Predicting Internet Path Dynamics and Performance with Machine Learning

Sarah Wassermann*, Pedro Casas†, Thibaut Cuvelier*, Benoit Donnet*

*Université de Liège, Belgium
sarah.wassermann@student.ulg.ac.be, tcuvelier@ulg.ac.be, benoit.donnet@ulg.ac.be

†AIT Austrian Institute of Technology, Austria
pedro.casas@ait.ac.at

***Abstract*—In this paper, we study the problem of predicting Internet path changes and path performance using `traceroute` measurements and machine learning models. Path changes are frequently linked to path inflation and performance degradation; therefore, predicting their occurrence is highly relevant for performance monitoring and dynamic traffic engineering. We introduce NETPerfTrace, an Internet Path Tracking system capable of forecasting path changes and path latency variations. By relying on decision trees and using empirical distribution based input features, we show that NETPerfTrace can predict (i) the remaining life time of a path before it actually changes and (ii) the number of path changes in a certain time-slot with high accuracy. Through extensive evaluation, we demonstrate that NETPerfTrace highly outperforms DTRACK, a previous system with the same prediction targets. NETPerfTrace also offers path performance forecasting capabilities. In particular, it can predict path latency metrics, providing a system which could not only predict path changes but also forecast their impact in terms of performance variations. As an additional contribution, we release NETPerfTrace as open software to the networking community.**

***Keywords*—Traceroute; Machine Learning; Prediction; Benchmarking; M-Lab; DTRACK.**

I. Introduction

Internet paths change frequently due to inter/intra-domain routing changes, load balancing and even misconfigurations and failures [1]. Some of these changes can seriously disrupt performance, causing longer round-trip times, congestion, or even loss of connectivity [2]. For example, in [3], Google reports that inter-domain routing changes caused more than 40% of the cases in which clients experienced a latency increase of at least 100 ms. These changes could not only impact the QoE of the end users, but also might turn to be quite costly: Amazon claims that every additional 100 ms of page load time could cost them 1% of their sales [4], and that a page load slowdown of just one second could turn into a $1.6 billion loss in sales each year [5]. Google has also calculated that, by slowing their search results down by 400 ms, they could lose 8 million searches per day, meaning they would serve up many millions fewer advertisements [5]. As such, predicting the time when a path is likely to change, as well as how such a change would impact end-to-end latency, becomes a highly relevant problem in practice.

The research leading to these results has been partially funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-129, "BigDAMA".

The most common approach to analyze Internet paths in the large-scale is by relying on active measurements. Systems such as DisNETPerf [6], iPlane [7], Reverse `traceroute` [8] and Sibyl [9] are all distributed measurement systems which rely on `traceroute` measurements to monitor Internet paths performance. Analyzing the performance of a certain path through active measurements requires to regularly measure or *sample* the path, by periodically launching traceroutes to retrieve relevant metrics. However, there is a constraint in how often measurements are performed, trading the accuracy of the analysis with the probing resource budget. As such, monitoring a large number of Internet paths through active measurements requires some smart ways to allocate a predefined probing budget. In particular, a desired property of an efficient path-sampling scheduling approach is to allocate measurements with finer granularity for more dynamic paths, and around those specific times when relevant path changes (i.e., causing performance degradation) are close to happen.

To this end, and similar to [10], [11], we propose to predict the time when a path change would occur by relying on `traceroute` measurements and supervised machine learning prediction models. We introduce NETPerfTrace, an Internet Path Tracking system capable of predicting paths with higher chances of change, forecasting the most likely time when these paths would actually change, as well as predicting their future path latency. Extensive evaluations using highly distributed `traceroute` measurements from M-Lab show that NETPerfTrace perfectly predicts (i) the remaining life time of a path (i.e., the time before a path change) in about 30% of the cases, (ii) the exact number of daily path changes in about 70% to 80% of the cases, and (iii) the average RTT of a path in about 50% of the cases. In addition, we show that NETPerfTrace highly outperforms DTRACK [10], [11], a previous system conceived to predict Internet path changes. In particular, NETPerfTrace outperforms DTRACK by a factor of 5 in forecasting the residual lifetime of a path with relative prediction errors below 10%, and by a factor of 7 in correctly predicting daily path changes. A closer look into results reveals that the input features used by NETPerfTrace have better forecasting power than those used by DTRACK, and that the selected prediction model is by far much better for the prediction task.

NETPerfTrace relies on a standard random forest model for prediction, which provides highly accurate results with very low computational overhead as compared to other evaluated models. In particular, we benchmark six different regression

models - including random forests, neural networks, SVM, linear regression, decision trees and bayesian regression, and select the best one for NETPerfTrace. We also perform extensive evaluation on the impact of different input features, studying the correlations between inputs and prediction targets, as well as by using wrapper and filter feature selection techniques.

NETPerfTrace is open-source and freely available on GitHub at https://github.com/SAWassermann/NETPerfTrace. The datasets used in this paper are also available at the GitHub repository, making all the results fully reproducible. We are currently extending our tool DisNETPerf [6] by adding an automatic approach to *dynamically* adapt the sampling rate of a path based on the remaining time until a next path change, similar to [10].

The remainder of this paper is organized as follows: Sec. II briefly reviews the related work. Sec. III describes the basic concepts behind NETPerfTrace, including the prediction targets and the corresponding input features. Sec. IV presents the benchmarking results related to the evaluation of multiple machine learning models, and reports initial results for NETPerfTrace using the most accurate model. Sec. V evaluates the impact of different input features, using multiple feature selection techniques. Sec. VI reports the results obtained in the comparative evaluation of NETPerfTrace and DTRACK. Finally, Sec. VII concludes this work.

II. Related work

There is a very rich literature in the problem of using `traceroute` measurements to track Internet path dynamics and performance. Since the early work of Paxson on the analysis of end-to-end Internet routing behavior [1], multiple research efforts have targeted the study of Internet paths at the large scale. Paxon's study was one of the first using a reasonably large number of distributed `traceroute` measurements to analyze relevant Internet routing and path properties such as stability, symmetry, and pathologies leading to performance degradation such as routing loops, misconfigurations and failures. He concluded that while Internet paths are heavily dominated by single long-lasting routes, the time periods over which routes persist show wide variation, ranging from seconds up to days. Closer in time, authors in [12] reappraised Paxson's results using larger datasets, and concluded that Paxson's observations on path stability still hold.

Systems such as DisNETPerf [6], iPlane [7], Reverse `traceroute` [8] and Sibyl [9] are all distributed measurement systems which rely on `traceroute` measurements to track and predict Internet paths performance. DisNETPerf and Reverse `traceroute` particularly target the problem of measuring paths from arbitrary selected sources. iPlane and Sibyl both offer a service for predicting the performance of Internet paths, by building a structural model of the Internet using `traceroute` and opportunistic measurements. In a nutshell, these systems combine multiple historical `traceroute` measurements with prediction techniques to reconstruct measurements on segments not necessarily measured before.

While the problem of analyzing path changes at the Internet scale has attracted important attention in the past, only few papers have focused on predicting such path changes [2], [3], [10], [11], which is the target of this paper. Papers such as [2], [3] study the potential causes leading to Internet path changes, particularly those causing highly increasing latency [3]. Close to our work, authors in [10], [11] study the problem of predicting path changes using both `traceroute` measurements and machine-learning based predictors. In particular, they develop a model based on K nearest-neighbors to predict both the remaining time of a established path before a change and the number of changes experienced by a path on a certain time period. Our work builds on these papers, using different modeling techniques and different input features for prediction.

Finally, in terms of predicting end-to-end path performance using machine learning models, papers such as [13], [14] build models to predict the RTT of a TCP connection at a small time scale, to better optimize the TCP protocol. Our path latency prediction problem is similar, but our target is on path performance and not on TCP optimization. In addition, we operate at the time granularity provided by the `traceroute` -based sampling of a path, and not at a per RTT granularity provided by a TCP connection.

This paper is an extension of our early work on path dynamics and performance prediction [15], where we presented some first results of the techniques described next.

III. Predicting Path Changes & Performance

In this section, we introduce some basic definitions to formulate the corresponding learning and prediction problem behind NETPerfTrace. We define a path P as a sequence of links connecting a certain fixed source s to a fixed destination d. At any time t, path $P(t)$ is realized by a specific route r: this route consists of a specific sequence of links connecting s to d, and has an associated initial time t_0 when the route becomes active or in-place, and a final time t_f which corresponds to the time when r changes to another route realization, i.e., when the actual route changes. From now on, we therefore refer to route changes instead of path changes. As such, a path $P(t)$ can be considered as a statistical time process, composed of a set of time-contiguous routes $r_i(t_0^i, t_f^i)$, each one with a duration $D(r_i) = t_f^i - t_0^i$. For the sake of notation, we say that $r_i \in P$.

We additionally define the duration of a route r as $D(r) = t_f - t_0$, its current life time or *route age* at time t as $L_r(t) = t - t_0$, and its remaining life (i.e., time before a route change) at time t as $R_r(t) = t_f - t$. Finally, we define $rc_P(t)$ as the total number of route changes observed so far at time t for path P and $rc_{P_T}(t)$ as the number of route changes observed so far at time t for path P in the current time-slot T.

Given a new `traceroute` measurement at time t, the prediction problem solved by NETPerfTrace includes three prediction targets: (i) the remaining life time $R_r(t)$ of route r, namely $\widehat{R}_r(t)$, (ii) the number of route changes a path P experiences over a specific future time-window of length T, defined as $\widehat{rc}_{P_T}$, and (iii) the average RTT that path P will experience in the next `traceroute` measurement, defined as $\widehat{avgRTT}_P(t+\varepsilon)$, where ε represents the duration until the next measurement. The first two targets correspond to path dynamics prediction, whereas the third target consists of path performance forecasting. In practice, when $\widehat{R}_r(t)$ comes closer to zero, we would increase the sampling rate to better

monitor the path performance in the event of a route change. Predicting $\widehat{rc}_{P_T}$ allows to dynamically identify which paths are more prone to frequent changes, and thus better allocate new `traceroute` measurements. Based on previous results on route stability [1], [12] and similar to [10], we focus on predicting the number of daily route changes for the next day, i.e., $T = 24$ hours from now on. At last, predicting the average RTT that a certain path P would experience next becomes highly relevant for dynamic traffic engineering purposes, and when combined with the prediction of route changes, it can provide a very powerful approach to forecast those performance-harmful route changes. We do not explore this combined analysis approach in this paper and leave it for future work.

To predict these three targets we use a rich set of input features describing the statistical properties of route dynamics and path latency. Tab. I describe these features, separated into three different groups. Note that we compute all these features from the raw `traceroute` measurements performed in an observation learning period T_{learn} of the monitored paths, during which we extract the following statistics for learning purposes. The first group of features, referred to as F_A, includes 11 features relevant to the prediction of $R_r(t)$. These features describe the statistical properties of the route duration $D(r)$ observed for each path P. More precisely, we compute the average duration of the corresponding routes, the shortest and longest observed Dr, and different percentiles for this metric. F_A also includes information about the currently active route r at time t, namely its route age $L_r(t)$.

The second group of features, referred to as F_B, includes 14 features relevant to the prediction of rc_{P_T}. F_B features take into account the statistical properties of rc_{P_T}, including the average, minimum, maximum, and different percentiles, as well as the total number of route changes observed in T_{learn}, the total number of route changes in current time slot T, and the number of route changes observed at time t within current time slot T. A binary feature indicating the occurrence of a route change in current time slot T is also included in F_B.

The third group of features, referred to as F_C, includes 44 features relevant to the prediction of $avgRTT_P(t+\varepsilon)$. F_C features account for the statistical properties (avg, min, max and percentiles) of the 4 RTT metrics reported in `traceroute` measurements, namely the average, minimum, maximum and standard deviation of the `traceroute` RTT. In addition, F_C also includes the current value of `traceroute` RTT metrics at time t, i.e., $avgRTT_P(t)$, $minRTT_P(t)$, $maxRTT_P(t)$ and $devRTT_P(t)$.

As we show next, these features are highly correlated to the corresponding prediction targets, resulting in a strong forecasting power.

IV. NETPERFTRACE ANALYSIS & PERFORMANCE

In this section, we present an in-depth analysis of the performance achieved by NETPerfTrace, considering different machine learning models. Firstly, we introduce the evaluation dataset and study the correlation among input features and prediction targets. Next, we benchmark several machine learning models and select the one which fits the best our prediction goals. Finally, using the best machine learning model, we assess the prediction power of NETPerfTrace by comparing the real and predicted values for the three targets.

Residual Life Time R_r feature set (F_A)	11
average of $D(r_i)$, $\forall r_i \in P$	1
minimum of $D(r_i)$, $\forall r_i \in P$	1
maximum of $D(r_i)$, $\forall r_i \in P$	1
5-, 10-, 25-, 50-, 75-, 90-, 95-percentiles of $D(r_i)$, $\forall r_i \in P$	7
$L_r(t)$: route age of route r at timer t for P	1
# Route Changes rc_{P_T} feature set (F_B)	14
average of rc_{P_T} in P	1
minimum of rc_{P_T} in P	1
maximum of rc_{P_T} in P	1
5-, 10-, 25-, 50-, 75-, 90-, 95-percentiles of rc_{P_T} in P	7
total number of route changes in P	1
total number of route changes in P in T	1
$rc_{P_T}(t)$: number of route changes in P at time t in T	1
binary indication of a route change in T	1
Path Latency $avgRTT_P$ feature set (F_C)	44
average of RTT stats in P: $mean\,(avg./max/min/dev\,RTT)$	4
minimum of RTT stats in P: $min\,(avg./max/min/dev\,RTT)$	4
maximum of RTT stats in P: $max\,(avg./max/min/dev\,RTT)$	4
5-, 10-, 25-, 50-, 75-, 90-, 95-percentiles of RTT stats ($avg./max/min/dev\,RTT$) in P	28
current RTT stats ($avg./max/min/dev\,RTT$) at time t	4

Table I. FEATURE SET USED BY NETPERFTRACE. THE FULL SET INCLUDES 69 FEATURES. FEATURE SETS F_A, F_B AND F_C INCLUDE 11, 14 AND 44 DISJOINT FEATURES RESPECTIVELY.

A. M-Lab Data Description

For the purpose of this study, we analyze a full week of Paris-`traceroute` measurements performed through the M-Lab[1] open Internet measurement initiative. The M-Lab infrastructure consists of a high number of servers distributed globally in multiple provider networks and geographic regions, mostly in the US. M-Lab makes all data available, including packet traces and supplementary path measurements data. The raw data files are publicly available through Google's BigQuery and Cloud Storage, see https://console.cloud.google.com/storage/browser/m-lab/.

The analyzed dataset corresponds to the first week of January 2016. During this week, we observe more than 450,000 different paths, sampled through Paris-`traceroute` measurements from more than 180 geo-distributed servers. Unfortunately, not all of these paths are periodically sampled during this week, as M-Lab `traceroute` measurements are normally triggered as part of other experiments; indeed, when analyzing the number of `traceroute` measurements for each of these paths, we found that only 15,725 paths have been sampled more than 10 times, and only 2346 paths have at least 100 `traceroute` associated measurements during the analyzed week. We use 100 as threshold to avoid reducing the useful dataset even more, but naturally, the more `traceroute` measurements or samples we have for a path, the higher the visibility on potential route changes. Having 100 samples in a week means a minimum path sampling rate of one `traceroute` every 100 minutes, which is quite low but a good starting point for the different analyses. Actually the time between `traceroute` measurements in the resulting dataset

[1]https://www.measurementlab.net/

is below 14 minutes for more than 50% of the measurements, and for more than 40% of the paths, the sampling rate is above one `traceroute` every 20 minutes. The total number of `traceroute` measurements in the resulting filtered dataset is above 550,000.

Regarding paths topology, the resulting 2346 paths are issued from 82 different sources, distributed in 33 different ASes to about 2000 different destinations in 125 different ASes. These paths traverse more than 260 different ASes, and have an average length of 10 hops and 4 ASes.

For each of these 2346 paths P, we compute the distribution of the aforementioned input features during an observation period $T_{learn} = 1$ week. Note that while we use the full week of measurements to compute the input features for NETPerfTrace, all performed evaluations in this paper are done on a 10-fold cross-validation basis, to avoid biased results.

B. Initial Feature Analysis

Let us start by analyzing the correlations among input features and prediction targets. This would let us perform a first raw selection of features for each prediction target. Fig. 1 depicts the Pearson linear correlation coefficients (PLCCs) between the full set of input features and the three prediction targets, discriminated by feature set F_A, F_B and F_C. The set is extended by adding the three prediction targets, which are flagged by a PLCC = 1 in the corresponding plot. As expected, features from each set present high positive correlation to the corresponding prediction target. Features from sets F_A and F_B are inversely correlated to targets rc_{P_T} and R_r respectively, which is coherent with the fact that more stable paths with smaller number of changes have longer life times. In addition, there is negligible correlation between path stability and path performance; indeed, features from set F_C are very weakly correlated to targets R_r and rc_{P_T}, and features from sets F_A and F_B are very weakly correlated to $avgRTT_P$.

Based on these initial observations, we shall consider each set of features F_A, F_B and F_C as individual inputs to predict R_r, rc_{P_T} and $avgRTT_P$ respectively: $\widehat{R_r}(t) =$ NETPerfTrace(F_A), $\widehat{rc}_{P_T} =$ NETPerfTrace(F_B), and $\widehat{avgRTT}_P(t+\varepsilon) =$ NETPerfTrace(F_C) for the rest of this section. Later on we show in Sec. V that a more careful feature selection can improve the performance of NETPerfTrace.

C. Benchmarking Different ML Models

We now evaluate different machine learning models to find the most appropriate one for NETPerfTrace. We benchmark seven different machine learning regression models, including decision trees (CART), random forests (RF) with 10 and 100 trees, support vector machines for regression with a Gaussian kernel (SVR), Bayesian ridge regression (BRR), linear regression (LR), and multi-layer perceptron neural networks (MLP). We compare these algorithms on the basis of PLCC coefficients, mean absolute error MAE = $mean(|\hat{X} - X|)$, root mean squared error RMSE = $\sqrt{mean((\hat{X} - X)^2)}$, and mean relative absolute error MRE = $mean(|\hat{X} - X|/X)$, where X and $\hat{X}$ are real and predicted values respectively. The MAE metric penalizes all the errors equally, whereas the RMSE metric puts a relatively high weight on larger errors.

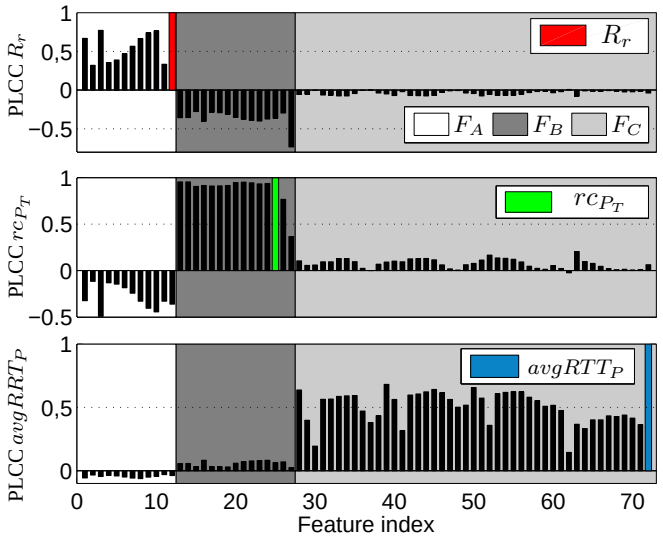


Figure 1. Linear correlation between input features and the three prediction targets, for feature sets F_A, F_B and F_C.

Benchmarking results are reported in Tabs. II, III, and IV for the three prediction targets respectively. Tree-based predictors perform the best. For the three targets, RF models yield the lowest MAE, and also the lowest RMSE and MRE in most cases. Interestingly, different models are not equally well suited for the three prediction problems. For instance, SVR performs very badly in predicting R_r and $avgRTT_P$, but highly improves in forecasting rc_{P_T}. The analysis also confirms our intuition that we are not confronted with linear regression problems, as both LR and BRR models are clearly outperformed by RF models.

When comparing the performance for each of the three prediction targets, we observe that predicting both R_r and $avgRTT_P$ is more challenging than predicting rc_{P_T}. Indeed, PLCCs are much higher and MREs much smaller in the latter case. In particular, and as already pointed out by previous work [10], [11], predicting R_r is difficult and error-prone.

Note that, in the case of rc_{P_T} prediction, we might have zero route change slots for which $rc_{P_T} = 0$; indeed, the fraction of stable routes is not negligible. About 25% of the 24hours time slots correspond to zero route change slots in the studied dataset. We leave those cases out of the computation of the MREs, and treat them independently. For this reason, Tab. III includes two additional metrics: the True Prediction Rate (TPR) when predicting zero route-change slots - TPR_{0rc}, and the TPR for all predictions - TPR_{rc}. RF models achieve the lowest MREs - 16% with almost perfect alignment between real and predicted values; MAEs are below 1 route change whereas RMSEs are below 3 changes. They correctly predict 38% of the zero route-change slots, and achieve an overall TPR of 60%. The MLP model yields a surprisingly high TPR_{rc} = 90%, but the other error metrics are poor, suggesting potential overfitting.

Besides prediction performance, Tab. IV additionally reports the total computation time taken by each model in the 10-fold cross-validation process, for the specific prediction of $avgRTT_P$ (similar results were obtained for the other two targets). Computations were performed in a single machine equipped with two Intel Xeon E5-2650 v4 processors (30M

Model	PLCC	MAE (s)	RMSE (s)	MRE (%)
BRR	0.86	31.5	54	466
LR	0.86	31.5	54	466
CART	0.81	23.7	64	244
RF (10 trees)	0.86	21.0	54	230
RF (100 trees)	0.87	20.7	53	230
MLP	0.89	26.5	48	446
SVR	0.19	58.1	117	335

Table II. BENCHMARKING OF DIFFERENT MODELS FOR PREDICTING ROUTE RESIDUAL LIFE TIME.

Cache, 2.20 GHz) including 12 physical cores and 128 GB of RAM. Out of the seven models, five required less than three minutes, while the MLP took approximately half an hour and the SVR even more than one day. This clearly shows that SVR is not suitable when the learning phase should be done in near real-time, for example, when targeting a more dynamic, periodic learning approach.

As a general conclusion, and based both on prediction performance and computational speed, we select RF as the underlying prediction model for NETPerfTrace. In particular, we take a RF model with 10 trees (RF10), which achieves almost the same performance as RF100, with a much smaller and less complex structure. This model will remain the core prediction engine used by NETPerfTrace for the rest of the paper. Next, we present a more detailed evaluation of NETPerfTrace using the RF10 model.

D. NETPerfTrace Performance with RF10 Prediction Model

Fig. 2 presents a closer look into the prediction performance achieved by NETPerfTrace using RF10 as the underlying model, and input features sets F_A, F_B and F_C for predicting R_r, rc_{P_T} and $avgRTT_P$ respectively. Figs. 2(a) and 2(d) report the (a) normalized real and predicted values for R_r and (d) the distribution of the relative prediction errors. NETPerfTrace correctly predicts R_r for about 20% of the samples, and achieves relative prediction errors below 100% for more than 70% of the samples. Fig. 2(a) suggests that NETPerfTrace predicts shorter residual life times worse. Finally, we found that NETPerfTrace underestimates R_r for about 40% of the samples.

Figs. 2(b) and 2(e) report the (b) normalized real and predicted values for rc_{P_T} and (e) the distribution of the relative prediction errors. Relative prediction errors are small, with about 70% of the samples being perfectly predicted and more than 90% of them with relative errors below 50%. As we said before, zero route-change cases are not included in Fig. 2(e), and the model correctly predicts 38% of the zero route-change slots, achieving an overall TPR of 60%.

Finally, Figs. 2(c) and 2(f) report the (c) normalized real and predicted values for $avgRTT_P$ and (f) the distribution of the relative prediction errors. In this case, relative prediction errors are almost zero for about 50% of the samples, and below 30% for almost 90% of them. Given that $avgRTT_P$ values are in general very small - below 130 ms for more than 75% of the samples, such small relative prediction errors are highly satisfactory.

V. IMPROVING NETPERFTRACE BY FEATURE SELECTION

In this section, we analyze in more detail the relevance of each of the used input features in terms of prediction power, and apply different feature selection techniques to select the most relevant ones for each prediction target. We consider two different feature selection approaches: a filter approach based on mutual information, and a wrapper approach based on a RF10 model. Whereas feature selection based on filter approaches evaluates the worth of a subset of features independently of the considered prediction model, wrapper approaches rank features based on their prediction power for a specific prediction model, in this case RF10.

A. Feature selection within independent feature sets

We start by selecting the most relevant features from each independent feature set F_A, F_B and F_C, for the three corresponding prediction targets. As expected, both feature selection approaches do not assign the same importance to each feature. In general terms, the wrapper approach is the most discriminative one, as it clearly splits the three input sets between relevant and irrelevant or less powerful features. On the contrary, filter-based selection does not provide a clear cut revealing the most relevant features. For example, wrapper-based selection takes the average number of route changes in T and the tail of its distribution as the most relevant features to predict rc_{P_T}. Also features providing information about the currently observed route (e.g., route age), time slot (e.g., current number of route changes) and `traceroute` sample (e.g., current $avgRTT_P$) are among the most important features.

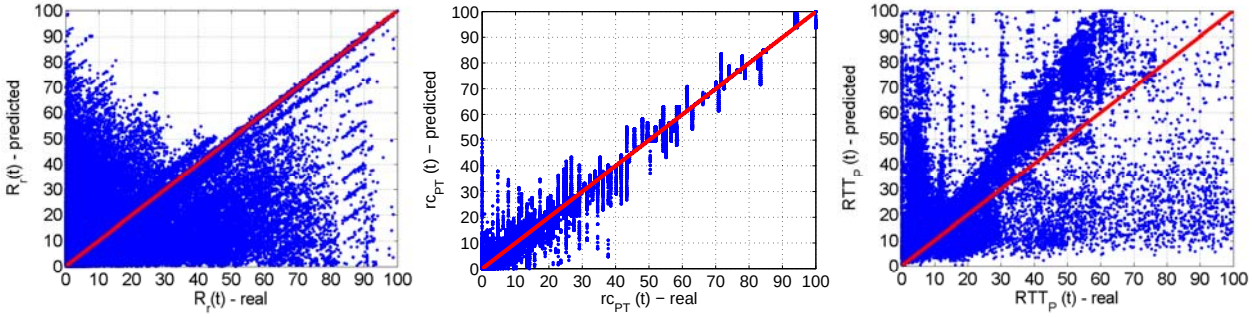
To verify the relevance of the selected features by both approaches, we compare the performance of NETPerfTrace using as input all the features of each independent set against (i) the top 5 features of each feature set as selected by the wrapper approach and (ii) the top 5 features selected by the filter approach. Fig. 3 depicts the obtained results in terms of relative prediction errors for (a) $R_r(t)$, (b) rc_{P_T} and (c) $avgRTT_P$. While the top 5 features selected by filter-based selection drastically reduce prediction performance for the three prediction targets, those features selected by wrapper-based selection provide almost the same results as the complete input sets F_A, F_B and F_C respectively. This shows that many of the input features used within each independent feature set are irrelevant for the prediction of each of the three targets. In particular, 6 out of 11 features for set F_A, 9 out of 14 features for set F_B and 39 out of 44 features for set F_C have a negligible impact on the prediction performance.

B. Feature selection using the full feature set

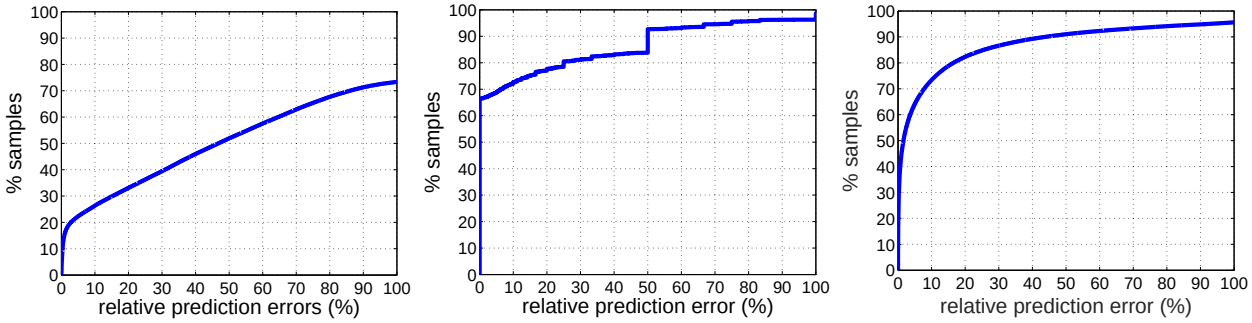
So far, we have tested NETPerfTrace using a split of features into groups F_A, F_B and F_C. However, based on the initial feature correlation results reported in Fig. 1, there is strong correlation between features of group F_A and F_B for the prediction of both $R_r(t)$ and rc_{P_T}, which could be exploited to improve prediction performance. We therefore explore now the performance of NETPerfTrace when using as input the full set of 69 input features $F_A \cup F_B \cup F_C$, and perform wrapper-based feature selection on top of this full set.

Model	PLCC	MAE (#)	RMSE (#)	MRE (%)	TPR_{0rc} (%)	TPR_{rc} (%)
BRR	0.96	3.89	6.77	48	34	17
LR	0.96	3.89	6.77	48	34	17
CART	0.99	0.89	2.43	16	38	60
RF (10 trees)	1.00	0.88	2.32	16	38	60
RF (100 trees)	1.00	0.87	2.31	16	38	60
MLP	0.96	4.01	6.75	51	62	91
SVR	0.96	3.47	6.70	42	74	30

Table III. BENCHMARKING OF DIFFERENT MODELS FOR PREDICTING THE NUMBER OF ROUTE CHANGES IN THE NEXT 24H TIME SLOT.

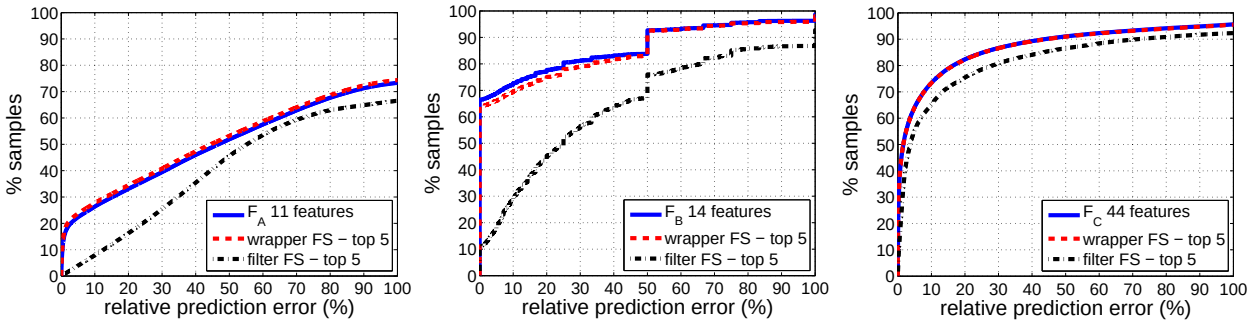


(a) $R_r(t)$ prediction (normalized values). (b) rc_{P_T} prediction (normalized values). (c) $avgRTT_P$ prediction (normalized values).



(d) $R_r(t)$ relative prediction errors. (e) rc_{P_T} relative prediction errors. (f) $avgRTT_P$ relative prediction errors.

Figure 2. Real vs. predicted values for the prediction of (a) $R_r(t)$, (b) $rc_{P_T}(t)$ and (c) $avgRTT_P(t)$. Relative errors for the prediction of (d) $R_r(t)$, (e) $rc_{P_T}(t)$ and (f) $avgRTT_P(t)$. NETPerfTrace uses RF10 and input features sets F_A, F_B and F_C for predicting R_r, rc_{P_T} and $avgRTT_P$ respectively.



(a) FS for prediction of $R_r(t)$. (b) FS for prediction of rc_{P_T}. (c) FS for prediction of $avgRTT_P$.

Figure 3. Relative errors for the prediction of (a) $R_r(t)$, (b) $rc_{P_T}(t)$ and (c) $RTT_P(t+\varepsilon)$ using all the features of each independent set against the top 5 features selected by the wrapper approach and the top 5 features selected by the filter approach.

Tab. V reports the top 5 features selected by wrapper-based selection out of the full set of features - we refer to these as 5/69 features, for the three prediction targets. We can easily spot out that the most important features are not necessarily the ones included in the subsets F_A, F_B, and F_C. A striking example are the top five features selected for predicting $R_r(t)$: only two out of the five features were already in the subset F_A. The other three are related to the number of route changes, included in F_B. We can see that features in F_A also help estimate rc_{P_T}. However, as expected, features in set F_C play a significant role only for the prediction of $avgRTT_P$.

To verify the prediction properties of the selected features, Fig. 4 reports the relative prediction errors for (a) $R_r(t)$, (b)

Model	PLCC	MAE (ms)	RMSE (ms)	MRE (%)	CT (s)
BRR	0.69	25.2	127	186	11
LR	0.69	25.2	127	186	9
CART	0.54	27.4	164	140	31
RF (10 trees)	0.63	25.1	140	139	28
RF (100 trees)	0.63	25	140	138	180
MLP	0.70	32.3	130	142	2.2 e3
SVR	0.38	39.8	171	53	97.2 e3

Table IV. BENCHMARKING OF DIFFERENT MODELS FOR PREDICTING THE AVERAGE RTT OF THE NEXT TRACEROUTE MEASUREMENT.

rc_{P_T} and (c) $avgRTT_P$, when considering (i) the features on each independent set (i.e, F_A, F_B and F_C), (ii) the full set of 69 features and (iii) the top 5/69 features reported in Tab. V. The performance increase for the prediction of $R_r(t)$ w.r.t. the one achieved with F_A features is astonishing, and just by using the top 5/69 features there is a major reduction in the relative prediction errors. Indeed, Fig. 4(a) shows that relative prediction errors are almost zero for about 30% of the samples with 5/69 features, and below 60% for about 80% of the samples. The MAE obtained with 5/69 inputs is 6.2 seconds, which is more than 3 times smaller than the MAE = 21 seconds attained with F_A features (cf., Tab. II). There is also a significant improvement in the other evaluation metrics: the PLCC goes up to 0.98, the RMSE decreases from 54 to 22 seconds, and the MRE goes down from 230% to 70%. Using the full set of 69 features reduces even more the MAE and the RMSE - by about 20%, but there are no significant changes in the relative prediction errors, thus it is not worth considering such a huge input set.

Regarding the estimation of rc_{P_T}, 4(b) shows that the top 5/69 features do not provide any relevant improvement w.r.t. F_B features. However, in this case there is a significant improvement when considering the full set of 69 features. Overall, the TPR$_rc$ increases from 60% (cf., Tab. III) to 83%, and the distribution of relative prediction errors shows an important decrease. Still, for the sake of reducing the model complexity and the number of input features, the final release of NETPerfTrace uses the top 5/69 features as input. Finally, and as expected, there are no significant changes in the prediction performance of $avgRTT_P$ when using either the top 5/69 or the full set of features.

As a general conclusion of the feature selection analysis, besides using RF10 as underlying prediction model, the final implementation of NETPerfTrace uses the top 5/69 features reported in Tab. V as input for the prediction of the three corresponding targets.

VI. NETPERFTRACE VERSUS DTRACK

Now that we have finally found the best configuration of NETPerfTrace for path dynamics and performance prediction, we compare its performance with the state of the art. In particular, we compare NETPerfTrace to DTRACK [10], [11]. DTRACK predicts only path dynamics and not path performance, as its focus is on the prediction of $R_r(t)$ and rc_{P_T}. The system uses a Nearest Neighbors (NN) based model as underlying prediction model, and takes as input the four features described in Tab. VI. Note that the term *route prevalence* corresponds to the proportion of time a certain route is active. The authors of [10], [11] named their algorithm as NN4, as it works on four aforementioned features. In a nutshell, the NN4 algorithm of DTRACK works as follows: first, the feature space is partitioned into polyhedrons which number of dimensions is equal to the number of features. The bin boundaries of the different features are chosen as equally-spaced percentiles. In the performed evaluations we set the number of bins for each feature to 10, as chosen by the authors in [10], [11]. The discretization process goes as follows: for each feature, the first bin contains the samples which value is below the 10th percentile, the second bin the values between the 10th and the 20th percentiles, and so on. Targets $R_r(t)$ and rc_{P_T} are predicted for a `traceroute` sample s as the average of the real values of these metrics over the training samples contained in the the polyhedron including the feature vector of s. This algorithm is basically equivalent to a decision tree with a fixed choice of thresholds.

The comparison of NETPerfTrace vs. DTRACK is performed along three distinct dimensions: features, model and system. Firstly, we compare the input features used by both systems, using a NNX model (X = 4 for DTRACK and X = 5 for NETPerfTrace) and a RF10 model; secondly, we compare the properties of the underlying prediction models, by using NETPerfTrace input features and the two different prediction models - NN5 and RF10; finally, we directly compare NETPerfTrace and DTRACK systems, using their default configurations (i.e., models and input features).

A. NETPerfTrace features vs. DTRACK features

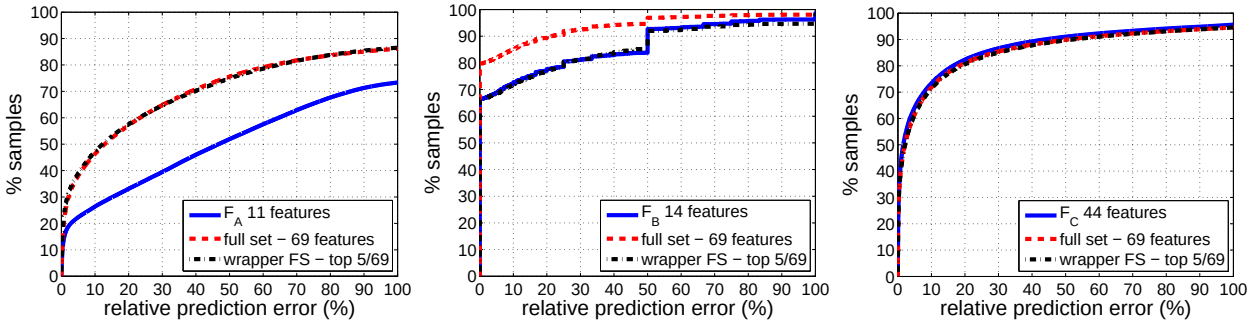
Fig. 5 compares the performance of NETPerfTrace and DTRACK using their corresponding input features and NNX as underlying prediction model. As shown in Fig. 5(a), there is only a slight reduction on the relative prediction errors for $R_r(t)$ when using NNX with NETPerfTrace top 5/69 input features (NPT-NN5) as compared to DTRACK features. Still, NPT-NN5 achieves a MAE of 20 seconds whereas DTRACK's MAE = 33 seconds, meaning a dramatic reduction. In addition, NPT-NN5 shows a PLCC = 0.94 vs. a PLCC = 0.81 for DTRACK. Fig. 5(b) shows that the performance improvement is much more relevant when considering the prediction of rc_{P_T}. NPT-NN5 correctly predicts more than 25% of the non-zero-change time slots, while DTRACK does it for only 10%. For TPR$_{0rc}$, NPT-NN5 rounds 47% of correct predications, whereas DTRACK's TPR$_{0rc}$ = 2%.

Repeating the same evaluations but using RF10 as underlying prediction model shows better results for both input feature sets. DTRACK's MAE for $R_r(t)$ prediction decreases from 33 seconds to 16 seconds when using RF10, whereas NETPerfTrace attains a MAE = 6 seconds. Interestingly, DTRACK becomes slightly better than NETPerfTrace when it comes to predict rc_{P_T}, but without a relevant difference.

As a first conclusion, the top 5/69 features used by NETPerfTrace provide in general much better results than those used by DTRACK.

B. NN5 vs. RF10 with NETPerfTrace

We now compare the prediction power of the two underlying models used by NETPerfTrace and DTRACK, using as input the top 5/69 features used by NETPerfTrace by default. Fig. 6 shows a significant performance improvement when using NETPerfTrace's RF10 model as compared to



(a) FS for prediction of $R_r(t)$. (b) FS for prediction of rc_{P_T}. (c) FS for prediction of $avgRTT_P$.

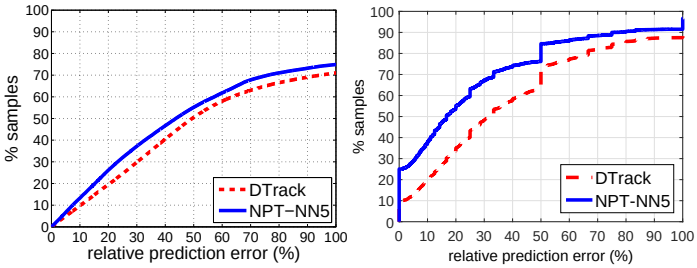
Figure 4. Relative errors for the prediction of (a) $R_r(t)$, (b) $rc_{P_T}(t)$ and (c) $RTT_P(t+\varepsilon)$ using all the features of each independent set against the full set of 69 features and the top 5 features selected by the wrapper approach out of the full set (i.e., 5/69).

Top 5 features	Residual life time	# route changes in timeslots	average RTT
#1 feature	# route changes in current time slot	head distribution # route changes	$mean(avgRTT_P)$
#2 feature	route age of current route	avg. # route changes in time slots	current $avgRTT_P$
#3 feature	max. of all $D(r_i)$	total # route changes	current $maxRTT$
#4 feature	route change binary flag	# route changes in current time slot	current $minRTT$
#5 feature	current # route changes in present time slot	avg. of $D(r_i)$, $\forall r_i \in P$	route age of current route

Table V. FEATURE SELECTION FOR THE THREE PREDICTION TARGETS WHEN CONSIDERING ALL THE 69 FEATURES.

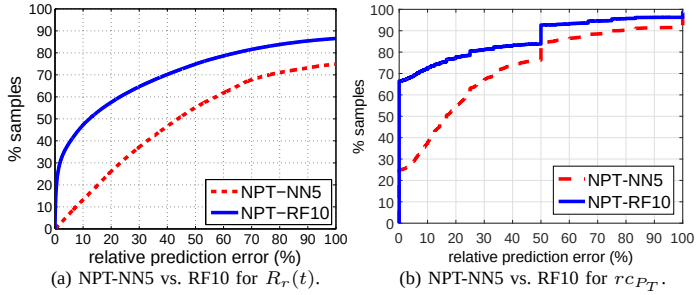
route age of route r
prevalence of route r in current time slot
number of previous occurrences of route r in current time slot for path P
total number of route changes in current time slot (rc_{P_T}) for path P

Table VI. FEATURE SET USED BY DTRACK



(a) NPT-NN5 vs. DTRACK for $R_r(t)$. (b) NPT-NN5 vs. DTRACK for rc_{P_T}.

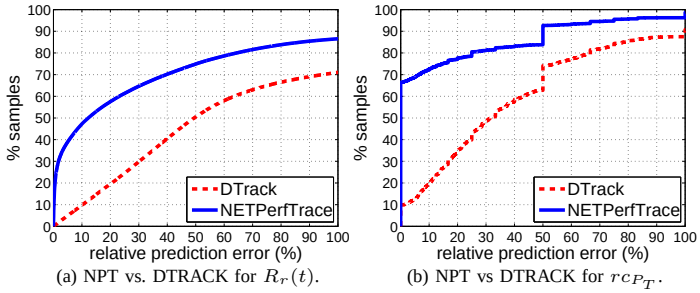
Figure 5. Performance of NETPerfTrace using NN5 vs. DTRACK.



(a) NPT-NN5 vs. RF10 for $R_r(t)$. (b) NPT-NN5 vs. RF10 for rc_{P_T}.

Figure 6. Performance of NETPerfTrace using NN5 and RF10 models.

DTRACK's NN5 model. For example, Fig. 6(a) shows that about 30% of the relative prediction errors are close to 0% when using RF10, whereas almost no zero relative prediction errors are observed for NN5. Fig. 6(b) shows that for nearly



(a) NPT vs. DTRACK for $R_r(t)$. (b) NPT vs DTRACK for rc_{P_T}.

Figure 7. NETPerfTrace vs. DTRACK. NETPerfTrace largely outperforms DTRACK for predicting path dynamics.

70% of the samples, RF10 predicts the correct number of non-zero route changes, which drops to only 25% for NN5. The prediction power of RF10 can be further underlined by the achieved MAE, which is as low as 0.92 vs. 3.55 for NN5.

As a second conclusion, the prediction model used by NETPerfTrace clearly outperforms the one used by DTRACK.

C. NETPerfTrace vs. DTRACK

To conclude the comparison, we now focus on the performance of both NETPerfTrace and DTRACK systems using their default configurations in terms of model and input features. Fig. 7 clearly shows that NETPerfTrace largely outperforms DTRACK for predicting path dynamics. According to Fig. 7(a), NETPerfTrace can predict $R_r(t)$ with relative errors below 10% for about 50% of the samples, whereas DTRACK only does so for 10% of the samples. In addition, almost 30% of the predictions with NETPerfTrace yield a relative error close to zero, whereas almost no zero relative prediction errors are observed for DTRACK. The PLCC of NETPerfTrace has a value of 0.97 while the one attained by DTRACK is only 0.81. Finally, NETPerfTrace's MAE is almost 80% smaller than the one obtained by DTRACK.

In terms of daily route changes, Fig. 7(b) shows that NETPerfTrace correctly predicts almost 70% of the non-zero route changes, whereas DTRACK falls to correctly predict only 10% of the changes. Overall, NETPerfTrace predicts the correct number of route changes TPR_{rc} for about 65% of the samples whereas DTRACK correctly does it for only 8% of the samples. Finally, the MAE is below 1 for NETPerfTrace and above 5 for DTRACK.

As a general conclusion, presented results evidence that NETPerfTrace largely outperforms DTRACK when forecasting both $R_r(t)$ and rc_{P_T}, by using only one additional feature to tackle both prediction problems. On the one hand, this is explained by the better prediction power of the selected features. Note that we have selected specific feature sets for the prediction of $R_r(t)$ and rc_{P_T} respectively, whereas DTRACK uses the same set of features for predicting both targets. On the other hand, NETPerfTrace relies on a much more powerful prediction model than DTRACK, which greatly contributes to the overall high accuracy of the system.

VII. Concluding Remarks

In this paper, we have addressed the problem of predicting Internet path changes and path performance using `traceroute` measurements and machine learning models. We have introduced and evaluated NETPerfTrace, an Internet Path Tracking system capable of forecasting (i) the remaining life time of a path before it actually changes, (ii) the daily number of path changes in the next day and (iii) the average RTT of the next `traceroute` measurement with relatively high accuracy. By carefully engineering NETPerfTrace underlying model and input features, we have shown that NETPerfTrace highly outperforms DTRACK, a previous system with the same prediction targets. In particular, NETPerfTrace outperforms DTRACK by a factor of 5 when forecasting the residual lifetime of a path with relative prediction errors below 10%, and by a factor of 7 in correctly predicting daily path changes. As an additional contribution, we have released NETPerfTrace as open software to the networking community.

References

[1] V. Paxson, "End-to-end routing behavior in the internet," in *Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '96. New York, NY, USA: ACM, 1996, pp. 25–38. [Online]. Available: http://doi.acm.org/10.1145/248156.248160

[2] U. Javed, I. Cunha, D. Choffnes, E. Katz-Bassett, T. Anderson, and A. Krishnamurthy, "Poiroot: Investigating the root cause of interdomain path changes," in *Proc. of the ACM SIGCOMM 2013*, August 2013, pp. 183–194.

[3] Y. Zhu, B. Helsley, J. Rexford, A. Siganporia, and S. Srinivasan, "Latlong: Diagnosing wide-area latency changes for CDNs," *IEEE Transactions on Network and Service Management*, vol. 9, no. 3, pp. 333–345, September 2012.

[4] G. Linden, "Make Data Useful," 2006, http://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-28.ppt.

[5] K. Eaton, "How One Second Could Cost Amazon $1.6 Billion in Sales," 2012, https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales.

[6] S. Wassermann, P. Casas, B. Donnet, G. Leduc, and M. Mellia, "On the Analysis of Internet Paths with DisNETPerf, a Distributed Paths Performance Analyzer," *Proc. 10th IEEE Workshop on Network Measurements (WNM)*, November 2016.

[7] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An information plane for distributed services," in *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[8] E. Katz-Bassett, H. Madhyastha, V. Adhikari, C. Scott, J. Sherry, P. van Wesep, A. Krishnamurthy, and T. Anderson, "Reverse traceroute," in *Proc. USENIX Symposium on Networked Systems Design and Implementations (NSDI)*, 2010.

[9] I. Cunha, P. Marchetta, M. Calder, Y.-C. Chiu, B. V. A. Machado, A. Pescapè, V. Giotsas, H. V. Madhyastha, and E. Katz-Bassett, "Sibyl: A practical internet route oracle," in *Proc. USENIX Symposium on Networked Systems Design and Implementations (NSDI)*, 2016, pp. 325–344.

[10] I. Cunha, R. Teixeira, D. Veitch, and C. Diot, "Predicting and tracking internet path changes," in *Proceedings of the ACM SIGCOMM 2011 Conference*, August 2011, pp. 122–133.

[11] ——, "DTRACK: A system to predict and track internet path changes," *IEEE/ACM Transations on Networking*, vol. 22, no. 4, pp. 1025–1038, August 2014.

[12] Í. Cunha, R. Teixeira, and C. Diot, *Measuring and Characterizing End-to-End Route Dynamics in the Presence of Load Balancing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 235–244. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-19260-9_24

[13] W. Hu, Z. Wang, and L. Sun, "Guyot: a Hybrid Learning- and Model-based RTT Predictive Approach," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 5884–5889.

[14] B. A. Arouche Nunes, K. Veenstra, W. Ballenthin, S. Lukin, and K. Obraczka, "A Machine Learning Framework for TCP Round-Trip Time Estimation," *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, no. 1, p. 47, 2014. [Online]. Available: http://dx.doi.org/10.1186/1687-1499-2014-47

[15] S. Wassermann, P. Casas, and B. Donnet, "Machine Learning based Prediction of Internet Path Dynamics," *CoNEXT 2016 Student Workshop*, 2016.