IP option probes (vs. the 35 reported in existing work [19]) and slightly more than 2 forward traceroutes. It may be possible to improve scalability in the future by focusing on measuring paths between "core" PoPs whose routing health and route changes likely influence many paths,[7] and by scheduling measurements to minimize the impact of router-specific rate limits.

What is the probing load for locating problems? Fault isolation requires approximately 280 probe packets per outage. We found that *LIFEGUARD* isolates failures on average much faster than it takes long-lasting outages to be repaired. For bidirectional and reverse path outages, potential candidates for poisoning, *LIFEGUARD* completed isolation measurements within 140 seconds on average.

[7]For example, 63% of iPlane traceroutes traverse at least one of the most common 500 PoPs (0.3% of PoPs) [17].

		$d = 5$ minutes		$d = 15$ min.		$d = 60$ min.	
		$T = 0.5$	1.0	0.5	1.0	0.5	1.0
I	0.01	393	783	137	275	58	115
	0.1	3931	7866	1370	2748	576	1154
	0.5	19625	39200	6874	13714	2889	5771

Table 2: Number of additional daily path changes due to poisoning for fraction of ISPs using *LIFEGUARD* (I), fraction of networks monitored for reachability (T), and duration of outage before poisoning (d). For comparison, routers currently make 110K–315K updates per day.

What load will poisoning induce at scale? An important question is whether *LIFEGUARD* would induce excessive load on routers if deployed at scale. Our earlier study showed that, by prepending, we can reduce the number of updates made by each router after a path poisoning. In this section, we estimate the Internet-wide load our approach would generate if a large number of ISPs used it. While our results serve as a rough estimate, we find that the number of path changes made at each router is low.

The number of daily path changes per router our system would produce at scale is $I \times T \times P(d) \times U$, where I is the fraction of ISPs using our approach, T is the fraction of ASes each ISP is monitoring with *LIFEGUARD*, $P(d)$ is the aggregate number of outages per day that have lasted at least d minutes and are candidates for poisoning, and U is the average number of path changes per router generated by each poison. Based on our experiments, $U = 2.03$ for routers that had been routing via the poisoned AS, and $U = 1.07$ for routers that had not been routing via the poisoned AS. For routers using the poisoned AS, BGP should have detected the outage and generated at least one path update in response. For routers not routing through it, the updates are pure overhead. Thus, poisoning causes affected routers to issue an additional 1.03 updates and and unaffected routers an additional 1.07 updates. For simplicity, we set $U = 1$ in this analysis.

We base $P(d)$ on the Hubble dataset of outages on paths between PlanetLab sites and 92% of the Internet's edge ASNs [20]. We filter this data to exclude cases where poisoning is not effective (e.g., complete outages and outages with failures in the destination AS). We assume that the Hubble targets were actually monitoring the PlanetLab sites and define $P(d) = H(d)/(I_h T_h)$, where $H(d)$ is the total number of poisonable Hubble outages per day lasting at least d minutes, $I_h = 0.92$ is the fraction of all edge ISPs that Hubble monitored, and $T_h = 0.01$ is our estimate for the fraction of total poisonable (transit) ASes on paths from Hubble VPs to their targets. Because the smallest d that Hubble provides is 15 minutes, we extrapolate the Hubble outage distribution based on the EC2 data to estimate the poisoning load for $d = 5$.

Scaling the parameters estimates the load from poisoning under different scenarios. In Table 2, we vary the fraction of participating ISPs (I), the fraction of poisonable ASes being monitored (T), and the minimum outage duration before poisoning is used (d). We scale the number of outages linearly with I and T. We scale with d based on the failure duration distribution from our EC2 study.

LIFEGUARD could cause a router to generate from tens to tens of thousands of additional updates. For reference, a single-homed edge router peering with AS131072 sees an average of approximately 110K updates per day [4], and the Tier-1 routers we checked made 255K-315K path updates per day [26]. For cases where only a small fraction of ASes use poisoning ($I \leq 0.1$), the additional load is less than 1%. For large deployments ($I = 0.5, T = 1$) and short delays before poisoning ($d = 5$), the overhead can become significant (35% for the edge router, 12-15% for Tier-1 routers). We note that reducing the number of monitored paths or waiting longer to poison can easily reduce the overhead to less than 10%.

6. LIFEGUARD CASE STUDY

To demonstrate *LIFEGUARD*'s ability to repair a data plane outage by locating a failure and then poisoning, we describe a failure between the University of Wisconsin and a PlanetLab node at National Tsing Hua University in Taiwan. *LIFEGUARD* announced production and sentinel prefixes via the University of Wisconsin BGP-Mux. *LIFEGUARD* had monitored the PlanetLab node for a month, gathering historical data. On October 3-4, 2011, nodes in the two prefixes exchanged test traffic with the PlanetLab node, and we sent traceroutes every 10 minutes from the PlanetLab node towards the test nodes to track its view of the paths. We use the measurements from the Taiwanese node to evaluate *LIFEGUARD*.

After experiencing only transient problems during the day, at 8:15pm on October 3, the test traffic began to experience a persistent outage. When *LIFEGUARD* isolated the direction of the outage, spoofed pings from Wisconsin reached Taiwan, but spoofed pings towards Wisconsin failed, indicating a reverse path problem. *LIFEGUARD*'s atlas revealed two historical paths from Taiwan back to Wisconsin. Prior to the outage, the PlanetLab node had been successfully routing to Wisconsin via academic networks. According to the atlas, this route had been in use since 3pm, but, prior to that, the path had routed through UUNET (a commercial network) for an extended period. The UUNET and academic paths diverged one AS before UUNET (one AS closer to the Taiwanese site). *LIFEGUARD* issued measurements to see which of the hops on these reverse paths still could reach Wisconsin. These measurements establish a reachability horizon with UUNET and the ASes before it behind the failure. During the failure, all routers along the academic path from the divergence point to Wisconsin were still responsive to pings, meaning they had a route to the University of Wisconsin. However, hops in UUNET no longer responded to probes. In fact, hand inspection of traceroutes from the PlanetLab node to Wisconsin showed that, at 8:15pm, the path had switched to go via UUNET, and the traceroutes had been terminating in UUNET.

Because the hops on the academic route had paths to Wisconsin, it was a likely viable alternative to the broken path, and we used *LIFEGUARD* to poison UUNET. For a brief period after *LIFEGUARD* announced the poison, test traffic was caught in a convergence loop. After convergence, the test traffic and a traceroute from Taiwan successfully reached the production prefix via academic networks. Traceroutes to the sentinel prefix continued to fail in UUNET until just after 4am on October 4, when the path through UUNET began to work again. In summary, *LIFEGUARD* isolated an outage and used poisoning to re-establish connectivity until the outage was resolved. Once repaired, the poison was no longer necessary, and *LIFEGUARD* reverted to the baseline unpoisoned announcement.

7. DISCUSSION

7.1 Poisoning Anomalies

Certain poisonings cause anomalous behavior. Some networks disable loop detection, accepting paths even if they contain their AS. Other networks do not accept an update from a customer if the path contains a peer of the network. We discuss these two issues.

Some networks with multiple remote sites communicate between sites across the public Internet, using the same AS number across sites. One approach is to disable BGP's loop prevention to exchange prefixes from the remote sites, even though they share an origin AS. The paths for these prefixes allow the remote sites to route to each other using BGP. Fortunately, best practices mandate that, instead of disabling loop detection altogether, networks should set the maximum occurrences of their AS number in the path. For instance, AS286 accepts updates if it is already in the AS path. Inserting AS286 twice into the AS path, however, causes it to drop the update, thus enabling the use of poisoning. We expect ASes that use the public Internet to communicate between remote sites to be stubs (meaning they have no customers). We use poisoning to bypass faulty transit networks, so have no need to poison stubs.

Some networks do not accept an update from a customer if the path contains one of the network's peers. For example, Cogent will not accept an update that includes one of its Tier-1 peers in the path, meaning that announcements poisoning one of these ASes via Georgia Tech did not propagate widely. However, we could poison them via BGP-Mux instances at universities that were not Cogent customers, and 76% of route collector peers were able to find a path that avoided a poisoned AS through which they had previously been routing. While the filtering reduced the coverage of our poisoning studies and will prevent poisoning from rerouting around some failures, it is likely not a big limitation. First, most failures occur outside of large networks (such as Cogent and Tier-1s) [32, 36]. Second, we used poisoning to experiment with AS avoidance, but most of the techniques and principles still apply to other implementations, such as a modification of BGP to include a new signed AVOID_PROBLEM(X,P) notification.

7.2 Address Use by the Sentinel Prefix

We proposed using a less-specific prefix with an unused sub-prefix as a sentinel. We discuss the trade-offs when using three alternative approaches. First, absent any sentinel, it is not clear how to check for failure recovery or to provide a backup unpoisoned route. Second, if an AS has an unused prefix that is not adjacent to the production prefix, it can use the unused prefix as a sentinel, even though it may lack a super-prefix that covers both prefixes. This approach allows the AS to check when the failed route is repaired. However, this scheme does not provide a "backup" route to the production prefix for networks captive behind the poisoned AS. Third, if an AS lacks unused address space to serve as a sentinel, a less-specific prefix will ensure that the poisoned AS and ASes captive behind it still have a route, even though the less-specific prefix does not include an unused sub-prefix. Pings to the poisoned AS and its captives will return via routes to the unpoisoned less-specific prefix, and so they can be used to check for repairs.

Alternatively, a provider with multiple prefixes hosting the same service can use DNS redirection to test when a problem has resolved, without using additional addresses. Such providers often use DNS to direct a client to a nearby data prefix. Our scheme relies on clients using the same route to reach all prefixes in the absence of poison. To establish that this property holds, at least for Google, we resolved a Google hostname at 20 PlanetLab sites around the world, yielding a set of Google IP addresses from various data centers. We then issued traceroutes from the PlanetLab sites to the set of IP addresses. Each PlanetLab site used a consistent path to reach Google's network for all IP addresses. Google then routed the traffic to a particular data center. With this consistent routing, if a provider discovers a routing problem affecting a set of clients *C*, it could poison the prefix *P1* serving those clients. It need not poison its prefix *P2* that serves other clients (possibly from a different data center). Periodically, the provider's DNS resolvers could give a client from *C* an address from *P2* and an address from *P1*, with *P1* serving as a failover. By checking server logs, the provider could discern if the client was able to reach *P2*. When clients can reach *P2*, the provider can remove the poison on *P1*.

8. RELATED WORK

Failure isolation. Hubble [20] informs our failure isolation approach. *LIFEGUARD* makes heavy use of reverse traceroute [19].

Feamster et al. measured failures between vantage points using pings and traceroutes [13]. Upon detected a problem, PlanetSeer triggered measurements from multiple vantage points [36]. However, after identifying partial outages, all further probes and analysis focusing only on the local measurements.

Feldmann et al. located the cause of BGP updates by assuming that it was on the old path or the new path. They could pinpoint the likely cause of a BGP update to one or two ASes [14]. iSpy [39] identified prefix hijacking using a technique similar to *LIFEGUARD*'s reachability horizon, but assumed that paths were symmetric. *LIFEGUARD* uses reverse traceroute and spoofed probes to more accurately determine the reachability horizon.

Failure Avoidance. Previous research used poisoning as a measurement tool to uncover network topology [10] and default routes [8]. While inspired by that work, we propose using poisoning operationally as a means to improve availability.

In the absence of a solution to long-lasting, partial outages, researchers and companies have proposed systems to detour traffic around outages using overlay paths [2, 6, 16]. These approaches provide a great alternative when no other solutions exist. In contrast to detouring, our poisoning approach does not require the expense of an overlay and can carry traffic at core Internet data rates along policy-compliant BGP paths that avoid the identified problem.

Entact used overlapping BGP prefixes to simultaneously measure alternative paths [38]. Bush et al. used an anchor prefix to determine whether a test prefix should be reachable [8]. Similarly, *LIFEGUARD* uses a sentinel prefix to test for repairs along the failing path while rerouting traffic to a path avoiding the outage.

MIRO proposed to enable AS avoidance and other functionality by allowing ASes to advertise and use multiple inter-domain paths [34]. The proposal retains much of the quintessence of BGP and routing policies, but it requires modifications to protocols and routers, meaning it faces a slow path to adoption. Our deployable solutions and results on failure avoidance could perhaps present some of the argument for the adoption of MIRO-like modifications.

9. CONCLUSION

Increasingly, Internet and cloud-based services expect the Internet to deliver high availability. Nevertheless, partial outages that last for hours occur frequently, accounting for a large portion of the Internet's overall end-to-end downtime. Such an outage can occur when an ISP advertises a BGP route but fails to deliver packets on the route. To address these problems, we introduce *LIFEGUARD*, a system that locates the AS at fault and routes around it. Using multiple vantage points and spoofed probes, we show that *LIFEGUARD* can identify the failing AS in the wild. We show that we can use BGP poisoning to cause routes to avoid an AS without disrupting working routes. An ISP can deploy our approach unilaterally today. *LIFEGUARD* enables experiments that we hope will motivate the need for and design of new route control mechanisms.

Acknowledgments

We gratefully acknowledge our shepherd Craig Labovitz and the SIGCOMM reviewers. The BGP-Mux system would not be available without the help of researchers and network administrators at the sites: Scott Friedrich, Michael Blodgett, Jeff Fitzwater, Jennifer Rexford, Larry Billado, Kit Patterson, and Schyler Batey. This work was partially funded by Google, Cisco, FAPEMIG, NSF (CNS-0905568 and CNS-1040663), and the NSF/CRA Computing Innovation Fellowship program. We are thankful for their support.

10. REFERENCES

[1] Abilene Internet2 network. http://www.internet2.edu/network/.
[2] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP*, 2001.
[3] R. Austein, S. Bellovin, R. Bush, R. Housley, M. Lepinski, S. Kent, W. Kumari, D. Montgomery, K. Sriram, and S. Weiler. BGPSEC protocol. http://tools.ietf.org/html/draft-ietf-sidr-bgpsec-protocol.
[4] The BGP Instability Report. http://bgpupdates.potaroo.net/instability/bgpupd.html.
[5] BGPMux Transit Portal. http://tp.gtnoise.net/.
[6] C. Bornstein, T. Canfield, and G. Miller. Akarouting: A better way to go. In *MIT OpenCourseWare 18.996*, 2002.
[7] M. A. Brown, C. Hepner, and A. C. Popescu. Internet captivity and the de-peering menace. In *NANOG*, 2009.
[8] R. Bush, O. Maennel, M. Roughan, and S. Uhlig. Internet optometry: assessing the broken glasses in Internet reachability. In *IMC*, 2009.
[9] K. Chen, D. R. Choffnes, R. Potharaju, Y. Chen, F. E. Bustamante, D. Pei, and Y. Zhao. Where the sidewalk ends: Extending the Internet AS graph using traceroutes from P2P users. In *CoNEXT*, 2009.
[10] L. Colitti. *Internet Topology Discovery Using Active Probing*. PhD thesis, University di "Roma Tre", 2006.
[11] I. Cunha, R. Teixeira, and C. Diot. Predicting and tracking Internet path changes. In *SIGCOMM*, 2011.
[12] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient algorithms for large-scale topology discovery. In *SIGMETRICS*, 2005.
[13] N. Feamster, D. G. Andersen, H. Balakrishnan, and M. F. Kaashoek. Measuring the effects of internet path faults on reactive routing. In *SIGMETRICS*, 2003.
[14] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs. Locating Internet routing instabilities. In *SIGCOMM*, 2004.
[15] L. Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM TON*, 2001.
[16] 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*, 2004.
[17] iPlane. `http://iplane.cs.washington.edu`.
[18] J. P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkataramani. Consensus routing: The Internet as a distributed system. In *NSDI*, 2008.
[19] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. van Wesep, A. Krishnamurthy, and T. Anderson. Reverse traceroute. In *NSDI*, 2010.
[20] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. Anderson. Studying black holes in the Internet with Hubble. In *NSDI*, 2008.
[21] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. Detection and localization of network black holes. In *INFOCOM*, 2007.
[22] N. Kushman, S. Kandula, and D. Katabi. R-BGP: Staying connected in a connected world. In *NSDI*, 2007.
[23] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. In *SIGCOMM*, 2000.
[24] K. K. Lakshminarayanan, M. C. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. In *SIGCOMM*, 2007.
[25] H. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane Nano: Path Prediction for Peer-to-Peer Applications. In *NSDI*, 2009.
[26] D. Meyer. RouteViews. `http://www.routeviews.org`.
[27] P. Mohapatra, J. Scudder, D. Ward, R. Bush, and R. Austein. BGP prefix origin validation. http://tools.ietf.org/html/draft-ietf-sidr-pfx-validate.
[28] Outages mailing list. `http://isotf.org/mailman/listinfo/outages`.
[29] Packet clearing house. http://www.pch.net/home/index.php.
[30] B. Quoitin and O. Bonaventure. A survey of the utilization of the BGP community attribute. Internet draft, draft-quoitin-bgp-comm-survey-00, 2002.
[31] RIPE RIS. `http://www.ripe.net/ris/`.
[32] C. Scott. LIFEGUARD: Locating Internet Failures Effectively and Generating Usable Alternate Routes Dynamically. Technical report, Univ. of Washington, 2012.
[33] UCLA Internet topology. `http://irl.cs.ucla.edu/topology/`.
[34] W. Xu and J. Rexford. MIRO: Multi-path Interdomain ROuting. In *SIGCOMM*, 2006.
[35] J. Yates and Z. Ge. Network Management: Fault Management, Performance Management and Planned Maintenance. Technical report, AT&T Labs, 2009.
[36] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer: Internet path failure monitoring and characterization in wide-area services. In *OSDI*, 2004.
[37] Y. Zhang, V. Paxson, and S. Shenker. The stationarity of Internet path properties: Routing, loss, and throughput. *ACIRI Technical Report*, 2000.
[38] Z. Zhang, M. Zhang, A. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian. Optimizing cost and performance in online service provider networks. In *NSDI*, 2010.
[39] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush. iSpy: detecting IP prefix hijacking on my own. In *SIGCOMM*, 2008.