Predicting and Tracking Internet Path Changes

Italo Cunha, Renata Teixeira, Darryl Veitch, Christophe Diot

▶ **To cite this version:**

Italo Cunha, Renata Teixeira, Darryl Veitch, Christophe Diot. Predicting and Tracking Internet Path Changes. ACM SIGCOMM, Aug 2011, Toronto, Canada. pp.122-133, 10.1145/2018436.2018451 . hal-00835405

HAL Id: hal-00835405

https://hal.sorbonne-universite.fr/hal-00835405

Submitted on 18 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Predicting and Tracking Internet Path Changes

Ítalo Cunha[†‡] Renata Teixeira[⋆‡] Darryl Veitch[♮] Christophe Diot[†]
[†]Technicolor [‡]UPMC Sorbonne Universités [⋆]CNRS
[♮]Dept. of Electrical and Electronic Eng., University of Melbourne
{italo.cunha, christophe.diot}@technicolor.com renata.teixeira@lip6.fr dveitch@unimelb.edu.au

ABSTRACT

This paper investigates to what extent it is possible to use traceroute-style probing for accurately tracking Internet path changes. When the number of paths is large, the usual traceroute based approach misses many path changes because it probes all paths equally. Based on empirical observations, we argue that monitors can optimize probing according to the likelihood of path changes. We design a simple predictor of path changes using a nearest-neighbor model. Although predicting path changes is not very accurate, we show that it can be used to improve probe targeting. Our path tracking method, called DTRACK, detects up to two times more path changes than traditional probing, with lower detection delay, as well as providing complete load-balancer information.

Categories and Subject Descriptors

C.2.3 [**Computer Systems Organization**]: Computer Communication Networks—*Network Operations—Network Monitoring*;
C.4 [**Computer Systems Organization**]: Performance of Systems—*Measurement Techniques*

General Terms

Design, Experimentation, Measurement

Keywords

Topology Mapping, Tracking, Prediction, Path Changes

1. INTRODUCTION

Systems that detect Internet faults [9, 15] or prefix hijacks [34] require frequent measurements of Internet paths, often taken with traceroute. Topology mapping techniques periodically issue traceroutes and then combine observed links into a topology [14, 17, 25]. Content distribution networks continuously monitor paths and their properties to select the "best" content server for user requests [10]. Similarly, overlay networks monitor IP paths to select the best overlay routing [1]. In all these examples, a source host issues traceroutes to a large number of destinations with the hope of tracking paths as they change.

[♮]The work was done while Darryl Veitch was visiting Technicolor.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCOMM'11, August 15–19, 2011, Toronto, Ontario, Canada.
Copyright 2011 ACM 978-1-4503-0797-0/11/08 ...$10.00.

The classical approach of probing all paths equally, however, has practical limits. First, sources have a limited probing capacity (constrained by source link capacity and CPU utilization), which prevents them from issuing traceroutes frequently enough to observe changes on all paths. Second, Internet paths are often stable [8, 12, 24], so probing all paths at the same frequency wastes probes on paths that are not changing while missing changes in other paths. Finally, many paths today traverse routers that perform load balancing [2]. Load balancing creates multiple simultaneous paths from a source to a given destination. Ignoring load balancing leads to traceroute errors and misinterpretation of path changes [8]. Accurately discovering all paths under load balancing, however, requires even more probes [31].

This paper shows that a monitor can optimize probing to track path changes more efficiently than classical probing given the same probing capacity. We develop DTRACK, a system that separates the tracking of path changes into two tasks: *path change detection* and *path remapping*. DTRACK only remaps (measures again the hops of) a path once a change is detected. Path remapping uses Paris traceroute's multipath detection algorithm (MDA) [31], because it accurately discovers all paths under load balancing. The key novelty of this paper is to design a probing strategy that predicts the paths that are more likely to change and adapts the probing frequency accordingly. We make two main contributions:

Investigate the predictability of path changes. We use traceroute measurements from 70 PlanetLab nodes to train models of path changes. We use RuleFit [13], a supervised machine learning technique, to identify the features that help predict path changes and to act as a benchmark (Sec. 3). RuleFit is too complex to be used online. Hence, we develop a model to predict path changes, called NN4, based on the K nearest-neighbor scheme, which can be implemented efficiently and is as accurate as RuleFit (Sec. 4). We find that prediction is difficult. Even though NN4 is not highly accurate, it is effective for tracking path changes, as it can predict paths that are more likely to change in the short term.

A probing strategy to track path changes. (Sec. 5) DTRACK adapts path sampling rates to minimize the number of missed changes based on NN4's predictions. For each path, it sends a single probe per sample in a temporally striped form of traceroute. We evaluate DTRACK with trace-driven simulations and show that, for the probing budget used by DIMES [25], DTRACK misses 73% fewer path changes than the state-of-the-art approach and detects 93% of the path changes in the traces.

DTRACK tracks path changes more accurately than previous techniques. A closer look at path changes should enable research on the fine-grained dynamics of Internet topology as well as ensure that failure detection systems, content distribution, and overlay networks have up-to-date information on network paths.

2. DEFINITIONS, DATA, AND METRICS

In this section we define key underlying concepts and present the dataset we use. We establish the low-level path prediction goals which underlie our approach to path tracking, and then present a spectrum of candidate path features to be exploited to that end.

2.1 Virtual paths and routes

Following Paxson [24], we use *virtual path* to refer to the connectivity between a fixed source (here a monitor) and a destination d. At any given time, a virtual path is realized by a route which we call the *current route*. Since routing changes occur, a virtual path can be thought of as a continuous time process $P(t)$ which jumps between different routes over time.

A *route* can be *simple*, consisting of a sequence of IP interfaces from the monitor toward d, or *branched*, when one or more *load balancing* routers are present, giving rise to multiple overlapping sequences (branched routes are called "multi-paths" in [31]). A route can be a sequence that terminates before reaching d. This can occur due to routing changes (e.g., transient loops), or the absence of a complete route to the destination. By *route length* we mean the length of its longest sequence, and we define the *edit distance* between two routes as the minimum number of interface insertions, deletions, and substitutions needed to make the IP interface sequences of each route identical. In the same way we can define *AS length* and *AS edit distance* for a general route.

Let a virtual path P be realized by route r at time t, i.e., $P(t) = r$. Suppose that the path will next jump to a new route at time t_d, and last jumped to the current route r at time t_b. Then the *age* of this instance of route r is $A(r) = t - t_b$, its *residual life* is $L(r) = t_d - t$, and its *duration* is $D(r) = A(r) + L(r) = t_d - t_b$. Typically, as we have just done, we will write $A(r)$ instead of $A(P(t))$, and so on, when the context makes the virtual path, time instant, and hence route instance, clear.

In practice we measure virtual paths only at discrete times, resulting effectively in a sampling of the process $P(t)$. A change can be detected whenever two consecutive path measurements differ, however the full details of the evolution of the virtual path between these samples is unknown, and many changes may be missed. Unless stated otherwise, by *(virtual) path change* we mean a change observed in this way. The change is deemed to have occurred at the time of the second measurement. Hence, the measured age of a route instance is always zero when it is first observed. This conservative approach underestimates route age with an error smaller than the inter-measurement period.

2.2 Dataset

For our purposes, an ideal dataset would be a complete record of the evolution of virtual paths, together with all sequences of IP interfaces for each constituent route. Real world traces are limited both in the frequency at which each virtual path can be sampled, and the accuracy and completeness of the routing information obtained at each sample. In particular, the identification of the multiple IP interface sequences for branched routes requires a lot of probes [31] and takes time, reducing the frequency at which we can measure virtual paths. For this identification we use Paris traceroute's Multipath Detection Algorithm (MDA) [31]. MDA provides strong statistical guarantees for complete route discovery in the presence of an unknown number of load balancers. It is therefore ideal for reliable change detection, but is conservative and can be expensive in probe use (see Sec. 5.4).

We address the above limitations by using traces collected with *FastMapping* [8]. FastMapping measures virtual paths with a modified version of Paris traceroute [2] that sends a single probe per hop. Whenever a new IP interface is seen, FastMapping remeasures the route using MDA. In this way, the frequency at which it searches for path changes is high, but when a change is detected, the new route is mapped out thoroughly.

We use a publicly-available dataset collected from 70 PlanetLab hosts during 5 weeks starting September 1st, 2010 [8]. Each monitor selects 1,000 destinations at random from a list of 34,820 randomly chosen reachable destinations. Each virtual path is measured every 4.4 minutes on average. We complement the dataset with IP-to-AS maps built from Team Cymru[1] and UCLA's IRL [23]. Although almost all monitors are connected to academic networks, the destinations are not. As such, this dataset traverses 7,842 ASes and covers 97% of large ASes [23].

We lack ground truth about path changes and the FastMapping dataset may miss changes; however, all changes the dataset captures are real. Fig. 1 shows the distribution of all route durations in the dataset. It is similar to Paxson's findings that most routes are short-lived: 60% of routes have durations under one hour.

2.3 Prediction goals and error metrics

We study three kinds of prediction: (i) prediction $\hat{L}(r)$ of the residual lifetime $L(r)$ of a route $r = P(t)$ of some path observed at time t, (ii) prediction $\hat{N}_\delta(P)$ of the number of changes in the path occurring in the time interval $[t, t+\delta]$, and (iii) prediction, via an indicator function $\hat{I}_\delta(r)$, of whether the current route will change in the interval $[t, t+\delta]$ ($I_\delta(r) = 1$), or not ($I_\delta(r) = 0$).

In the case of residual lifetime, we measure the relative prediction error $E_L(r) = (\hat{L}(r) - L(r))/L(r)$. This takes values in $[-1, \infty)$, with $E_L(r) = 0$ corresponding to a perfect prediction. For $\hat{N}_\delta$, we measure the absolute error $E_{N_\delta}(P) = \hat{N}_\delta(P) - N_\delta(P)$ because the relative prediction error is undefined whenever $N_\delta(P) = 0$. For $\hat{I}_\delta$, we measure the error E_{I_δ}, the fraction of time $\hat{I}_\delta(r) \neq I_\delta(r)$. This takes values in $[0, 1]$, with $E_{I_\delta} = 0.5$ corresponding to a random predictor.

2.4 Virtual path features

A virtual path predictor needs to determine and exploit those features of the path and its history that carry the most information about change patterns.

Paxson characterized virtual path stability using the notions of route *persistence*, which is essentially route duration $D(r)$, and route *prevalence* [24], the proportion of time a given route is active. In the context of prediction, where only metrics derivable from past data are available, these two measures translate to the following two features of the route r which is current at time t: (i) the route age $A(r)$, and (ii) the (past) prevalence, the fraction of time r was active over the window $[t - \tau, t]$. We set the *timescale* τ to $\tau = \infty$ to indicate a window starting at the beginning of the dataset.

Route age and prevalence are important prediction features. A first idea of their utility is given in Figs. 2(a) and 2(b) respectively, where the median, 25th, and 75th percentiles of route residual lifetimes are given as a function of the respective features (these were computed based on periodic sampling of all virtual paths in the dataset with period five minutes). In Fig. 2(a) for example we observe that younger routes have shorter residual lifetimes than older routes, a possible basis for prediction. Similarly, Fig. 2(b) shows that when prevalence is measured over a timescale of $\tau = 1$ day, routes with lower prevalence are more likely to die young.

Although route age and prevalence are each useful for prediction, they are not sufficient, as shown by the high variability in the

[1] http://www.team-cymru.org/Services/ip-to-asn.html

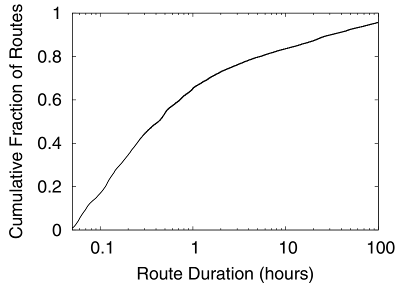

Figure 1: Distribution of all route durations in the dataset.

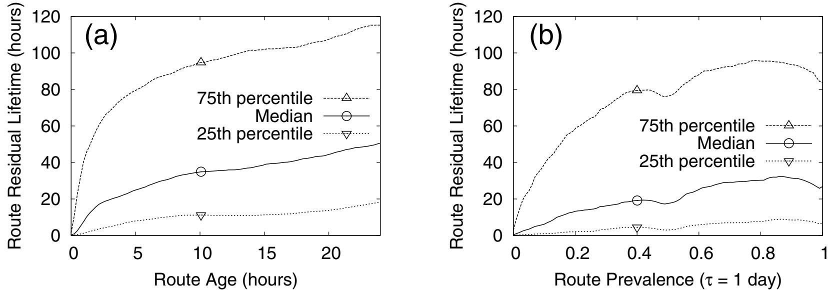

Figure 2: Relationship between virtual path features and residual lifetime: residual lifetime as a function of (a) route age and (b) route prevalence.

data (wide spread of the percentiles in Figs. 2(a) and 2(b)). To do better, additional features are needed. Our aim here is to define a spectrum of features broad enough to capture essentially all information computable from the dataset which may have predictive value. We do not know at this point which features are the important ones, nor how to combine them to make accurate predictions. This is a task we address in Sec. 3.

We do not attempt to exploit spatial dependencies in this paper for prediction, although they clearly exist. For example, changes in routing tables impact multiple paths at roughly the same time. The reason is that including spatial network information in RuleFit requires one predictive feature per link in the network, which is computationally prohibitive. However, we *can* exploit spatial dependencies to improve path tracking efficiency through the probing scheme, as we detail in Sec. 5.3.

Table 1 partitions all possible features into four categories: (i) Current route – characterize the current route and its state; (ii) Last change – capture any nearest neighbor interactions; (iii) Timescale-based – metrics measured over a given timescale; (iv) Event-based – metrics defined in 'event-time'. We use this scheme only as a framework to guide the selection of individual features. We aim to capture inherently different kinds of information and measures both of average behavior and variability. Only features that are computable based on the information in the dataset, together with available side-information (we use IP-to-AS maps), are allowed.

The last four features in the Timescale-based category allow us to identify virtual paths that are highly unstable and change repeatedly, as observed by previous work [22, 24, 30]. The features in the Event-based category may involve time but are not defined based on a preselected timescale. Instead, they try to capture patterns of changes in the past, like oscillation between two routes. For computational reasons we limit ourselves to looking up to the 5 most recent virtual path changes. In most of the cases this is already sufficient to reach the beginning of the dataset.

Feature properties. Paths in the FastMapping dataset are stable 96% of the time, but experience short-lived instability periods. Similar to Zhang et al. [32], we find that path changes are local and usually close to destinations: 86% of changes are inside an AS and 31% of path changes impact the destination's AS. We also find that 38% of path changes impact the path length, and 14% change the AS-level path. Our previous work [8] presents a more detailed characterization of path changes.

Tab. 1 shows the correlation between path features and residual lifetime, computed by sampling the dataset with a Poisson process with an average sampling period of 4 hours. For timescale-based

	Correlation with $L(r)$
Current route	
Route Age	0.17
Length	-0.10
AS length	-0.10
Number of load balancers (i.e., hops with multiple next-hops)	-0.04
Indicator of whether the route reaches the destination	-0.03
Last change	
Duration of the previous route	0.03
Length difference	-0.07
AS length difference	-0.02
Edit distance	0.05
AS edit distance	0.07
Timescale-based (computed over $[t-\tau, t]$)	
Prevalence of the current route	0.20
Average route duration	-0.11
Standard deviation of route durations	-0.13
Number of previous occurrences of the current route	-0.11
Number of virtual path changes	-0.14
Event-based	
Times since the most recent occurrences of the current route	-0.08
Number of changes since the most recent occ. of the cur. route	-0.09

Table 1: Set of candidate features underlying prediction.

features we show correlation values for $\tau = 1$ day, and for event-based features we show the highest correlation. These low correlation values indicate that no single feature can predict changes; in the next section we study how to combine features for prediction.

3. PREDICTION FOUNDATIONS

Our path tracking approach is built on the ability to predict (albeit imperfectly) virtual path changes. We seek a predictor based on an intuitive and parsimonious model rather than a black box. However, virtual path changes are characterized by extreme variability and are influenced by many different factors, making model building, and even feature selection, problematic. We employ RuleFit [13], a state-of-the-art supervised machine learning technique, to bootstrap our modeling efforts. We use RuleFit for two main purposes. First, to comprehensively examine the spectrum of features of Tab. 1 to determine the most predictive. Second, to act as a benchmark representing in an approximate sense the best possible prediction when large (off-line) resources are available for training.

3.1 RuleFit Overview

RuleFit [13] trains predictors based on rule ensembles. We choose it over other alternatives (against which it compares favorably) for two reasons: (i) it ranks features by their importance for

prediction, (ii) it outputs easy-to-interpret rules that allow an understanding of how features are combined. We give a brief overview of RuleFit, referring the reader to the original paper for details [13].

Rules combine one or more features into simple 'and' tests. Let $\boldsymbol{x}$ be the feature vector in Tab. 1 and s_f a specified subset of the possible values of feature f. Then, a rule takes the form

$$r(\boldsymbol{x}) = \prod_f I(x_f \in s_f), \tag{1}$$

where $I(\cdot)$ is an indicator function. Rules take value one when all features have values inside their corresponding ranges, else zero.

RuleFit first generates a large number of rules using decision trees. It then trains a predictor of the form

$$\hat{\phi}(\boldsymbol{x}) = a_0 + \sum_k a_k r_k(\boldsymbol{x}), \tag{2}$$

where the vector $\boldsymbol{a}$ is computed by solving an optimization problem that minimizes the *Huber loss* (a modified squared prediction error robust to outliers) with an L1 penalty term. RuleFit also employs other robustness mechanisms, for example it trains and tests on subsets of the training data internally to avoid overfitting.

Rule ensembles can exploit feature interdependence and capture complex relationships between features and prediction goals. Crucially, RuleFit allows rules and features to be ordered by their importance. *Rule importance* is the product of the rule's coefficient and a measure of how well it splits the training set:

$$I_k = |a_k| \sqrt{s_k(1 - s_k)},$$

where s_k is the fraction of points in the training set where $r_k(\boldsymbol{x}) = 1$. *Feature importance* is computed as the sum of the normalized importance of the rules where the feature appears:

$$I_f = \sum_{k: f \in r_k} I_k / m_k, \tag{3}$$

where m_k is the number of active features in r_k.

3.2 RuleFit training sets

RuleFit, like any supervised learning algorithm, requires a training set consisting of training points that associate features with the true values of metrics to be predicted. In our case, a training point, say for residual lifetime, associates a virtual path at some time t, represented by the features in Tab. 1, with the true residual lifetime $L(r)$ of the current route $r = P(t)$. Separate but similar training is performed for $N_\delta(P)$ and $I_\delta(r)$.

To limit the computational load of training, which is high for RuleFit, we control the total number of training points. For training point selection, first note that a given virtual path has a change history that is crucial to capture for good prediction of its future. We therefore build the required number of training points by extracting rich path change information from a subset of paths, rather than extracting (potentially very) partial information from each path. We retain path diversity through random path selection, and the use of multiple training sets (at least five for each parameter configuration we evaluate), obtained through using different random seeds.

For a given virtual path, we first include all explicit path change information by creating a training point for each entry in the dataset where a change was detected. However, such points all have (measured) current route age equal to zero (Sec. 2.1), whereas when running live predictions in general are needed at any arbitrary time point, with arbitrary route age. To capture the interdependence of features and prediction targets on route age we include additional synthetic points which do not appear in the dataset but which are functions of it. To achieve this we discretize route age into bins and create a training point whenever the age of a route reaches a bin boundary. We choose bin boundaries as equally-spaced percentiles of the distribution of route durations in the training set, as this adapts naturally to distribution shape. Using five bins as example, we create training points whenever a route's age reaches zero seconds, 3.5 min., 12 min., 48 min., and 4 hours.

3.3 Test sets

Like training sets, test sets consist of test points which associate virtual path features with correct predictions. Unlike training sets, where the primary goal is to collect information important for prediction and where details may depend on the method to be trained, for test sets the imperative is to emulate the information available in the operational environment so that the predictor can be fairly tested, and should be independent of the prediction method.

The raw dataset has too many points for use as a test set. To reduce computational complexity, we build test sets by sampling each virtual path at time points chosen according to a Poisson process, using the same sampling rate for each path. This corresponds to examining the set of paths in a neutral way over time, which will naturally include a diversity of behavior. For example, our test sets include samples inside bursts of path changes, many samples from a very long-lived route, and rare events such as of an old route just before it changes.

We use an average per-path sampling period of four hours, resulting in at least two orders of magnitude more test points than training points. We test each predictor against eight test sets (from different seeds), for a total of 40 different training–test set combinations.

We ignore routes active at the beginning or the end of the dataset when creating training and test sets, as their duration, age, and residual lifetime are unknown. Similarly, we ignore all virtual path changes in the first τ hours of the dataset (if $\tau \neq \infty$) to avoid biasing timescale-dependent features.

3.4 RuleFit configuration

In this section we study the impact of key parameters on prediction accuracy, pick practical default values, and justify our use of RuleFit as a benchmark for predicting virtual path changes.

We study the impact of four parameters on prediction error: the number of rules generated during training, the number of age thresholds, the timescale τ, and the training set size. Each plot in Fig. 3.1 varies the value of one parameter while keeping the others fixed. We show results for E_{I_δ} with $\delta = 4$ hours because this is the prediction goal where the studied parameters have the greatest impact. Results for other values and other prediction goals are qualitatively similar. We compute the prediction error rate only for test points with route age less than 12 hours to focus on the differences between configurations. As we discuss later, prediction accuracy is identical for routes older than 12 hours regardless of configuration. We plot the minimum, median, and maximum error rate over 40 combinations of training and test sets for each configuration.

Fig. 3.1(a) shows that the benefit of increasing the number of generated rules is marginal beyond 200 for this data. Our interpretation is that at 200 or so rules, RuleFit has already been able to exploit all information relevant for prediction. Therefore, we train predictors with 200 rules unless stated otherwise.

Fig. 3.1(b) shows that prediction error decreases when we add additional points with age diversity into training sets as described in Sec. 3.2. However, as few as three age bins are enough to achieve accurate predictions, and improvement after six is minimal. Therefore, we train predictors with six age bins unless stated otherwise.

Fig. 3.1(c) shows that the timescale τ used to compute timescale-dependent features has little impact on prediction accuracy. A pos-

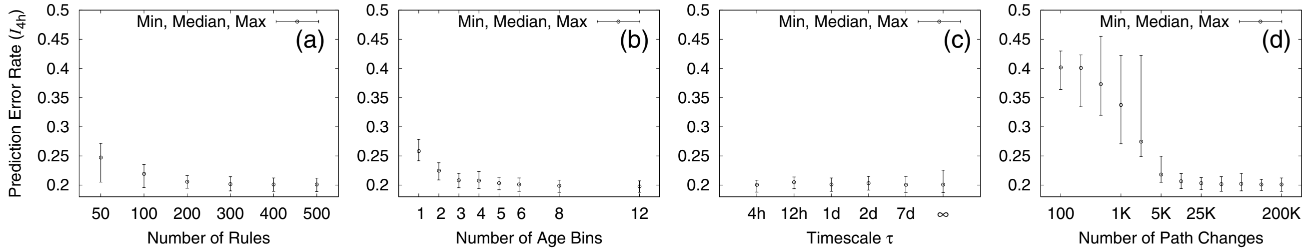

Figure 3: Impact of the (a) number of rules, (b) number of age bins, (c) timescale τ, and (d) training set size on RuleFit accuracy (test points with route age less than 12 hours).

PATH FEATURE	IMPORTANCE
Prevalence of the current route ($\tau = 1$ day)	1.0
Num. of virtual path changes ($\tau = 1$ day)	.624
Num. of previous occ. of the current route ($\tau = 1$ day)	.216
Route age	.116
Times since most recent occs. of the current route	$\leq .072$
Edit distance (last change)	.015
Duration of the previous route	.014
Standard deviation of route durations ($\tau = 1$ day)	.014
Length difference (last change)	.012
All other features	$\leq .010$

Table 2: Feature importance according to RuleFit.

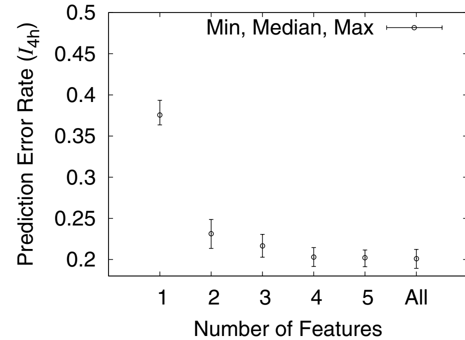

Figure 4: $E_{I_{4h}}$ for predictors trained with the most important features (test points with route age < 12 hours).

sible explanation is that only the long term mean value of timescale-dependent features is predictive, and that RuleFit discovers this and only builds the means of these features into the predictor (or ignores them). Therefore, we train predictors with timescale-dependent features computed with $\tau = 1$ day.

Finally, Fig. 3.1(d) shows the impact of the number of virtual path changes in a training set. Training sets with too few changes fail to capture the virtual path change diversity present in test sets, resulting in predictors that do not generalize. Prediction accuracy increases quickly with training set size before flattening out. We use training sets with 200,000 virtual path changes (around 2.4% of those in the dataset) unless stated otherwise.

We justify our use of RuleFit as a benchmark for predicting changes, based on a given (incomplete) dataset, on: (i) we provide RuleFit with a rich feature set, (ii) RuleFit performs an extensive search of feature combinations to predict residual lifetimes, and (iii) our evaluation shows that changing RuleFit's parameters is unlikely to improve prediction accuracy significantly. This is an empirical approach to approximately measure the limits to prediction using a given dataset. Determining actual limits would only be possible given information-theoretic or statistical assumptions on the data, which is beyond the scope of this paper.

3.5 Feature selection

We compute feature importance with Eq. (3) and normalize using the most important feature. Tab. 2 shows the resulting ordered features, with normalized importance averaged over 50 predictors for each of residual lifetime, number of changes, and I_δ.

Route prevalence is the most important feature, helped by its correlation with route age. It is clear why route prevalence alone is insufficient. It cannot differentiate a young current route that also occurred repeatedly in the time window of width τ, from a middle-aged current route, as both have intermediate prevalence values.

The second, third, and fourth most important features are the number of virtual path changes, the number of occurrences of the current route, and route age. Predicted residual lifetimes increase as route age and prevalence increase, but decrease as the number of virtual path changes and occurrences of the current route increase. Results for the number of changes and I_δ are similar.

The fifth most important feature is the times (1st up to 5th) of the most recent occurrences of the current route. The low importance of this and the other event-based feature suggests that, contrary to our initial hopes, patterns of changes are too variable, or too rare, to be useful for prediction.

To evaluate more objectively the utility of RuleFit's feature importance measure, Fig. 4 shows $E_{I_{\delta=4h}}$ for predictors trained with training sets containing only the top p features, for $p = 1$ to 5. The improvements in performance with the addition of each new feature are consistent with the importance rankings from Tab. 2. Importantly, we see that the top four features generate predictors which are almost as accurate as those trained on all features.

4. NEAREST–NEIGHBOR PREDICTOR

We design and evaluate a simple predictor which is almost as accurate as RuleFit while overcoming its slow and computationally expensive training, its difficult integration into other systems, and the lack of insight and control arising from its black box nature.

4.1 NN4: Definition

We start from the observation that the top four features from Tab. 2 carry almost all of the usable information. Since virtual paths are so variable and the RuleFit models we obtained are so complex, simple analytic models are not serious candidates as a basis for prediction. We select a nearest-neighbor approach as it captures empirical dependences effectively and flexibly. Using only four features avoids the dimensionality problems inherent to such predictors [5] and allows for a very simple method, which we name NN4.

4.1.1 Method overview

Like all nearest-neighbor predictors, we compute predictions for a virtual path with feature vector $\boldsymbol{x}$ based on training points with

feature vectors that are 'close' to $\boldsymbol{x}$. The first challenge is to define a meaningful distance metric. This is difficult as feature domains differ (prevalence is a fraction, the number of changes and previous occurrences are integers, and route age is a real), have different semantics, and impact virtual path changes differently.

To avoid the pitfalls of some more or less arbitrary choice of distance metric, we instead partition the feature space into 4 dimensional 'cubes', or partitions, based on discretising each feature. Discretisation creates artifacts related to bin boundaries and resolution loss; however, the advantages are simplicity and the retention of a meaningful notion of distance for each feature individually. To avoid rigid fixed bin boundaries, for each feature we choose them as equally-spaced percentiles of their corresponding distribution, computed over all virtual path changes in the training set (as we did for route age in Sec. 3.2).

We denote the partition containing the feature vector of path P at time t as $\mathcal{P}(P,t)$ or simply $\mathcal{P}(P)$. We predict the residual lifetime of $r = P(t)$ and the number of changes in the next δ interval as the averages of the true values of these quantities over all training points in the partition $\mathcal{P}(P)$:

$$\hat{L}(r) = E[\{L(P_s(t_s)) \mid s \in \mathcal{P}(P)\}],$$
$$\hat{N}_\delta(P) = E[\{N_\delta(P_s) \mid s \in \mathcal{P}(P)\}],$$

where training point s corresponds to the path P_s at time t_s. Similarly, we predict $\hat{I}_\delta(r) = 1$ if more than half the training points in $\mathcal{P}(P)$ change within a time interval δ:

$$\hat{I}_\delta(r) = \lfloor E[\{I_\delta(P_s(t_s)) \mid s \in \mathcal{P}(P)\}] + 0.5\rfloor.$$

The cost of a prediction in NN4 is O(1), while in RuleFit it is O(r), where r is the number of rules in the model. NN4 can be easily implemented, while RuleFit is available as a binary module that cannot be accessed directly and requires external libraries.

4.1.2 Training

To allow a meaningful comparison in our evaluation, each training for NN4 reuses the virtual paths of some RuleFit training set.

Consider a virtual path $P(t_s)$ chosen for training. As t_s progresses, the associated feature vector $\boldsymbol{x}(t_s)$ moves between the different partitions. For example, for long-lived routes, $\boldsymbol{x}(t_s)$ evolves toward the partition with 100% prevalence, zero changes, no previous occurrences, and the oldest age bin (before resetting to zero age etc. when/if the path changes). We need to sample this trajectory in a way that preserves all important information about the changes in the three prediction goals (L, N_δ, I_δ). Just as in RuleFit, we need to supplement the changes that occur explicitly in the dataset with additional training points occurring inbetween change points. Here we need to add additional samples to capture the diversity not only of age, but also the other three dimensions. In fact we can do much better than a discrete sampling leading to a set of training time points. From the dataset we can actually calculate when the path enters and exits the partitions it visits, its sojourn time in each, and the proportions of the sojourn time when a prediction goal takes a given value. For each partition (and prediction goal) we are then able to calculate the exact time-weighted average of the value over the partition. The result is a precomputed prediction for each partition traversed by the path that emulates a continuous-time sampling. Final per-partition predictions are formed by averaging over all paths traversing a partition.

4.1.3 Configuration

Apart from δ, the only parameter of our predictor is the number of bins b we use to partition each feature. We choose a shared

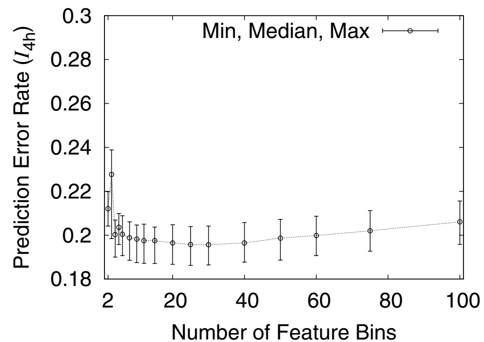

Figure 5: Impact of the number of feature bins on prediction accuracy (test points with age < 12 hours).

number of bins for parsimony, since when studying each feature separately (not shown) the optimal point was similar for each. The tradeoff here is clear. Too few bins and distinct change behaviors important for prediction are averaged away. Too many bins and partitions contain insufficient training information resulting in erratic predictions. We found in Sec. 3.4 that six bins were sufficient for route age. We now examine the three remaining features.

Fig. 5 shows E_{I_δ} with $\delta = 4$ hours as a function of b, restricting to test points with route age below 12 hours where the b dependence is strongest. We see that values in $[6, 20]$ achieve a good compromise. We use $b = 10$ in what follows.

4.2 NN4: Evaluation

We evaluate the prediction accuracy of NN4 and compare it to our operational benchmark, RuleFit, discovering in the process the limitations of this kind of prediction in general. We will find that only very rough prediction is feasible, but in the next section we show that it is nonetheless of great benefit to path tracking. For each method we generate new training and test sets in order to test the robustness of the configuration settings determined above.

4.2.1 Predicting residual lifetime

Fig. 6(Top) shows the distribution of $E_L(r)$, the relative error of $\hat{L}(r)$. An accurate predictor would have a sharp increase close to $E_L = 0$ (dotted line), but this is not what we see. Specifically, only 33.5% of the RuleFit and 31.1% of the nearest-neighbor predictions have $-0.5 \leq E_L \leq 1$ (see symbols on the curves). Predictions miss the true residual lifetimes by a significant amount around 70% of the time. As this is true not only of NN4 but also for RuleFit, we conjecture that accurate prediction of route residual lifetimes is too precise an objective with traceroute-based datasets. It does not follow, however, that $\hat{L}(r)$ is not a useful quantity to estimate. We can still estimate its order of magnitude well in most cases, and this is enough to bring important benefits to path tracking, as we show later. The error of NN4 is considerable larger than that of the benchmark but it is of the same order of magnitude.

4.2.2 Predicting number of changes

Fig. 6(Bottom) shows the distribution of E_{N_δ}, the error of $\hat{N}_\delta$, for NN4 for all test points with route age less than 12 hours. The errors for RuleFit are similar. Errors for test points in routes older than 12 hours are significantly smaller (not shown) because a predictor can perform well simply by outputting "no change" ($\hat{N}_\delta = 0$). We focus here on the difficult case of $A < 12$h.

Unlike residual lifetimes, the sharp increase near zero means most predictions are accurate. For example, 90.2% of test points have $-2 < E_{N_{4h}} < 2$, and accuracy increases for smaller val-

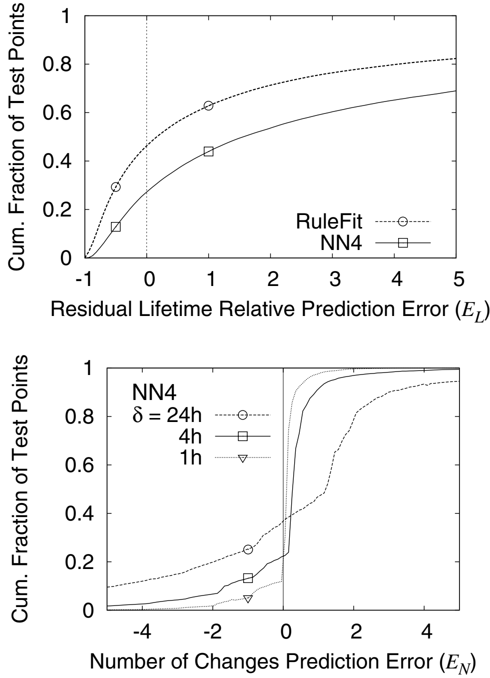

Figure 6: Distribution of prediction error. Top: L; Bottom: N_δ based on NN4 (for age < 12h).

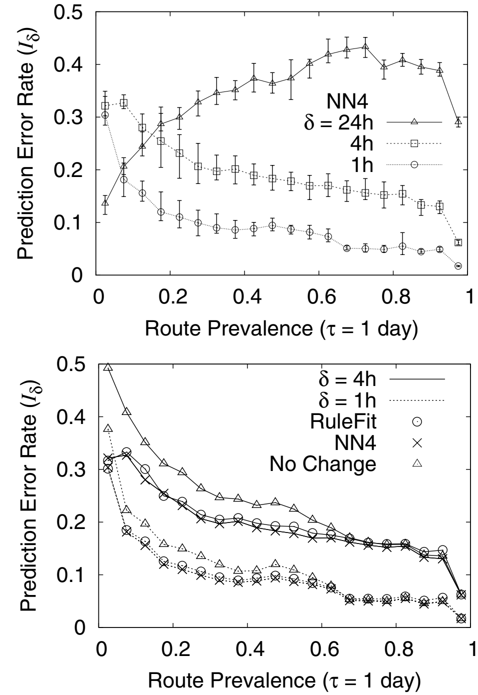

Figure 7: E_{I_δ} as a function of route prevalence for various values of δ. Top: NN4; Bottom: RuleFit comparison.

ues of δ. However, predicting the number of changes over long intervals such as 24 hours cannot be done accurately. Note that simply guessing that $N_\delta = 0$ also works well for very small δ. Although N_δ is a less ambitious target than L, it remains difficult to estimate from traceroute-type data. Again, however, prediction is sufficiently good to bring important tracking benefits.

4.2.3 Predicting a change in next δ interval

We now study whether the current route of a given path will change within the next time interval of width δ. We expect I_δ to be easier to predict than L or N_δ.

Fig. 7(Top) shows NN4's prediction error as a function of route prevalence for δ between 1 hour and 1 day (results for RuleFit are very similar and are omitted for clarity). We group route prevalence into fixed-width bins and compute the error from all test points falling within each bin (these bins are distinct from the constant-probability bins underlying NN4's partitions). For each bin, we show the minimum, median, and maximum error among the 40 training and test set combinations. Such a breakdown is very useful as it allows us to resolve where prediction is more successful, or more challenging. For example, since routes with prevalence 1 are very common, a simple global average over all prevalence values would drown out the results from routes with prevalence below 1.

First consider the results for $\delta = 1\text{h}$ and 4h. The main observation is that error drops as prevalence increases. This is because routes with high prevalence are unlikely to change, and a prediction of "no change" ($\hat{I}_\delta(r) = 0$), which the predictors output increasingly often, becomes increasingly valid as prevalence increases. We also see that, for all prevalence values, error is lower for smaller δ. This makes intuitive sense since prediction further into the future is in general more difficult. More precisely, the probability that a route will change in a time interval δ decreases as δ decreases, and predictors exploit this by predicting "no change" more often.

The situation is more complex when $\delta = 24\text{h}$, with errors beginning low and increasing substantially before finally peaking and then decreasing at very high prevalence. This happens because for larger values of δ, routes with low prevalence have a high probability of changing. Predictors exploit this and output $I_{24\text{h}}(r) = 1$ more often (in fact more than 80% of the time for paths with prevalence under 0.2). Prediction error is highest at intermediate prevalence values, as these routes have a probability close to 50% of changing in the next 24 hours. Finally, prediction error decreases for routes with high prevalence: as routes become stable the same mechanism noted above for smaller δ kicks in.

We now provide a comparison against RuleFit, focusing on small to medium δ. Fig. 7(Bottom) shows that NN4 and RuleFit have equivalent prediction accuracy across all values of prevalence. In fact NN4 is marginally (up to 2%) better here, where we used the default RuleFit configuration. Their performance is close to identical when using the more generous RuleFit configuration (see Sec. 3) with 500 rules and 12 age bins.

The plot also shows results for a simple baseline predictor that always predicts $\hat{I}_\delta(r) = 0$ (no change). Our predictor is better for routes with prevalence smaller than 0.7 which are more likely to change than not, but for high-prevalence routes all predictors predict "no change" and are equivalent. For routes with prevalence below 0.7, NN4 reduces the baseline predictor's $E_{I_{4\text{h}}}$ from 0.296 to 0.231 (22%), and $E_{I_{1\text{h}}}$ from 0.163 to 0.131 (20%).

Summary. Prediction is easiest when δ is small and prevalence is high. This is a promising result as most Internet routes are long-lived and have high prevalence; moreover, applications like topology mapping need to predict changes within short time intervals. NN4 predicts I_δ reasonably well, and errors ultimately fall to just a few percent as route prevalence increases and as δ decreases. We have tested the sensitivity to training and test sets, monitor choice, and overall probing rate, and found it to be very low.

5. TRACKING VIRTUAL PATH CHANGES

We now apply our findings to the problem of the efficient and accurate tracking of a set of virtual paths over time. We describe and evaluate our tracking technique, DTRACK.

5.1 DTRACK overview

Path tracking faces two core tasks: path change detection (how best to schedule probes to hunt for changes), and path remapping (what to do when they are found). For the latter, inspired by Fast-Mapping [8], DTRACK uses Paris traceroute's MDA to accurately measure the current route of monitored paths both at start up and after any detection of change. This is vital, since confusing path changes with load balancing effects makes 'tracking' meaningless.

For change detection, DTRACK is novel at two levels.

Across paths: paths are given dedicated sampling rates guided by NN4 to focus effort where changes are more likely to occur. Without this, probes are wasted on paths where nothing is happening.

Within paths: a path 'sample' is a single probe rather than a full traceroute, whose target interface is carefully chosen to combine the benefits of Paris Traceroute over time with efficiencies arising from exploiting links shared between paths. This allows changes to be spotted more quickly.

DTRACK monitors operate independently and use only locally available information. Each monitor takes three inputs: a predictor of virtual path changes, a set $\mathcal{D}$ of virtual paths to monitor, and a probing budget; and consists of three main routines: sampling rate allocation, change tracking, and change remapping. When a change is detected in a path through sampling, that path is remapped, and sampling rates for all paths are recomputed. A probing budget is commonly used to control the average resource use [14, 25].

5.2 Path sampling rate allocation

For each path p in $\mathcal{D}$, DTRACK uses NN4 to determine the rate λ_p at which to sample it. Sampling rates are updated whenever there is a change in the predictions, i.e., whenever any virtual path's feature vector changes its NN4 partition. This can happen as a result of a change detection or simply route aging.

We constrain sampling rates to the range $\lambda_{\min} \leq \lambda_p \leq \lambda_{\max}$. Setting $\lambda_{\min} > 0$ guarantees that all paths are sampled regularly, which safeguards against poor predictions on very long lived paths. An upper rate limit is needed to avoid probes appearing as an attack ($\lambda_{\max}$ implements the "politeness" of the tracking method [18]).

Based on the monitor's probe budget of B probes per second, a sampling budget of B_s samples per second for change detection alone can be derived (Sec. 5.4). To be feasible, the rate limits must obey $\lambda_{\min} \leq B_s/|\mathcal{D}| \leq \lambda_{\max}$, where $|\mathcal{D}|$ is the number of paths.

We now describe three allocation methods for the sampling rates λ_p. The first two are based on residual life and the third minimizes the number of missed changes.

Residual lifetime allocation. Since $1/L$ is precisely the rate that would place a sample right at the next change, allocating sampling rates proportional to $1/\hat{L}$ is a natural choice. We will see that despite the poor accuracy of $\hat{L}$ found before, this is far better than the traditional uniform allocation. To approximate this we define rates to take values in

$$\lambda_p \in \{\lambda_{\max}, a/\hat{L}(p), \lambda_{\min}\} \tag{4}$$

and require that $\lambda_p \geq \lambda_q$ if $L(p) < L(q)$ and $\lambda_{\min} \leq \lambda_p \leq \lambda_{\max}$ for all p, where a is a renormalisation constant which respects $\sum_p \lambda_p = B_s$ while minimizing the number of paths with rates clipped at $\lambda_{\min}$ or $\lambda_{\max}$.

We define two variants depending on the definition of $\hat{L}(p)$:

RL: $\hat{L}(p)$ is estimated by NN4,

RL-AGE: $\hat{L}(p)$ is predicted as the average residual lifetime of all route instances in the dataset with duration larger than $A(r)$, i.e.,

$$\hat{L}'(r) = \mathrm{E}[\{D(s) \mid s \in \mathcal{R} \text{ and } D(s) > A(r)\}] - A(r),$$

where $\mathcal{R}$ is the set of all route instances in the dataset.

Finally, for comparison we add an oracular method which knows the true $L(p)$ and is not subject to rate limits:

RL-ORACLE: $\lambda_p = a'/L(p)$ where $a' = B_s \sum_q 1/L(q)$.

Minimizing missed changes (MINMISS, used in DTRACK). We use a Poisson process as a simple model for when changes occur. With this assumption we are able to select rates that minimize the expected number of missed changes over the prediction horizon δ. This combines prediction of N_δ with a notion of sampling more where the pay off is higher. The rate $\mu_c(p)$ of the Poisson change process is estimated as $\mu_c(p) = \hat{N}_\delta(p)/\delta$.

We idealize samples as occurring periodically with separation $1/\lambda_p$. By the properties of a Poisson process, the changes falling within successive gaps between samples are i.i.d. Poisson random variables with parameter $\mu = \mu_c(p)/\lambda_p = \hat{N}_\delta/(\delta\lambda_p)$. Let C be the number of changes in a gap and M the number of these missed by the sample at the gap's end. It is easy to see that $M = \max(0, C-1)$, since a sample can see at most one change (here we assume that there is at most one instance of any route in the gap). The expected number of missed changes in a gap is then

$$\begin{aligned}\mathrm{E}[M(\mu)] &= \sum_{m=0}^{\infty} m\Pr(M=m) = \sum_{m=1}^{\infty} m\Pr(C=m+1) \\ &= e^{-\mu}\sum_{m=1}^{\infty}\frac{m\mu^{m+1}}{(m+1)!} = \mu - 1 + e^{-\mu}\,. \end{aligned} \tag{5}$$

Summing over the $\delta\lambda_p$ gaps, we compute the sampling rates as the solution of the following optimization problem:

$$\min_{\{\lambda_p\}} : \sum_p \delta\lambda_p(\mu - 1 + e^{-\mu}) = \sum_p \hat{N}_\delta + \delta\lambda_p(e^{-\hat{N}_\delta/(\delta\lambda_p)} - 1)$$

$$\text{such that } \sum_p \lambda_p = B_s,\ \lambda_{\min} \leq \lambda_p \leq \lambda_{\max}, \forall p.$$

We also evaluated I_δ as the basis of rate allocation, but as it is inferior to MINMISS, we omit it for space reasons.

Implementation. Path sampling in DTRACK is controlled to be 'noisily periodic'. As pointed out in [4], strictly periodic sampling carries the danger of phase locking with periodic network events. Aided by the natural randomness of round-trip-times, our implementation ensures that sampling has the noise in inter-sample times recommended to avoid such problems [3].

DTRACK maintains a FIFO event queue which emits a sample every $1/B_s$ seconds on average. Path p maintains a timer $T(p)$. When $T(p) = 0$ the next sample request is appended to the queue and the timer reset to $T(p) = 1/\lambda_p$. Whenever DTRACK updates sampling rates, the timers are rescaled as $T_{\text{new}}(p) = T_{\text{old}}(p)\lambda_{\text{old},p}/\lambda_{\text{new},p}$. Path timers are staggered at initialization by setting $T(p_i) = i/B_s$, where i indexes virtual paths.

5.3 In-path sampling strategies

By a *sample* of a path we mean a measurement, using one or more probes, of its current route. At one extreme a sample could correspond to a detailed route mapping using MDA; however, when checking for route changes rather than mapping from scratch, this is too expensive. We now investigate a number of alternatives that

are less rigorous (a change may be missed) but cheaper, for example sending just a single probe. In each case however the sample is load-balancing aware, that is we make use of the flow-id to interface mapping, established by the last full MDA, to target interfaces to test in an informed and strategic way. Thus, although a single sample takes only a partial look at a path and may miss a change, it will not flag a change where none exists, and can still cover the entire route through multiple samples over time.

In what follows we describe a single sample of each technique applied to a single path.

Per-sequence A single interface sequence from the route is selected, and its interfaces are probed in order from the monitor to the destination using a single probe each. Subsequent samples select other sequences in some order until all are sampled and the route is covered, before repeating. This strategy gives detailed information but uses many probes in a short space of time. FastMapping has a similar strategy only it probes a single sequence repeatedly rather than looping over all sequences.

Per-probe The interface testing schedule is exactly as for per-sequence, however only a single probe is sent, so the probing of each sequence (and ultimately each interface in the route) is spread out over multiple samples.

The above methods treat each path in isolation, but paths originated at a single monitor often have shared links. Doubletree [11] and Tracetree [17] assume that the topology from a monitor to a set of destinations is a tree. They reduce redundant probes close to the monitor by doing backwards probing (from the destinations back to the monitor). Inspired by this approach, we describe methods that exploit spatial information, namely knowledge of shared links, to reduce wasteful probing while remaining load-balancing-aware. We define a *link* as a pair of consecutive interfaces found on some path, which can be thought of as a set of links. Many paths may share a given link.

Per-link A single probe is sent, targeting the far interface of the least recently sampled link. The per-link sample sharing scheme means that the timestamp recording the last sampling of a given link is updated by *any* path that contains it. The result is that a given path does not have to sample shared links as often, instead focussing more on links near the destination. Globally over all links, the allocation of probes to links becomes closer to uniform.

Per-safelink As for per-link, except that a shared link only triggers sample sharing when in addition an entire subsequence, from the monitor down to the interface just past the link, is shared.

Any method that tries to increase probe efficiency through knowledge of how paths share interfaces can fail. This happens when a change occurs at a link (say ℓ) in some path p, but the monitor probes ℓ using a path other than p, for which ℓ has not changed. To help reduce the frequency of such events, per-link strengthens the definition of sharing from an interface to a link, and per-safelink expands it further to a subsequence.

Finally, for comparison we add an oracular method:

Per-oracle A single probe is sent, whose perfect interface targeting will always find a change if one exists.

5.4 Evaluation methodology

We describe how we evaluate DTRACK and compare it to other tracking techniques.

Trace-driven simulation. We build a simulator that takes a dataset with raw traceroutes as input, and for each change in each path extracts a timestamp and the associated route description. It then simulates how each change tracking technique would probe these paths, complete with their missed changes and estimated (hence inaccurate) feature vectors.

We use the traces described in Sec. 2.2 as input for our evaluation. Different monitors in this dataset probe paths at different frequencies. Let r_{min} be the minimum interval between two consecutive path measurements from a monitor. We set $\lambda_{\text{max}} = 1/r_{\text{min}}$ per-sequence samples per second (the average value over all monitors is $1/190$), and this is scaled appropriately for other sampling strategies. This setting is natural in our trace-driven approach: probing faster than $1/r_{\text{min}}$ is meaningless because the dataset contains no path data more frequent than every r_{min}, and lower λ_{max} would guarantee that some changes would be missed. We set $\lambda_{\text{min}} = 0$ for all monitors.

Setting probe budgets. The total probe budget B is the sum of a *detection budget* B_d used in sampling for change **d**etection, and a *remapping budget* or cost B_r for route **r**emapping. Let the number of probes per sample be denoted by $n(sam)$, where $sam \in \{s, p, l, sl, o\}$ is one of sampling methods above. The total budget (in probes per second) can be written as

$$B = B_d + B_r = n(sam)B_s + N_r \cdot \overline{\text{MDA}}, \quad (6)$$

where $\overline{\text{MDA}}$ is the average number of probes in a remapping, and N_r is the average number of remappings per second.

When running live in an operational environment, typical estimates of N_r and $\overline{\text{MDA}}$ can be used to determine B_s based on the monitor parameter B. Our needs here are quite different. For the purposes of a fair comparison we control B_d to be the same for all methods, so that the sampling rates will be determined by $B_s = B_d/n(sam)$ where *sam* is the sampling method in use. This makes it much easier to give each method the same resources, since we cannot predict how many changes different methods may find. More importantly, it does not make sense in this context to give each method the same total budget B, since the principal measure of success is the detection of as many changes as possible. More detections inevitably means increased remapping cost, but it would be contradictory to focus on B_r and to view this as a failing. The remapping cost is essentially just proportional to the number of changes found and, although important for the end system, is not of central interest for assessing detection performance. We provide some system examples below based on equal B.

The default MDA parameters are very conservative, leading to high probe use. However, it is stated [31] that much less conservative parameters can be used with little ill effect. In this paper we use default parameters for simplicity, since the change detection performance is our main focus.

Performance metrics. We evaluate two performance metrics for tracking techniques: the fraction of missed virtual path changes, and the change detection delay.

A change can be missed through a sample failing to detect a change, or because of undersampling. We give two examples of the latter. If a path changes from r_1 to r_2 and back to r_1 before a sample, then the tracking technique will miss two changes and think that the path is stable between the two probes. If instead the path changes from r_1 to r_2 to r_3, then tracking will detect a change from r_1 to r_3. For each detected change (and only for detected changes), we compute the detection delay as the time of the detection minus the time of the last true change.

Alternative tracking techniques. We compare DTRACK against two other techniques: FastMapping [8] (Sec. 2.2) and Tracetree [17] (Sec. 5.3).

Comparing Tracetree against FastMapping and DTRACK is difficult because Tracetree assumes a tree topology, and is also obliv-

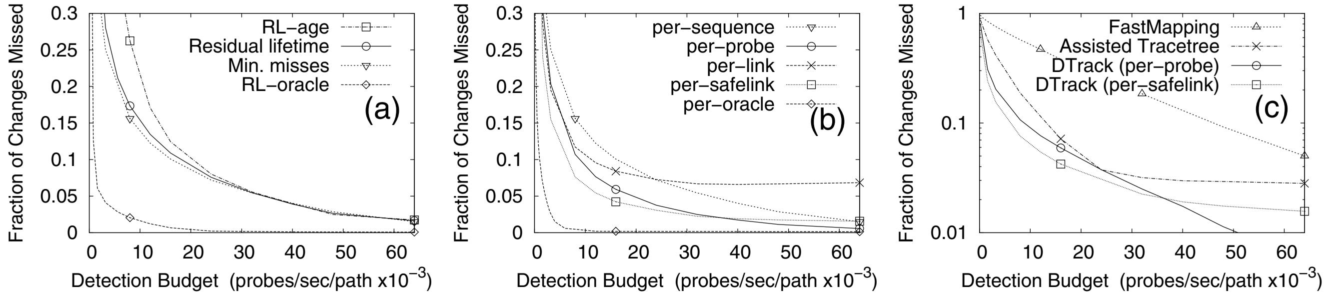

Figure 8: Fraction of missed changes versus detection budget per path ($B_d/|\mathcal{D}|$): (a) comparing path sampling rate allocation (using per-sequence) (b) comparing sampling methods (using MINMISS), (c) comparing DTRACK to alternatives.

ious to load balancing. As such, Tracetree detects many changes that do not correspond to any real change in any path. To help quantify these false positives and to make comparison more meaningful, in addition to the total number of Tracetree 'changes' detected we compute a cleaned version by assisting Tracetree in three ways. We filter out all changes induced by load balancing; ignore all changes due to violation of the tree hypothesis; and whenever a probe detects a change, we consider that it detects changes in all virtual paths that traverse the changed link (even though they were not directly probed). The result is "Assisted Tracetree".

5.5 Evaluation of path rate allocation

This section evaluates RL, RL-AGE, and MINMISS, using per-sequence, the simplest sampling scheme. Fig. 8(a) shows the fraction of changes missed as a function of $B_d/|\mathcal{D}|$, the detection budget per path. Normalizing per-path facilitates comparison for other datasets. For example, CAIDA's Ark project [14] and DIMES [25] use approximately 0.17×10^{-3} and 8.88×10^{-3} probes per second per virtual path, respectively.

When the budget is too small, not even the oracle can track all changes; whereas in the high budget limit all techniques converge to zero misses. We see that Ark's probing budget is the range where even the oracle misses 72% of changes. To track changes more efficiently Ark would need more monitors, each tracking a smaller number of paths.

Comparing RL-AGE and RL shows that NN4 reduces the number of missed changes over the simple age-based predictor by up to 47% when the sampling budget is small. For sampling budgets higher than 30×10^{-3} both RL-AGE and RL perform similarly as most missed changes happen in old, high-prevalence paths where predictors behave similarly. MINMISS reduces the number of missed changes by less than 11% compared to RL. We adopt MINMISS in DTRACK. It is unlikely that we can improve its performance, even if we could it would require a significantly more complex model.

5.6 Evaluation of in-path sampling

We now use MINMISS as the path rate allocation method, and compare the performance of the in-path sampling strategies using Fig. 8(b) ("minimize misses" in Fig. 8(a) and "per-sequence" in Fig. 8(b) are the same).

The per-probe strategy improves on per-sequence by up to 54%. Per-sequence sampling often wastes probes as once a single changed interface is detected, there is no need to sample the rest of the sequence or route, the route can be remapped immediately, and so the search for the next change begins earlier. Per-probe also has a large advantage in spotting short-lived routes, as its sampling rate is $n(s)$ times higher (around 16 times in our data) than per-sequence, greatly decreasing the risk of skipping over them.

Each of per-link and per-probe use a single probe per sample, but from Fig. 8(b) the latter is clearly superior. This is because the efficiency gains of the sample-sharing strategy of per-link are outweighed by the inherent risks of missed changes (as explained at the end of Sec. 5.3). This tradeoff becomes steadily worse as probing budget increases, in fact for this strategy the error saturates rather than tending to zero in the limit.

Per-safelink sampling addresses the worst risks of per-link, and over low detection budgets is the best strategy, with up to 28% fewer misses than per-probe. However, at high sampling rates a milder form of the issue affecting per-link still arises, and again the error saturates rather than tending to zero. These results show that exploiting spatial information (like shared links) must be done with great care in the context of tracking, as the very assumptions one is relying on for efficiencies are, by definition, changing (see Tracetree results below).

By default we use per-safelink sampling in DTRACK, as we expect most deployments to operate at low sampling budgets (e.g., DIMES and CAIDA's Ark). At very high sampling budgets we recommend per-probe sampling.

5.7 Comparing DTRACK to alternatives

Fig. 8(c) replots the per-probe and per-safelink curves from Fig. 8(b) on a logarithmic scale, and compares against FastMapping and the assisted form of Tracetree. Each variant of DTRACK outperforms FastMapping by a large margin, up to 89% at intermediate detection budgets. DTRACK also outperforms Assisted Tracetree for all detection budgets, despite the significant degree of assistance provided. We attribute this mainly to the failure of the underlying tree assumption because of load balancing, traffic engineering, and typical AS peering practices. Real (unassisted) Tracetree also suffers from false positives, which in fact grow linearly in probing budget. Already for a detection budget of 8×10^{-3} probes per second per path, Tracetree infers 17 times more false positives than there are real changes in the dataset!

As an example of the benefits that DTRACK can bring, DIMES, which uses $B_d/|\mathcal{D}| = 8.88 \times 10^{-3}$ probes per second per path, would miss 86% fewer changes (detect 220% more) by using DTRACK instead of periodic traceroutes.

Fig. 9 shows the average remapping cost as a function of sampling budget for DTRACK and FastMapping. Real deployments can reduce remapping costs compared to the results we show by configuring MDA to use less probes [31]. We omit Tracetree as it does not perform remapping.

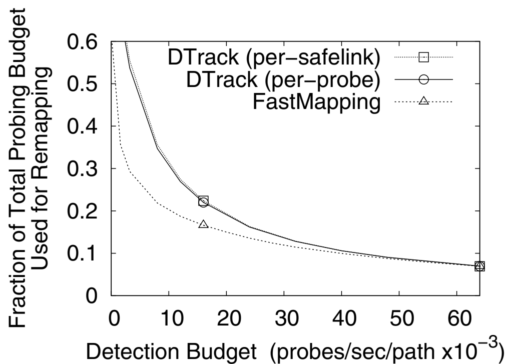

Figure 9: Remapping cost for a given detection budget.

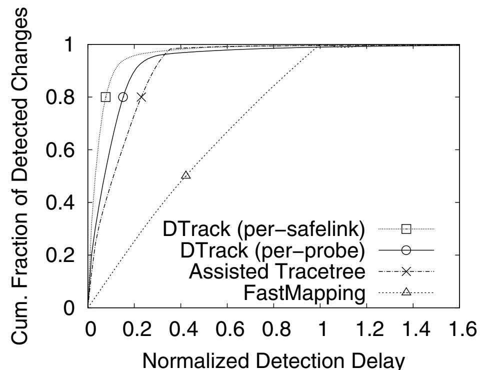

Figure 10: Distribution of detection delay normalized by FastMapping's virtual path sampling period.

Fig. 9 gives $B_r/B = (B - B_d)/B$, the fraction of the probing budget that is used for remapping. At low detection budgets, sampling frequency is lower and each sample has a higher probability to detect a change (as well as to miss others). In such scenarios the remapping cost is comparable to the total budget. As the sampling budget increases, the number of changes detected stabilizes and the remapping cost becomes less significant relative to the total.

Taking again the example of DIMES, even including DTRACK's remapping cost, DIMES would miss 73% less (or detect twice as many) changes using DTRACK instead of periodic traceroutes, while providing complete load balancing information.

Fig. 9 allows an operator to compute an initial sampling budget so that DTRACK respects a desired total probing budget B in a real deployment. After DTRACK is running, the operator can readjust the sampling budget as a function of the actual remapping cost.

Fig. 10 shows the distribution of the detection delay of detected changes for the different tracking techniques, given a detection budget of $B_d/|\mathcal{D}| = 16 \times 10^{-3}$ probes per second per path. Results for other detection budgets are qualitatively similar. We normalize the detection delay by FastMapping's virtual path sampling period (which is common to all paths).

We see that FastMapping detection delay is in a sense the worst possible, being almost uniform over the path sampling period. Tracetree samples paths more frequently and achieves lower detection delay. However, both FastMapping and Tracetree are limited by sampling all paths at the same rate. DTRACK (per-safelink) reduces average detection delay by 57% over FastMapping and has lower delay 99.8% of the time, the exceptions being, not surprisingly, on paths with low sampling budgets.

Low detection delay is important to increase the fidelity of fault detection and tomographic techniques. To see the benefits, say that a monitor uses a total budget B of 64 kbits/sec to track 8,000 paths. It would detect 52% more changes by replacing periodic traceroutes with DTRACK (using safelink) and it would detect 90% of path changes with a delay below 125 seconds. Replacing classic traceroute by MDA also has the benefit of getting complete and accurate routes.

Summary. Our results indicate that DTRACK not only detects more changes, but also has lower detection delay, which should directly benefit applications that need up-to-date information on path stability and network topology.

6. RELATED WORK

Forwarding vs. routing dynamics. Internet path dynamics and routing behavior have captured the interest of the research community since the mid-90s with Paxson's study of end-to-end routing behavior [24] and Labovitz et al.'s findings on BGP instabilities [16]. In this paper, we follow Paxson's approach of using traceroute-style probing to infer end-to-end routes and track virtual path changes. Traceroute is appealing for tracking virtual paths from monitors located at the edge of the Internet for two main reasons. First, traceroute directly measures the forwarding path, whereas AS paths inferred from BGP messages may not match the AS-level forwarding path [21]. Second, traceroute runs from any host connected to the Internet with no privileged access to routers, whereas the collection of BGP messages requires direct access to routers. Although RouteViews and RIPE collect BGP data from some routers for the community, public BGP data lacks visibility to track all path changes from a given vantage point [7, 29]. When BGP messages from a router close to the traceroute monitor are available, they could help tracking virtual path changes. For instance, Feamster et al. [12] showed that BGP messages could be used to predict about 20% of the path failures in their study. We will study how to incorporate BGP messages in our prediction and tracking methods in future work.

Characterization and prediction of path behavior. Some of the virtual path features that we study are inspired by previous characterizations of Internet paths [2, 12, 24] as discussed in Sec. 2.4. None of these characterization studies, however, use these features to predict future path changes. Although to our knowledge there is no prior work on predicting path changes, Zhang et al. [33] studied the degree of constancy of path performance properties (loss, delay, and throughput); constancy is closely related to predictability. Later studies have used past path performance (for instance, end-to-end losses [28] or round-trip delays [6]) to predict future performance. iNano [20] also "predicts" a number of path properties including PoP-level routes, but their meaning for route prediction is different than ours. Their goal is to predict the PoP-level route of an arbitrary end-to-end path, even though the system only directly measures the route of a small sub-set of paths. iNano only refreshes measurements once per day and as such cannot track path changes.

Topology mapping techniques. Topology mapping systems [14, 17, 19, 25] often track routes to a large number of destinations. Many of the topology discovery techniques focus on getting more complete or accurate topology maps by resolving different interfaces to a single router [26, 27], selecting traceroute's sources and destinations to better cover the topology [27], or using the record-route IP option to complement traceroutes [26]. DTRACK is a good complement to all these techniques. We argue that to get more accurate maps, we should focus the probing capacity on the paths that are changing, and also explore spatio-temporal alternatives to simple traditional traceroute sampling. One approach to tracking the

evolution of IP topologies is to exploit knowledge of shared links to reduce probing overhead and consequently probe the topology faster as Tracetree [17] and Doubletree [11] do. As we show in Sec. 5, Tracetree leads to a very large number of false detections. Thus, we choose to guarantee the accuracy and completeness of measured routes by using Paris traceroute's MDA [31]. Most comparable to DTRACK is FastMapping [8]. Sec. 5 shows that DTRACK, because of its adaptive probing allocation (instead of a constant rate for all paths) and single-probe sampling strategy (compared to an entire branch of the route at a time), misses up to 89% fewer changes than FastMapping.

7. CONCLUSION

This paper presented DTRACK, a path tracking strategy that proceeds in two steps: path change detection and path remapping. We designed NN4, a simple predictor of path changes that uses as input: route prevalence, route age, number of past route changes, and number of times a route appeared in the past. Although we found that the limits to prediction in general are strong and in particular that NN4 is not highly accurate, it is still useful for allocating probes to paths. DTRACK optimizes path sampling rates based on NN4 predictions. Within each path, DTRACK employs a kind of temporal striping of Paris traceroute. When a change is detected, path remapping uses Paris traceroute's MDA to ensure complete and accurate route measurements. DTRACK detects up to two times more path changes when compared to the state-of-the-art tracking technique, with lower detection delays, and whilst providing complete load balancer information. DTRACK finds considerably more true changes than Tracetree, and none of the very large number of false positives. More generally, we point out that any approach that exploits shared links runs the risk of errors being greatly magnified in the tracking application, and should be used with great care.

To accelerate the adoption of DTRACK, our immediate next step is to implement DTRACK into an easy-to-use system and deploy it on PlanetLab as a path tracking service. For future work, we will investigate the benefits of incorporating additional information, such as BGP messages, to increase prediction accuracy, as well as the benefits of coordinating the probing effort across monitors to further optimize probing.

Acknowledgements. We thank Ethan Katz-Bassett, Fabian Schneider, and our shepherd Sharon Goldberg for their helpful comments. This work was supported by the European Community's Seventh Framework Programme (FP7/2007-2013) no. 223850 (Nano Data Centers) and the ANR project C'MON.

8. REFERENCES

[1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. *SIGOPS Oper. Syst. Rev.*, 35(5):131–145, 2001.

[2] B. Augustin, T. Friedman, and R. Teixeira. Measuring Load-balanced Paths in the Internet. In *Proc. IMC*, 2007.

[3] F. Baccelli, S. Machiraju, D. Veitch, and J. Bolot. On Optimal Probing for Delay and Loss Measurement. In *Proc. IMC*, 2007.

[4] F. Baccelli, S. Machiraju, D. Veitch, and J. Bolot. The Role of PASTA in Network Measurement. *IEEE/ACM Trans. Netw.*, 17(4):1340–1353, 2009.

[5] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When Is "Nearest Neighbor" Meaningful? In *Proc. Intl. Conf. on Database Theory*, 1999.

[6] A. Bremler-Barr, E. Cohen, H. Kaplan, and Y. Mansour. Predicting and Bypassing End-to-end Internet Service Degradations. *IEEE J. Selected Areas in Communications*, 21(6):961–978, 2003.

[7] R. Bush, O. Maennel, M. Roughan, and S. Uhlig. Internet Optometry: Assessing the Broken Glasses in Internet Reachability. In *Proc. IMC*, 2009.

[8] I. Cunha, R. Teixeira, and C. Diot. Measuring and Characterizing End-to-End Route Dynamics in the Presence of Load Balancing. In *Proc. PAM*, 2011.

[9] I. Cunha, R. Teixeira, N. Feamster, and C. Diot. Measurement Methods for Fast and Accurate Blackhole Identification with Binary Tomography. In *Proc. IMC*, 2009.

[10] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally Distributed Content Delivery. *IEEE Internet Computing*, 6(5):50–58, 2002.

[11] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient Algorithms for Large-scale Topology Discovery. In *Proc. ACM SIGMETRICS*, 2005.

[12] N. Feamster, D. Andersen, H. Balakrishnan, and F. Kaashoek. Measuring the Effects of Internet Path Faults on Reactive Routing. In *Proc. ACM SIGMETRICS*, 2003.

[13] J. Friedman and B. Popescu. Predictive Learning via Rule Ensembles. *Annals of Applied Statistics*, 2(3):916–954, 2008.

[14] k. claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov. Internet Mapping: from Art to Science. In *Proc. IEEE CATCH*, 2009.

[15] E. Katz-Bassett, H. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. Anderson. Studying Black Holes in the Internet with Hubble. In *Proc. USENIX NSDI*, 2008.

[16] C. Labovitz, R. Malan, and F. Jahanian. Internet Routing Instability. In *Proc. ACM SIGCOMM*, 1997.

[17] M. Latapy, C. Magnien, and F. Ouédraogo. A Radar for the Internet. In *Proc. Intl. Workshop on Analysis of Dynamic Networks*, 2008.

[18] D. Leonard and D. Loguinov. Demystifying Service Discovery: Implementing an Internet-Wode Scanner. In *Proc. IMC*, 2010.

[19] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: an Information Plane for Distributed Services. In *Proc. USENIX OSDI*, 2006.

[20] H. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane Nano: Path Prediction for Peer-to-peer Applications. In *Proc. USENIX NSDI*, 2009.

[21] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz. Towards an Accurate AS-level Traceroute Tool. In *Proc. ACM SIGCOMM*, 2003.

[22] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. N. Chuah, Y. Ganjali, and C. Diot. Characterization of Failures in an Operational IP Backbone Network. *IEEE/ACM Trans. Netw.*, 16(4):749–762, 2008.

[23] R. Oliveira, D. Pei, W. Willinger, B. Zhang, and L. Zhang. Quantifying the Completeness of the Observed Internet AS-level Structure. *IEEE/ACM Trans. Netw.*, 18(1):109–122, 2010.

[24] V. Paxson. End-to-end Routing Behavior in the Internet. *IEEE/ACM Trans. Netw.*, 5(5):601–615, 1997.

[25] Y. Shavitt and U. Weinsberg. Quantifying the Importance of Vantage Points Distribution in Internet Topology Measurements. In *Proc. IEEE INFOCOM*, 2009.

[26] R. Sherwood, A. Bender, and N. Spring. DisCarte: a Disjunctive Internet Cartographer. In *Proc. ACM SIGCOMM*, 2008.

[27] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, 2002.

[28] S. Tao, K. Xu, Y. Xu, T. Fei, L. Gao, R. Guerin, J. Kurose, D. Towsley, and Z.-L. Zhang. Exploring the Performance Benefits of End-to-End Path Switching. In *Proc. ICNP*, 2004.

[29] R. Teixeira and J. Rexford. A Measurement Framework for Pin-pointing Routing Changes. In *Proc. SIGCOMM Workshop on Network Troubleshooting*, 2004.

[30] D. Turner, K. Levchenko, A. Snoeren, and S. Savage. California Fault Lines: Understanding the Causes and Impact of Network Failures. In *Proc. ACM SIGCOMM*, 2010.

[31] D. Veitch, B. Augustin, T. Friedman, and R. Teixeira. Failure Control in Multipath Route Tracing. In *Proc. IEEE INFOCOM*, 2009.

[32] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-area Services. In *Proc. USENIX OSDI*, San Francisco, CA, 2004.

[33] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the Constancy of Internet Path Properties. In *Proc. IMW*, 2001.

[34] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush. iSPY: Detecting IP Prefix Hijacking on My Own. In *Proc. ACM SIGCOMM*, 2008.