PoiRoot: Investigating the Root Cause of Interdomain Path Changes

Umar Javed
University of Washington
ujaved@cs.washington.edu

Italo Cunha
Universidade Federal de Minas Gerais
cunha@dcc.ufmg.br

David R. Choffnes
University of Washington
Northeastern University
choffnes@cs.washington.edu

Ethan Katz-Bassett
University of Southern California
ethan.kb@usc.edu

Thomas Anderson
University of Washington
tom@cs.washington.edu

Arvind Krishnamurthy
University of Washington
arvind@cs.washington.edu

ABSTRACT

Interdomain path changes occur frequently. Because routing protocols expose insufficient information to reason about all changes, the general problem of identifying the root cause remains unsolved. In this work, we design and evaluate *PoiRoot*, a real-time system that allows a provider to accurately isolate the root cause (the network responsible) of path changes affecting its prefixes. First, we develop a new model describing path changes and use it to provably identify the set of all potentially responsible networks. Next, we develop a recursive algorithm that accurately isolates the root cause of any path change. We observe that the algorithm requires monitoring paths that are generally not visible using standard measurement tools. To address this limitation, we combine existing measurement tools in new ways to acquire path information required for isolating the root cause of a path change. We evaluate *PoiRoot* on path changes obtained through controlled Internet experiments, simulations, and 'in the wild' measurements. We demonstrate that *PoiRoot* is highly accurate, works well even with partial information, and generally narrows down the root cause to a single network or two neighboring ones. On controlled experiments *PoiRoot* is 100% accurate, as opposed to prior work which is accurate only 61.7% of the time.

Categories and Subject Descriptors

C.2.2 [**Communication Networks**]: Network Operations: Network Monitoring

Keywords

Path Changes, Root Cause Analysis, Measurement, Monitoring, BGP

1. INTRODUCTION

Internet paths change frequently, as links fail or become available and as traffic engineering and policies evolve. Although many of these changes are benign, some can seriously disrupt the performance or availability of distant networks and services. Disruptive changes can cause unwelcome changes in flow volumes over ingress links, longer round-trip times, less available bandwidth, and loss of connectivity [17, 41]. For example, Google recently found that interdomain routing changes caused more than 40% of the cases in which clients experienced a latency increase of at least 100ms [44].

These changes in performance and availability can be costly for service providers. Amazon found that every additional 100ms of delay in loading a web page costs them 1% of their sales [22]. Similarly, a Yahoo! study found that, when a page took an additional 400ms to load, the latency caused 5-9% of users to browse away, rather than letting the page load to completion [36]. Outages at large data centers cost an average of $5,000 per minute [33]. Previous work demonstrates that these providers can work with remote networks to resolve problems [19], but doing so requires knowing which network to contact. Taken together, these facts suggest that content and service providers would like to quickly understand what caused a path change that impacts their clients' performance.

There are other reasons to understand the root cause of a path change. Research has established that Internet routing policies can create unstable configurations [12, 13] and that some paths experience frequent updates [21], making it important to understand the underlying causes of these updates. Networks may want to avoid providers or paths based on a historical understanding of which ISPs cause many changes [2]. Routing policies have a significant impact on performance [35], and understanding the causes of path changes and selections provides some visibility into the policies in use [23]. An understanding of how path changes ripple through the Internet could prove helpful in developing new protocols for routing, for distributing routing updates [14], or for moving toward a more stable Internet.

The goal of this work is to accurately isolate the root cause of a path change within minutes of it happening. For our purposes, the root cause is the network or router whose link availability or policy adjustment initially triggered the sequence of route updates leading to the path change in question. Operators can use this information to debug and address performance problems as they occur, and it could eventually drive a system that automatically reroutes traffic to improve performance (similar to LIFEGUARD [18]).

Previous work provided initial inroads towards our goal [2, 9], but opaque policies and poor network visibility limit our ability to understand the root cause of path changes and how they propagate through the Internet. In particular, we identify the following open challenges that these limitations present.

First, interdomain routing policies can interact in complex ways, so the network responsible for a change may appear neither on the old path nor the new path [9]. Existing algorithms assume that the change is on the old or new path [2, 9], and they cannot properly account for these *induced path changes*. Second, because path changes can have cascading effects, proper root cause identification can require monitoring a potentially large number of paths, many of which are invisible to traditional measurement vantage points such as public BGP feeds and traceroute servers. Third, little to no ground truth information is available to validate assumptions about Internet routing made by root cause identification algorithms. Our results from controlled experiments on real Internet paths demonstrate that these challenges and others cause an existing approach [9] to correctly identify the root cause in only 62% of path changes.

This paper describes how we address these issues to build a system, which we call *PoiRoot* [4], that allows a provider to quickly and accurately isolate the root cause of path changes affecting their clients. We derive new constraints on how interdomain path changes propagate and use them to develop an algorithm to accurately isolate the root cause. We then deploy an extensive measurement system and demonstrate that our approach is both precise (*i.e.*, produces a small set of suspected root causes) and accurate (*i.e.*, the suspect set always includes the root cause of a path change) for real Internet paths. Our main contributions are as follows.

First, we identify a provable upper bound for the set of paths that must be monitored to identify the root cause of an interdomain path change. Because this set might be prohibitively large for an arbitrary change, we use a simple, yet effective, routing model to refine the set to improve scalability. Our bound reduces the average number of ASes we need to monitor to perform root cause analysis by up to 65%.

Second, we combine existing measurement tools in new ways to acquire path information required to isolate the root cause of a path change. In addition to public BGP feeds, we use BGP prepending on prefixes we control to reveal alternative, less preferred, paths toward our prefixes from distant ASes. We then use data-plane active measurements (traceroutes) toward our prefixes to identify the paths before and after a given path change. In our experiments, we are able to collect the necessary path information within minutes of a path change, allowing our approach to quickly perform root cause analysis.

Third, we develop an algorithm – based on our model and using our measurements – that allows us to isolate the root cause of a path change, even with partial routing information. We use controlled experiments on real Internet paths to demonstrate the effectiveness of our approach. In particular, we use BGP announcements to cause path changes for prefixes we control, then validate our algorithm using this ground-truth information. We show that *PoiRoot* accurately identifies the root cause of a path change *in every case in our experiments*, including induced path changes that previous approaches do not address. Finally, we show that path changes in the wild cause significant performance problems, and that *PoiRoot* is able to identify small suspect sets containing the network responsible for them.

The rest of the paper is organized as follows. In the following section, we provide background related to the problem of root cause analysis and motivate *PoiRoot* by describing limitations of previous work. In §3, we develop a general algorithm for identifying the root cause of an arbitrary path change, then use a simple model of BGP path changes to derive the set of paths that must be monitored to identify the root cause. In §4, we describe the design of a measurement system to gather this path information. §5 uses experiments on real Internet paths, simulations, and "in the wild" path changes to demonstrate the effectiveness of our approach. We discuss related work in §6 and conclude in §7.

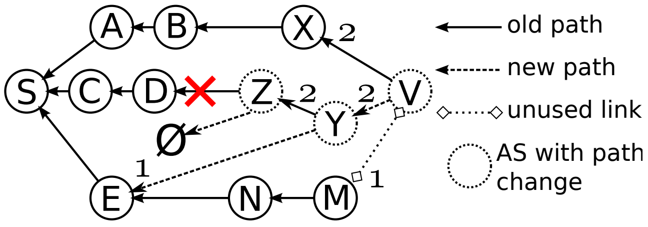

Figure 1: Example of an induced path change at v after the link d–z fails. Numbers on edges show the local preference (LocalPref) of a peer, higher is more preferred.

2. BACKGROUND AND CHALLENGES

In this section, we discuss closely related work and how it fails to address several open challenges for *root cause* analysis, *i.e.*, identifying the AS that originates a routing change in the Internet.[1] We discuss other related work in §6.

2.1 *NOOR* Approach

The first general approach for root cause analysis was proposed by Feldmann et al. [9]. The key idea behind this heuristic is that the root cause of a path change observed at an AS is likely either on the old path used before the change or the new path used after the change. We refer to this approach as *NOOR*, because it identifies changes on the New Or Old Route. (We use the terms 'route' and 'path' interchangeably in this paper.) The heuristic then combines simultaneous path change observations from different ASes and builds a suspect set containing the ASes that appear in all paths that changed and that never appear in paths that did not change. Although not evaluated explicitly, to reduce the set of suspect ASes further, *NOOR* assumes that the root cause is on the most preferred path (otherwise AS A would keep using the preferred path and there would be no change).

The authors use simulation to inform their design, then evaluate their approach using experiments on real-world data with BGP beacons announcements for ground truth. In this context, Feldmann et al. were able to isolate the origin of a BGP path change to a single AS or inter-AS link for 76% of the cases they study. Applying further heuristics, they achieve this precision in 96% of cases. While these results may suggest that we can close the book on root cause analysis, we list several challenges below that may impact *NOOR*'s accuracy and precision.

2.2 Open Challenges

Induced Path Changes. The root cause of a path change observed at an AS v may lie outside the old and new paths used by v. Feldmann et al. [9] recognize this challenge but did not have an approach to address it. Fig. 1 shows an example network topology where an induced change happens. The solid arrows show the paths ASes chose toward s before the link between ASes d and z failed. The dashed arrows show the path ASes chose to AS s after the failure. ASes with one outgoing arrow (solid circles) did not change paths. Note that, before the failure, AS y prefers the longer route through z (LocalPref 2, higher is more preferred) than the shorter

[1]We do *not* identify reasons for a routing change, *e.g.*, whether a change was caused by a link failure or traffic engineering.

route through e (LocalPref 1). When the failure happens, AS z cannot find a path to s and withdraws its path. AS y then changes to the shorter, less preferred route through e. When y announces the new route to v, AS v changes to the new route $[y, e, s]$ because it is shorter than the route through x. Note that the root cause, *i.e.*, the link between ASes d and z, does not lie in either the old or the new path used by AS v. *NOOR* cannot identify the root cause of induced path changes; at best, it would find the AS where the change was induced, *e.g.*, AS y. Moreover, because *NOOR* correlates changes across all observation points, it suffices that one observation point experiences an induced path change to make identification fail. We address such cascading path changes in §3.1.

Extensive Path Monitoring. BGP limits the amount of path-change information one can gather passively [28, 38]. For example, an AS that receives updated paths from different downstream neighbors will forward only its best path upstream. One consequence is that BGP and traceroute measurements from a small number of vantage points are insufficient to observe an arbitrary path change. We potentially need BGP measurements from each AS in the network [37], or techniques such as Reverse Traceroute [16] that provide path measurements from arbitrary ASes, to have complete information to perform root cause analysis. We determine the set of required measurements in §3.2 then use a simple routing model to bound this set in §3.3.

Inferring Route Preferences. When an AS switches from path P to P', it may be unclear which of the two paths it prefers. Feldmann et al. proposed a heuristic that considers the shorter path the preferred path. However, in practice inter-AS business relationships (*i.e.*, policy routing) take precedence over path length in the route selection process. Incorrect assumptions about routing policies can prevent root cause analysis from identifying the correct AS responsible for a path change. We describe our preference inference mechanism in §3.3.

3. GENERAL ROOT CAUSE ANALYSIS

In this section we frame the general problem of locating the root cause of an arbitrary interdomain path change. For the sake of exposition, we use an AS as the basic routing element. First we make a set of simple observations to reason about all possible path changes at a given AS and their implications for root cause analysis. These lead to a a recursive algorithm for isolating the root cause (§3.1). Then we provide a proof as to what is the upper bound on the set of ASes that our recursive algorithm traverses for root cause analysis (§3.2). We observe that the number of paths to monitor to obtain this subgraph can be large for arbitrary path changes. To address this, we use commonly accepted assumptions about routing decisions to further restrict the set of monitored paths (§3.3).

3.1 Root Cause Analysis Algorithm

We now consider locating the root cause of a change observed on the path from an AS v towards an AS s that originates a prefix. In the following discussion we isolate the root cause at the granularity of ASes, but our approach makes few assumptions about routing and can equally apply to other groupings such as quasi-routers [29] or PoPs as long as information at these finer granularities is available. For simplicity, we assume that at any time there is only one routing event causing a path change to a particular prefix, and that we have complete and accurate path information. We revisit these assumptions in §4. We also make the reasonable assumptions that (i) an AS v changes paths either due to modifications in its route preference or due to a change in one of its neighbor's paths or export policies; and that (ii) if v did not change its next hop to s nor its export policy, it is not responsible for the route change.

Algorithm 1 Recursive algorithm for general root cause analysis.

Precondition: A path change observed on the path from v to s. Let $O(v) = [v, x, \ldots, s]$ and $N(v) = [v, y, \ldots, s]$.

RootCause(v, s):

```
1 if $x = y$:                          // downstream change
2     return RootCause($x, s$)
3 if $O(x) \neq N(x)$:                 // neighbor path change
4     return RootCause($x, s$)
5 if $O(y) \neq N(y)$:                 // neighbor path change
6     return RootCause($y, s$)
7 return $\{v, x, y\}$                 // local change
```

Below we enumerate all possible observations of path changes at an AS v and what inference we can make from each observation for root cause analysis. Let $O(v)$ and $N(v)$ denote the old and the new paths AS v used before and after the event, respectively.[2]

Local change. If AS v changed its next hop toward s, *i.e.*, $O(v) = [x, \ldots, s]$ and $N(v) = [y, \ldots, s]$ such that $x \neq y$, but both next hops did not change paths, *i.e.*, $O(x) = N(x)$ and $O(y) = N(y)$, then the root cause is in $\{v, x, y\}$. Either v changed its route preference or either x or y changed their export policy. If there is no working path before or after a change, either x or y is null. Note also that if there was no change at v, *i.e.*, $O(v) = N(v) = [x, \ldots, s]$, then downstream ASes $\{x, \ldots, s\}$ are not the root cause of the event. AS v can still be the root cause if it changed its export policy (*i.e.*, started or stopped announcing the route upstream).

Neighbor path change. If AS v changed its next hop toward s, *i.e.*, $O(v) = [x, \ldots, s]$ and $N(v) = [y, \ldots, s]$, and one of the neighbors changed paths, *i.e.*, $O(x) \neq N(x)$ or $O(y) \neq N(y)$, then the change at v was induced by the change in its next hop neighbor and v is not the root cause.

Downstream change. If the old and new paths at an AS v share the first hop but downstream hops are different, *i.e.*, $O(v) = [x, \ldots, m, \ldots, s]$ and $N(v) = [x, \ldots, n, \ldots, s]$, then v is not the root cause of the event.

The observations above can be combined to build a generic root cause identification algorithm as shown in Alg. 1. The algorithm starts from an AS v where a change was observed. If v uses the same next hop AS toward s before and after the change, v is not the root cause and we continue our search downstream (*downstream change*). If v uses different next hop ASes and one of them changed paths, then we search downstream from the next hop that changed paths (*neighbor path change*). If both next hops of an AS v change paths, the changes are due to the same (unique) root cause and Alg. 1 finds the root cause regardless of the next hop chosen in the recursive call. If v uses different next hop ASes before and after the change and the next hops did not change paths, then we have found the root cause (*local change*).

We can simplify inference from a *local change* if we assume that ASes do not change their export policy, *i.e.*, if an AS is (is not) exporting a particular path to a particular neighbor, it will not stop (start) exporting to that neighbor unless it changes paths. Under this assumption, ASes x and y cannot be the root cause in a *local change*. Simply put, if $O(x) = N(x)$, x is not the root cause. As a result, line 7 of Alg. 1 would return only $\{v\}$ as the root cause.

As an example, Tab. 1 shows the steps taken by Alg. 1 when applied to the path change in Fig. 1. The algorithm starts at v,

[2]Even though the text focuses on the sequence of ASes in a path, comparison of paths includes other relevant data such as BGP communities and multi-exit discriminators when available.

Recursion Depth	Target AS	$O(\cdot)$	$N(\cdot)$	Observation
1	v	$[x, b, a, s]$	$[y, e, s]$	*neighbor path change*
2	y	$[z, d, c, s]$	$[e, s]$	*neighbor path change*
3	z	$[d, c, s]$	$\emptyset$	*local change*

Table 1: An example of Alg. 1 applied to the path change shown in Fig. 1.

observes a *neighbor path change* since the old and new first hops are different, and calls itself recursively to search the root cause downstream from y. At y, the algorithm observes another *neighbor path change* and searches the root cause downstream from z. At z, the algorithm observes a *local change* and identifies $\{z, d, \emptyset\}$ as the root cause as both neighbors (d and $\emptyset$) did not change paths ($\emptyset$ is z's neighbor since z has no new path due to the failed link $z - d$). Under the assumption that export policies are fixed, we would get $\{z\}$ as the root cause instead.

3.2 General Candidate and Monitored Sets

We now consider the set of ASes that Alg. 1 could traverse for an arbitrary path change observed at v, which yields the full set of path information that the algorithm could require to identify the root cause.

We define the *general candidate set* of an AS v, denoted $\mathcal{C}(v)$, as the set of ASes Alg. 1 may identify as the root cause of a path change observed at v. For a given path change for prefix p observed at AS v, $\mathcal{C}(v)$ is a set containing v itself and the ASes in the general candidate sets of its old and new downstream neighbors. Let $O_{\rm n}(v)$ and $N_{\rm n}(v)$ denote the downstream **n**eighbors AS v chose toward p before and after the change, respectively. Then $\mathcal{C}(v) = v \cup \mathcal{C}(O_{\rm n}(v)) \cup \mathcal{C}(N_{\rm n}(v))$.

As an example, in Fig. 1, $O_{\rm n}(v) = x$, $N_{\rm n}(v) = y$, and the general candidate set of v contains the ASes in v's old and new paths (v, x, b, a, y, e, s) as well as the ASes in y's old path (z, d, c). If z had a new path, its ASes would also be in $\mathcal{C}(v)$. ASes like m and n that are not involved in the change are not in v's general candidate set. Now we can prove the following result.

THEOREM 1. *If there is only one routing event in the network, the root cause of a path change at an AS v lies in* $\mathcal{C}(v)$.

Proof The claim is trivially true if the root cause is v. If there is another AS x in $O(v)$ or $N(v)$ that changed paths, the change at v was induced by the downstream change at x (*neighbor path change* or *downstream change*). Such an AS x is in $\mathcal{C}(v)$. Similarly, if there is another AS y in $O(x)$ or $N(x)$ that changed paths, the change at x was induced by the change at y. Such an AS y is in $\mathcal{C}(x) \subset \mathcal{C}(v)$. We can follow this process recursively until we find the AS $c \in \mathcal{C}(v)$ that is the AS closest to the source that changed paths (*i.e.*, there is no AS downstream c that changed paths). Without loss of generality, we need to prove that the root cause of the change at c is in $\mathcal{C}(v)$. An AS not in $O(c) \cup N(c)$ cannot be the cause of the change at c because it cannot change the relative preference between the (unchanged) paths c receives from its downstream neighbors $O_{\rm n}(c)$ and $N_{\rm n}(c)$. Thus, either c changed its route preference and is the root cause, or one of its downstream neighbors changed their export policy and are the root cause (*local change*). This completes the proof. ∎

Alg. 1 is guaranteed to identify the root cause of path changes. However, it requires path information from a potentially large set of ASes in the network. Identifying the root cause of a change at

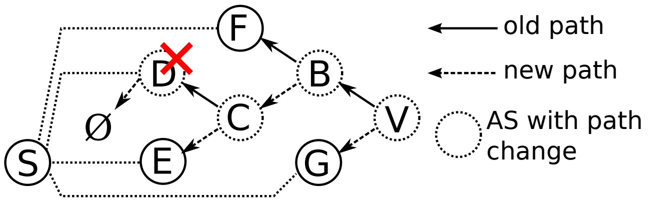

Figure 2: General topology subgraph demonstrating a three-level induced path change starting at c and propagating up to v (impossible under the simplified routing model). The ASes shown can be anywhere in the path.

an AS v requires information about the old path used by an AS x in the new path chosen by v, *i.e.*, we need to know $O(x)$, where $x \in N(v)$. Because we cannot know the ASes in $N(v)$ until after the path change, we need to track the old paths from all ASes in all possible paths v can change into. For example, in Fig. 1, we need to monitor paths from x, y, and m, as we do not know *a priori* what neighbor v will choose after a change. Given the recursive calls in Alg. 1, the set of ASes we need to track paths from may grow exponentially with the path length, subject to the upper bound of the size of the AS graph.

Formally, we define the *general monitored set* of an AS v, denoted $M(v)$, as the set of ASes whose old paths we need to track to run Alg. 1 on path changes observed at AS v. Let $L(v)$ denote the set of neighbor ASes v can use to reach an AS s. The set $M(v)$ includes the neighbors v can use to reach s and all ASes in the general monitored sets of each of v's neighbors. This gives $M(v) = L(v) \cup \{M(x) \mid x \in L(v)\}$. While the general candidate set, $\mathcal{C}(v)$, is specific to a given change, $M(v)$ is not and covers all possible changes. Note that $\mathcal{C}(v) \subset M(v)$, as $\mathcal{C}(v)$ includes the general candidate set of only the neighbors v chose before and after the path change. For example, in Fig. 1, $\mathcal{C}(v)$ includes $\mathcal{C}(x)$ and $\mathcal{C}(y)$, while $M(v)$ includes $M(x)$, $M(y)$, and $M(m)$.

3.3 Bounding Candidate and Monitored Sets

To address the challenge of monitoring a prohibitively large set of ASes, we provide a bound on the general candidate set $\mathcal{C}(v)$. We then use this bound to reduce the set of ASes we need to monitor to perform root cause analysis.

Our bound on the candidate set is derived from the simple and widely used Gao-Rexford routing model [10]; *i.e.*, we assume each AS has a total ordering over the preferences for all available paths toward a prefix. For example, an AS typically prefers a path learned from a customer over another learned from a peer or provider, and a path learned from a peer over another learned from a provider.

We also assume an AS considers its LocalPref (which can encode policies such as business relationships and/or traffic engineering) for each path before AS path length when selecting among paths announced by multiple neighbors toward the same prefix. This assumption is reasonable as it is used in the BGP route selection process. The shorter path is picked only if two or more paths have the same LocalPref (*e.g.*, both are customer paths).

We further assume that assigning LocalPref values only on neighboring ASes. In other words, path preference at an AS does not depend on which specific ASes are in the path, except for the next hop ASes only, a property that holds empirically the vast majority of the time [11].

Based on this routing model we identify the ASes from the general candidate set most likely to have caused a path change. We overload our notation for the old and new paths used by an AS v, $O(v)$ and $N(v)$, as follows. $N(O(v))$ is the set of ASes that appear on the new paths originating from the ASes in $O(v)$, and

$O(N(v))$ is the set of ASes that appear on the old paths originating from ASes in $N(v)$. For example, in Fig. 2, $O(v) = [b, f, \ldots, s]$ and $N(O(v)) = O(v) \cup N(b) \cup N(f) \cup \cdots \cup N(s) = \{b, f, c, e, \ldots, s\}$.

THEOREM 2. *Under the simplified routing model, the root cause of a path change observed at an AS a is contained in the* bounded candidate set $\mathcal{B}(v) = N(O(v)) \cup O(N(v))$.

Proof We first show that the root cause AS *may* appear in $\mathcal{B}(v)$. It is obvious that the root cause may lie on either the old or the new path, *i.e.*, ASes like g and b. ASes in the set $N(O(v))$, like c, can be the root cause as well. Using Fig. 2 as an example, suppose a failed link c–b comes back up and c starts announcing a path to b. AS b may start routing through c if it prefers paths from c over paths from f. AS v may then change its route from b to g if the new path through $[b, c, \ldots, s]$ is longer. The case of $O(N(v))$ is similar.

Now we show that ASes in the set $\mathcal{C}(v) - \mathcal{B}(v)$ cannot be the root cause. These include ASes like d, which lies in $O(N(O(v)))$. Proceeding by contradiction, assume that the path change shown in Fig. 2 happens and that the root cause is d, causing c to switch to a different path through e and causing b to switch to c's new path. This implies that b's criterion for path selection between f and c is path length. If b had a policy favorable to c (*e.g.*, a higher LocalPref for c) then it would have picked the path through c in the first place. If b had a policy favorable to f it would not change paths. Shortest path routing at b implies that $[b, c, e, \ldots, s] \leq [b, f, \ldots, s] \leq [b, c, d, \ldots, s]$, where the '$\leq$' operator compares path lengths. This is because b prefers the path through f before the routing event and the path through c–e after. Now, if v switches to the new path through g, then it implies that v is using shortest path routing to choose between b and g. As v prefers the path through b–f over the path through g before the routing event and the path through g over the path through b–c after, we have that $[v, b, f, \ldots, s] \leq [v, g, \ldots, s] \leq [v, b, c, e, \ldots, s]$. Looking at the paths from v we have that $[b, f, \ldots, s] \leq [b, c, e, \ldots, s]$. This contradicts the earlier result from b that $[b, c, e, \ldots, s] \leq [b, f, \ldots, s]$. Hence, d or any other AS in $O(N(O(v)))$ cannot be the root cause. The argument is similar for $N(O(N(v)))$. ∎

Observe that $\mathcal{B}(v) = N(O(v)) \cup O(N(v))$ is contained in v's general candidate set $\mathcal{C}(v)$. Hence, if the root cause lies in $\mathcal{B}(v)$, it also lies in $\mathcal{C}(v)$, as per Theorem 1. Note also that if our assumptions and routing model are wrong, the root cause of a path change at v can lie outside $\mathcal{B}(v)$. However, we *can* still detect violations of the assumptions checking that the root cause is outside $\mathcal{B}(v)$.

Even if we limit our search for the root cause in the bounded candidate set $\mathcal{B}(v) = N(O(v)) \cup O(N(v))$, the $O(N(v))$ term implies we need to track the old paths from all ASes that may appear in any new path v can choose. This makes the set of ASes whose old paths we need to track the same as the general monitored set $M(v)$.

To bound the monitored set, we only track the old paths from ASes in the most preferred paths v can change to. AS v may change away from $O(v)$ if a more preferred path becomes available. To cover this case, we monitor old paths from all ASes in paths that are more preferred than $O(v)$. Conversely, AS v may change away from $O(v)$ to a less preferred path if $O(v)$ becomes unavailable. Monitoring old paths from all ASes in paths that are less preferred than $O(v)$ can be costly and wasteful because v should use preferred paths most of the time and the less preferred a path, the less likely it will be used. Instead, we monitor old paths from all ASes in the next less preferred paths from each AS x in $O(v)$. This approach only misses information when an AS $x \in O(v)$ changes to a path that is not its next less preferred path (which may happen, *e.g.*, when the network lacks redundancy and an event impacts both of an AS's current and next preferred path).

We define the *bounded monitored set*, denoted $T(v)$, as the set of ASes in all paths that v prefers over $O(v)$ plus the ASes from the next less preferred path from each AS in $O(v)$. If path preference information is unavailable for an AS v, we take a conservative approach and add the most preferred path from each of v's downstream neighbors to $T(v)$. We have that $T(v) \subset M(v)$. We also note that while $\mathcal{B}(v)$ is specific to a given change, $T(v)$ is not and covers most possible changes.

Tracking paths from ASes in the most preferred paths reduces the monitored set when identifying the root cause in the bounded candidate set $\mathcal{B}(v)$ because we need to know only $O(N(v))$. The heuristic would not be as effective at reducing the monitored set if we were identifying the root cause using the general candidate set $\mathcal{C}(v)$. The reason is that we would need to monitor ASes in the most preferred paths for any AS that can show up in new paths, recursively. For example, we would need to track ASes that can show up in the most preferred paths of all ASes in $N(O(v))$ to know $O(N(O(v)))$.

4. DESIGN AND IMPLEMENTATION

The previous sections laid down theoretical foundations for root cause analysis when complete and accurate routing information is available. We now describe a practical system to identify the root cause of a path change in the Internet with possibly incomplete and inaccurate data. We now state the goals of the system and how we achieve them.

4.1 Goals

Our work is motivated by the fact that Internet path changes can have a significant impact on performance and availability, affecting service providers, operators, and customers. When such changes occur, we would like to to be able to quickly and accurately identify the network responsible for the change. This allows operators to more quickly debug problems and take corrective action (*e.g.*, by contacting the responsible network). To enable such a system, we identify the following system goals.

- *Accuracy:* The output candidate should include the AS that caused the change.
- *Precision:* The system should identify a small set of ASes as candidates for being the root cause of a path change, ideally a single AS or single AS link.
- *Robustness:* The system should be able to deal reasonably well with uncertainty or absence of information.
- *Scalability:* The system needs to be selective in its choice of ASes to monitor and should introduce as little measurement overhead as possible. It should conduct measurements and quickly identify root causes of path changes (*e.g.*, within minutes), enabling operators to quickly react when an unexpected change happens.

4.2 System Overview

PoiRoot employs a number of measurement components (§4.3 and §4.4) to identify the root cause of path changes observed on paths from a set of target ASes $\mathcal{V}$ to a set of monitored prefixes $\mathcal{P}$. As shown in Fig. 3, *PoiRoot* operates in two modes for each monitored prefix. In the absence of path changes, the system stays in *monitoring* mode. The first task of monitoring mode is to estimate

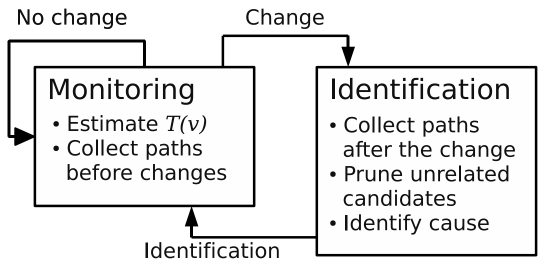
No change
Change
Monitoring
• Estimate T(v)
• Collect paths before changes
Identification
• Collect paths after the change
• Prune unrelated candidates
• Identify cause
Identification

Figure 3: *PoiRoot* operation modes.

the set of ASes we need to monitor to perform root cause analysis for all target ASes, *i.e.*, $\bigcup T(v)$ for all $v \in \mathcal{V}$. The second task is to measure paths from the monitored ASes toward the monitored prefixes in $\mathcal{P}$.

PoiRoot enters *identification* mode whenever the path from a target AS v to a monitored prefix changes. Identification uses information about the old paths used before the change, collected while in monitoring mode. The first step is to collect information about new paths used by ASes in v's old path, *i.e.*, measure $N(O(v))$. *PoiRoot* then runs the identification algorithm to identify the root cause of the change. Finally, *PoiRoot* returns to monitoring mode after identification is complete.

4.3 Path Measurement

PoiRoot maintains an atlas of paths from each AS $v \in \mathcal{V}$ to each monitored prefix. To this end, we combine publicly-available BGP feeds and active traceroute measurements. To satisfy the assumption that there is only one routing event in the network, we need path measurements from each AS at high enough frequency such that the path is available before and after each routing event. We also need to collect all necessary information and identify the root cause before the next routing event happens. We describe both control- and data-plane monitoring measurements we use below.

Passive BGP Monitoring. *PoiRoot* uses BGP paths made public by over 100 RouteViews peers [28]. We download data from RouteViews every 15 minutes. For each monitored prefix, we store historical and current BGP path information. We group updates into convergence periods if they are less than 4 minutes apart to identify stable routes. Previous work has shown that varying the grouping threshold between 2 and 8 minutes has negligible impact on the resulting convergence periods and stable paths [18, 41].

Traceroute Monitoring. We supplement our view of paths toward a monitored prefix using traceroute measurements that we translate to AS paths using IP-to-AS translation. Specifically, every 10 minutes, we issue forward traceroutes from 175 geographically distributed PlanetLab sites toward the monitored prefixes. Forward traceroutes allow us to find paths from ASes not present in route collector feeds, but are still limited to the locations where PlanetLab nodes are located.

Reverse Traceroute [16] allows us to address this by potentially measuring paths to monitored prefixes from routers in arbitrary ASes. In particular, we use Reverse Traceroute to measure paths from three IPs in each AS on the BGP feeds and forward traceroutes toward the monitored prefixes. We refresh reverse traceroutes every 15 minutes. For a variety of reasons (*e.g.*, routers not responding to IP Options probes), Reverse Traceroute may not be able to collect measurements from all ASes.

Combining Data Sources. In some cases, we obtain a path between two ASes using both traceroute measurements and BGP feeds. Because IP-to-AS mappings can be inaccurate [3, 27], *PoiRoot* uses the control-plane paths from BGP feeds instead of data-plane paths in this case.

4.4 Identifying the Set of ASes to Monitor

PoiRoot identifies the set of ASes whose paths we need to monitor to run root cause analysis, $\bigcup T(v)$, while in monitoring mode. We consider the following data sources.

Passive Monitoring of Paths to Prefixes. One way to identify the ASes in the monitored set $T(v)$ is to continuously monitor public BGP feeds [28]. Monitoring changes for a prolonged period of time allows us to discover a set of paths an AS may use to reach a prefix and build the set of ASes we need to monitor to perform root cause analysis. Although a passive approach reveals actual paths used in the Internet and is minimally invasive, not all paths appear in BGP feeds because ASes only propagate their best path, apply export filters, and perform prefix aggregation. Waiting for path changes to take place naturally is likely to be slow (on the order of months or years [30]) for the purposes of identifying the ASes we need to monitor.

Active Monitoring of Paths to Prefixes. We complement passive BGP data with active periodic traceroute measurements. We store all paths observed with passive and active monitoring and build a database of historical paths toward the monitored prefixes.

Revealing Alternative Paths via BGP Announcements. *PoiRoot* reveals alternative paths in the Internet using the Transit Portal platform [40], which lets us announce prefixes using five different US universities as our providers (University of Washington, University of Wisconsin, Georgia Tech, Princeton University, and Clemson University). Suppose AS v has AS x in its path toward a prefix we control in AS s. To reveal an alternative path from v to s that does not traverse x, we can prepend our BGP announcements with $[s, x, s]$. This approach is generally referred to as "BGP poisoning" [1, 5, 18].[3]

When a poisoned update for our prefix arrives at x, BGP's loop-prevention mechanism causes x to discard the route, leaving it routeless. AS x then withdraws its path to our prefix from its upstream neighbors (including v), forcing them to choose new paths. For example, in Fig. 2 poisoning c emulates a link failure between c and b. This causes c to withdraw the route to b, causing b to route through d.

To discover all possible paths to our prefixes from an AS x, we first discover all neighbors $L(x)$ that x can use to reach our prefix and then call the discovery process recursively on each AS in $L(x)$. To discover the neighbors $L(x)$ that an AS x can use, we first poison the neighbor $O_{\mathrm{n}}(x)$ that x is using to reach our prefix. When x changes to a new path, we poison its new neighbor. We repeat this process until x has no new path to our prefix. Note that we include multiple ASes in our announcements as the discovery process proceeds. In particular, to call the discovery process recursively on an AS $y \in L(x)$, we need to poison all other ASes in $L(x)$ that x prefers over y. For example, suppose our measurements go through a provider x that has y and z as the two most preferred neighbors to reach our prefixes. We will keep y poisoned throughout the discovery of the paths z can use toward our prefixes, otherwise x would change back to y and we will be unable to observe paths through z.

We argue that this approach to learning alternative available paths

[3]Similar to the study we conducted for LIFEGUARD [18], these announcements affect only paths toward our experimental prefixes and we impose a conservative rate limit for announcements to ensure our experiments do not generate unduly large numbers of updates. We gave operators an opportunity to opt out. We have received zero complaints during several months of experiments.

is reasonable for the context of a large cloud provider or ISP. Such networks already are BGP speakers, and we expect they can dedicate a small prefix (*e.g.*, a /24) for the purposes of exploring alternate paths and inferring route preferences of different ISPs routing towards them.

Path Preference Estimation. We use the order in which x chooses neighbors in the path discovery procedure above to rank paths by preference. When the discovery procedure provides insufficient information to estimate preference, we use path usage time as in indicative of preference as proposed by Oliveira *et al.* [31].

4.5 Root Cause Analysis Algorithm

The root cause analysis algorithm in Alg. 1 requires complete and accurate information. *PoiRoot* uses a modified version, shown in Alg. 2, that is more robust to missing and inconsistent measurements. Because it is impossible to determine whether an AS changed its export policy without BGP feeds from its neighbors, we assume that ASes do not change their export policy (*i.e.*, unless an AS changes its path, it will not change whether or not it is exporting the path to a neighbor). If an AS changes its export policy in practice, our system will blame the root cause's upstream. We believe this case is uncommon. Further, this information is useful in that the upstream AS is the only one that could detect the change in export policy, information necessary to debug the path change.

We now describe how our practical root cause analysis algorithm works. Suppose a path change happens on the path from an AS v to another AS s that originates a prefix. The algorithm adds the ASes in $O(v)$ and $N(v)$ to the suspect set $\mathcal{S}$. At each step, the algorithm visits the unvisited AS x in the suspect set that is closest to v (ties can be broken arbitrarily). If a measurement from x is missing (lines 3–4), the algorithm uses several heuristics to resolve uncertainty, described in §4.6. If the visited AS x did not change paths, then x cannot be the root cause and we remove x and all downstream ASes from the suspect set (*local change*, lines 5–7). If x changed paths, we know that either it is the root cause or the root cause is downstream of x (*neighbor path change* or *downstream change*). We remove all ASes upstream of x from the suspect set (line 9). If x is in $O(v)$ or $N(v)$, then we add any new ASes downstream of x to the suspect set and visit them to determine whether they are the root cause of the path change. Note that this algorithm has no recursive call and that lines 11 and 13 only execute if $x \in O(v)$ or $x \in N(v)$. Only ASes in $N(O(v)) \cup O(N(v))$ may be added to the suspect set. The algorithm finishes when all ASes in the suspect set have been visited.

Our algorithm will misidentify the root cause if our simplified routing model is wrong; *i.e.*, the root cause AS is not in $N(O(v)) \cup O(N(v))$. Say Alg. 2 identified x as the root cause. We can check for violations of the model (and misidentifications) by extending Alg. 2 to check if another downstream AS in $O(x) \cup N(x)$ changed paths. Our algorithm further assumes there is a unique path change when performing root cause analysis. For concurrent changes where the root causes are in one AS (or at an AS-AS boundary), *PoiRoot* will correctly identify the AS responsible. *PoiRoot* is equally effective when concurrent changes affect disjoint sets of paths. The case of distant, unrelated concurrent changes may lead to misidentifications (*e.g.*, identifying multiple, incorrect root causes).

4.6 Dealing with Uncertainty

Alg. 2 can find the root cause of a path change using a small subset of all monitored ASes in $T(v)$. However, some path measurements might be unavailable. In particular, measurements for the new path of an AS $x \in O(v)$ may be unavailable because no

Algorithm 2 Practical algorithm for root cause analysis of a change on the path from v to s with potentially incomplete information.

```
PracticalRootCause(v, s):
1  $\mathcal{S} \leftarrow O(v) \cup N(v)$
2  for each unvisited $x \in \mathcal{S}$ in order of distance from $v$:
3     if $O(x)$ not found or $N(x)$ not found:
4         // resolve uncertainty (§4.6)
5     if $O(x) = N(x)$:
6         remove $x$ from $\mathcal{S}$
7         remove ASes downstream of $x$ from $\mathcal{S}$
8     else:
9         remove ASes upstream of $x$ from $\mathcal{S}$
10        if $x \in O(v)$:
11            add ASes in $N(x) - O(x)$ to $\mathcal{S}$
12        else if $x \in N(v)$:
13            add ASes in $O(x) - N(x)$ to $\mathcal{S}$
14 return $\mathcal{S}$
```

vantage point has a path that traverses x after the change. Similarly, measurements for the old path of an AS $y \in N(v)$ may be missing. We employ the following heuristics to deal with missing measurements in Alg. 2.

FailedPath. If AS x experiences an outage, *i.e.*, $O(x) \neq N(x)$ and $N(x) = \emptyset$, then the root cause is downstream of x and all ASes upstream of x cannot be the root cause (*local change* or *neighbor path change*) and we remove ASes upstream of x from the suspect set. Similarly, **FailedPath** removes x's upstream ASes in the case of a link restoration, *i.e.*, $N(x) \neq O(x) = \emptyset$. We keep track of reachability from ASes using pings to historically reachable routers inside ASes.

MissingPath. If a path from AS x is unavailable and x is reachable, we use information from our atlas of historical paths (§4.4) to predict the missing path. When either $O(x)$ or $N(x)$ is missing, **MissingPath** adds all ASes in x's bounded monitored set, $T(x)$, to the suspect set, as these are the ASes that x is most likely to be using after the change.

Correlation. When Alg. 2 runs and encounters missing measurements, the resulting candidate sets may contain more than one AS. Because routing is generally destination-based, the same root cause can appear in multiple paths from vantage points toward the monitored prefix. We use this observation to reduce the suspect set by correlating suspect sets across path changes from the same event observed at different vantage points.

Note that the candidate sets from different vantage points may be disjoint for two reasons. First, our information about routing preferences for each AS in the candidate set may be incorrect (*e.g.*, because it is stale) leading the system to monitor the wrong set $T(v)$ of ASes. Second, there may be multiple concurrent and independent path changes. Thus, taking the intersection of suspect sets, as done by Feldmann et al. [9], would filter out a potential root cause.

To avoid this, we compute the suspect set for each path change resulting from a network event, then count the number of occurrences of each suspect AS over all suspect sets. Then, each suspect set is pruned to only those ASes with the highest number of occurrences.

5. EVALUATION

In this section, we evaluate *PoiRoot*'s algorithm on a large set of path changes. Our evaluation goals are to demonstrate that our approach is 1) *accurate* in terms of always including the root cause in the suspect set; 2) *precise* in that it identifies a small suspect set (ideally of size one); 3) *robust* to missing measurement from vantage points; and 4) *scalable* in that it does not require monitoring a prohibitive number of paths and it can identify root causes within minutes of paths changing. To evaluate these goals, we use controlled path changes on real Internet paths, simulations for larger topologies, and path changes "in the wild." In contrast to previous work, our approach is both precise and accurate, largely because it accounts for induced path changes. Further, we show that our results are similar even with only partial information from paths in the monitoring set for a prefix.

5.1 Controlled Internet Experiments

In this section, we use controlled experiments on real Internet paths to demonstrate the accuracy and precision of *PoiRoot* for identifying the root cause of a path change. We begin by describing our experiment methodology, then present a case study of how our algorithm works on a real induced path change. Last, we use a large set of AS-link failures to evaluate our approach and compare it with *NOOR* [9].

5.1.1 Methodology

A key challenge for evaluating root cause analysis algorithms is the lack of control or ground truth for interdomain path changes. Our work addresses this using the Transit Portal (TP) platform [40] to craft announcements that cause controllable path changes for real Internet paths. Specifically, we use BGP poisoning [1] to induce changes for paths toward a prefix we control.[4] Because such announcements cause the targeted AS to withdraw routes from its upstream neighbors, we know that the "root cause" for the corresponding path change is the targeted AS. Likewise, when unpoisoning a prefix, we effectively emulate a "link up" event for the targeted AS.

We use five /24 *poisoning* prefixes to cause path changes. Each prefix is announced from one of the five TP sites: University of Washington, University of Wisconsin, Georgia Tech, Princeton University, and Clemson University. We perform a sequence of poisonings using the same topology discovery scheme described in §4.4, starting the discovery process from the ASes where we have a PlanetLab vantage point. We change announcements (add or remove poisonings) at most once every 90 minutes to allow for BGP convergence and to avoid flap dampening effects. In addition to changes caused by our poisoned announcements, paths may change in response to exogenous events. To account for this, we announce a separate *sentinel* /24 prefix from each TP site and do not vary its AS-Path. If the sentinel and poisoning prefixes change at the same time, then it is likely caused by an exogenous event and we filter the change from our dataset.

We let the experiment run for one week starting November 25th, 2012, yielding a total of 105 distinct poisonings for each prefix. The BGP updates and traceroutes collected throughout the experiment traverse 351 ASes and include 2572 path changes. Many of these path changes are duplicates, and are therefore ignored. Similarly, we also ignore experiments where we detect that one of the poisoned AS's peers filtered the poisoned announcement. After applying the filters, our experiments yield 638 unique path changes.

5.1.2 Induced Path Change Case Study

We begin by describing how *PoiRoot* locates the root cause AS for an induced path change via a case study. Note that since the root cause AS lies neither on the old nor the new path, *NOOR* will not be able to find it. The path change takes place between

[4]We use the same conservative approach to poisoning as described in §4.4.

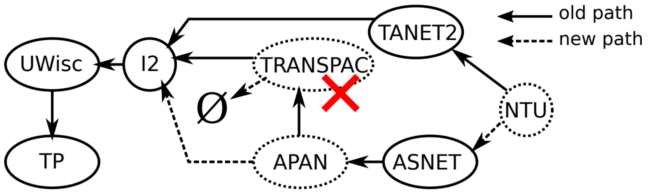

Figure 4: Case study for an induced path change. The path change is caused by a poisoned announcement that emulates a link failure at TRANSPAC (AS: 22388).

Step	Visit	Suspect Set
1	TANET2	NTU, TANET2, I2, ASNET, APAN
2	ASNET	NTU, ASNET, APAN
3	APAN	APAN, TRANSPAC, ASNET
4	TRANSPAC	TRANSPAC, APAN
5	∅	TRANSPAC

Table 2: Illustration of the suspect set when recursing through Alg. 2 for the induced path change shown in Fig. 4.

the National Taiwan University (NTU, ASN: 17716) and our experimental AS (ASN: 47065). We used TP to announce prefix 184.164.249.0/24 from the University of Wisconsin (ASN: 2381) and poisoned TRANSPAC (ASN: 22388) at 2:22 AM PST on November 27th, 2012. After convergence a path change was observed at a PlanetLab node located in NTU at 2:35 AM. The AS-level path change is shown in Fig. 4.

Upon observing the path change at NTU, *PoiRoot* runs Alg. 2 to identify the root cause. It begins by adding ASes in NTU's old and new paths to the suspect set. Then, it searches for the root cause in order of distance from NTU. It first checks that TANET2 uses the same path before and after the change, so TANET2 and all its downstream ASes cannot be the root cause of the change (*local change*, lines 5–7). We illustrate the algorithm in Tab. 2, where each row shows the AS being currently visited and the suspect set from the previous step. For lack of space, we do not show the TP AS and UWisc, which are trivially eliminated as root causes. When *PoiRoot* visits ASNET, it (i) removes NTU from the suspect set because ASNET changed paths (line 9), and (ii) adds TRANSPAC from ASNET's old path to the suspect set (line 13). Similarly when it visits APAN, it removes ASNET from the suspect set since APAN experienced a path change. Finally, the algorithm visits TRANSPAC and removes APAN from the suspect set (because TRANSPAC changed paths) and terminates returning a suspect set containing only TRANSPAC, the correct root cause. To validate our approach for obtaining ground truth via poisoning, we confirmed with TRANSPAC's NOC that it withdrew its announcement for our prefix.

5.1.3 Controlled Internet Experiments

In this section, we use controlled experiments to show that *PoiRoot* is accurate (always includes the root cause), precise (identifies suspect sets of size 2 or smaller 96% of the time) and robust to missing path information (more than 50% of paths must be missing to significantly impact suspect set size). By comparison, previous work cannot be both accurate and precise in the face of induced path changes, which occur frequently. Further, we show that our approach is scalable in that it monitors only a reasonable number of paths and provides results within minutes of path changes.

Accuracy and Precision. We use the path changes collected in the

	accuracy of suspect set			suspect set size when accurate			
	ground truth in set	empty	inaccurate	mean	median	25th perc.	75th perc.
PoiRoot	100.0%	0.0%	0.0%	1.66	1	1	2
***PoiRoot*-correlation across VPs**	100.0%	0.0%	0.0%	1.32	1	1	1
***NOOR*-standard**	61.7%	38.3%	0.0%	1.20	1	1	1
***NOOR*-individualPath**	91.1%	0.0%	8.9%	2.70	2	1	3

Table 3: Comparison of *PoiRoot* with *NOOR*, the approach used in previous work [9]. *PoiRoot* is both accurate and precise: it never misses the root cause and nearly always produces a suspect set size of two or smaller. *NOOR* trades off precision for accuracy in the face of induced path changes: *NOOR*-standard misses the root cause in many cases but has small suspect set sizes; *NOOR*-individualPath misses fewer root causes but exhibits larger suspect set sizes.

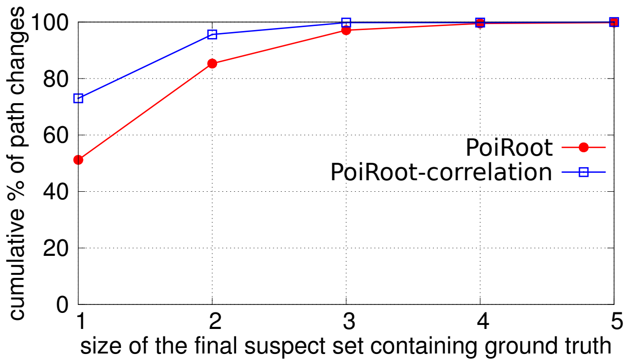

Figure 5: CDFs of the size of the final suspect set. *PoiRoot*, combined with an optimization that correlates the results across sources, results in a small suspect set on average as well as high accuracy.

previous section to evaluate *PoiRoot*. Tab. 3 presents our results and compares them with *NOOR*, which we discuss in the next section.

The table shows the fraction of paths where the suspect set (i) contains the ground truth (*i.e.*, the poisoned or unpoisoned AS), (ii) is empty, and (iii) is nonempty but does not contain the ground truth. We also show the mean and the quartiles of the suspect set size for correctly identified changes.

PoiRoot always includes the root cause in the suspect set, the average suspect set size is 1.66 ASes, and the median size is 1 AS. When *PoiRoot* identifies a suspect set of size 2, the ASes are adjacent, meaning the algorithm still isolates the root cause to the endpoints of an AS link. The median of the maximum candidate set size observed during executions of Alg. 2 is 5, and the quartiles are 4 and 6, respectively.

PoiRoot may return a suspect set with more than one AS when we have incomplete path information from some ASes. To reduce the impact of this issue we use the **Correlation** heuristic described in §4.6. Fig. 5 compares **Correlation**'s performance with the standard version of *PoiRoot*. For the standard algorithm, the suspect set contains one AS for 51% and two ASes for 84% of the path changes. Correlation across vantage points improves suspect set size: it is a single AS for 73%, and two ASes for more than 95% of the path changes. Again, in all cases of suspect sets larger than one, the ASes lie adjacent to each other on a path. The remainder of the evaluation uses **Correlation**.

Comparison with *NOOR*. We now compare our results with those from using *NOOR*, which is based on the approach proposed by Feldmann et al. [9].

The standard *NOOR* algorithm (see §2) builds a suspect set that contains the root cause for only 61.7% of the path changes, but with an average suspect set size smaller than that of our algorithm. This suggests an inclusion policy that is too conservative and an elimination policy that is too aggressive. We now detail the reasons behind these issues.

A key reason for inaccuracy is that the standard *NOOR* algorithm may output an empty set when an induced path change is observed. For example, when AS x causes an induced path change at another AS y, x may be identified as the root cause when considering paths from x, but it does not appear on the new or old paths used by AS y. Because the suspect sets from x and y may not overlap, the intersection may produce an empty set. The impact of this limitation is significant: in our experiments, 38.3% of the path changes were due to an event that generated an induced change.

The existence of even a small number of induced path changes can significantly decrease *NOOR*'s accuracy. In fact, a larger set of available vantage points leads to a higher probability that at least one induced change will be observed for a network event. Out of the 292 RouteViews peers and PlanetLab nodes used in our system, 15.8% observed at least one induced path change over the course of the experiment.

One way to address the induced path change problem is to modify the standard *NOOR* algorithm so that every path change is treated on an individual basis instead of grouping all changes for an event (*NOOR*-individualPath). This approach avoids taking the intersection of the individual candidate sets. It still fails to deal with an induced change due to the assumption that the root cause lies on either the old or new path. However, it improves results for non-induced changes, increasing accuracy to 91.1%. This comes at the cost of precision: the elimination policy is less aggressive without correlation, so the median and mean suspect set size for correct identification is larger (2 and 2.7, respectively). This highlights a key limitation of *NOOR*: the algorithm is either precise but not accurate; or in this case, accurate but not precise.

Robustness to Missing Information. We now address the question of whether *PoiRoot* is robust to missing path information. Note that *PoiRoot* always includes the root cause in our suspect set unless a new path from a monitored AS a uses an AS b that never appeared in previous measurements. This case never occurred in our experiments. Thus we quantify the robustness of our approach by evaluating the precision of *PoiRoot* when previously available paths are removed from the dataset.

In particular, consider AS x in our suspect set first discovered on the old path originating from AS v, hence $O(x)$ is known. Now our system measures $N(x)$. To determine how *PoiRoot* would perform with missing information, we set $N(x) = \textit{missing}$ with probability p. Similarly if x were originally discovered on the new path from v, *i.e.*, $N(x)$ is known, we set $O(x) = \textit{missing}$ with probability p. Increasing p causes *PoiRoot* to use the **MissingPath**

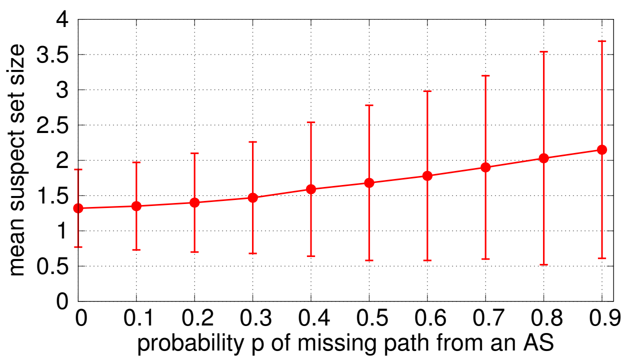

Figure 6: Mean suspect set size (and standard deviation) as we vary the probability of a missing path from an AS. *PoiRoot* is reasonably robust to missing path information: on average the suspect set size is only 2 when 70% of paths are missing.

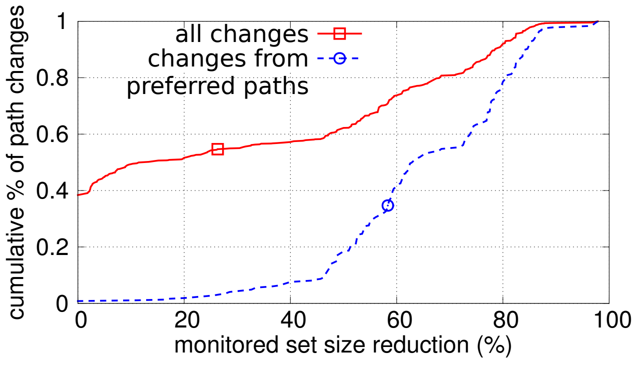

Figure 7: Our bound on the candidate set size allows us to reduce the number of ASes we have to monitor to perform root cause analysis.

heuristic (from §4.6) more frequently; this in turn can increase the suspect set size.

In Fig. 6 we plot the average suspect set sizes (and standard deviations) as we vary p. For each value of $p, 0 \leq p < 1$ we run the algorithm ten times on each path change and take the average suspect size. The final average for p is taken over all these averages. We note that the mean suspect set size increases slowly as p is increased. In fact the mean suspect size is less than 2 for a missing probability as large as 0.7.

PoiRoot is robust to missing measurements because paths from multiple vantage points tend to overlap as they converge toward the destination prefix. A single path measurement can remove multiple ASes from the suspect set, even ASes added to the suspect set due to missing measurements. For example, if *PoiRoot* observes a *local change* or a *neighbor path change* at an AS x (§3.1), *i.e.*, $O(x) \neq N(x)$, all ASes upstream from x are removed since they cannot be the root cause.

Prevalence of Induced Path Changes. One of the key strengths of *PoiRoot* is its success in dealing with induced path changes. We showed earlier that even a small number of induced changes can significantly decrease the effectiveness of previous approaches, such as *NOOR*. Now using a combination of our measurement data and the UCLA Internet graph [39], we argue that a larger set of vantage points would lead to significantly more observations of induced changes than in our controlled setting.

To simulate the scenario where we have more vantage points, and thus more paths, available to *PoiRoot*, we extend the empirically derived AS topology used in the previous section. Specifically, for each AS x in our measured paths, we find its set of upstream neighbors from the UCLA Internet graph. We use the corresponding annotated business relationships to determine whether these neighbors are upstream. Then, for each upstream neighbor u, we calculate the shortest valley-free path from the neighbor toward a TP prefix. AS u will use this path only if it is shorter than the path exported by x. We use path changes from our poisonings to determine which ASes x choose a longer path over a short one. These ASes may cause an induced change at an upstream AS u if a failure occurs on x's old path and x selects a new shorter path, *i.e.*, $N(x) < O(x)$. This is identical to the example in Fig. 1, where y chooses a longer path and induces changes at v.

We find 302 cases where AS x prefers a longer BGP path. This leads to a total of 7555 induced path changes caused by just 55 simulated link failures. [[[**drc: What are we supposed to compare this to?**]]] Hence with a larger set of paths being monitored, the number of induced path changes increases. Therefore an approach that does not account for induced changes is more likely to be inaccurate as the number of vantage points increases.

We also run *PoiRoot* on this set of simulated induced path changes. *PoiRoot* continues to always include the root cause in its suspect set; the suspect set contains only the failed AS. We also model missing paths with the parameter p similar to Fig. 6. The results are similar to Fig. 6. For example, $p = 0.7$ results in a mean suspect size of 1.65.

Candidate and Monitored Set Bounds. We now quantify the scalability gains obtained with our bound on the candidate set. We compute the sizes of the general and bounded candidate sets $\mathcal{C}(v)$ and $\mathcal{B}(v)$ for all path changes in our poisoning experiments and also the sizes of the general and bounded monitored sets $M(v)$ and $T(v)$. We use exhaustive poisoning experiments—we need topology information as complete as possible to compute the general candidate and monitored sets $\mathcal{C}(v)$ and $M(v)$

The further an AS is from the origin AS (*i.e.*, our TP providers), the longer its paths and the larger its candidate and monitored sets. We compute candidate and monitored sets for the ASes where we have vantage points because they are the furthest from the origin AS in our dataset. We also note that the bounded candidate and monitored sets $\mathcal{B}(v)$ and $T(v)$ depend on v's old path $O(v)$; in particular, the size of $T(v)$ depends heavily on the preference rank of its old path. The general candidate and monitored sets $\mathcal{C}(v)$ and $M(v)$ are independent of $O(v)$.

The solid line in Fig. 7 shows the distribution of the reduction in monitored set size (*i.e.*, $|M(v) - T(v)| \div |M(v)|$) across all path changes in the data set. Overall, we reduce the monitored set size by more than 50% for 38% of path changes. The average size of the general and bounded monitored sets are $E[M(v)] = 38.9$ and $E[T(v)] = 27.0$, and the average reduction in size is 31%. The bounded monitored set is identical to the general monitored set for 38% of the path changes (where the solid line intersects the y-axis). This happens whenever an AS is using its least preferred path, as we have to monitor all other paths that are more preferred. Even if an AS is not using its least preferred path, the set of ASes to monitor quickly converges to the general monitored set as the number of more preferred paths we need to monitor increases.

The dashed line in Fig. 7 shows the reduction in monitored set size across the subset of path changes when an AS v changes from its most preferred path to an arbitrarily less preferable path. Because we only need to monitor a few less preferred paths when v is using its most preferred path, the reduction in monitored set size is significantly higher. The average reduction in this case is 65%.

The reduction in candidate set sizes (*i.e.*, $|\mathcal{C}(v) - \mathcal{B}(v)| \div |\mathcal{C}(v)|$) is much smaller (not shown), as induced path changes are rare in practice and we do not observe any 3-level induced path changes.

Detection Speed. To allow providers and operators to debug problems caused by path changes in real time, we would like *PoiRoot* to provide root cause analysis as changes occur. Our current system refreshes paths from traceroutes every 10 minutes and from BGP feeds every 15 minutes. The algorithm for generating the suspect set takes seconds to run, so our system currently can provide results at the same rate that paths are refreshed (*i.e.*, 15 minutes or less). We note that we can further reduce detection time by monitoring real-time BGP feeds (*e.g.*, via BGPMon [42]) and triggering path measurements on demand in response to issues such as performance problems.

5.2 In the Wild Path Changes

Having shown that our approach is accurate and precise for path changes obtained by controlled BGP poisonings and simulations, we now demonstrate its effectiveness for path changes seen "in the wild." In particular, we focus on path changes that significantly worsen end-to-end latency, as they are most likely to warrant further debugging from service providers and operators that optimize for performance.

5.2.1 Methodology

We identify path changes as follows. We issued traceroutes between all PlanetLab sites every ten minutes for a period of three days starting November 1st, 2012, and perform IP-to-AS translation to convert traceroutes to AS-level paths. We extract from this dataset AS-path changes that last for at least an hour. In addition we require that the path before the change is stable for at least 30 minutes to allow for convergence.

Since network operators will be most interested in further investigating those path changes that degrade performance significantly, we use the relative change in RTT as our performance metric and consider only those path changes that experience an increase in RTT larger than 10ms and whose relative increase in RTT is greater than 25%. Each RTT value is averaged over at least three samples. This gives us a set of 643 path changes for the three day period.

5.2.2 Results

Fig. 8 shows the distribution of the size of the suspect set built by *PoiRoot* with the **Correlation** heuristic for the path changes identified above, as well as the corresponding distribution from controlled experiments §5.1.3. The results are similar: the suspect set size is one AS for more than 64%, and to two ASes for more than 83% of the path changes. We find that 96% of the suspect sets with more than one AS include a sequence of consecutive ASes (*i.e.*, ASes are adjacent).

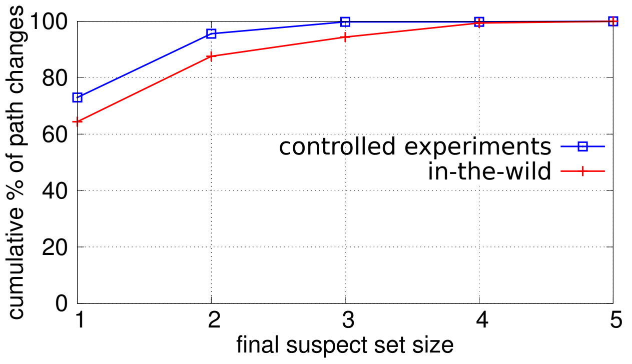

Figure 8: Distribution of suspect set sizes built by *PoiRoot* running on path changes observed "in the wild". The corresponding CDF from §5.1.3 is shown as well. Our algorithm identifies the root cause as being one or two ASes for more than 84% of the cases.

6. RELATED WORK

Measuring path changes. Several previous studies have introduced new path measurement techniques [7, 16, 17, 24, 31, 34, 43], *e.g.*, to discover inter-AS links, quantify BGP path exploration, or infer an AS's path preferences. Our system combines multiple sources of data to collect Internet path measurements, including public BGP feeds [28] and traceroute measurements collected from PlanetLab. Previous work has used BGP poisoning to quantify the use of default routes in the Internet [1]. We use BGP poisoning to perform topology discovery in a way equivalent to that of Colitti *et al.* [5]. However, ours is the first work we are aware of that uses BGP poisoning to obtain ground truth for evaluating root cause analysis in the Internet.

Impact of path changes. Several previous studies highlight the impact of route changes [8, 15, 20, 25, 26], *e.g.*, causing increased delays and transient packet loss due to convergence, in addition to outages due to misconfiguration. In this work, we identify the root cause of such pathological changes, which can assist in debugging and correcting the problem. Previous work also studied how paths propagate in the Internet [12, 13]. Our work uses a basic routing model to understand how interdomain path changes propagate, building on previous work that models the BGP decision process [10].

Root cause analysis. As described in §2, the work by Feldmann *et al.* [9] and Caesar *et al.* [2] are most closely related to ours. We demonstrated that the *NOOR* assumption prevents their approaches from achieving both accuracy and precision. Several other papers have looked at the problem of root cause analysis from different perspectives. Wu et al. [41] propose a system to identify high-impact network events affecting an AS, using routing changes observed at an AS's border routers and correlating it with traffic changes. The system helps operators identify important events, but is limited to one network and cannot identify the cause of events. In particular, if an event is caused by a distant AS, their system can only indicate the border routers involved. Pei et al. [32] propose a modification to BGP that includes information about the networks responsible for a path change. Our work demonstrates that one can identify root causes without requiring protocol modification. LIFEGUARD [18] and NetDiagnoser [6] identified a set of routers responsible for packet-forwarding outages, but did not focus on which network's path change (if any) caused the outage.

7. CONCLUSION

In this paper, we described an algorithm and a scalable system, *PoiRoot*, for accurately and precisely identifying the network responsible for Internet path changes, even in the face of induced path changes and incomplete information. We developed a general algorithm for identifying the root cause of path changes, then used simple, validated assumptions about routing behavior to derive new constraints on the networks that can possibly be responsible for a path change. We then designed a system to gather the requisite path information and produced an algorithm that identifies the root cause of a path change even with missing measurements. Finally, we evaluated *PoiRoot* using ground truth from path changes induced on real Internet paths, empirically informed simulations and natural path changes "in the wild." We showed that previous approaches to root cause analysis can either be accurate or precise, while our approach achieves both. As part of our future work, we are investigating ways to use this root cause information to automatically address routing changes that negatively impact performance.

Acknowledgments

We gratefully acknowledge our shepherd Craig Labovitz and the SIGCOMM reviewers. The Transit Portal 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, Schyler Batey, and Nick Feamster. This work was partially funded by Google, Cisco, NSF (CNS-0905568 and CNS-1040663), CAPES, CNPq (Universal 477932), and the NSF/CRA Computing Innovation Fellowship program. We are thankful for their support.

8. REFERENCES

[1] R. Bush, O. Maennel, M. Roughan, and S. Uhlig. Internet optometry: assessing the broken glasses in Internet reachability. In *IMC*, 2009.
[2] M. Caesar, L. Subramanian, and R. H. Katz. Towards localizing root causes of BGP dynamics. Technical report, University of California, Berkeley, 2003.
[3] 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.
[4] A. Christie. *Murder on the Links*. The Bodley Head, 1923.
[5] L. Colitti. *Internet Topology Discovery Using Active Probing*. PhD thesis, University di Roma Tre, 2006.
[6] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot. NetDiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data. In *CoNEXT*, 2007.
[7] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, kc claffy, and G. Riley. As relationships: inference and validation. *SIGCOMM Comput. Commun. Rev.*, 37(1):29–40, 2007.
[8] N. Feamster, D. G. Andersen, H. Balakrishnan, and M. F. Kaashoek. Measuring the effects of Internet path faults on reactive routing. In *SIGMETRICS*, 2003.
[9] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs. Locating Internet routing instabilities. In *SIGCOMM*, 2004.
[10] L. Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM TON*, 9(6):733–745, 2001.
[11] P. Gill, S. Goldberg, and M. Schapira. A survey of interdomain routing policies. NANOG 56, 2012. http://www.nanog.org/meetings/nanog56/presentations/Monday/mon.general.gill.11.pdf.
[12] T. Griffin and G. Huston. BGP wedgies. Network Working Group, RFC 4264, Nov. 2005.
[13] T. G. Griffin, B. F. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM ToN*, 10(2):232–243, 2002.
[14] N. Gvozdiev, B. Karp, and M. Handley. LOUP: The principles and practice of intra-domain route dissemination. In *NSDI*, 2013.
[15] Y. Huang, N. Feamster, A. Lakhina, and J. J. Xu. Diagnosing network disruptions with network-wide analysis. In *SIGMETRICS*, 2007.
[16] 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.
[17] 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.
[18] E. Katz-Bassett, C. Scott, D. R. Choffnes, I. Cunha, V. Valancius, N. Feamster, H. V. Madhyastha, T. Anderson, and A. Krishnamurthy. LIFEGUARD: Practical repair of persistent route failures. In *SIGCOMM*, 2012.
[19] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao. Moving beyond end-to-end path information to optimize CDN performance. In *IMC*, 2009.
[20] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. In *SIGCOMM*, 2000.
[21] C. Labovitz, G. R. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM TON*, 6(5):515–528, 1998.
[22] G. Linden. Make data useful. http://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-28.ppt, 2006.
[23] H. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane Nano: Path Prediction for Peer-to-Peer Applications. In *NSDI*, 2009.
[24] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An information plane for distributed services. In *OSDI*, 2006.
[25] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *SIGCOMM*, 2002.
[26] Z. M. Mao, R. Bush, T. G. Griffin, and M. Roughan. BGP beacons. In *IMC*, 2003.
[27] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz. Towards an accurate AS-level traceroute tool. In *SIGCOMM*, 2003.
[28] D. Meyer. RouteViews. http://www.routeviews.org.
[29] W. Mühlbauer, A. Feldmann, O. Maennel, M. Roughan, and S. Uhlig. Building an AS-topology model that captures route diversity. In *SIGCOMM*, 2006.
[30] R. Oliveira, D. Pei, W. Willinger, B. Zhang, and L. Zhang. The (in)completeness of the observed internet as-level structure. *IEEE/ACM Trans. Netw.*, 18(1):109–122, 2010.
[31] R. Oliveira, B. Zhang, D. Pei, and L. Zhang. Quantifying path exploration in the internet. *IEEE/ACM Trans. Netw.*, 17(2):445–458, 2009.
[32] D. Pei, M. Azuma, D. Massey, and L. Zhang. BGP-RCN: Improving BGP convergence through root cause notification. *Computer Networks*, 48(2):175–194, 2005.
[33] Ponemon Institute. Calculating the cost of data center outages, 2011. http://www.emersonnetworkpower.com/en-US/Brands/Liebert/Documents/White%20Papers/data-center-costs_24659-R02-11.pdf.
[34] A. Shaikh and A. Greenberg. OSPF monitoring: Architecture, design and deployment experience. In *NSDI*, 2004.
[35] N. Spring, R. Mahajan, and T. Anderson. Quantifying the causes of path inflation. In *SIGCOMM*, 2003.
[36] S. Stefanov. Yslow 2.0. In *CSDN SD2C*, 2008.
[37] R. Teixeira and J. Rexford. A measurement framework for pin-pointing routing changes. In *ACM SIGCOMM Workshop on Network Troubleshooting*, 2004.
[38] A. Toonk. BGPmon. In *NANOG 45*, 2009. http://www.nanog.org/meetings/nanog45/presentations/Sunday/Toonk_bgpmon_N45.pdf.
[39] UCLA Internet topology collection. http://irl.cs.ucla.edu/topology/.
[40] V. Valancius, N. Feamster, J. Rexford, and A. Nakao. Wide-area route control for distributed services. In *ATC*, 2010.
[41] J. Wu, Z. M. Mao, J. Rexford, and J. Wang. Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network. In *NSDI*, 2005.
[42] H. Yan, D. Matthews, R. Oliveira, L. Zhang, K. Burnett, and D. Massey. Bgpmon: A real-time, scalable, extensible monitoring system. In *In Cybersecurity Applications and Technologies Conference for Homeland Security(CATCH*, 2009.
[43] 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.
[44] Y. Zhu, B. Helsley, J. Rexford, A. Siganporia, and S. Srinivasan. Latlong: Diagnosing wide-area latency changes for cdns. *IEEE Transactions on Network and Service Management*, 9, 2012.