Sibyl: A Practical Internet Route Oracle

Ítalo Cunha, *Universidade Federal de Minas Gerais;* Pietro Marchetta, *University of Napoli Federico II;* Matt Calder, Yi-Ching Chiu, and Brandon Schlinker, *University of Southern California;* Bruno V. A. Machado, *Universidade Federal de Minas Gerais;* Antonio Pescapè, *University of Napoli Federico II;* Vasileios Giotsas, *University of California, San Diego/CAIDA;* Harsha V. Madhyastha, *University of Michigan;* Ethan Katz-Bassett, *University of Southern California*

https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/cunha

This paper is included in the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16).

March 16–18, 2016 • Santa Clara, CA, USA

ISBN 978-1-931971-29-4

Open access to the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16) is sponsored by USENIX.

Sibyl: A Practical Internet Route Oracle

Ítalo Cunha[†*] *Pietro Marchetta*[‡*] *Matt Calder*[★] *Yi-Ching Chiu*[★]
Brandon Schlinker[★] *Bruno V. A. Machado*[†] *Antonio Pescapè*[‡]
Vasileios Giotsas[◇] *Harsha V. Madhyastha*[+] *Ethan Katz-Bassett*[★]

[†]Universidade Federal de Minas Gerais [‡]University of Napoli "Federico II"

[★]University of Southern California [◇]University of California San Diego/CAIDA [+]University of Michigan

Abstract

Network operators measure Internet routes to troubleshoot problems, and researchers measure routes to characterize the Internet. However, they still rely on decades-old tools like traceroute, BGP route collectors, and Looking Glasses, all of which permit only a single query about Internet routes—what is the path from here to there? This limited interface complicates answering queries about routes such as *"find routes traversing the Level3/AT&T peering in Atlanta,"* to understand the scope of a reported problem there.

This paper presents *Sibyl*, a system that takes rich queries that researchers and operators express as regular expressions, then issues and returns traceroutes that match even if it has never measured a matching path in the past. *Sibyl* achieves this goal in three steps. First, to maximize its coverage of Internet routing, *Sibyl* integrates together diverse sets of traceroute vantage points that provide complementary views, measuring from thousands of networks in total. Second, because users may not know which measurements will traverse paths of interest, and because vantage point resource constraints keep *Sibyl* from tracing to all destinations from all sources, *Sibyl* uses historical measurements to predict which new ones are likely to match a query. Finally, based on these predictions, *Sibyl* optimizes across concurrent queries to decide which measurements to issue given resource constraints. We show that *Sibyl* provides researchers and operators with the routing information they need—in fact, it matches 76% of the queries that it could match if an oracle told it which measurements to issue.

1 Introduction

Operators and researchers need Internet route measurements to keep the Internet running smoothly, to understand its behavior, and to improve it for the future [61, 75]. Route measurements help identify performance problems caused by circuitous routing [34, 58, 73], loops and loss caused by inconsistency during route convergence [11, 23, 28, 35, 36, 52, 60, 69], and outages caused by misconfigurations [7, 31, 32, 51, 74]. Route measurements can reveal malicious hijacks [76] and inadvertent routing leaks [24]. Route measurements are also used to understand the Internet's structure [2, 6, 29, 41, 57, 71] and performance [42].

The ideal: An Internet route oracle. Given the importance of route measurements, one can imagine a centralized platform that could be queried for any Internet route of interest. Which end-points in Europe route to each other circuitously via networks in other continents? Which routes traverse the Atlanta peering between Level3 and AT&T that seems to be experiencing congestion? Is the problem more widespread—which routes traverse a peering between Level3 and AT&T that is not in Atlanta? Which routes go through Level3 in Atlanta without going through AT&T? Which Tor exit nodes have routes to my destination that do not traverse the US? A platform that can answer such questions would enable better understanding and faster troubleshooting for researchers and operators.

The reality: Traceroute. While such a platform would be enormously useful, the reality today is far from it. We are stuck with tools like traceroute. While traceroute is simple, widely used, and has been incredibly useful [1, 2, 6, 7, 13, 22, 27, 29, 31, 32, 41, 42, 51, 57, 61, 71, 73, 74, 75, 76], it offers a very limited capability–it can only answer *"what is the path from here to there?"* We are used to asking this question, so it seems natural, but in fact it is only one of the many questions we might ask about Internet routes, limiting the ability of operators and researchers to access the routing information they need. The Outages network operators mailing list [51] illustrates the problem—operators frequently send a traceroute to the mailing list when experiencing problems [7],

[*]The two lead authors are listed alphabetically. They conducted some of this research as visiting scholars at USC.

asking other operators to send traceroutes from their vantage points, in the blind (and often unsuccessful) hope that someone will issue a measurement that illuminates the problem.

Our contribution: A practical traceroute-based oracle. While a complete oracle for Internet routing is clearly infeasible without radical changes to the network, we demonstrate that we can come surprisingly close using only available vantage points and measurement tools.

We present *Sibyl*,[1] our system that can serve a rich set of queries about Internet routes. *Sibyl*'s interface is simple yet powerful: a user submits a regular expression describing paths of interest (§3.2), and *Sibyl* returns routes that match. Users need not worry about which vantage points to use, how to access or configure them, or which destinations to target. Behind the scenes, *Sibyl* issues traceroutes from a diverse set of vantage points, with the goal of satisfying a query if any vantage point has routes that match. Our evaluation in Section 8.2 shows that combining vantage points from multiple measurement platforms achieves unprecedented coverage, better than even successful crowd-sourced measurements [55, 56].

The problem: Resource constraints limit measurement budgets. Although the integration of multiple sets of vantage points offers the *potential* to improve coverage [8, 64], most vantage points are severely constrained in the number of measurements they can issue. This constraint occurs because the most diverse sets of vantage points are in home networks [55, 56, 57, 62], on personal phones [73], and on production devices [63], settings in which measurements cannot be allowed to interfere with other uses of already scarce resources. Thus, exhaustive probing to answer a query is infeasible, and allocating limited measurements to maintain an up-to-date atlas in the face of path changes is an extremely hard problem [15].

The main challenge in building *Sibyl* to serve any query is that, due to its limited probing budget, it may have never previously measured a path that matches the query or, even if it did, the path may have changed since. As a result, it needs to serve queries despite uncertainty about which measurements match the queries.

Our approach: Allocate measurement budget based on predictions. Our primary technical contributions to overcome this challenge are three-fold. First, we demonstrate how *Sibyl* can use the structure of a query to focus its attention on a small number of traceroutes to consider issuing (§6). Second, we design a prediction engine which uses an atlas of previously-issued traceroutes to predict which unissued traceroutes are likely to match input queries (§5). Third, we develop an optimization framework that uses the predictions to allocate *Sibyl*'s probing budget to measurements that maximize how well it satisfies input queries (§4).

Building an effective prediction engine requires addressing potential causes of inaccuracy. First, the prediction engine could make an incorrect prediction from even an up-to-date atlas, due to inaccuracies in the modeling of routing policy. Second, measurements in the atlas may become out-of-date. So, we develop techniques to evaluate how likely a prediction is to be correct (§5.2), allowing *Sibyl* to incorporate the likelihood into its optimizations, and we develop lightweight approaches *Sibyl* can use to identify and patch or discard paths that may no longer be correct (§7).

Our evaluation (§8) shows that, using this prediction approach, *Sibyl* can serve 32% more queries than it could without calculating likelihoods and can, despite stringent rate limits, serve 76% of the test queries that it could if it had an oracle informing it which measurements to issue.

[1]Named for the oracular **Siby**ls of ancient Greece, not the pesky **Sybi**ls who keep undermining our P2P systems.

2 Motivating *Sibyl*'s approach

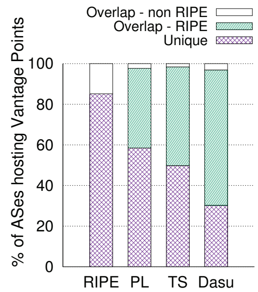
Traceroute is widely supported, and when the right traceroute measurement is at hand, it can prove useful for a range of tasks. Therefore, we use traceroute measurements as the basis of *Sibyl* and strive to overcome its limitations.

Opportunity: Combining platforms improves coverage. Today, one can use a number of publicly accessible measurement platforms that offer vantage points (VPs) across the world in order to issue traceroute measurements. In this paper, we focus on platforms at two extremes—small numbers of powerful VPs in a somewhat homogeneous deployment (PlanetLab) versus large numbers of severely limited VPs in networks around the world (RIPE Atlas and traceroute servers). In addition, Dasu and DIMES each offer several hundreds to several thousands of VPs from which one can issue traceroutes. For a few of these platforms, Figure 1a presents the number of VPs they offer and the number of ASes across which these VPs are spread. Although Figure 1a shows that the number of ASes in which RIPE Atlas offers VPs is much higher than in other platforms, we see in the *Unique* portions of the bars of Figure 1b that each of the other platforms contributes significantly to improving the number of distinct ASes covered by VPs. For all three of PlanetLab, Dasu, and traceroute servers, 30%–60% of ASes in which they have VPs do not host VPs for any of the other platforms.

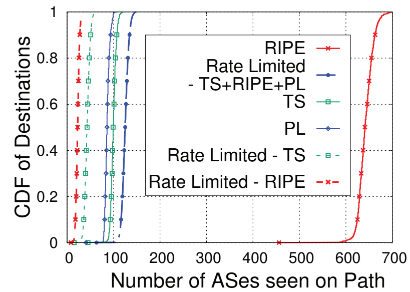
Challenge: Resource constraints limit probing rates. The wide spread of RIPE Atlas and the presence of other VPs in ASes without Atlas probes show promising coverage for a unified system. But, effective use is compli-

Platform	# of VPs	# of ASes
PlanetLab (PL)	422	260
RIPE Atlas (RIPE)	7699	2716
Traceroute servers (TS)	499	494
DIMES	<400	–
Dasu	–	288

(a) Sizes of measurement platforms.



(b) Vantage point uniqueness.



(c) Path diversity. Dashed lines obey rate limits.

Figure 1: Utility of combining measurement platforms: (a) Comparison of deployment sizes. (b) % of ASes hosting a platform's vantage points (VPs) that: (i) only host that platform; (ii) also host a RIPE Atlas VP; (iii) or host a VP from another non-Atlas platform. (c) Path diversity, with/without rate limits that exist in practice.

cated because the most diverse sets of VPs have severe and inevitable resource constraints. Good visibility requires an ability to measure from many networks low in the AS hierarchy [48], and researchers have argued and demonstrated that the way to achieve this viewpoint—especially in remote and developing regions—is to gather measurements from mobile devices [70, 73] and home networks [12, 17, 22, 55, 56, 57, 62], settings in which resources are scarce and researchers are guests. To give a sense of the constraints, measuring a traceroute every 5 minutes to all 500,000 BGP prefixes would take more than 40 Mbps—much higher than typical uplink bandwidth in many parts of the world.[2] And, to avoid interfering with the hosts, the platforms limit measurements to only a small fraction of this rate. Traceroute servers also offer diverse VPs, but these machines serve an operational role and so do not allow a fast rate of measurement. Future faster rates will still strain in the face of measurement-hungry use cases such as network tomography [9, 14] and studying route convergence [36, 69], which require consistent snapshots or rapid tracking of changes, respectively.

Figure 1c depicts one measure of the impact of limited probing budgets. It plots the number of unique ASes seen when using the vantage points in PlanetLab, RIPE Atlas, and traceroute servers in isolation and in combination, with and without rate limits. In each case, we consider traceroutes to the .1 address in the same 1000 IP prefixes, and we do not count the source AS (which we accounted for in Figure 1b). We have measurements from every PlanetLab site, RIPE vantage points in 2000 ASes not covered by PlanetLab, and traceroute servers in 200 ASes not covered by RIPE. Ignoring rate limits, RIPE vantage points provide routes to most destinations through >600 transit ASes, versus only ≈ 100 when using PlanetLab or traceroute servers alone. Figure 1c also depicts the path diversity we can uncover if we allocate a day's Atlas probing budget and a day's traceroute server rate limit evenly across the 1000 destinations. RIPE enforces a per user aggregate rate limit across all sources and destinations. Here, we split it across a quarter of the Atlas VPs and 0.2% of the Internet's prefixes. We follow established research best practices [40, 59] and limit ourselves to one traceroute every 5 minutes per public traceroute server. These limits result in us randomly choosing 16–17 RIPE vantage points and 57–58 traceroute servers from which to probe each destination, and the graph shows results averaged across 10 trials. Given these severe rate restrictions, the benefit of PlanetLab—and its very high achievable probe rate—becomes clear, and the route diversity is much better if we combine the rate-limited traceroute servers and Atlas platform with the smaller, but less restrictive PlanetLab platform.

Challenge: Rate limits necessitate decisions in the face of uncertainty. The vast gap between the full diversity of paths seen in Figure 1c and the diversity seen when subject to rate limits shows that we have to be quite discerning in how we allocate a limited probing budget, to make sure we are issuing the measurements most useful to the queries at hand. We cannot issue measurements fast enough to have up-to-date paths to large numbers of destinations—the rate limits imposed by Atlas and traceroute servers [40, 59] mean that it would take years to measure routes from their VPs to all 500K BGP prefixes. Therefore, to serve queries well, it is necessary to reason effectively about which traceroutes to issue despite uncertainty about routes that measurements will traverse and, hence, which traceroutes will satisfy queries.

3 System overview

Goal. Our goal is to provide researchers and operators with route measurements of interest to them. Our system should allow them to express properties of interest in a natural way, without the user needing to know a

[2] Beyond constraints on the VPs, we do not want to overload upstream devices or links with measurement traffic, and routers and other devices increasingly rate-limit and de-preference these probes.

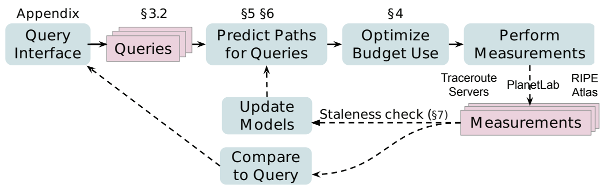


Figure 2: Sibyl architecture.

priori which (source, destination) pairs will yield paths with those properties. Section 2 shows existing traceroute platforms offer rich path diversity, if the system can respect resource constraints while efficiently measuring only the paths most useful in serving received queries.

3.1 Basic architecture

Figure 2 depicts *Sibyl*. Users submit queries to the system (§3.2). It operates in rounds, queueing up queries in between rounds. This round-by-round operation allows us to formulate the decision as a clean optimization, simplifies rate limiting, and aids in efficient use of a probing budget by batching requests. RIPE Atlas, for instance, is designed to perform more efficiently when given batches of measurements. Each round, the system predicts which traceroutes might be useful to match pending queries (§5 and §6). It then formulates an optimization to select the traceroutes to issue given measurement resource constraints (§4.1–4.2) and solves the optimization greedily (§4.3). It issues the measurements, collects the results, and returns results that match user queries.

Sibyl currently uses vantage points from PlanetLab, RIPE Atlas, and traceroute servers. We developed a controller for each that exposes them via a common API, including information on available vantage points and rate limits, and commands to request and collect traceroutes. A central controller integrates the three platform controllers to present the rest of *Sibyl* with a unified view. In the background, we issue daily traceroutes from all PlanetLab sites to responsive destinations across the Internet [25] to bootstrap *Sibyl*'s knowledge of routing.

3.2 Specifying queries

Just as other work found regular expressions to be a natural way to express properties of paths [46, 67], we support queries in a form that we refer to as symbolic regular expressions over IP addresses. Symbolic regular expressions are an analogue to symbolic finite automata [65], in which transitions are labeled with Boolean predicates on IP addresses, rather than directly with IP addresses. These predicates allow, for example, the natural expression of `Sprint(x)&CHI(x)` rather than listing all Sprint IP addresses in Chicago. We will use the notation `Sprint&CHI`. A predicate can delineate any subset of IP addresses, but our UI currently supports ASes, cities, and countries, [3] and also prefixes for sources/destinations.[4]

Users augment their queries with a utility function that indicates how well a set of traceroutes satisfies their needs. *Sibyl*'s UI currently supports two types of utility functions that we believe cover a wide range of queries. For *existence queries*, the user wants one matching path, e.g., the user may want to know the path from a particular network to a specific destination. The utility is zero if no measurements match, or a constant value if one or more measurements match. For *diversity queries*, the user wants a set of paths matching the query in as diverse ways as possible. For example, the user may want to know all paths that pass through a given AS link, in order to learn the set of (source, destination) pairs that use that link. The utility is a function of path diversity, which we model as a constant times the number of distinct elements seen in the set of measurements that match the query. The user specifies the granularity of elements by selecting any combination of (AS, city, and country). Again, if none of the traceroutes match the query, the utility is zero.

Now, let us consider a few example queries, in POSIX ERE-like syntax with dashes in between symbols for clarity. Parentheses create a group, whereas curly braces indicate that the query is a diversity query and delineate the portion of the query to diversify over.

Reverse traceroute [30]: To query for a path from a network *r* back to a source *s*, the user requests:

```
r-.*-s$
```

Detecting prefix hijacks with iSpy [76]: iSpy monitors paths towards a prefix *p* in the background. When the AS loses reachability to other destinations, iSpy considers it a normal outage if the destinations share common subpaths to the AS, or a hijack if the destinations represent a large cut in the graph towards *p*. To identify diverse AS paths for iSpy to monitor, an operator could query for:

```
^{.*}-p$ by <AS>
```

Troubleshooting a problem [51]: On January 6, 2015, an operator emailed the Outages mailing list suspecting a problem on paths that went between Level3 in LA and GTT in Seattle, and he wanted to check other paths with that subpath. He was requesting:

```
^{.*-(GTT&SEA-.*-Level3&LAX |
Level3&LAX-.*-GTT&SEA)-.*}$ by <AS,city>
```

[3]Mapping IP addresses to PoPs, ASes, and locations are active areas of research. We use iPlane's PoP and AS mappings and MaxMind's location data. *Sibyl* is agnostic to how mappings are generated, and its results will improve as mappings do.

[4]Our techniques for deciding which measurements to issue in response to a query (§5) base decisions on previous measurements of routing, so implicitly encode routing policies and hence avoid wasting measurements trying to match unlikely regular expressions, such as one that asks for a path that traverses every Tier-1 network.

Another operator replied that traceroutes with the problem seemed to traverse a Seattle peering between GTT and NTT. To see if the problems occurred on GTT paths with other peers as well, one might query for:

```
^.*-{[^NTT-Level3]}-GTT&SEA-
      {[^NTT-Level3]}-.*$ by <AS>
```

Appendix I presents screenshots of *Sibyl*'s query interface for the last of these examples.

3.3 Key problems to solve

Section 2 described two key challenges *Sibyl* must overcome in integrating traceroute platforms to serve these types of queries: severe probing rate limits induced by resource constraints, and the need to decide how to allocate these limited probes despite not knowing definitively which traceroutes will satisfy queries. To overcome these challenges, we address the following sub-problems:

- (§4) Suppose we can address uncertainty by capturing the likelihood that a traceroute, if issued, will match a query. How should *Sibyl* allocate its probing budget to best serve queries?
- (§5) How can *Sibyl* calculate those likelihoods?
- (§6) Given that the set of possible measurements is large, how can *Sibyl* limit the set of traceroutes it has to consider issuing (and hence calculate likelihoods for)?

4 Maximizing returns from rate limits

Since *Sibyl* cannot issue every traceroute—or even every traceroute that would match the queries—in a given round, it needs to intelligently allocate its probing budget to best serve a set of queries. Because *Sibyl* must issue a traceroute in order to know definitively whether it matches a query, *Sibyl*'s goal is to maximize the expected utility of the traceroutes it issues. This section describes how *Sibyl* allocates its budget, assuming it has an oracle that answers, for every possible traceroute, the likelihood that the traceroute, if issued, will match a particular query. Section 5 describes how *Sibyl* estimates these likelihood values to approximate such an oracle.

4.1 Accounting for rate limits

Since we want *Sibyl* to incorporate different sets of VPs to improve coverage and path diversity, we need to account for the different *kinds* of rate limits across platforms. The rate at which a PlanetLab node can probe is limited by the ability of our traceroute tool to send and receive and by the available bandwidth. ISPs make traceroute servers available through websites, but restrict how often one can issue traceroutes from a website. RIPE Atlas users earn credits for hosting probes, then spend credits by issuing measurements. We host a number of Atlas probes in order to earn credits, but RIPE caps the number of credits a user can spend in a day regardless of credit balance.

We unify these different types of rate limits as follows. First, we group together each set of vantage points that are subject to a shared aggregate rate limit. For the i'th such set, we will use the notation $V_i = \{v_{i,1}, v_{i,2}, v_{i,3}, \ldots\}$ to indicate the vantage points in set V_i, and let $\mathcal{V} = \{V_1, V_2, V_3, \ldots, V_n\}$ be the collection of n sets used. For PlanetLab, each host is in a singleton set, since the number of traceroutes sent from one PlanetLab site does not affect the number that can be sent from another. For traceroute servers, we group the hosts behind a common web interface (generally the hosts in one ISP), since we are limited in how often we can query a website without drawing complaints. For RIPE Atlas, we group together all vantage points in the platform, since they are subject to a platform-wide credit budget and daily limit.

Second, in each round, *Sibyl* has a multi-element budget of traceroutes it can issue, with one budget per set of vantage points in $\mathcal{V}$. For rate-limited vantage points like PlanetLab or traceroute servers, the per-round traceroute budget for a set reflects the rate limit on the set and the duration of the round. For credit-based vantage point platforms like RIPE Atlas, we set a per-round aggregate budget for all traceroutes from the platform to reflect the number of credits we earn in a round.[5]

4.2 Formulating the optimization

In a given round r, we have a set of queries $Q = \{q_1, q_2, \ldots, q_m\}$, each with a corresponding utility function $f_{q_1}, f_{q_2}, \ldots, f_{q_m}$ that maps a set of traceroutes to a score. For each set of vantage points $V \in \mathcal{V}$, we have a per round budget C_V. Each V defines a set of possible traceroutes $T_V = \{t_{v,d} \mid v \in V, d \in$ the set of all Internet destinations $D\}$, where $t_{v,d}$ is the traceroute from v to d, and we have to select a subset $T_{r,V} \subseteq T_V$ to issue in round r such that $|T_{r,V}| \leq C_V$.

Our goal is to select traceroutes, subject to budget constraints, to maximize the combined utility across queries:

$$\begin{aligned} &\max_{T_r} f(T_r), \text{ where } T_r = \bigcup_{V \in \mathcal{V}} T_{r,V} \\ &\qquad \text{and } f(T_r) = \sum_{q \in Q} f_q(T_r) \\ &\qquad \text{subject to } |T_{r,V}| \leq C_V \quad \forall V \in \mathcal{V} \end{aligned} \tag{1}$$

Since we cannot know whether a traceroute satisfies a query before issuing it, in practice, *Sibyl* maximizes the

[5] We adjust the exact budget round-by-round to allow overspending when we have banked a surplus or exercise caution when reserves run low, as well as to cap it to not exceed the daily platform limit.

expected utility. We use the notation $t \in q$ to indicate that traceroute t satisfies query q. Assuming an ability to determine the likelihood $p(t \in q_{\text{exist}})$ for any traceroute t matching an existence query q_{exist},[6] *Sibyl* calculates the expected utility $\mathbb{E}\left[f_{q_{\text{exist}}}(T_r)\right]$ as the probability that at least one traceroute matches the query:

$$\mathbb{E}\left[f_{q_{\text{exist}}}(T_r)\right] = 1 - \prod_{t \in T_r} (1 - p(t \in q_{\text{exist}})) \qquad (2)$$

Sibyl calculates the expected utility for a diversity query in a similar way, except that the likelihood values capture, for every path element h at the diversification granularity as defined by a Boolean predicate on IP addresses, the probability that t satisfies q_{div} *and* traverses h (Appendix D.3 presents an example):

$$\mathbb{E}\left[f_{q_{\text{div}}}(T_r)\right] = \sum_h \left(1 - \prod_{t \in T_r} (1 - p(t \in q_{\text{div}} \wedge \exists i \in t : h(i))\right) \qquad (3)$$

(In practice, *Sibyl* scales down diversity utility scores, which are per (query,hop), to balance vs existence queries, which are per (query).)

4.3 Solving the optimization

We apply a greedy algorithm to select the measurements to issue in every round. At each step, *Sibyl* chooses to issue the traceroute that fits in the budget (meaning that the source VP must be part of a set V for which budget remains) and that provides the largest marginal expected utility on top of those already chosen.[7] It stops when no budget remains for the round or when no traceroutes provide additional expected utility. While it may seem like a complicated problem, with a multi-part budget, multiple queries, and queries that desire diverse sets of traceroutes, in fact the greedy algorithm is known to have a provably good approximation bound for this class of problems. See Appendix A for details.

In addition to having good approximation performance, the runtime of our greedy algorithm is reasonable. The runtime is reasonable because both existence and diversity queries allow us to calculate the marginal expected benefit of each possible traceroute in time proportional to the number of queries, without growing with the number of traceroutes already issued. Appendix A describes other utility functions *Sibyl* supports. The worst-case runtime is thus proportional to the size of the budget (the number of greedy steps) times the number of traceroutes under consideration (to assess marginal benefit of each during each greedy step) times the number of queries (to calculate the marginal utility of the traceroute). Most traceroutes under consideration do not match most queries, i.e., $f_q(t) = 0$ most of the time, simplifying the calculation of marginal benefit in practice. *Sibyl* also limits the number of traceroutes under consideration based on the structure of queries (§6).

[6] For simplicity, we assume independence in how well traceroutes satisfy different queries, and in whether different traceroutes satisfy a query.

[7] The marginal expected utility of adding a traceroute t to a set of previously selected traceroutes T is $\mathbb{E}[f(T \cup \{t\})] - \mathbb{E}[f(T)]$.

5 Estimating likelihood of satisfying queries

Sibyl approximates an oracle by using the subset of paths for which it has relatively fresh measurements to predict other paths, checking whether the predictions match queries, and estimating how confident it is in the predictions. PlanetLab paths are stable relative to how often we can refresh PlanetLab measurements. Further, while paths from diverse RIPE Atlas and traceroute server VPs in general change more than PlanetLab paths and cannot be refreshed frequently, the portions of paths near these VPs tend to be quite stable.[8] Based on these observations, our design predicts paths by composing the relative freshness of paths to destinations from PlanetLab with the long-term stability of the beginning portions of paths from other VPs in order to predict unknown paths from these VPs, overcoming the rate limits that keeps us from measuring a full map in a timely fashion.

5.1 Predicting unknown paths

We adapt iPlane's path splicing approach [39] to predict whether a particular unmeasured path is likely to match a query. To predict the path from s to d, iPlane splices a path from s (to some destination) with a path to d (from some source), if they traverse a common point of presence (PoP, a set of routers in the same location and same AS), which we refer to as the splice PoP.

Although iPlane's approach provides a basic mechanism for using measured paths to predict unknown paths, it has two major limitations for our needs. First, iPlane's predictions can be wrong; our experiments found 32% of its AS path predictions to be incorrect. Second, iPlane does not calculate how confident it is in its prediction. Even if iPlane predicts (vantage point, destination) pairs as candidates to match a query, it fails to provide guidance on which paths are more likely to match the query than others, given limited measurement budgets.

We overcome these shortcomings in iPlane's path splicing approach as follows. For a (vantage point v, destination d) pair, while iPlane selects a single best guess for the route between them, we instead consider all possible ways to splice previously measured paths from v with previously measured paths to d. We then estimate

[8] Measurements supporting these claims appear in Appendix B.

our confidence in the correctness of each spliced path in the set S of all possible spliced paths from v to d. We denote the confidence as $p(v \rightarrow d = s)$ (normalized such that $\sum_{s \in S} p(v \rightarrow d = s) \leq 1$). Given these confidence estimates, we compute the likelihood $p(v \rightarrow d \in q)$ of the traceroute from v to d matching a query q as the sum of confidence in the spliced paths that match the query:[9]

$$p(v \rightarrow d \in q) = \sum_{s \in S \wedge s \in q} p(v \rightarrow d = s) \qquad (4)$$

Appendix D.2 illustrates how the above process works.

5.2 Assigning confidence to predictions

To assess the confidence in each spliced path, we employ RuleFit, a supervised machine learning technique [20]. We describe how we train and apply a RuleFit model. The model takes the set of spliced paths of a particular prediction and assigns confidence to each based on features of the paths.

Training the RuleFit model RuleFit is a supervised machine learning technique based on rule ensembles. In our case, we supply RuleFit with a training set that maps from features of a predicted (spliced) path (e.g., the predicted path's AS-path length and the latency from the source to the splice point; Appendix C describes all features) to the similarity between the spliced path and the actual path. As a measure of similarity, we use the PoP-level Jaccard index. RuleFit then generates thousands of rules that combine features in logical expressions and builds a model using rules that help predict the Jaccard index. RuleFit automatically generates and selects rules (and indirectly, features) using techniques such as decision trees and lasso constraints. See the RuleFit paper for details [20]. Each rule selected by RuleFit has an associated value, with positive (negative) values for rules meant to identify predicted paths similar (dissimilar) to the real path, indicating high (low) Jaccard index. Important features may change over time, as the Internet and the set of *Sibyl* VPs evolves. We track the accuracy of predictions over time to identify if performance drops and can re-initiate training. For the evaluation in Section 8, we use traceroutes from 100 PlanetLab sites to 500 destinations and from all RIPE Atlas (Atlas) and traceroute server (TS) sites to 50 destinations to generate spliced paths from Atlas and TS sites to the 500 destinations. We randomly chose 2.5% of the spliced paths to train a RuleFit model.

[9]The likelihood value for an (v, d) pair need not directly or inversely correlate with the number of spliced paths between the pair, as it depends on how confidence varies across spliced paths and on which spliced paths match the query. For example, if $p(v \rightarrow d = s_1) = 0.5$ and $p(v \rightarrow d = s_2) = 0.25$, $p(v \rightarrow d \in q)$ could be 0.25 ($s_2 \in q$), 0.5 ($s_1 \in q$), 0.75 ($s_1 \in q$ and $s_2 \in q$), or 0 (neither matches).

Using the RuleFit model To use the model to estimate the Jaccard index of a predicted (spliced) path from v to d, *Sibyl* calculates the features of the predicted path, then uses the RuleFit model to score the path. The score for a spliced path is the sum of rule values for rules that match the spliced path's features; e.g., if the spliced path's AS-path length is among the shortest, then increase the confidence (score) that it is very similar to the actual path. It repeats this process for every spliced path between (v, d).

Sibyl translates these estimates of similarity between a (known) spliced path and (unknown) actual path into a confidence estimate that a query that matches (does not match) the spliced path will also match (not match) the actual path. We assign each predicted path a confidence proportional to its RuleFit score, normalized to sum to the highest predicted Jaccard index among all spliced paths for $v \rightarrow d$. *Sibyl* uses these confidence values to estimate the likelihood that a traceroute will match a query, using Eq. 4, which it then uses to optimize the expected utility of the traceroutes it chooses to issue, in Eq. 1.

Section 8.3 evaluates the accuracy of *Sibyl*'s likelihood estimates, Appendix E evaluates the accuracy of its Jaccard index predictions, and Appendix C describes the RuleFit model in more detail.

6 Limiting traceroutes to consider

Thus far, our description has assumed that we estimate the likelihood of matching a query for every possible traceroute from every vantage point, and then use these likelihood values to choose the subset of traceroutes that *Sibyl* should measure in order to maximize utility, given rate limits. However, due to the non-negligible computation associated with the estimation of likelihood values, running this computation on all (vantage point, destination) pairs is not practical.

Instead, *Sibyl* computes the likelihood of matching a query q only on a subset of candidate paths it deems likely to match the query. The goal of candidate generation is to identify (vantage point v, splice PoP r, destination d) tuples such that *Sibyl* has a previous traceroute from v going through r that matches a prefix of the query q (possibly the empty prefix), and has a traceroute to d through r that matches the remaining suffix (possibly empty). For example, candidate generation for the query `Level3-Cogent-.*-SmallISP` could find a path that traverses a `Level3-Cogent` link on the way to some r, then another path that traverses r on its way to `SmallISP`. The process works as follows.

1. Given the query q, construct a symbolic finite automaton A_q that accepts L_q, the language of paths that match the expression q.

2. Run A_q over all traceroutes previously gathered from *Sibyl*'s VPs, which consists of evaluating the hops' IP addresses against A_q's transition predicates, testing, for example, AS membership. Label each (source, PoP) tuple with all of the state-to-state transitions that A_q can follow in consuming one of the PoP's IP addresses when processing a traceroute from that source.

3. Build A_q^R by swapping A_q's initial and final states and reversing transitions. A_q^R accepts the language L_q^R consisting of the reverse of all paths in L_q.

4. Run A_q^R over all traceroutes, starting from the destinations and proceeding backwards, labeling each (destination d, PoP r) tuple with all the transitions that A_q^R can follow in consuming r starting from d.

5. If a PoP ends up labeled as following a transition in one direction from a source and in the opposite direction from a destination, then the spliced path matches the entire query.

Appendix D.1 presents an example of this sequence of steps.

7 Patching & pruning stale measurements

Previous sections assume the availability of an atlas of historical measurements that serve as the basis for predictions. Resource-constrained VPs do not have enough resources to refresh all measurements regularly, and so routes may change between measurements. Therefore, *Sibyl* needs to balance between discarding old measurements to reduce the risk of out-of-date ones causing faulty predictions, versus using them in predictions to aid coverage (since many old measurements may still be valid). Given that most (*s, d*) pairs use a single route the vast majority of the time [15, 52], we err on the side of retaining routes and apply three mechanisms to infer and remove stale data from *Sibyl* 's atlas. In the first two, a traceroute from s to d reveals a change in one path, and we use the new path to patch other paths either from s or to d that overlapped the old path from s to d.

Traceroute-based destination patching. Since Internet routing is destination-based, if two traceroutes to the same destination (possibly from different sources) converge, we patch the old measurement to match the new measurement from the convergence point to the destination. Flach et al. found that the most common reason for violations of destination-based routing is load balancing [18], which can be filtered using Paris traceroute [5, 66]. Excluding load balancing, that study found only 10% of routers caused IP-level deviations from destination-based routing and only 2% caused AS-level deviations, for reasons including traffic engineering and tunneling. In the future, we could apply these earlier techniques to identify and exclude these exceptions from our patching.

Traceroute-based source patching. Destination-based routing helps us keep the tail of paths collected from constrained VPs up-to-date using measurements from less-constrained VPs such as PlanetLab. To remove stale data from the beginning of paths, we assume a path change observed on the path from a constrained VP to one destination will also impact its paths to other destinations that traverse the path segment that changed. Violations to these assumptions result in incorrect updates to paths. However, a single error is unlikely to impact *Sibyl* 's predictions, as *Sibyl* can continue to make equivalent predictions if it knows other non-stale paths from the VP that traverse a subset of the PoPs that were on the stale segment.

BGP-based destination pruning. Traceroute-based patching still requires issuing a measurement to detect the change. We supplement these approaches with lightweight BGP monitoring, which requires only passive observation of BGP feeds via the following steps. First, we convert all traceroutes in the atlas into AS paths in the following process. (a) We use PeeringDB data to build a database of IXP prefixes and remove these IP addresses from traceroutes. (b) We map remaining IP addresses to the ASes that originate their prefixes. (c) We group addresses into routers using CAIDA's Midar [33] for IP aliasing resolution, assigning a router to an AS only if all its interfaces belong to the same AS. (d) We partition the traceroute into segments in which every router has been assigned an AS (but a segment can contain multiple ASes). Second, we monitor RouteViews and RIPE RIS BGP feeds for BGP changes. When we observe an AS A change its next hop AS to a destination d, we mark as stale any traceroutes that routed via A and its old path to reach d, and we do not use these traceroutes to make predictions. Whereas traceroute-based staleness checks provide a way to patch old measurements, BGP checks on their own do not.

8 Evaluation

We evaluate *Sibyl* from two perspectives. First, we show that *Sibyl* is able to serve queries effectively. On a large set of test queries, it satisfies three-quarters of the queries it could if it had an oracle to provide the result of a traceroute before issuing it. Thereafter, we evaluate individual components of the system in isolation to show that *Sibyl*'s components operate efficiently and make decisions that enable it to make good use of its probing budget.

8.1 Efficiency in serving queries

Datasets and experimental design. To evaluate *Sibyl* end-to-end, we run the system in an offline mode, stubbing out the component that issues traceroutes. We first collect a large set of traceroutes. We then run *Sibyl* as normal, except that, when it decides to issue a traceroute from a vantage point to a destination, instead of issuing a new measurement, it fetches the existing measurement between that pair. Offline analysis allows us to compare choices made by *Sibyl* with other measurements it chose not to issue or that we did not give it access to.

Between January 13–16 2016, we issued traceroutes from 2660 vantage points–2000 RIPE Atlas vantage points, 560 traceroute servers, and 100 PlanetLab sites–to 1000 destinations, chosen at random from a list known to be responsive [25]. Within a platform (Atlas, traceroute servers, or PlanetLab), each vantage point is in a different AS, although these is some overlap across platforms.[10]

In each experiment, we generate a starting corpus of paths that *Sibyl* has access to. The corpus includes all traceroutes from PlanetLab sites, giving it traceroutes to destinations to splice to for predictions. For rate limited platforms (traceroute servers and RIPE Atlas), the corpus starts with 10 randomly chosen measurements from each vantage point, a number previous work shows captures upstream diversity for path prediction [40].

The experiments test how efficiently *Sibyl* can allocate a limited number of additional traceroutes from rate-limited vantage points in order to serve queries. We emulate a series of rounds, with a per round measurement budget and query arrival rate configured per experiment and described with the experiments below. In each round, *Sibyl* decides how to allocate its probing budget to issue traceroutes, we assess how well these traceroutes matched the queries, and then we add these traceroutes to *Sibyl*'s corpus for the next round. Unsatisfied queries do not carry over to the next round.

Existence queries. We first evaluate *Sibyl*'s ability to serve *existence queries*, where the goal is to find one traceroute that matches. To generate test queries, we select one of the traceroutes not (yet) available to *Sibyl* and generate a query that will match it. This way, we know that there is at least one measurement that *Sibyl* could issue to match the query. To create a query, we sampled hops in the path to generate regular expressions according to four different Sibyl use cases (e.g., find paths that traverse a given link toward a destination; more details in Appendix F). We evaluated *Sibyl* with a range of budgets and query volumes, and the results are qualitatively similar, so we present results for just one setting, a per-round probing budget that allowed an average of one traceroute per query.

[10]We worked with the RIPE Atlas staff to gather data faster than their normal rate limits. They allowed this just for the purpose of our evaluation, it required tight coordination between our team and theirs, and it does not appear they will support this on a regular basis.

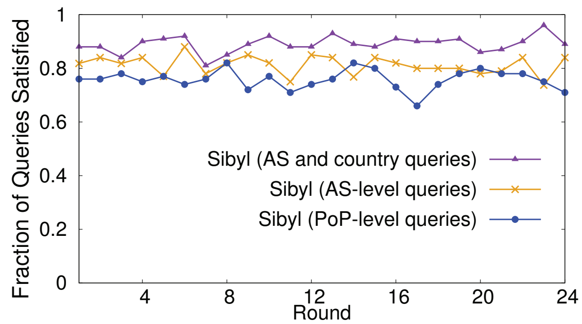


Figure 3: Fraction of queries satisfied when they are specified at different granularities.

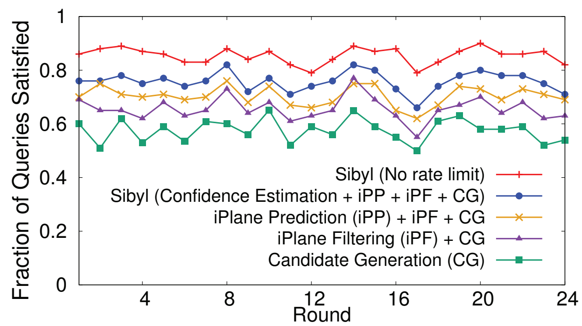


Figure 4: Incremental contribution of *Sibyl*'s components to its ability to satisfy existence queries. Combined, the techniques allow it to allocate budget smartly and approach the performance it would see if it could issue every candidate traceroute it generated (*no rate limit* line).

Performance by query granularity. We used this query generation approach at different granularities (by mapping the traceroutes to PoPs, ASes, and a mix of ASes and countries). Figure 3 shows the fraction of existence queries that *Sibyl* can satisfy at these granularities in each round. At all granularities, *Sibyl* satisfies a high fraction of queries. As expected, the coarser the granularity, the higher the fraction of satisfied queries, from around 75% at the PoP level to 90% at the AS/country level. Also, *Sibyl* is able to efficiently allocate its budget at different granularities, including answering queries that combine ASes and country codes, which may overlap in complex ways (e.g., traverse a link between AT&T and Level3 in the US on the way to Europe).

Incremental contribution of Sibyl components. *Sibyl*'s performance is good across queries of different granularities, and so we focus the rest of our analysis on fine-grained PoP-level queries to stress the system. For PoP-level queries, *Sibyl* allocates its probing budget well, satisfying 32% more queries than a baseline approach that relies only on existing measurements to answer queries. Figure 4 breaks down the incremental benefit of the system's various modules. First, *candidate generation* uses *Sibyl*'s module that splices previously measured paths to identify (s,d) pairs that may match a query (§6). Without access to the rest of *Sibyl*, it then assumes that each of these pairs will indeed match the query and distributes measurements uniformly across queries, picking candidates at random for each query. Second, we add *iPlane filtering* to candidate generation, using iPlane to predict a (more accurate) PoP-level path for each candidate and filtering out candidates whose predicted paths do not match any query. Third, *iPlane prediction* extends iPlane filtering to consider all spliced paths iPlane can generate for a given candidate (s,d) pair (§5.1). Unlike the full system, this comparison point assigns an equal confidence value to every spliced path between the pair when calculating likelihood of matching a query. Finally, the *Sibyl* line adds in confidence (§5.2) into the likelihood estimation to arrive at the full system. As we add the techniques, each contributes to satisfying 5-8% additional queries, justifying their use.

Why does Sibyl fail to satisfy some queries? The two central challenges are (a) limited probing budgets and (b) uncertainty about whether traceroutes will match queries before issuing them. Because the evaluation uses a 1:1 query:budget ratio and all queries have at least one traceroute that satisfies them, without uncertainty, we could satisfy 100% of queries. *Sibyl* could miss satisfying a query either because it failed to generate any candidate traceroutes that, if issued, would satisfied the query, or because it did generate the candidate but calculated that it was unlikely to match the query. The *Sibyl (no rate limit)* teases apart these two causes, as it allows *Sibyl* to issue every candidate traceroute. Without a rate limit, *Sibyl* satisfies 88% of queries (vs. 76% with a 1:1 ratio), suggesting that half of *Sibyl*'s unsatisfied queries were because its corpus of measured paths did not suffice to generate candidates that could satisfy them, and half were instances in which *Sibyl* generated a candidate that would have satisfied the queries, but rated them as having less expected value than other candidates, so did not allocate probing budget to them. This result suggests the potential benefit of future work to improve candidate generation and likelihood estimation.

Can Sibyl efficiently service satisfiable queries in the face of unsatisfiable ones? Our evaluation thus far is on queries that are satisfiable, generated from traceroutes *Sibyl* could choose to issue. Appendix H presents an experiment demonstrating that the fraction of these queries that *Sibyl* can satisfy is robust to the simultaneous introduction of realistic but unsatisfiable queries.

Diversity queries. Next, we assess *Sibyl*'s ability to respond to *diversity queries* by finding a set of paths that match the query in diverse ways. We create diversity queries by supplementing the PoP-level queries from above by asking *Sibyl* to maximize diversity of all wildcard tokens (`.*`) in the regular expression. The diversity utility function for a query awards a unit of utility for each unique PoP in the set of matching traceroutes. We set a 1:4 query:budget ratio (but *Sibyl* may distribute this budget unevenly across queries according to its expected diversity optimization in Eq. 3).

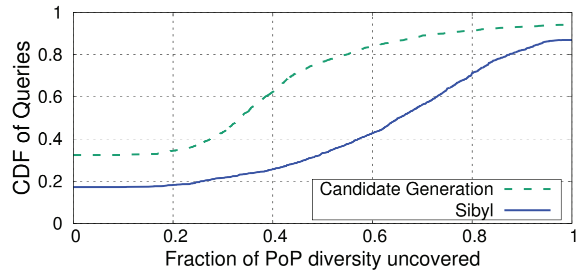


Figure 5: Fraction of distinct PoPs returned for diversity queries, relative to the number of distinct PoPs that could be returned if *Sibyl* had budget to issue every candidate traceroute it considered. Even with limited RIPE Atlas and traceroute server budgets, *Sibyl* usually uncovers a significant fraction of the relevant diversity, providing substantial benefit over an approach that lacks the ability to predict which paths are most likely to provide diversity in order to optimize use of the budget.

Figure 5 depicts the ratio between the number of distinct PoPs on matching paths that *Sibyl* uncovers subject to the rate limit versus the number it could have uncovered with an unlimited budget to probe all candidates it generates. The *candidate generation* baseline—which already uses some of *Sibyl*'s novel functionality to identify promising traceroutes to issue—is unable to find any matching paths for 32% of queries, and it uncovers less than half of the matching path diversity for 75% of queries. In contrast, by optimizing based on its estimation of the expected chance of a given traceroute traversing each PoP while satisfying the query, *Sibyl* satisfies 83% of queries with at least one traceroute, uncovers half the path diversity for 67% of queries, and, for 13% of queries, uses its very limited budget to uncover all of the diversity that was found using unlimited probes.

8.2 Coverage of vantage point platforms

Coverage by vantage point AS. Our ideal is to service *any* routing query, but *Sibyl* is limited by available vantage points. No one platform has achieved overwhelming coverage, and the types of ASes that host vantage points can vary across platforms, so we designed *Sibyl* to accommodate a range of platforms. Figure 6 depicts the locations of the vantage points of different platforms in terms of their coverage of ASes by customer cone sizes [3]; the customer cone of an AS is its customers, its customers' customers, etc.. For example, even though Atlas covers

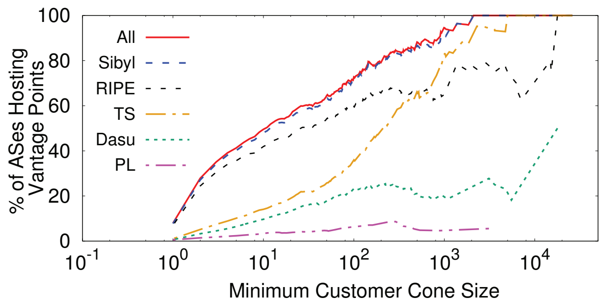


Figure 6: *Sibyl* combines platforms to maximize AS coverage, with especially strong coverage of larger ASes.

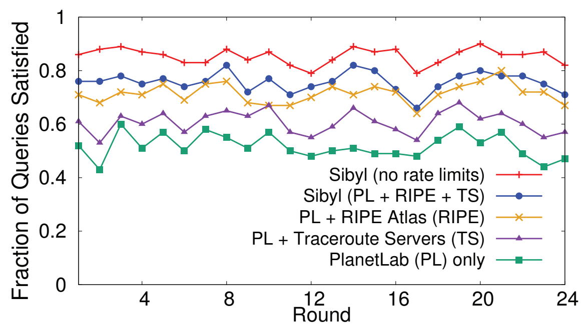


Figure 7: Fraction of existence queries satisfied given different VPs. Combining platforms improves performance.

by far the most ASes (Table 1a), the figure shows that traceroute servers have better presence in large ASes, enabling *Sibyl*'s union of platforms to have presence in all ASes with customer cone size greater than 2000, whereas this coverage is below 80% when combining all other platforms excluding traceroute servers. Based on data provided by the Dasu team, incorporating Dasu would not significantly improve *Sibyl*'s vantage point diversity, although it would increase probing budget. Coverage across AS sizes will improve as existing measurement platforms expand and new ones become available.

Impact of combining vantage point platforms on ability to satisfy queries. We now consider how combining platforms helps *Sibyl* satisfy queries. Using the same queries and probing budget as in Section 8.1, Figure 7 shows the fraction of queries *Sibyl* can satisfy using only PlanetLab, PlanetLab plus traceroute servers, PlanetLab plus RIPE, and all three platforms combined. The *Sibyl* lines from the Figure 4 are the same as the *Sibyl* lines in this graph. We observe that all platforms contribute to the number of satisfied queries. Even though the number of RIPE Atlas vantage points is four times larger than the number of traceroute servers, traceroute servers provide additional diversity and are useful in satisfying queries.

8.3 Accuracy of likelihood estimation

We evaluate *Sibyl*'s likelihood estimation (§5) during our end-to-end evaluation of existence queries (§8.1). Figure 8 shows the fraction of candidates that match a query as a function of estimated likelihood (Eq. (4)) for 20,895 candidates generated for 2273 queries. We bucketed candidates by rounding the estimated likelihood to the closest 0.1. The graph shows high correlation between likelihood and the probability of satisfying a query. Appendix E shows the number of candidates in each bucket.

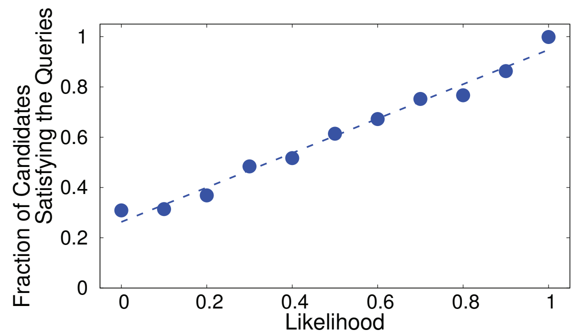


Figure 8: (Bucketed) estimated likelihood of candidates matching queries vs actual fraction matching queries.

8.4 Impact of staleness

Sibyl always issues and returns fresh traceroutes to serve queries, so staleness cannot result in false query matches. Staleness can however lead to wrong predictions and suboptimal allocation of probing budget.

We evaluate the impact of staleness on *Sibyl*'s ability to service queries over time, using weekly traceroute measurements from 1800 RIPE Atlas nodes toward a set of 1000 destinations collected between Jun. 20th and Aug. 20th, 2015. We partition the set of RIPE Atlas VPs in two: we choose 150 VPs at random to use as *constrained* VPs, and use the remaining 1650 VPs as *unconstrained* VPs. As in Section 8.1, we consider existence queries, give *Sibyl* a probing budget of one traceroute per query, and build an initial corpus of traceroute paths that includes 10 measurements from each of the 150 constrained VPs plus all measurements from the 1650 unconstrained VPs.

We test the performance of three different strategies for dealing with stale traceroutes. *Keep last 14 days* uses only paths collected during the last 14 days and discards older paths. *Keep all* accumulates all the traceroute paths collected by *Sibyl* regardless of their age, without applying any sanitation technique to mitigate staleness. *Sibyl(patching and pruning)* also accumulates all the collected traceroutes, but attempts to filter-out stale hops using *Sibyl*'s techniques described in Section 7.

Figure 9 measures *Sibyl*'s ability to service the queries over time. We also show linear fits for each curve to make the trends more clear. *Keep last 14 days* loses path diversity over time, as it only keeps traceroutes from parts of the Internet that were recently targeted by queries, and this narrow focus over time limits its ability to serve

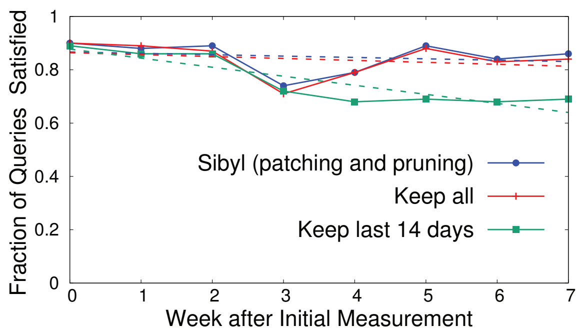


Figure 9: Queries satisfied over time using different approaches to maintain historical traceroutes as a basis for predictions. By pruning/patching paths it detects as stale, *Sibyl* performs much better than an approach that only keeps recent measurements and slightly better than an approach that keeps all measurements. The dip at week 3 is caused by corrupted traceroute files that week.

queries about some other parts of the Internet. *Keep all* maintains diversity, but loses some accuracy due to staleness. *Sibyl (patching and pruning)* strikes a balance, adding measurements to its corpus over time to generate more candidates while minimizing the impact of staleness by patching paths likely to be out of date. Appendix G evaluates the coverage and accuracy of *Sibyl*'s various approaches to patching and pruning stale paths.

9 Related work

Traceroute tool: Van Jacobson's traceroute tool [26] first enabled measurements of the Internet route from the machine on which the tool is executed to any destination. Followup work addressed limitations in the tool. Paris traceroute modified traceroute to account for load balancing [4]; reverse traceroute enabled a source to measure the route back to it from any destination [30]; and researchers assessed how common interpretations of the tool's output can lead to overestimating route changes [44]. *Sibyl* goes beyond enabling measurement of the route from/to a specific source, and instead chooses (source, destination) pairs that it should measure in order to obtain routes that match specified input criteria.

Measurement platforms and systems: Many distributed platforms have been deployed to cater to the needs of researchers and network operators to perform measurements of Internet routing. DIMES [57], Ark [2], public traceroute servers [63], and RIPE Atlas [55] explicitly serve this goal, whereas other platforms such as PlanetLab [53], MobiPerf [72], and Dasu [56] enable traceroutes among several other capabilities. Leveraging the measurement capabilities offered by these platforms, a large number of systems have been developed that rely on making measurements of the Internet for various purposes such as topology discovery [2], fault diagnosis [31, 32, 74], prefix hijack detection [76], etc. In all of these cases, researchers have relied on issuing traceroutes along paths whose routes match particular criteria relevant to their system, but they have only used small numbers of vantage points due to the overhead of incorporating different platforms and the difficulty in discerning which measurements will be most useful. *Sibyl* can enable these prior systems as well as future ones to take advantage of available measurement platforms.

Studies of Internet routing: Several research efforts have studied the temporal stability of Internet routes [15, 52], attempted to infer routing policies [1, 3, 21, 27, 45], and modeled the evolution of the Internet's topology [49]. We similarly model properties of Internet routing, in our case in service of identifying the measurements that are most beneficial for *Sibyl* in serving user queries.

Route prediction: Many prior efforts have developed techniques to predict Internet routing at the AS [43, 54] and PoP [37, 40, 41] levels. However, in our results, even the state of the art prediction techniques offer only 68% accuracy in correctly predicting AS-level paths. Therefore, instead of attempting to predict a single route for any (source, destination) pair, we focus on estimating the probability that the route will match a query; our approach shows significant gains in prediction accuracy.

10 Conclusion

Internet route measurements are crucial to our ability to troubleshoot and understand the Internet, yet our interface to them remains crude: for decades, the only query that has been easy to answer is, *"What is the path from here to there?"* This limitation leads to inefficient approaches and incomplete understanding. We built and evaluated *Sibyl*, a system that accepts regular expression-based queries and returns fresh path measurements matching the queries. To achieve broad coverage, *Sibyl* includes vantage points (such as traceroute servers and RIPE Atlas probes) that are severely rate-limited, which led to the central challenge in building the system—how can it accurately respond even though, for many queries, it will not have issued traceroutes that match in the recent past? Therefore, we designed *Sibyl* to predict which measurements, if issued, will help fulfill queries, in order to efficiently service requests while subscribing to rate limits. Our evaluation shows that these predictions allow *Sibyl* to easily outpace other schemes in its ability to answer questions about Internet routes, performing nearly as well as if it had access to an oracle to tell it which measurements to issue.

Acknowledgements

We would like to thank our shepherd Monia Ghobadi and the anonymous reviewers for their valuable feedback. The RIPE NCC and Comcast supported our use of RIPE Atlas. Conversations with Shaddin Dughmi and Nate Foster shaped *Sibyl*'s optimization and query language, respectively. This work was supported in part by the National Science Foundation grants CNS-1351100 and CNS-1413978, CNPq, and FAPEMIG.

Appendices

A Optimization details

***Sibyl*'s optimization has good greedy performance.** While the constraints in Eq. 1 (§4.2) enforce multiple budgets, we designed them so that each traceroute only counts against one budget, and so the constraints function as a partition matroid [68]. The utility functions we use for *existence queries* and *diversity queries* exhibit diminishing returns as we add to the set of traceroutes to issue, and so the objective function is submodular (essentially, a set function that displays diminishing returns) [47]. The greedy optimization of submodular functions given partition constraints both has a good theoretical lower bound [16, 68][11] and has been frequently observed to be near-optimal in practice.

In addition to the greedy heuristic only being guaranteed to find a solution within a factor of optimal, the optimization problem itself can lose utility compared to a global optimal due to the following factors:

- Candidate generation can miss useful traceroutes, if no previous traceroutes splice to generate the candidate.
- Prediction errors can lead to errors in expected utility.
- Our formulation assumes the correctness of different predictions is independent, but destination-based routing [19] and other factors mean that the correctness of different predictions may be intertwined.

Section 8.1 assessed the first two factors. The third is an interesting future direction for improving predictions.

Utility functions supported by *Sibyl*. Section 4.2 formalizes the utility functions supported by our UI, but, in general, *Sibyl* will work with any utility function f_q for a query q that satisfies the following properties:

- f_q takes a set of traceroutes T and returns a nonnegative value.
- $f_q(T) > 0$ if and only if $\exists t \in T$ that satisfies q.
- Non-decreasing: $f_q(T) \leq f_q(T \cup \{t\})\ \forall T, \forall t$.
- $S \subseteq T \Rightarrow f_q(S \cup \{t\}) - f_q(S) \geq f_q(T \cup \{t\}) - f_q(T)$ $\forall S, \forall T, \forall t$. In other words, adding additional traceroutes provides diminishing returns.
- The expected utility of issuing a set of traceroutes must be computable within *Sibyl*'s prediction framework, in which a traceroute is predicted as a set of PoP-level paths, each with a confidence.

[11] The greedy algorithm we use has an approximation ratio of 0.5. A randomized variant has a ratio of $1 - 1/e \approx 0.63$ [68].

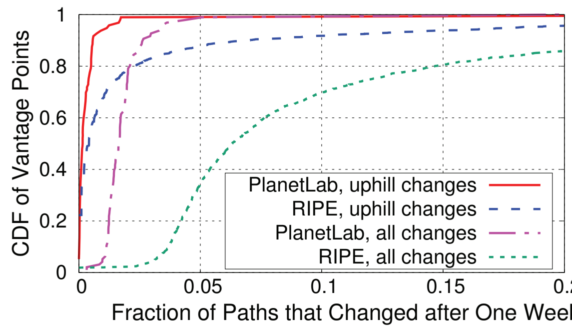


Figure 10: Fraction of paths for which AS-level routes differ in snapshots measured a week apart.

To be computationally efficient, the expected utility function should also be "incrementally computable": if *Sibyl* already calculated $\mathbb{E}[f_q(T)]$, then computing $\mathbb{E}[f_q(T \cup \{t\})]$ takes time proportional to the time to calculate $\mathbb{E}[f_q(\{t\})]$, not proportional to $|T|$.

B Assessing path stability

Section 5 describes how *Sibyl* predicts paths by splicing the small number of traceroutes from resource-constrained vantage points onto traceroutes from less-constrained vantage points to a large number of destinations. We assessed path stability to justify this approach. We probed 1000 prefixes from all PlanetLab sites and from 2000 RIPE Atlas vantage points. We repeated these measurements twice, a week apart. Figure 10 shows, for every vantage point, the fraction of prefixes for which the AS-level routes differ across a week, revealing more RIPE Atlas paths change than PlanetLab paths.

Internet paths are generally considered to have an uphill portion, traversing from customers to providers, followed by a downhill portion from providers to customers, possibly with a peering link in between. Figure 10 also plots the fraction of prefixes that have different AS-level routes in the two snapshots if we consider only the uphill portions of the paths. The uphill paths differ much less frequently than the full paths, implying that most of the differences are on the downhill portions of paths. By combining the uphill (more stable) portion of paths from rate-limited RIPE Atlas/traceroute server vantage points with the downhill portions of (frequently-refreshed) PlanetLab paths, *Sibyl* minimizes the impact of path instability on its predictions.

C Features used by RuleFit

In this section we provide more details on the RuleFit model we train to estimate the correctness of a path prediction (§5.2). To identify important features, we adopted a multi-round refinement process, starting from a large set of features that we reduced each round, retaining features RuleFit found to have predictive power. We describe features retained at the end of this process.

Source path features: The greatest challenge in identifying which spliced path is correct is to pick the correct route out from the source, since Internet routing is predominantly destination-based and the source's portion of the source path has a destination different from the one we are predicting. Traffic engineering practices such as hot and cold potato routing may also exacerbate this issue. We characterize the source path using the following features: the number of PoPs and ASes along the source path, the round-trip latency from the source to the splice point [39], and the degree of the source AS (from CAIDA data [3]). The intuition behind these features is that a prediction is more likely to be correct if the source's part of the path is short (so quickly intersects a known path to the destination) and if the source AS is small (so has fewer routing options we can incorrectly pick). We do not consider the age of measurements as a feature since we take steps to prune out-of-date measurements, as described in Section 7. Most (source, destination) pairs have a path that is prevalent over long time periods [15, 52].

Splice point features: We considered the characteristics of the splice point as additional features such as (i) the type of AS splice point (i.e. educational, transit, access, transit/access, content, enterprise, educational/research, non-profit network, from CAIDA data [10]); and (ii) the business relationship between the AS of the splice point and its neighbors in the predicted path (from CAIDA data [38]). The type and the AS relationships allow RuleFit to learn to favor spliced paths that follow common routing policies, such as valley-free [21].

iPlane-derived features: iPlane picks the correct spliced path more often than not [39], and so, for each spliced path, we calculate the features that iPlane uses, as well as comparisons between that spliced path and the one that iPlane picks (inflation in terms of RTT up to the splice point and in terms of AS- and PoP-level path lengths). We also included the rank order assigned by iPlane to the spliced path, to account for mechanisms added to improve iPlane's prediction accuracy [41].

Spliced Path Feature	Importance
1. PoP-level similarity with the other paths	1
2. PoP-level path length inflation vs iPlane's top-ranked path	.90
3. Total number of PoP splice points	.60
4. Total number of AS splice points	.55
5. AS splice point type	.52
6. AS splice point relationship with neighbors	.49
7. Number of PoPs in iPlane's top-ranked path	.44
Other features	$\leq$ .34

Table 1: Feature importance according to RuleFit.

Spliced path set features: Finally, we compute some features by comparing the spliced path with the other spliced paths from the vantage point to the destination. We used the Jaccard Index to estimate the average similarity between the spliced path and other paths both at the PoP and AS level. We aim to inform RuleFit whether or not the other paths confirm this one. We also include as features the total number of spliced paths and the total number of ASes containing splice points.

Most important features: RuleFit computes the importance of each rule as a function of how often it gets applied and how much it impacts the correctness of the prediction. For each feature, it computes this as the sum of the importance of the rules that use the feature.

Table 1 reports the resulting ordering of features with normalized importance computed by RuleFit. Several features turned out to play an important role in estimating the similarity of a spliced path to the true path. The first, third, and fourth most important features capture how similar the spliced paths are; intuitively, if there are few splicing points and all spliced paths are similar, then there is less diversity and spliced paths are likely similar to the true path. The second and seventh most important feature follows from Internet routing protocols that prefer short paths. The fifth and sixth most important features capture AS routing relationships at the splicing point, which enables RuleFit, e.g., to reduce confidence in splices that violate the valley-free model.

D Examples

D.1 Candidate generation

We first provide an example of how *Sibyl* generates candidate traceroutes to consider issuing (§6). For ease of exposition, assume that an IP address maps to an AS and PoP corresponding to the address's first octet (e.g., 1.0.0.1 is in AS1 and PoP1; 5.0.0.1 is in AS5, PoP5). Assume *Sibyl* has three existing traceroutes it can combine to generate new candidates:

1. 1.0.0.1 (AS1, PoP1), 2.0.0.1 (AS2, PoP2), 3.0.0.1 (AS3, PoP3), 4.0.0.1 (AS4, PoP4), 5.0.0.1 (AS5, PoP5)
2. 6.0.0.1 (AS6, PoP6), 7.0.0.1 (AS7, PoP7), 8.0.0.1 (AS8, PoP8), 9.0.0.1 (AS9, PoP9), 10.0.0.1 (AS10, PoP10)

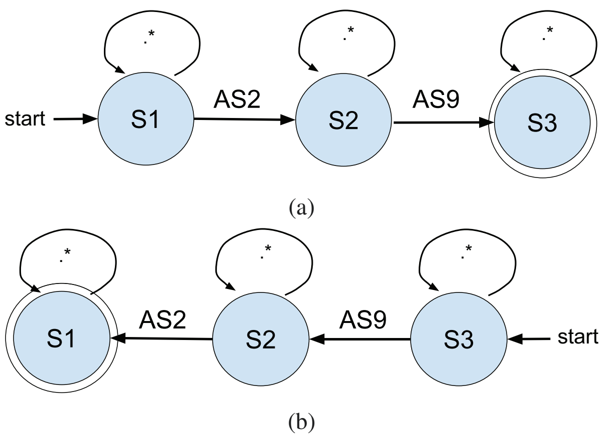


Figure 11: (a) Forward and (b) reverse FSAs corresponding to a query for a traceroute through AS2 and AS9.

3. 11.0.0.1 (AS11, PoP11), 12.0.0.1 (AS12, PoP12), 3.0.0.1 (AS3, PoP3), 9.0.0.1 (AS9, PoP9), 13.0.0.1 (AS13, PoP13)

Say a user issues an existence query: "I want a traceroute that traverses AS2 and AS9, in that order, consecutively or not," is expressed as the following regular expression:

```
^.*AS2.*AS9.*$.
```

This regular expression is then translated to an FSA shown in Figure 11(a). *Sibyl* then runs the FSA over each traceroute, maintaining a record of the transitions in the FSA taken when consuming the PoPs in each of its existing traceroutes, as shown in Table 2(a).

Next, the FSA is reversed (Figure 11(b)), and the reverse FSA is run over the traceroutes from destination to source. Table 2(b) shows the transitions used in this case.

In our example, PoP 3 is labeled with the transition ($S_2 \xrightarrow{.*} S_2$) when the forward FSA is applied on Trace 1, and the same PoP is labeled with the reverse of that transition when the reverse FSA is applied on Trace 3. Hence, *Sibyl* splices Traceroute 1 (PoP1→PoP2→PoP3...) and Traceroute 3 (...PoP3→PoP9→PoP13) at PoP3 to generate a candidate. The candidate pair is constructed from the source of Traceroute 1 and the destination of Traceroute 3 which gives (1.0.0.1, 13.0.0.1).

D.2 Likelihood estimation

We next walk through an example of how *Sibyl* calculates how likely a traceroute is to satisfy a query (Eq. 4 in §5). Assume that, in addition to (1.0.0.1, 13.0.0.1), *Sibyl* also finds (15.0.0.1, 16.0.0.1) as a possible candidate. Once *Sibyl* identifies the candidates for a query, it uses iPlane to generate a set of possible paths for each candidate (source, destination) pair. *Sibyl* uses its RuleFit-trained model to estimate the Jaccard indexes for each spliced path compared to the corresponding (unknown) actual path. It uses these estimates to compute the likelihood of each candidate matching the query. Consider the example paths and estimated Jaccard indexes in Table 3, where we show AS-level paths for ease of exposition.

Forward FSA Transition	PoPs Traversed
$S_1 \xrightarrow{.*} S_1$	Trace 1: PoP 1, 2, 3, 4, 5 Trace 2: PoP 6, 7, 8, 9, 10 Trace 3: PoP 11, 12, 3, 9, 13
$S_1 \xrightarrow{AS2} S_2$	Trace 1: PoP 2
$S_2 \xrightarrow{.*} S_2$	Trace 1: PoP 3, 4, 5
$S_2 \xrightarrow{AS9} S_3$	-
$S_3 \xrightarrow{.*} S_3$	-

(a)

Reverse FSA Transition	PoPs Traversed
$S_3 \xleftarrow{.*} S_3$	Trace 1: PoP 5, 4, 3, 2, 1 Trace 2: PoP 10, 9, 8, 7, 6 Trace 3: PoP 13, 9, 3, 12, 11
$S_2 \xleftarrow{AS9} S_3$	Trace 2: PoP 9 Trace 3: PoP 9
$S_2 \xleftarrow{.*} S_2$	Trace 2: PoP 8, 7, 6 Trace 3: PoP 3, 12, 11
$S_1 \xleftarrow{AS2} S_2$	-
$S_1 \xleftarrow{.*} S_1$	-

(b)

Table 2: Transitions activated by PoPs in each traceroute on the (a) forward and (b) reverse FSAs.

Candidate	Splice	Jaccard	Predicted AS-Level Path
(1.0.0.1, 13.0.0.1)	A	0.7	AS1 AS2 AS3 AS9 AS13
	B	0.5	AS1 AS20 AS21 AS9 AS13
(15.0.0.1, 16.0.0.1)	C	0.6	AS15 AS2 AS3 AS9 AS16
	D	0.6	AS15 AS2 AS4 AS9 AS16

Table 3: All spliced paths for each candidate and their RuleFit-predicted Jaccard indexes.

For the candidate pair (1.0.0.1, 13.0.0.1), *Sibyl* estimated that spliced path A is more likely to be correct than spliced path B (0.7 vs 0.5), which (via §5.2) normalize to $0.41 = 0.7 \times 0.7/(0.7+0.5)$ and $0.29 = 0.7 \times 0.5/(0.7+0.5)$. Spliced path A matches the user's query, whereas B does not traverse AS2. The final likelihood that candidate (1.0.0.1, 13.0.0.1) matches the query is 0.41, from Eq. 4.

For (15.0.0.1, 20.0.0.1), spliced paths C and D have lower estimated Jaccard indexes than spliced path A, but both satisfy the user's query. These spliced paths result in a likelihood of matching the query equal to $0.6 = 0.6 \times 0.6/(0.6+0.6) + 0.6 \times 0.6/(0.6+0.6)$, making (15.0.0.1, 16.0.0.1) a stronger candidate to satisfy the user's query.

D.3 Diversity queries

To illustrate the usefulness of diversity queries, we will use an example of *Sibyl* finding diverse AS paths that a system such as iSpy [76] can use to monitor for prefix hijacks in BGP. Since issuing traceroutes from all vantage points is not feasible, we want *Sibyl* to find a set of vantage points to use that maximizes AS path coverage to a prefix 204.57.0.0/21. Since we want to diversify over AS, we build the following query:

```
^{.*}-204.57.0.0/21$ by AS
```

For simplicity of exposition, we assume *Sibyl* predicts a single path for each candidate and has complete confidence in all predictions, removing the probabilistic expected value calculation of Eq. 3. *Sibyl* predicts traceroutes with the following AS paths toward 204.57.0.0/21:

1. AS3356, AS209, AS2722, AS47
2. AS1299, AS10490, AS2722, AS47
3. AS3257, AS209, AS2722, AS47
4. AS1273, AS209, AS2722, AS47
5. AS6939, AS226, AS2914, AS2497, AS47
6. AS3257, AS209, AS2722, AS47
7. AS701, AS2914, AS209, AS2722, AS47

Sibyl greedily selects traceroutes that offer the highest diversity utility first. The greedy selection starts out with an empty AS set. Traceroutes are then selected based on how many new ASes a path is predicted to add. In the above example, Traceroute 5 is selected first since it has a utility of 5 ASes (contains 5 new ASes). Traceroute 7 would be greedily selected next since it has a marginal utility of 4 ASes. The current AS set is now:

```
AS6939, AS226, AS2914, AS2497, AS47, AS701,
AS2914, AS209, AS2722
```

Of the remaining traceroutes, Traceroutes 1, 3, 4, and 6 each offer only one new AS compared to the above set, whereas Traceroute 2 has 2 new ASes. Hence, Traceroute 2 is selected. In subsequent rounds, Traceroutes 3 and 4 would be selected if budget allowed, but Traceroute 6 would not be since it adds no new ASes.

E Evaluation of RuleFit model

Section 8.3 showed that *Sibyl*'s estimates of how likely a candidate traceroute is to satisfy a query are accurate enough to use as expected utilities. In this section we look at the distribution of likelihood values across candidates and at the accuracy of the Jaccard index estimates (§5.2) that *Sibyl* uses to calculate the likelihoods.

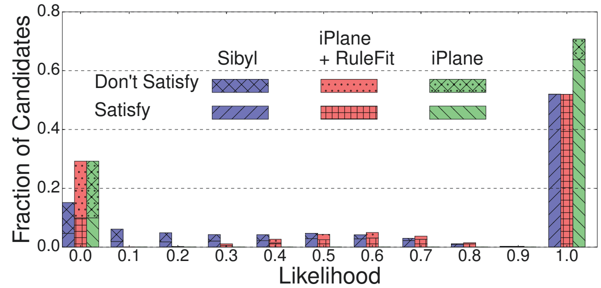


Figure 12: Distribution of candidates by likelihood.

Distribution of likelihood estimates. Figure 12 partitions the range of likelihood values ([0, 1]) into 11 buckets ([0, 0.05], [0.05, 0.15], ..., [0.95, 1]), and shows the number of candidates in each bucket, broken down by whether the candidates satisfy their queries or not. We also add two comparison points: (1) iPlane: iPlane provides a single predicted path for a candidate and does not have a notion of varying confidence [40], and so we assign a candidate a likelihood of 1 if iPlane's prediction matches the query and 0 if it does not. (2) iPlane with confidence ranking: for iPlane predictions that match their queries, we extend iPlane by assigning a likelihood equal to our RuleFit model's estimated confidence in the prediction. As seen in the graph, *Sibyl*'s likelihood estimation provides benefit over iPlane. In the bucket of likelihood [0.95, 1], *Sibyl* only includes candidates that satisfy queries, while iPlane includes some candidates that do not satisfy queries. *Sibyl* only assigns a likelihood of 1 to a candidate when all its spliced paths satisfy the query *and* RuleFit rates it high confidence. *Sibyl* also provides benefit over iPlane by removing some candidates that can satisfy queries from the [0, 0.05] bucket. This improvement comes at the cost of moving some candidates that do not satisfy queries from the [0, 0.05] likelihood bucket to other low-likelihood buckets, which we consider to be acceptable since *Sibyl* gives low priority to issue measurements for candidates with low likelihood.

Together, Figures 8 and 12 show that *Sibyl* computes likelihoods that can reasonably reflect the probability of matching a query, and it assigns most candidates either very high or very low likelihood values, enabling it to distinguish between candidates that it should or should not select to satisfy queries.

Accuracy of confidence values. Figure 13 evaluates RuleFit's capability to predict the PoP-level similarity of spliced paths to the actual paths they are predicting, which it does without access to the actual paths. We use RuleFit to estimate the Jaccard index for 4 million spliced paths (not included in the training set), then calculate the actual Jaccard index by comparing the spliced path to

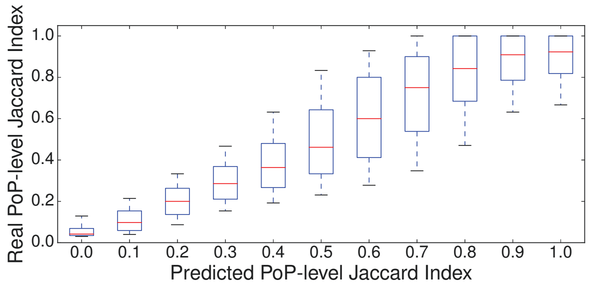


Figure 13: Spliced paths' predicted vs actual Jaccard index with respect to the actual path.

the actual path. We group the spliced paths by their predicted Jaccard index and show the 10th, 25th, 50th, 75th, and 90th percentiles of the true Jaccard values for each group. We see that our estimated Jaccard indexes are well correlated to the true Jaccard values.

F Queries used in evaluation

We used several types of queries when evaluating *Sibyl* (§8.1). For each traceroute that *Sibyl* does not have access to, we generate all possible queries that the traceroute matches for the following query types:

1. `^.*A.*D$` Traverse A on the way to destination D.
2. `^[^A]+A.*D$` Traverse, but do not start at, A on the way to destination D.
3. `^.*AB.*D$` Traverse link A-B on the way to destination D.
4. `^.*A.*B.*C.*$` Traverse A, B, and C in sequence.

Among all possible queries of these types, our evaluation randomly selected an equal number of each type. Queries of types 1 and 2 represent queries reverse traceroute uses as part of its measurements [30]. Query type 3 represents queries operators might ask when troubleshooting performance problems towards a destination, to assess paths that use a particular link. All three look for routes toward a destination *D* traversing a specific network region. Query type 4 does not specify a destination and could be used to study inter-AS routing policing and business relationships [38] or to look for routes that take long detours in between two nearby hops (e.g., [22]).

G Efficacy of staleness patching & pruning

Section 8.4 evaluated the impact of staleness on *Sibyl*'s end-to-end ability to satisfy queries, showing that its techniques for dealing with stale measurements allow it to outperform techniques that either keep or discard all old measurements. In this section we evaluate the accuracy and coverage of its techniques (§7) in isolation.

Traceroute-based source/destination patching. First, we validate *Sibyl*'s approaches of using a path change observed on one path to update other previously measured paths (from the same source or to the same destination) that traverse the path segment that changed. For this, we issued traceroutes from all PlanetLab sites to 150K prefixes on Dec. 5 and on Dec. 6 2014. We calculated the probability that paths undergo identical path changes, given their Dec. 5 routes traversed a shared segment that changed in one of the routes on Dec. 6. For 65% of path changes, all paths experience an identical change. Results on measurements one week apart are similar.

BGP-based destination pruning. We evaluate *Sibyl*'s BGP-based filtering of stale paths on RIPE Atlas measurements gathered between July 2 and August 27, 2015, using daily BGP paths from BGPStream [50]. We mapped the traceroute destinations to the longest prefix in the collected BGP data, excluding prefixes longer than 24.

First, for coverage, of the (AS, destination) pairs in our traceroutes, only 5% of the ASes appear in BGP feed paths towards the destinations, demonstrating both the superior coverage of our traceroute vantage points compared to available BGP feeds and also a limitation with BGP-based filtering. However, 84% of our traceroutes include at least one pair seen in the BGP feeds. Of the pairs seen in both data sources, the AS paths are the same in 57% of cases. The other 43% reflect a mix of large ASes using multiple paths, of errors in translating traceroutes to AS paths, and of misalignment in time because we do not have an exact timestamp for the traceroutes.

Second, we evaluate the accuracy of BGP-based filtering. Every time we refreshed an Atlas traceroute to a destination *d*, for every AS *A* on the traceroute, we check three conditions. 1:(*BGP-change*) Is the BGP path to *d* different than it was at the time of the original traceroute to *d*? 2:(*TR-change*) Did *A*'s traceroute AS path change between the two measurements? 3:(*TR-match*) Did *A*'s original traceroute AS path match *A*'s BGP path at the time it was issued? Comparing every instance of *BGP-change* with the subset that are also *TR-change*, 72% of BGP changes were also reflected in traceroutes. Comparing instances that are both *BGP-change* and *TR-change* with the subset that are also *TR-match*, the percentage increases to 77% if we add the stricter condition that the BGP and traceroute paths matched to begin with. Overall, BGP monitoring prunes 9% of the traceroute changes if we require the *TR-match* check and 13.8% if we do not.

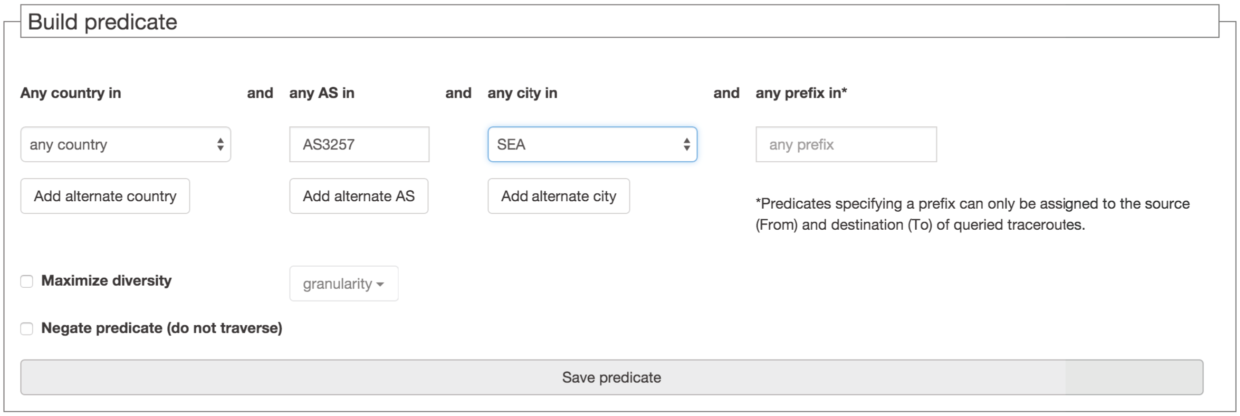
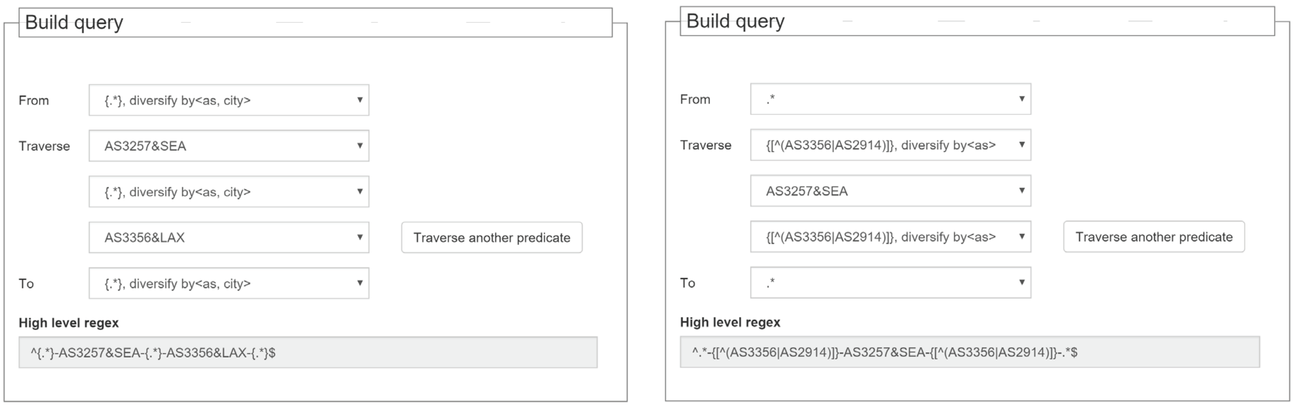


Figure 14: Screenshot of *Sibyl*'s interface to build predicates.



(a) All paths traversing GTT in Seattle then Level3 in Los Angeles. (b) Paths through GTT peers other than NTT and Level3 in Seattle.

Figure 15: Screenshots of example queries from Section 3.2, built by composing predicates such as the one in Fig. 14.

H Unsatisfiable Queries

Section 8 uses queries that are satisfiable–since we generate them from traceroutes *Sibyl* could choose to issue. Here we evaluate whether *Sibyl* can avoid wasting budget on queries it has no hope of satisfying, to avoid having them impede its performance on queries it can satisfy. We generated sensible unsatisfiable queries by generating existence queries as in Section 8.1, removing *Sibyl*'s access to 10% of the RIPE Atlas and traceroute server VPs, then identifying queries that can only be satisfied by measurements from the removed vantage points.[12]

In our experiment, we add unsatisfiable queries to the set of queries submitted to *Sibyl* while keeping the probing budget fixed. As we move from all queries satisfiable to an even mix of satisfiable and unsatisfiable, *Sibyl* still matches just as many queries, 76% on average as in Figure 3. It does generate some candidates to consider issuing for some of the unsatisfiable queries. However, *Sibyl*'s ability to rate the likelihood of matching allows it to prioritize measurements with high expected utility, concentrating budget on queries that can be satisfied. In practice, it could inform a user when it had no candidates likely to match the user's query.

To verify that this result was because the system assessed that its vantage points were unable to satisfy the queries, not because it found the queries to be unsatisfiable in general, we reintroduced the 10% of vantage points back into the system and ran it with just the previously unsatisfiable queries. *Sibyl* satisfied an average of 48% of the queries, suggesting that they are hard but not impossible when suitable vantage points are available. When we then combined the two batches of queries, increasing the absolute traceroute budget to maintain the 1:1 query:budget ratio, *Sibyl* satisfied an average of 58% of queries, balancing the budget well across the two sets to nearly equal the $(76+48)/2 = 62\%$ average performance when it could dedicate itself to one set.

[12]We do not include trivial unsatisfiable queries such as asking for paths originated from ASes hosting the removed VPs.

I *Sibyl*'s query interface

We built a web-based user interface to guide users in specifying queries. Figure 14 presents a screenshot of the widgets used to build a predicate. Users can build a broad class of predicates that accept (or, via negation, reject) a user-specified set of values (e.g., particular cities or ASes) at any or all granularities. Users can then build queries by specifying a sequence of predicates they want paths to traverse. Figures 15(a) and (b) show examples of simplified versions of queries from Section 3.2.

References

[1] Anwar, R., Niaz, H., Choffnes, D., Cunha, I., Gill, P., and Katz-Bassett, E. Investigating interdomain routing policies in the wild. In *IMC* (2015).

[2] Archipelago measurement infrastructure. `http://www.caida.org/projects/ark/`.

[3] AS Rank: AS Ranking. `http://as-rank.caida.org/`.

[4] Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., and Teixeira, R. Avoiding traceroute anomalies with Paris traceroute. In *IMC* (2006).

[5] Augustin, B., Friedman, T., and Teixeira, R. Measuring multipath routing in the Internet. *IEEE/ACM TON* (2011).

[6] Augustin, B., Krishnamurthy, B., and Willinge, W. IXPs: mapped? In *IMC* (2009).

[7] Banerjee, R., Chiang, L., Mishra, A., Razaghpanah, A., Sekar, V., Choi, Y., and Gill, P. Internet outages, the eyewitness accounts: Analysis of the outages mailing list. In *PAM* (2015).

[8] Bourgeau, T., Augé, J., and Friedman, T. TopHat: supporting experiments through measurement infrastructure federation. In *TridentCom* (2010).

[9] Bu, T., Duffield, N., Presti, F. L., and Towsley, D. Network tomography on general topologies. In *SIGMETRICS* (2002).

[10] CAIDA UCSD AS classification dataset. `http://www.caida.org/data/as_classification.xml`.

[11] Chang, D., Govindan, R., and Heidemann, J. The temporal and topological characteristics of BGP path changes. In *ICNP* (2003).

[12] Chen, K., Choffnes, D. R., Potharaju, R., Chen, Y., Bustamante, F. E., Pei, D., and Zhao, Y. Where the sidewalk ends: Extending the Internet AS graph using traceroutes from P2P users. In *CoNEXT* (2009).

[13] Chiu, Y.-C., Schlinker, B., Radhakrishnan, A. B., Katz-Bassett, E., and Govindan, R. Are we one hop away from a better Internet? In *IMC* (2015).

[14] Coates, M., Hero, A., Nowak, R., and Yu, B. Internet tomography. *IEEE Signal Processing Magazine* (2002).

[15] Cunha, I., Teixeira, R., Veitch, D., and Diot, C. Predicting and tracking Internet path changes. In *SIGCOMM* (2011).

[16] Dughmi, S. Submodular functions: Extensions, distributions, and algorithms. A survey (PhD qualifying exam report). Tech. rep., Stanford, 2009.

[17] Fanou, R., Francois, P., and Aben, E. On the diversity of interdomain routing in Africa. In *PAM* (2015).

[18] Flach, T., Katz-Bassett, E., and Govindan, R. Quantifying violations of destination-based forwarding on the Internet. In *IMC* (2012).

[19] Flach, T., Katz-Bassett, E., and Govindan, R. Quantifying violations of destination-based forwarding on the Internet. In *IMC* (November 2012).

[20] Friedman, J. H., and Popescu, B. E. Predictive learning via rule ensembles. *The Annals of Applied Statistics* (2008), 916–954.

[21] Gao, L., and Rexford, J. Stable Internet routing without global coordination. In *SIGMETRICS* (2000).

[22] Gupta, A., Calder, M., Feamster, N., Chetty, M., Calandro, E., and Katz-Bassett, E. Peering at the Internet's frontier: A first look at ISP interconnectivity in Africa. In *PAM* (2014).

[23] Hengartner, U., Moon, S., Mortier, R., and Diot, C. Detection and analysis of routing loops in packet traces. In *IMW* (2002).

[24] Hiran, R., Carlsson, N., and Gill, P. Characterizing large-scale routing anomalies: a case study of the China Telecom incident. In *PAM* (2013).

[25] Internet address hitlist dataset, PREDICT ID USC LANDER internet_address_hitlist_it28wbeta20090914. `http://www.isi.edu/ant/lander`.

[26] Jacobson, V. Traceroute. `ftp://ftp.ee.lbl.gov/traceroute.tar.gz`.

[27] Javed, U., Cunha, I., Choffnes, D. R., Katz-Bassett, E., Anderson, T., and Krishnamurthy, A. PoiRoot: Investigating the root cause of interdomain path changes. In *SIGCOMM* (2013).

[28] John, J. P., Katz-Bassett, E., Krishnamurthy, A., Anderson, T., and Venkataramani, A. Consensus routing : The Internet as a distributed system. In *NSDI* (2008).

[29] Katz-Bassett, E., John, J. P., Krishnamurthy, A., Wetherall, D., Anderson, T., and Chawathe, Y. Towards IP geolocation using delay and topology measurements. In *IMC* (2006).

[30] Katz-Bassett, E., Madhyastha, H. V., Adhikari, V., Scott, C., Sherry, J., van Wessep, P., Anderson, T., and Krishnamurthy, A. Reverse traceroute. In *NSDI* (2010).

[31] Katz-Bassett, E., Madhyastha, H. V., John, J. P., Krishnamurthy, A., Wetherall, D., and Anderson, T. Studying black holes in the Internet with Hubble. In *NSDI* (2008).

[32] Katz-Bassett, E., Scott, C., Choffnes, D. R., Cunha, I., Valancius, V., Feamster, N., Madhyastha, H. V., Anderson, T. E., and Krishnamurthy, A. LIFEGUARD: Practical repair of persistent route failures. In *SIGCOMM* (2012).

[33] Keys, K., Hyun, Y., Luckie, M., and kc claffy. Internet-scale IPv4 alias resolution with MIDAR: System architecture. Tech. rep., Cooperative Association for Internet Data Analysis (CAIDA), 2011.

[34] Krishnan, R., Madhyastha, H. V., Srinivasan, S., Jain, S., Krishnamurthy, A., Anderson, T., and Gao, J. Moving beyond end-to-end path information to optimize CDN performance. In *IMC* (2009).

[35] Kushman, N., Kandula, S., and Katabi, D. Can you hear me now?! It must be BGP. *SIGCOMM CCR* (2007).

[36] Labovitz, C., Ahuja, A., Bose, A., and Jahanian, F. Delayed Internet routing convergence. In *SIGCOMM* (2000).

[37] Lee, D., Jang, K., Lee, C., Iannaccone, G., and Moon, S. Scalable and systematic Internet-wide path and delay estimation from existing measurements. *Computer Networks* (2011).

[38] Luckie, M., Huffaker, B., Dhamdhere, A., Giotsas, V., and Claffy, K. AS relationships, customer cones, and validation. In *IMC* (2013).

[39] Madhyastha, H. V., Anderson, T., Krishnamurthy, A., Spring, N., and Venkataramani, A. A structural approach to latency prediction. In *IMC* (2006).

[40] Madhyastha, H. V., Isdal, T., Piatek, M., Dixon, C., Anderson, T., Krishnamurthy, A., and Venkataramani, A. iPlane: An information plane for distributed services. In *OSDI* (2006).

[41] Madhyastha, H. V., Katz-Bassett, E., Anderson, T., Krishnamurthy, A., and Venkataramani, A. iPlane Nano: Path prediction for peer-to-peer applications. In *NSDI* (2009).

[42] Mahajan, R., Zhang, M., Poole, L., and Pai, V. Uncovering performance differences among backbone ISPs with Netdiff. In *NSDI* (2008).

[43] Mao, Z. M., Qiu, L., Wang, J., and Zhang, Y. On AS-level path inference. In *SIGMETRICS* (2005).

[44] Marchetta, P., Persico, V., Katz-Bassett, E., and Pescape, A. Don't trust traceroute (completely). In *CoNEXT Student Workshop* (2013).

[45] Mühlbauer, W., Uhlig, S., Fu, B., Meulle, M., and Maennel, O. In search for an appropriate granularity to model routing policies. In *SIGCOMM* (2007).

[46] Narayana, S., Tahmasbi, M., Rexford, J., and Walker, D. Compiling path queries. In *NSDI* (2016).

[47] Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. An analysis of approximations for maximizing submodular set functions I. *Mathematical Programming 14* (1978), 265–294.

[48] Oliveira, R., Pei, D., Willinger, W., Zhang, B., and Zhang, L. In search of the elusive ground truth: The Internet's AS-level connectivity structure. In *SIGMETRICS* (2008).

[49] Oliveira, R. V., Zhang, B., and Zhang, L. Observing the evolution of Internet AS topology. In *SIGCOMM* (2007).

[50] Orsini, C., King, A., and Dainotti, A. BGPStream: a software framework for live and historical BGP data analysis. Tech. rep., Center for Applied Internet Data Analysis (CAIDA), Oct 2015.

[51] Outages mailing list. `http://isotf.org/mailman/listinfo/outages`.

[52] Paxson, V. End-to-end routing behavior in the Internet. *IEEE/ACM TON* (1997).

[53] PlanetLab website. `http://www.planet-lab.org`.

[54] Qiu, J., and Gao, L. AS path inference by exploiting known AS paths. In *GLOBECOM* (2006).

[55] RIPE Atlas. `https://atlas.ripe.net/`.

[56] Sánchez, M. A., Otto, J. S., Bischof, Z. S., Choffnes, D. R., Bustamante, F. E., Krishnamurthy, B., and Willinger, W. A measurement experimentation platform at the Internet's edge. *IEEE/ACM TON* (2014).

[57] Shavitt, Y., and Shir, E. DIMES: Let the Internet measure itself. *SIGCOMM CCR* (2005).

[58] Spring, N., Mahajan, R., and Anderson, T. Quantifying the causes of path inflation. In *SIGCOMM* (2003).

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

[60] Sridharan, A., Moon, S. B., and Diot, C. On the correlation between route dynamics and routing loops. In *IMC* (2003).

[61] Steenbergen, R. A. A practical guide to (correctly) troubleshooting with traceroute. In *NANOG 45* (2009). `http://www.nanog.org/meetings/nanog45/presentations/Sunday/RAS_traceroute_N45.pdf`.

[62] Sundaresan, S., de Donato, W., Feamster, N., Teixeira, R., Crawford, S., and Pescape, A. Broadband Internet performance: A view from the gateway. In *SIGCOMM* (2011).

[63] Traceroute.org. `http://www.traceroute.org/`.

[64] Trammell, B., Casas, P., Rossi, D., Bar, A., Houidi, Z., Leontiadis, I., Szemethy, T., and Mellia, M. mPlane: an intelligent measurement plane for the Internet. *Communications Magazine, IEEE 52*, 5 (2014), 148–156.

[65] Veanes, M. Applications of symbolic finite automata. In *Implementation and Application of Automata*. Springer, 2013.

[66] Veitch, D., Augustin, B., Teixeira, R., and Friedman, T. Failure control in multipath route tracing. In *INFOCOM* (2009).

[67] Vissicchio, S., Tilmans, O., Vanbever, L., and Rexford, J. Central control over distributed routing. In *SIGCOMM* (2015).

[68] Vondrak, J. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC* (2008).

[69] Wang, F., Mao, Z. M., Wang, J., Gao, L., and Bush, R. A measurement study on the impact of routing events on end-to-end Internet path performance. In *SIGCOMM* (2006).

[70] Wang, Z., Qian, Z., Xu, Q., Mao, Z., and Zhang, M. An untold story of middleboxes in cellular networks. *SIGCOMM CCR* (2011).

[71] Wong, B., Stoyanov, I., and Sirer, E. G. Octant: A comprehensive framework for the geolocalization of Internet hosts. In *NSDI* (2007).

[72] Xu, Q., Huang, J., Wang, Z., Qian, F., Gerber, A., and Mao, Z. M. Cellular data network infrastructure characterization and implication on mobile content placement. In *SIGMETRICS* (2011).

[73] Zarifis, K., Flach, T., Nori, S., Choffnes, D., Govindan, R., Katz-Bassett, E., Mao, Z. M., and Welsh, M. Diagnosing path inflation of mobile client traffic. In *PAM* (2014).

[74] Zhang, M., Zhang, C., Pai, V., Peterson, L., and Wang, R. PlanetSeer: Internet path failure monitoring and characterization in wide-area services. In *OSDI* (2004).

[75] Zhang, Y., Mao, Z. M., and Zhang, M. Effective diagnosis of routing disruptions from end systems. In *NSDI* (2008).

[76] Zhang, Z., Zhang, Y., Hu, Y. C., Mao, Z. M., and Bush, R. iSpy: detecting IP prefix hijacking on my own. In *SIGCOMM* (2008).