Prediction Models for Long-Term Internet Prefix Availability

Ravish Khosla, Sonia Fahmy, Y. Charlie Hu, Jennifer Neville [1]
Purdue University
Email: ravish_03@yahoo.com,{fahmy, ychu, neville}@purdue.edu

Abstract

The Border Gateway Protocol (BGP) maintains inter-domain routing information by announcing and withdrawing IP prefixes. These routing updates can cause prefixes to be unreachable for periods of time, reducing prefix availability observed from different vantage points on the Internet. The observed prefix availability values may not meet the standards promised by Service Level Agreements (SLAs).

In this paper, we develop a framework for predicting long-term availability of prefixes, given short-duration prefix information from publicly available BGP routing databases like RouteViews, and prediction models constructed from information about other Internet prefixes. We compare three prediction models and find that machine learning-based prediction methods outperform a baseline model that predicts the future availability of a prefix to be the same as its past availability. Our results show that mean time to failure is the most important attribute for predicting availability. We also quantify how prefix availability is related to prefix length and update frequency. Our prediction models achieve 82% accuracy and 0.7 ranking quality when predicting for a future duration equal to the learning duration. We can also predict for a longer future duration, with graceful performance reduction. Our models allow ISPs to adjust BGP routing policies if predicted availability is low, and are generally useful for cloud computing systems, content distribution networks, P2P, and VoIP applications.

Keywords: Prefix Availability, Border Gateway Protocol (BGP), Prediction, Machine Learning

1. Introduction

The Border Gateway Protocol (BGP), the de-facto Internet inter-domain routing protocol, propagates reachability information by announcing paths to prefixes, which are aggregates of IP addresses. Autonomous Systems (ASes) maintain paths to prefixes in their routing tables, and (conditionally) update this information when route update messages are received. These update messages can be *announcements*, which announce an AS path to a prefix, or *withdrawals*, which indicate that no path is available to the prefix. Continuous prefix reachability over time is crucial for the smooth operation of the Internet. This is captured using the metric of *availability*, defined as the time duration

[1]Ravish Khosla, the corresponding author (Tel:+1 (765) 325-7284, Fax: +1 (765) 494-0739), and Prof. Y. Charlie Hu are with the Department of Electrical and Computer Engineering, 465 Northwestern Avenue, West Lafayette, IN 47907, USA. Prof. Sonia Fahmy and Prof. Jennifer Neville are with the Department of Computer Science, 305 N. University St., West Lafayette, IN 47907–2107, USA.

when the prefix is deemed reachable divided by the total time duration we are interested in. While typical system availability metrics for telephone networks exceed *five 9s*, i.e., 99.999%, computer networks are known to have lower availability [1, 2, 3]. The five 9s availability value amounts to the system being down for about five minutes in a year's period and is usually too stringent a requirement for Internet prefixes.

Prefixes belonging to highly popular services such as CNN, Google, and YouTube need to be highly available, and a disruption of more than a few minutes is generally unacceptable. Internet Service Providers (ISPs) such as AT&T and Sprint usually provide availability guarantees on their backbone network through Service Level Agreements (SLAs) [4, 5]. However, content providers are more interested in their website availability as observed from various points in the Internet, and a routing path being advertised is critical to maintaining traffic flow to their data centers. Attempts at defining policies so that SLAs can be extended to several ISPs [6] and at defining and estimating service availability between two end points [7] in the Internet have had limited success. Meanwhile, several reachability problems have occurred, such as the YouTube prefix hijack which lasted about two hours [8], and several undersea cable cuts, e.g., [9, 10], which caused significant disruptions and increase in web latencies to much of the Middle East, Asia, and North Africa for a period of several weeks [11].

Measuring prefix availability is non-trivial without an extensive measurement infrastructure comprising many vantage points. Additionally, *data plane* measurements are inherently discontinuous, as they take reachability samples at periodic time instants. The reachability estimate they compute increases in accuracy as the sampling interval is made smaller, at the cost of increased burden on the prober and elevated network traffic. Moreover, the observations need to be made over a long period of time to obtain a reasonable estimate. A shortfall in measured availability requires a *reactive* approach that corrects the problem after the fact. Our work takes a *predictive* approach to solve the *availability prediction problem*, i.e., predicting the advertised availability of prefixes, as observed from multiple vantage points in the Internet.

Our framework predicts long-term *control plane* availability, i.e., the availability of the paths to prefixes as advertised by BGP. However, previous work has shown that the control plane-advertised paths may not always imply that the paths are usable in the *data plane* [12, 13, 14]. Wang et al. [14] studied the correlation between control plane and data plane events and found that control plane changes mostly result in data plane performance degradation, showing that the two planes are correlated. BGP routing dynamics have been used to predict data plane failures in previous work [13, 15]. Zhang et al. [13] found that data plane failures can be predicted using routing updates with about 80-90% accuracy for about 60-70% of the prefixes. Feamster et al. [15] predict end-to-end path failure using the number of BGP messages observed during a 15 minute window. This indicates that the control plane does indeed have a positive correlation with the data plane.

Transient events like routing convergence and forwarding loops result in temporary reachability loss in the data plane, most of which last less than 300 seconds [13]. However, since we are concerned with the long term availability metric considering at least a few days at a time, the percentage of time that the control plane and data plane paths mismatch should be insignificant compared to the time over which our availability values are computed.

Data plane reachability can exist even when control plane paths are withdrawn due to the presence of default routes [12]. However, it is not possible to predict the existence of default routes, as they depend on intermediate ASes between the source and the destination. There is no agreed upon method to detect the existence of default routes, though some initial efforts have been made by the authors of [12] by controlling announcements and withdrawals of certain prefixes allocated to their ASes. Our work considers only control plane availability and hence actual prefix availability could be higher in the data plane if default routes are present. As can be seen from the discussion above, establishing the correlation between the two planes is by itself a challenging topic [12] and detailed study of this is beyond the scope of this work.

In this work, we compute attributes during a short duration observation period of publicly available routing information (e.g., from RouteViews [16]) and develop a prediction model based on information on other Internet prefixes. Thus, our approach does not need additional measurement infrastructure apart from RouteViews [16], which has been maintained by the University of Oregon for several years.

A predicted long-term advertised availability value which falls short of requirements could lead to changes in BGP policies of the ISP regulating the advertisement of these prefixes to the rest of the Internet. For example, one can increase the penalty threshold associated with route flap damping for the routes to a high availability requirement prefix (like a business customer) to ensure higher availability [17]. Changing BGP attributes such as MED and community, or aggregating prefixes, can increase the perceived prefix availability or aid traffic engineering [17]. We will make our prediction tool publicly available through a web page so it can be used for monitoring the predicted availability, e.g., of prefixes of an ISP.

Our work can optimize Hubble [18]– a system that studies black holes in the Internet by issuing traceroutes to potentially problem prefixes, and then analyzing the results to identify data plane reachability problems. Currently, Hubble uses BGP updates for a prefix as one of the potential indicators of problems, focusing on withdrawals and AS path changes. We can enhance this technique by using the prefixes for which the predicted availability falls below a threshold as the potentially problem prefixes. This will increase detection accuracy of black holes. Our work also complements a data plane loss rate prediction system such as iPlane [19].

Applications of our work include Content Distribution Networks (CDNs), cloud computing applications, VoIP applications, and P2P networks. CDNs and cloud computing applications can use the highest predicted availability replica/server to redirect the clients to. VoIP implementations can use predicted availability of relay nodes along with latency and loss rate estimates for better performance. Our work can also be applied to peer to peer networks, where ensuring content availability is a primary concern amid extensive peer churn. One can modify the incentive mechanisms of BitTorrent [20] by unchoking the BitTorrent peers which are parts of a highly available prefix, in addition to considering their download rate and latency/loss rate estimates. Our system eliminates the need for storing information about peers at clients that are not currently downloading from these peers but may do so in the future.

The key premise in this paper is that Internet prefix characteristics convey valuable information about prefix availability. We argue that prediction models are viable even if prefixes whose availability is to be predicted and

prefixes used for learning prediction models are unrelated (e.g., learning and predicted prefixes are not in the same AS). This is because an important factor causing paths to prefixes from various vantage points to go up or down is BGP path convergence, caused by BGP reaction to path failure or policy changes. This, combined with the fact that operator reaction to path failures is relatively standard, and that AS policy changes, e.g., AS de-peering, typically affect several prefixes at a time, supports this premise. We therefore use randomly selected prefixes from RouteViews to learn models, and then predict availability of other prefixes. This theme is common in other disciplines, such as medicine, where one uses known symptoms of patients with a diagnosed disease to try to diagnose patients with an unknown condition. To the best of our knowledge, no other work has exploited the similarity of prefixes in the Internet; a few studies, e.g., [13], applied predictive modeling in the context of BGP, but they only examined problem ASes in the path to a particular prefix.

While we focus on predicting prefix availability using observed routing updates, our prediction framework can be easily extended to predict other prefix properties of interest. We formulate hypotheses about how attributes of a prefix such as prefix length and update frequency relate to its availability, and prove or refute them based on our data. We show that past availability of a prefix is inadequate for accurately predicting future availability. Our availability predictions from three models are compared to measured availability values from RouteViews.

This paper extends our previous work [21] as follows:

1. In addition to varying the ratio of the learning duration to the prediction duration as in [21], we vary the learning duration itself. This is important because the availability distribution depends on the duration over which it is computed, and hence this impacts prediction performance.
2. We consider an additional machine learning model, namely the Naïve Bayes model, for availability prediction. This is a popular model in the machine learning literature [22], known to be simpler than the bagged decision trees considered in [21] but potentially less accurate. We find that the performance of this model is better or worse than bagged decision trees, depending on the learning duration (Section 6.4). We also conduct a more thorough investigation of the prediction models used in [21].
3. We study the distribution of the prefix attributes and show, using statistical tests, that the attributes indeed demarcate availability classes (Section 5.3).
4. We predict availability of a large number of prefixes, thereby showing that the prediction models are scalable (Section 7).
5. All results presented in this paper are for the time period of January to October 2009, as opposed to [21], where one month of data was considered at a time. This leads to higher diversity of the visible prefixes, since some prefixes are only visible for short time periods. As availability is a long-term metric, this 10-month evaluation of the prediction models is more realistic.

The remainder of this paper is organized as follows. Section 2 summarizes related work. We define the problem that we study in Section 3. Section 4 describes our datasets, and Section 5 describes our methodology and metrics. In

Section 6, we compare results from three prediction models and study the effect of classification attributes and using certain more predictable prefixes on prediction results. Section 7 describes our results of applying prediction models to large sets of combinations. Section 8 concludes the paper and discusses future work.

2. Related Work

Rexford *et al.* [23] find that highly popular prefixes have relatively stable BGP routes, and experience fewer and shorter update events. Their results fit into our prediction framework, with the prefix popularity being a feature that can be used to predict stability, specifically the number of update events associated with a prefix. Our work goes a step further by predicting prefix availability, not just the events associated with a prefix, using easily computable attributes. Prefix attributes like activity, update count, reachability from various monitors, prefix churn, and growth, have been studied, e.g., in [24, 25, 26, 27], but the attributes are not used to classify prefixes or predict prefix features, which we address in this paper.

Chang *et al.* [28] cluster routing updates into events based on the announcing peers and similarity of AS paths using *descriptive modeling* as the data mining technique. This technique is used for summarizing the data and improving understanding of the data features. In contrast, we use *predictive modeling* to predict prefix behavior, specifically availability, given the observed values of prefix attributes.

Zhang *et al.* [13] predict the impact of routing changes on the data plane. They aim to predict reachability problems based on problematic ASes in AS paths in the routing updates observed for a prefix. Our work is orthogonal to theirs in the sense that we consider control plane availability, utilizing four simple attributes computed from observing RouteViews data, and we investigate three prediction models that may learn from other prefixes. Predictability of network performance has been extensively studied, e.g., in [3, 29, 2]. These studies focused on end-to-end loss, delay, and throughput, measured by active probes, whereas we consider prefix availability indicated by routing update messages. Recently, Hubble [18] and iPlane [19, 30] have been developed at University of Washington for detecting data plane reachability problems and predicting data plane paths and their properties, respectively. Our work is complementary to iPlane [19] and iPlane Nano [30], since we predict control plane availability (or existence of routing paths) to a prefix from multiple Internet vantage points, while iPlane samples data plane metrics like latency, bandwidth and loss rates to end hosts in the prefix at a low frequency. These predictions, taken together, can increase knowledge about prefixes and end hosts in the Internet, which can improve the performance of several applications like VoIP, P2P and CDNs. In cases where no responsive hosts within a prefix can be found by iPlane, iPlane cannot make predictions, in which case our availability predictions will be the only available ones for applications.

3. Problem Definition

We define the *availability prediction problem* to be the prediction of the BGP-advertised availability of a prefix, given its attributes computed by observing BGP updates (for example, through RouteViews), and the availability and

attribute information of other prefixes, collected for a short duration of time. Advertised availability is critical in maintaining smooth traffic flow to these prefixes. Going back to our patient analogy, given the symptoms and known diseases of some patients, one can use test results of a new patient to diagnose the new patient's disease. Our "test results" are the updates observed for a prefix for a limited period of time, which are used to predict its long-term availability.

In this paper, we compute availability in the *control plane* by marking the time of an announcement of a prefix as the time when it goes up and a withdrawal as the time when it goes down and matching our predictions against this computed availability. Spurious announcements and withdrawals are filtered as described in Section 4.

Rather than predicting continuous values of availability, we discretize availability, and predict the availability *class* of a prefix for some time period in the future, based on information collected from the past that is used to train prediction models. This is because, for diagnosis or detection purposes, our interest lies in predicting whether the availability value is above or below an acceptable threshold (e.g., that advertised in SLA), and not the specific value of the availability. Discretizing also gives us an added advantage of use of confusion matrix-based measures, e.g., false positives, to assess prediction performance. Using continuous availability values causes problems in defining error measures because a miss in high availability values (e.g., 99% predicted as 94%) counts more than a miss in lower values (e.g., a predicted 35% instead of 40%) because of attached importance to higher values. In this paper, we validate our predictions by computing the "future" availability class and comparing it with the predicted class. However, this is purely for validation of our prediction schemes – in a real deployment, we will not have the availability classes of the future, just our predictions.

In this paper, we seek answers to the following questions for our framework:

1. How to discretize availability? How many classes and what threshold values should be used?
2. Given a set of prefixes with their associated attributes and availability classes, how accurately can one predict the availability classes of other prefixes, and which prediction models work best?
3. How to extract and represent prefix attributes from RouteViews data? Which attributes of a prefix are most important in predicting availability? For example, are more specific prefixes (ones with longer length) less available than less specific ones? Do prefixes that generate more updates have lower availability?
4. How large should a set of prefixes be such that if we learn our prediction model from this set, it will give accurate results on unseen prefixes?
5. How long should one observe prefix attributes so that its availability can be accurately inferred?

4. Datasets

The routing tables (RIB files) and updates available from RouteViews [16] are in .bz2 format with typical sizes of 0.8 GB per day of RIB files (sampled every 2 hours) and about 25 MB per day of update files (written every 15 minutes), which total about 25 GB per month of data. We preprocess the data using libbgpdump version 1.4.99.7 [31]

to convert the files from the MRT format to text. We reduce the storage space required by removing unused fields. We only keep the timestamp, peer IP, prefix, and the type of update (announcement or withdrawal), except when studying additional attributes of Announcements in Section 6.5.1. After preprocessing and filtering table transfers (as described below), we have about 14-18 GB of gzipped RIB and update files per month of data.

We utilize data from January to October 2009 to build and test our prediction models. The months span a reasonable time period to prevent biasing our model selection process towards datasets from a particular timeframe when some event (such as an undersea cable cut) may have occurred.

A problem with using raw updates from RouteViews is that they also include routing table transfers which are caused by session resets between a monitor and a peer [32]. These spurious updates are an artifact of the update collection methodology. Zhang *et al.* [32] developed the Minimum Collection Time (MCT) algorithm to identify BGP routing table transfers. We executed scripts (contributed by the authors of [32]) from the point of view of every peer in our dataset. We define a peer as any vantage point gathering routing information which is present in any routing table entry and at least one update. This definition yields 41-43 peers in our dataset. We developed a script that removes the table transfer updates from the update files obtained from RouteViews. We use these filtered updates for all further processing.

5. Methodology

We define a *combination* as a (peer, prefix) tuple, which implies that the prefix was observed by the peer in the RouteViews dataset. We compute the availability of these combinations and use that for building our prediction models. The notion of availability of a prefix is with reference to an observation point in the Internet. For the RouteViews data, these observation points are the peers. They are fairly well spread out over the world, enabling one to observe the availability of prefixes from various points in the Internet. Note that these peers are not the same as the RouteViews monitors, which passively collect data about routing tables and updates from the AS routers (peers) which actually observe prefixes. It is these peers and the prefixes they observe that we refer to as *combinations*. In what follows, a combination is *up* or *down* when the peer associated with the combination has the corresponding prefix in an announced or withdrawn state, respectively.

BGP supports aggregation of prefixes [33], and prefixes are frequently aggregated and deaggregated for implementing routing policies like traffic engineering [17]. In a routing table, there can be several prefixes which are more specific versions (subprefixes) of other prefixes in the table [24]. However, the relationship between the prefixes and their subprefixes can be complicated since these can be announced from different origin ASes. This can happen if the customer of an ISP announces a subportion of the prefix allocated to the ISP. Routing policies can change over time and the announcements of the subprefixes can vary depending upon transient conditions like network load. Misconfigurations can also cause a subprefix to be announced for a short duration, making it indistinguishable from the announcements caused by traffic engineering. Since routing policies are unknown, distinguishing the time when the

prefix is unannounced because of a covering prefix or when it is withdrawn due to BGP or network conditions is difficult, and this needs to be handled by an availability metric aggregated across prefixes. Hence, computing an aggregate long-term availability of prefixes which are subprefixes of other prefixes is a challenging task. In this work, we treat each announced prefix separately as a part of the (peer, prefix) tuple defined above. Formulation and computation of aggregate availability across more specific prefixes and their covering prefixes is left as future work.

We learn the prediction models from a *training set*, which consists of the combinations with known attributes computed during the learning period and availability class labels during the period. We then predict the availability of a disjoint set of combinations, which we call the *test set*. The disjointness is necessary to prevent overfitting [22] so that the model performs well on unseen test data and to permit a realistic evaluation of the model. After the prediction model is learned using the combinations from the training set and the information from the learning period, it is applied to the attributes of the combinations of the test set (computed during the learning period) to predict their availability classes in the future. Thus, the training and test sets are disjoint in both the combinations used and in the time period they span. If we denote the learning period as t_l and the future prediction duration as t_p, then for each test combination, we apply the prediction model to its attributes learned from t_l and we validate the availability prediction by comparing it to its availability during t_p. The learning and future prediction durations are contiguous, i.e., the prediction duration starts right after the learning duration ends.

In this paper, the combinations present in the training and the test sets are randomly chosen from the set of combinations "visible" in the training and test durations t_l and t_p. We define a combination to be visible in a time duration t if it exists in the first routing table of the period t (for preventing boundary effects) or in any of the updates in the time duration. Thus, a combination has an equal chance of appearing in the training and the test sets if it appears at least once in the first routing table of t_l or an update in the period $t_l + t_p$. Since the learning period t_l and prediction period t_p are contiguous, $t_l + t_p$ represents the total time starting from the first update of t_l to the last one of t_p. This random selection of combinations prevents biasing our prediction results towards a specific group of combinations which may be related, e.g., combinations containing prefixes from a specific AS may make it easier to predict availability of combinations containing prefixes from the same AS.

We define the *percentage learning duration* as the ratio $t_l/(t_l + t_p)$ which evaluates the percentage of the duration $t_l + t_p$ that is used in learning. The larger this ratio, the easier the prediction since less of the future is unknown. We evaluate the quality of our prediction models by varying this ratio among 0.1, 0.25, 0.5, 0.75, and 0.9. For each of the values of this ratio, we experiment with values of t_l, where t_l = 1, 7, 19 and 30 days. Thus, we have 20 data points for evaluating each prediction model. The rationale behind this is that the availability distribution may be different when computed over different periods of time. We want to investigate this difference and the effect it has on prediction for the same values of $t_l/(t_l + t_p)$, but different values of t_l.

The prediction models considered in this paper are described in detail in Section 6. We use Weka [22], a Java open-source data mining software, for evaluating the models. Weka provides implementations of standard prediction models and data mining techniques for analyzing the performance of the models.

5.1. Discretizing Availability

We discretize the continuous availability value into availability classes which we predict using observed attributes. The process of discretization uses thresholds as parameters, the number and values of which have to be decided. The choice of these parameters is based on the prediction goal. If one aims to find prefixes that do not meet high availability requirements, a single threshold can discretize availability into *high* and *low* classes. If one aims to find prefixes which have both high and low availability values, one should use two thresholds to discretize availability into *high*, *medium*, and *low* classes.

The computation of the availability of a combination for a particular time period proceeds as follows. The first routing table of the period is used to initialize the state of each combination present in the table to up (or announced). Thus, the learning duration of t_l considers all the combinations found in the first routing table of Jan. 09 and in the updates recorded in the duration t_l. We maintain the state of each combination at each point in time, and at the time of each state change (as indicated by an update), we record a downtime or an uptime. If the state of a combination changes from Announced (A) to Withdrawn (W), an uptime is recorded, whereas a change from W to A leads to the recording of a downtime. After processing all update files, we add an extra up or downtime depending upon the last state of the combination. For example, if the last state change was to W and was reported at time t_1, and if the data period ended at time t_2, we add a downtime with value $t_2 - t_1$. The availability of the combination is computed by noting the time that the combination was up (cumulative uptime) divided by the total time in which we are interested. Hence, a combination that only appears in the first routing table of the month and has no updates for the duration under consideration will have an availability of 1.

We use data from Jan. 09 to study the effect of discretization. Table 1 shows the availability statistics for four values of t_l starting from the beginning of Jan. 09 (i.e., t_l=19 days means data from Jan. 1 to Jan. 19). The second column shows the number of (non-trivial) availability values that are considered in computing these statistics, where one value corresponds to one combination. The first quartile, median, and third quartile are the values below which 25%, 50%, and 75% of the availability ordered combinations lie, respectively. The table shows a steady increase in the number of combinations as more days are considered, since previously undiscovered combinations are found in newer update files. These new combinations were not present in the first routing table of the month; otherwise they would have been found for t_l = 1 day. These are expected to be low availability combinations. This is validated by the fact that the first quartile and mean of the combinations show a decreasing trend with these newly added combinations. The variance of the availability increases as lower values are added to the set of predominantly higher availability values.

This trend of lower availability values with longer durations motivates us to study four different values of t_l with the same $t_l/(t_l + t_p)$ ratio. The difference in availability distributions for durations t_l and t_p not only depends on the value of $t_l/(t_l + t_p)$, but also on the value of t_l. This effect is seen in Table 2 which shows the percentage difference in the mean availability of the learning and test durations (t_l and t_p) for different values of t_l and the same value of $t_l/(t_l + t_p)$.

Table 1: Availability statistics for Jan. 09 for different values of t_l. Median and 3^{rd} quartile are 1 for all t_l.

t_l	Number of Combinations	1^{st} Quartile	Mean	Variance
1 day	10545170	1	0.9975	0.0018
7 days	10700675	1	0.9897	0.0078
19 days	10959231	0.999988	0.9743	0.02041
30 days	11476218	0.999882	0.9604	0.02966

Table 2: Percentage difference of mean availability between the training and test sets for different t_l, $t_l/(t_l + t_p)$=0.1

t_l	Mean availability of learning duration t_l	Mean availability of test duration t_p	% Difference in availability of t_l w.r.t. t_p
1 day	0.9975	0.9853	-1.22
7 days	0.9897	0.9204	-6.99
19 days	0.9743	0.8367	-14.13
30 days	0.9604	0.7996	-16.74

These statistics play an important role in the choice of discretization thresholds. To study this, we start with a ternary class label (values *high*, *medium*, and *low*), and choose two different threshold sets of (0.99, 0.50) and (0.99999, 0.50), with the higher threshold demarcating *high* and *medium* and the lower one differentiating the *medium* and *low* classes. This enables us to compare the percentage share of *high* under the two threshold sets, which is listed in Table 3. The *medium* percentage can be easily calculated since the percentages add up to 100%. If we choose a relatively lower valued threshold for *high*, e.g., 99%, the class distribution will be highly skewed, with most combinations (around 91-94%) having *high* availability. With a 0.5 threshold for the *low* class label, about 1-4% of combinations fall into that category. However, the prediction problem is more difficult with a 0.99999 threshold for *high* than with 0.99, since there is a higher chance of combinations that have *high* availability in the learning period to fall below the 0.99999 threshold in the test period. We verified this observation by evaluating the prediction models of Section 6 on datasets with the two thresholds for the *high* class and found that the model performance for 0.99 threshold is indeed higher than that with 0.99999, validating that the former is an easier prediction problem. Based on these observations and the significance of "five nines" availability [1], we use a single threshold of 0.99999 and a binary class label. However, to find combinations with very low availability, we can easily extend our framework to two thresholds and a ternary class label.

Table 3: Class distributions when discretizing availability

t_l	% *High* with 0.99 Threshold	% *High* with 0.99999 Threshold	% *Low* with 0.5 Threshold
1 day	93.89%	67.92%	1.02 %
7 days	93.09%	67.25%	1.68%
19 days	91.89%	66.19%	2.74%
30 days	91.09%	66.13%	3.59%

5.2. Computing Attributes

We now investigate the attributes of the (peer, prefix) combinations to be extracted from the RouteViews data. The attributes are computed for the learning period with the aim of predicting (future) availability classes for the test set. Our goal is to compute the attributes from publicly available information from RouteViews, which contains both routing tables and updates for various combinations. We choose not to use the routing tables because they provide time snapshots of prefixes which can be reached by peers, and we are interested in availability, which is a continuous time metric. The updates collected from RouteViews have the advantage that (barring errors) all the updates for a particular combination will be recorded. Knowing the announcement and withdrawal times for a combination, we can easily compute its availability. Comparing this computed availability with the predicted availability validates prediction results.

The attributes of a combination are selected to relate to its availability (Section 5.3), and to be easily computable given the observed updates for the learning period so that the learning system is fast. It is important to note that the attributes we select do not necessarily cause high/low availability; we are looking for correlation *not causality*. Correlation is sufficient for a prediction model to be successful.

We hypothesize that longer prefixes will have lower availability since they represent smaller networks which are more likely to go up or down. From [23], it is known that popular destinations, which are expected to have high availability, are stable, i.e., have fewer updates. Hence, in addition to prefix length, we also compute update frequency, which is the average number of updates observed for the combination in a time window of one hour (averaged over the learning period). The period of one hour is chosen so that the update frequency numbers are neither too large nor too small.

Furthermore, by recording the time when a combination goes up/down, we compute two additional attributes, mean uptime and mean downtime, called the Mean Time to Failure (MTTF) and Mean Time to Recovery (MTTR), respectively. It is important to note that MTTF and MTTR are computed for the learning period, and hence the predicted availability for the time-disjoint test set is not a direct function of these values.

In summary, we compute the following attributes for the learning period from routing updates observed through RouteViews: (1) Prefix length, (2) Update frequency, (3) Mean Time to Failure (MTTF), and (4) Mean Time to Recovery (MTTR).

We opt not to use information about to which AS a prefix belongs or the AS path to a prefix in this work. This is because we want to keep our prediction model free from constraints of specific ASes or AS paths that can change. We defer the investigation of how prefixes are similar across the same AS or neighboring ASes in the AS topology to future work.

Although we compute the attributes of every combination with at least one recorded uptime or downtime, we downsample this set of combinations (of about 11 million as in Table 1) to a set of 10,000 combinations with their attributes, and use that to build and test models. Downsampling does not significantly affect the accuracy of models since prediction models typically learn well with a few hundred instances. We evaluate the performance of the models

with increasing number of learning instances in Section 6 and on larger test sets in Section 7. An advantage of downsampling is the computational efficiency of building and testing the models.

5.3. Demarcating Availability using Attributes

In this section, we quantify whether the four attributes discussed in the last section indeed convey information about the availability class. We divide the 10,000 combinations into ones that have *high* and *low* availability for the month and compute statistics for the attributes of each of the two groups. We show the means and variances of all the attributes for a typical value of t_l = 19 days in Table 4.

Table 4: Attribute statistics of each class for learning period of t_l = 19 days

Attribute	*High* Class		*Low* Class	
	Mean	Variance	Mean	Variance
Prefix length	22.04	6.41	22.72	4.25
MTTF (s)	1587480	3.76E+10	777844	2.92E+11
MTTR (s)	0.201002	3.52	58882.2	4.57E+10
Update frequency (/hr)	0.0244	0.7339	0.0795915	0.5694

We use the paired t-test to test for equality of the means of each of the attributes of the two classes. We employ the Welch t-test [34, 35] which assumes that the two populations have normal distributions and their variances cannot be assumed to be equal (which is true for our data). The normality assumption is valid due to the Central Limit Theorem (CLT) and because we have about 3000-7000 samples in each class. We find that the means of each of the four attributes are significantly different at 1% significance level for each of the four learning periods. This shows that the attributes show a statistically significant correlation with the availability class labels. For most of the attributes, their variances for the *low* class are higher because the class covers a wider range of availability values.

Our intuition that the combinations with longer prefix lengths have lower availability is confirmed. The mean prefix lengths of the *high* and the *low* availability classes usually differ by about 0.7, or about 3% (which is statistically significant) while the median and first quartile differ in length by 1 and 2 respectively, with the higher value for the *low* class. The consistency of the results across each of the four values of t_l is convincing of the correlation between prefix length and availability class. We conjecture that this is because shorter prefixes represent larger, more stable networks while small portions of the address space can be announced and withdrawn frequently for multihoming or load balancing purposes. Further, it is more likely that a longer prefix representing a smaller network goes down than a larger network.

The MTTF of a *high* availability combination is higher than that of a *low* availability one by about 85% on average, whereas the MTTR is almost 100% lower. The difference becomes larger as t_l increases. This result is intuitive: a *high* availability prefix has a long uptime before it fails, and when it does fail, it quickly comes back up (well within one second on average). The average frequency of updates observed for a *high* availability prefix is about 77% lower than for *low* availability ones. These results are explained by the fact that a high availability combination stays up

for a long period of time, and hence has fewer updates. The difference in attribute values of the *high* and *low* classes increases with t_l, showing that these attributes correlate well with the availability class since availability computed over a longer duration is more indicative of the actual availability.

Assuming update frequency distribution in each month is a normally distributed random variable (valid because of CLT), we construct a 99% *t*-Confidence Interval (CI) for the average update frequency of a combination. The mean update frequency of a combination, averaged over all 11.5 million combinations of Table 1 is about 0.03/hr and the variance is about 0.28. The upper bound of the CI is computed to be about 1.4 updates an hour. Thus, if we observe more than an update for a combination in about 43 minutes, on average, we are 99% certain that it will have *low* availability.

The conclusion from this section is that the selected prefix attributes perform well in demarcating the availability classes. The correlation of the attributes with the availability class is consistent with our intuition.

5.4. Learning and Evaluation

We learn several models in this paper to predict the availability class of combinations. The performance of each model is studied using *n-fold incremental cross-validation*. The dataset is divided randomly into *n* parts, called *folds*, while maintaining the class distribution of the dataset in the fold (i.e., *supervised sampling*). The model is then learned using the known attributes and class labels of *n*-1 folds (called the *training set*), and applied to predict the class labels of the remaining fold (the *test set*). Each fold is left out at a time, resulting in *n* learned models and corresponding performance results. The training and the test sets are disjoint in order to get an unbiased estimate of model error. The algorithm is run *k* times, each time with a different random seed so that different *n* folds are constructed in each run. Thus, for each training set size, we have *nk* performance values, and we report the mean value.

As the number of instances to learn a model increases, the model performance on test data typically improves, but with diminishing returns. We study this using *learning curves.* A model is successively learned using increasing training set sizes (from each of the *n* training sets) and its performance on the test set is plotted against the training set size. A typical shape of a learning curve is an increasing exponential; the performance increases, and then flattens after a certain number of instances is reached.

5.5. Performance Metrics

We now describe the performance metrics used to evaluate a model when it is applied to the test set. Any classification algorithm can be studied using a confusion matrix, which gives all possible combinations of the true and predicted class. In what follows, the class label *high* is treated as a *positive* class, and the label *low* is treated as a *negative* class. The confusion matrix can be used to compute several performance measures, the most common of which is *accuracy*, defined as: $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$, where TP and TN are the true positives and negatives respectively, and FP and FN are the false positives and negatives respectively. The True Positive Rate (TPR) and the False Positive Rate (FPR) are defined as: $TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$, and $FPR = \frac{FP}{N} = \frac{FP}{FP+TN}$.

The *Kappa statistic* measures the agreement between predicted and observed values, correcting for agreement that occurs by chance. It is computed as: $\kappa = \frac{P(o)-P(e)}{1-P(e)}$, where $P(o)$ is the proportion of observed agreement between the observed and predicted values, and $P(e)$ is the proportion of times the values are expected to agree by chance. Complete agreement corresponds to κ = 1, which will be the best predictor, whereas κ = 0 for a random predictor, and κ = -1 indicates complete disagreement between the values.

Unfortunately, confusion matrix-based measures can be misleading with a skewed class distribution, which happens when the proportion of *high* availability (positive) and *low* availability (negative) instances in the sample are unequal. For example, a trivial algorithm which predicts every availability value as *high* will have 90% accuracy on a dataset which has 90% *high* values. The measures use data from both columns of a confusion matrix, and hence are sensitive to the proportion of instances in the two columns [36]. From Table 3, we observe that there can be significant class skew, which render these measures inappropriate. A better metric is obtained by using Receiver Operating Characteristic (ROC) curves [22], which plot the TPR versus the FPR. ROC curves are independent of class skew because they use a strict columnar ratio from the confusion matrix [37, 36]. We use the Area Under the ROC Curve (AUC) as a performance metric. The AUC of a classifier is equivalent to the probability that it will rank a randomly chosen *high* instance higher than a randomly chosen *low* instance. A perfect classifier has an AUC of 1.

We compare the results from our prediction models to those obtained using a random classifier, which acts as a baseline for comparison. A random classifier randomly chooses either of the class labels with equal probability. Such a classifier has an AUC of 0.5, since it has about as many TPs as FPs. The reason for comparison to a random classifier is that we need to be sure that any learning-based model performs better than the random classifier. Otherwise, one could effectively toss a coin and decide the class label, making a trivial predictor the best one.

While ROC curves work well for most classifiers, they are not directly applicable for models which do not produce any ranking of instances in terms of probabilities of being classified as *high* and *low*. This is because one plots a ROC curve by varying the threshold that decides between *high* and *low* to produce various (FPR, TPR) points. A model which does not produce instance ranking has no threshold to vary; hence, it gives a single point in the ROC space instead of a curve. For such a model, an option is to randomly order the instances predicted as *high* and *low*, and then rank them to produce a ROC curve. We describe the details of this scheme in Section 6.1.

6. Model Space

In this section, we study three prediction models using the metrics in Section 5.5. As mentioned in Section 5.2, we work with 10,000 combinations and their attributes, downsampled from all the combinations in each of four different months. We do 10-fold incremental cross-validation as described in Section 5.4; thus *n*=10. We conduct *k*=5 runs, generating a different set of 10 folds each time. Hence, we have 50 performance measures for each model averaged to give an output measurement.

We start with a simple baseline prediction model in Section 6.1. This model does not learn based on other combinations, and simply predicts the availability of one combination at a time. We then investigate more sophisticated

machine learning based models.

6.1. Simple Prediction

The simplest approach to predict the availability of a combination is based on the simplistic assumption that the future is the same as the past. The *past availability* of a combination is its availability during the learning period t_l. This prediction approach does not learn a model based on other combinations, but merely predicts the same availability for a combination as the discretized value of its *past availability*. Thus, if the past availability exceeds 99.999%, the predicted class label is *high*, otherwise it is *low*.

This is a model where no instance ranking is performed; only hard classifications are made. Therefore, we compute confusion matrix-based measures. These measures, computed for various values of t_l and $t_l/(t_l + t_p)$ and averaged over *nk* = 50 runs, are listed in Table 5.

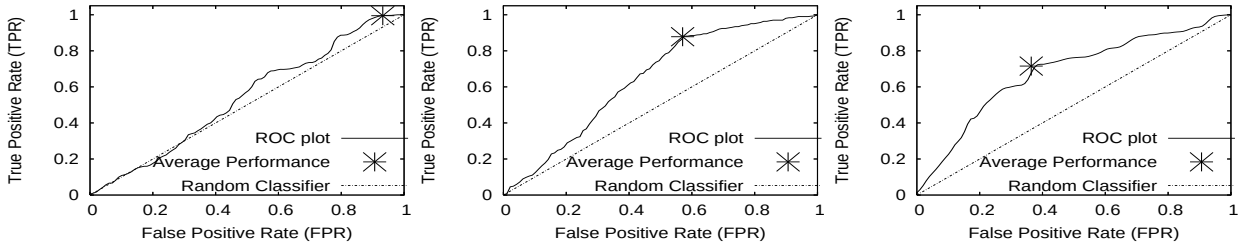
Table 5: Results with the simple prediction model

t_l	$t_l/(t_l + t_p)$	Accuracy (%)	TPR	FPR	κ	AUC
1 day	0.1	88.60	0.9950	0.9322	0.1022	0.5261
	0.25	96.79	0.9940	0.8085	0.2641	0.5877
	0.5	97.99	0.9928	0.7160	0.3207	0.6224
	0.75	98.69	0.9911	0.5670	0.3175	0.6900
	0.9	98.90	0.9900	0.4766	0.1803	0.7272
7 days	0.1	60.02	0.9717	0.8451	0.1353	0.5599
	0.25	73.87	0.9502	0.8013	0.1928	0.5774
	0.5	83.98	0.9421	0.7410	0.2403	0.5962
	0.75	89.37	0.9271	0.7242	0.1575	0.5813
	0.9	91.22	0.9224	0.5907	0.1281	0.6713
19 days	0.1	54.10	0.9107	0.6777	0.1917	0.6163
	0.25	67.98	0.8933	0.6281	0.2816	0.6326
	0.5	76.01	0.8620	0.5315	0.3481	0.6641
	0.75	78.30	0.8201	0.4652	0.2726	0.6748
	0.9	78.66	0.7953	0.4082	0.1355	0.7015
30 days	0.1	57.17	0.8613	0.5720	0.2242	0.6414
	0.25	65.53	0.8379	0.5231	0.3133	0.6548
	0.5	70.81	0.7961	0.4752	0.3242	0.6606
	0.75	73.45	0.7544	0.3723	0.2955	0.6895
	0.9	71.08	0.7154	0.3641	0.1346	0.6728

The results show that while the TPR of the simple model is high, its FPR is high as well. However, this simple classifier outperforms a random classifier (as indicated by the κ statistic) and hence forms a baseline model to which other sophisticated models can be compared. As $t_l/(t_l+t_p)$ increases, the prediction problem becomes easier as more data is available for learning. Hence, the accuracy of the model increases, while its FPR reduces. As t_l increases, the availability distribution becomes more diverse and hence the model typically performs worse.

We now use ROC-based metrics to evaluate this classifier. The model gives a single point in the ROC space (since it does not perform instance ranking), so we modify the algorithm to draw a ROC curve. We take a typical run of the model with confusion matrix measures close to their average values. The instances which are classified as *high*

and *low* by the model are randomly reordered within their respective groups, and then the instances are ranked with the (predicted) *high*s higher than the *low*s. We vary the prediction threshold, and record the TPR and FPR for each threshold, as in Algorithm 2 of [36] to compute the points on a ROC curve.



(a) t_l = 1 day, $t_l/(t_l + t_p)$ = 0.1 (b) t_l = 30 days, $t_l/(t_l + t_p)$ = 0.1 (c) t_l = 30 days, $t_l/(t_l + t_p)$ = 0.9

Figure 1: ROC plots for the simple prediction model.

The ROC curves for the simple prediction model for some typical values of t_l and t_p are depicted in Figure 1. The plots show the original model performance (in Table 5) as a point ("star") on the ROC plots, along with the performance of a random classifier. The performance of simple prediction is clearly better than a random classifier for most cases, but there are occasions when it performs as good as or slightly worse than a random one as in Figure 1(a). This is especially true when t_l is small, and hence future availability is quite different from past availability. As the average accuracy in Table 5 is reasonably high, this emphasizes the inadequacy of accuracy as a metric to evaluate performance models. Hence, we use ROC metrics, like area under the ROC curve (AUC). The AUC is computed, using Algorithm 3 of [36], for a typical run (confusion matrix based measures close to their average values of 50 runs). Because of inherent randomness in reordering and ranking the instances, the typical run will give different AUC values when run with different random seeds; the average of 50 different AUC values is reported in Table 5.

The results highlight the importance of ROC curves. For example, classifier A for t_l = 30 days, $t_l/(t_l + t_p)$ = 0.1 (Figure 1(b)) is worse than classifier B for t_l = 30 days, $t_l/(t_l + t_p)$ = 0.9 (Figure 1(c)) using AUC as the metric. However, examining the ROC curve, we see that for higher FPRs (around 0.8), classifier B outperforms classifier A. The overall inferior performance of classifier A is because it performs similar to a random classifier for low FPRs. Hence, if our operating region is at low FPR, classifier B is better, whereas classifier A is better for high FPRs.

6.2. Naive Bayes Model

The Naïve Bayes model predicts a *high* or *low* class label using the attribute vector $\mathbf{X} \equiv \{X_1, X_2, \ldots, X_m\}$ based on Bayes rule [22]. It makes the "naïve" assumption that the attributes are conditionally independent given the class label. However, the model is often used even when its assumption does not hold due to its simplicity. The model computes, for each instance, the probability of each class label given its attribute set and the independence

assumption, using the training set to estimate $P(X_i|C)$ and $P(C)$, where C is the class label. Hence, instance ranking is naturally produced by the model which can be used to produce ROC curves.

We evaluate the model on each of the values of t_l and t_p using learning curves. The model is learned on increasing size training sets, and its performance is evaluated on the 10 different test sets produced by incremental cross-validation. We plot a typical learning curve in Figure 2, using both accuracy and AUC as performance measures. The plots for the other time durations lead to similar conclusions. The accuracy initially increases at a fast rate when the number of training instances is increased, and tapers off afterwards. However, the AUC remains relatively stable with the increase in number of training instances. In what follows, we use the entire training set to train the Naïve Bayes model to achieve the maximum accuracy without sacrificing AUC. This ensures that the model is trained to its potential.

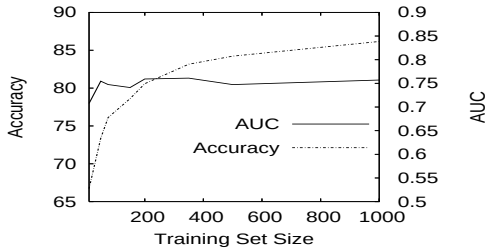


Figure 2: Naïve Bayes learning curves for t_l=30 days, $t_l/(t_l + t_p)$=0.9

We now compare the Naïve Bayes model to the simple prediction model of Section 6.1. We use the accuracy and AUC as measures for comparison. The results are given in Table 6. The results show that the Naïve Bayes model yields a higher AUC than the simple model for all cases. The accuracy values of Naïve Bayes are close to those of the simple model, except when learning from 30 days of data, where for a smaller prediction period t_p, the accuracy is significantly better with a high variance (around 26.3), while for a higher prediction period t_p, the accuracy is significantly lower with a low variance (around 2.3). This is because this model assumes that the attributes are conditionally independent given the class label. The model uses the frequencies in the training set as estimates of the probability distributions of the attributes. These estimated distributions are valid only when the period of parameter estimation, i.e., learning period, is not too different from the prediction period. When t_l = 30 days, the period of the training and test sets differ by a few days to months (except when $t_l/(t_l + t_p)$ = 0.5) and hence have different distributions. This leads to different accuracies since this metric is highly dependent on class skew.

We consider the better metric, AUC, and investigate whether the higher AUC values of the Naïve Bayes model are statistically significant. If so, Naïve Bayes would be a better prediction model than the simple model. We use the Welch t-test [34, 35] to test for equality of the performance measures (means) of the distributions of the two samples (simple and the Naïve Bayes). We perform the test on the AUCs of the two models for each of the four months, using the mean values shown in Tables 5 and 6, and the sample variances computed using the *nk* = 50 data points.

Table 6: Results with Naïve Bayes model and % change from simple model

t_l	$t_l/(t_l+t_p)$	Accuracy (%)	% Change in Accuracy from Simple Model	AUC	% Change in AUC from Simple Model
1 day	0.1	88.51	-0.09	0.6044	14.89
	0.25	96.70	-0.09	0.6568	11.76
	0.5	97.94	-0.045	0.7097	14.02
	0.75	98.66	-0.034	0.7924	14.84
	0.9	98.82	-0.074	0.8159	12.19
7 days	0.1	59.85	-0.288	0.6341	13.25
	0.25	74.20	0.444	0.6290	8.96
	0.5	84.06	0.10	0.6355	6.59
	0.75	87.89	-1.65	0.6473	11.35
	0.9	89.95	-1.39	0.6990	4.12
19 days	0.1	54.61	0.94	0.6761	9.70
	0.25	68.04	0.08	0.6956	9.95
	0.5	76.09	0.10	0.7173	8.01
	0.75	77.35	-1.21	0.7173	6.31
	0.9	77.38	-1.63	0.7304	4.12
30 days	0.1	46.14	-19.29	0.6930	8.04
	0.25	59.23	-9.61	0.7009	7.04
	0.5	70.29	-0.73	0.7009	6.10
	0.75	80.03	8.95	0.7394	7.24
	0.9	83.40	17.33	0.7538	12.05

We compute the degree of freedom ν using the Welch-Satterthwaite equation, and round it to the nearest integer for t table lookup using [38]. We find that the null hypothesis of equality of the means is rejected for every month at 5% significance level. This means that the AUC of the Naïve Bayes model indeed exceeds that of the simple model at 5% significance level. Table 7 shows the details of the test for some typical values of t_l and t_p.

Table 7: Paired *t*-test results of comparing AUC of Naïve Bayes model and the simple model

t_l	$t_l/(t_l+t_p)$	Statistic Value	ν	t-value for 5% Significance
1 day	0.1	15.80	98	1.984
7 days	0.25	8.46	64	1.998
19 days	0.5	8.75	98	1.984
30 days	0.75	7.28	81	1.99

Finally, we compare the Naïve Bayes model to the simple model using ROC curves. The plot for t_l = 30 days and $t_l/(t_l+t_p)$ = 0.1 is illustrated in Figure 3. The figure shows that the Naïve Bayes model dominates the simple model throughout most of the ROC space. For the same FPR, its TPR is higher and hence it is closer to the ideal point in ROC space. The implication of these results is that a model which learns based on other prefix combinations like the Naïve Bayes classifier will typically outperform prediction without learning, despite its naïve assumptions. This confirms that availability is predictable using the attributes we measure. It is also worth noting that this better performance in terms of TPR and FPR in the ROC space again points to the inadequacy of accuracy as a metric: even

though the Naïve Bayes model has much lower accuracy than the simple model for these values of t_l and t_p, it is better in ROC space.

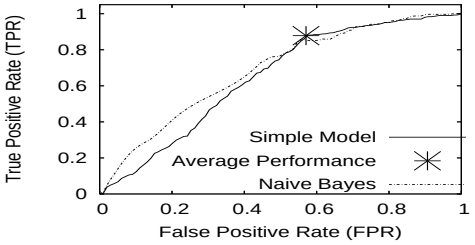


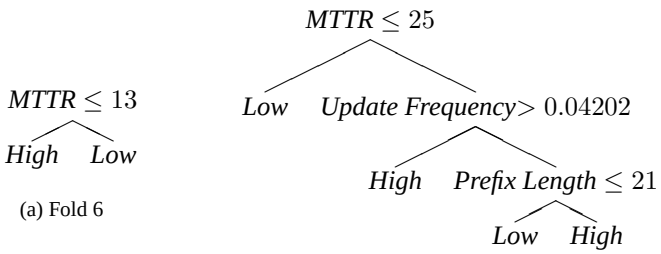
Figure 3: ROC plots for Naïve Bayes and simple model for t_l=30 days, $t_l/(t_l + t_p)$=0.1

6.3. Decision Trees

A *decision tree* is a recursive divide-and-conquer classifier, which divides the instances based on one attribute at a time in a top-down fashion until the leaves of the tree are reached [22]. A decision tree can be easily transformed into if-then rules. This classifier has the advantage that it is interpretable, since the attributes of the classifier are ranked from the root node downwards in the order of importance, and rules to classify an instance can be read off the decision tree. We use the C4.5 algorithm developed by Quinlan [39] to build decision trees, which uses reduction in entropy (measure of randomness) when splitting the instance set S based on an attribute A as the information gain metric to build the tree.

Pruning the tree is necessary to avoid overfitting to the training data, and for constructing a general enough tree to perform well on unseen test data. In Weka, the J4.8 classifier implements the C4.5 algorithm [22], and one can choose to consider the unpruned tree, or prune it based on different criteria. C4.5 pruning (the default) uses an estimate of the error on the training set. An alternative is to use Reduced Error Pruning (REP) [40], which holds back some of the training data as a fold and uses that to estimate the error. The advantage of REP is that it can lead to more unbiased estimates of the error; the disadvantage is that it uses less data for tree building.

We use the unpruned, C4.5-pruned, and REP trees, and find that the accuracy and AUC metrics are not significantly different among them. However, at very small training set sizes, holding out instances for REP can lead to insufficient training data, which results in lower AUC. Nonetheless, we decided to use REP because of the advantages of a tree which avoids overfitting and because we will work with sufficiently large datasets. Our results have a high variance. This is a typical property of decision trees, since a small difference in the training data can cause different branches to be constructed. For example, with 200 training instances in each of the 10 folds, we find decision trees with different structure and attribute values (two are shown in Figure 4). The right branches of all nodes are for a “Yes” decision and the left branches are for a “No” decision. While the decision trees shown all use *MTTR* as their root node, different trees use different numbers and values of attributes to make decisions. This increases variance in classification results, causing mean results to appear worse.



(a) Fold 6

(b) Fold 8

Figure 4: Decision trees for t_l = 30 days, $t_l/(t_l + t_p)$ = 0.1 constructed with 200 training instances.

A method to reduce the variance of decision trees is to use *bootstrap aggregating (bagging)* [22]. Bagging combines an ensemble of unstable, high variance, predictors into a stable predictor. We apply the bagged decision tree classifier to predict availability with the underlying baseline classifier chosen to be decision trees with REP. Ten decision trees are learned for each of the 10 folds of the dataset, and they are then voted on to produce the *high* or *low* class label. The learning curve for a typical case is shown in Figure 5. The curve demonstrates that the performance measures flatten with increase in training set size, which confirms that pruning is successful in preventing overfitting.

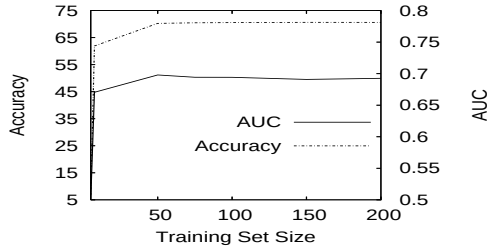


Figure 5: Learning curve for bagged decision trees, t_l = 30 days, $t_l/(t_l + t_p)$ = 0.1

We now apply the bagged decision tree model learned from the entire training dataset of around 9000 combinations to predict availability for the values of t_l and t_p considered earlier. The average results over *nk* = 50 points are given in Table 8. As before, we perform significance tests, and find that AUC increases for t_l = 1 and 7 days are significant at 5% significance level, except for $t_l/(t_l + t_p)$ = 0.9 for t_l = 1 day, and $t_l/(t_l + t_p)$ = 0.75 and 0.9 for t_l = 7 and 19 days. The results reveal that bagged decision trees perform well w.r.t. Naïve Bayes when the learning period is shorter (up to a couple of weeks) and the prediction period is longer, i.e., when $t_l/(t_l + t_p)$ is small. This is because as diversity of the data increases, the bagged decision trees adapt to the diversity by building complex trees, which do not generalize well to future datasets. This cannot be corrected by pruning since the diversity is in the time domain and occurs in nearly every combination, so holding out a set of combinations for pruning does not necessarily help.

Table 8: Results with bagged decision trees (% change from Naïve Bayes model given within parentheses)

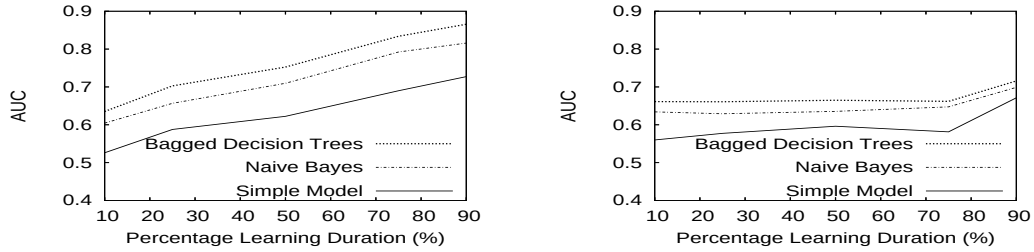
t_l	$t_l/(t_l+t_p)$	Accuracy (%)	% Change in Accuracy from Naïve Bayes Model	AUC	% Change in AUC from Naïve Bayes Model
1 day	0.1	87.81	-0.80	0.6352	5.10
	0.25	95.54	-1.21	0.7027	6.99
	0.5	96.61	-1.37	0.7525	6.04
	0.75	97.22	-1.46	0.8339	5.24
	0.9	97.36	-1.48	0.87	6.09
7 days	0.1	60.24	0.67	0.6613	4.29
	0.25	74.91	0.96	0.6609	5.05
	0.5	83.42	-0.76	0.6648	4.60
	0.75	87.41	-0.55	0.6619	2.26
	0.9	90.23	0.32	0.7159	2.41
19 days	0.1	54.95	0.6147	0.6726	-0.52
	0.25	68.35	0.46	0.6976	0.28
	0.5	75.96	-0.17	0.7188	0.21
	0.75	77.45	0.13	0.7218	0.62
	0.9	76.63	-0.96	0.7235	-0.94
30 days	0.1	56.83	23.17	0.6671	-3.73
	0.25	65.24	10.15	0.6745	-3.75
	0.5	70.85	0.79	0.6771	-3.39
	0.75	73.44	-8.23	0.7018	-5.09
	0.9	70.69	-15.23	0.6945	-7.87

6.4. Learning Duration

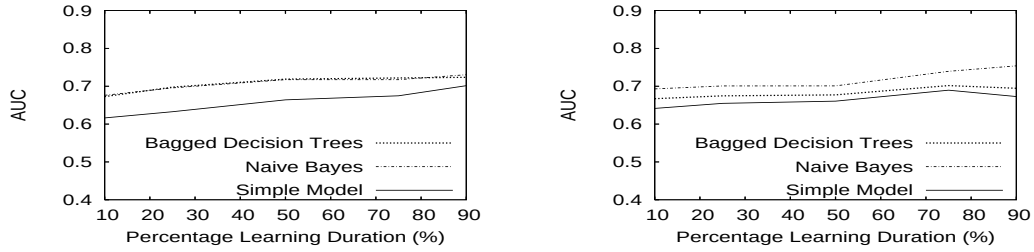
We now study the effect of learning duration on the prediction results of all the models we have considered. There are two facets to this problem: the learning duration as a percentage of the overall period of interest, i.e., $t_l/(t_l+t_p)$, and the value of the learning duration itself. Lowering the percentage learning duration means that we have a shorter time to learn the attributes of various combinations, leading to a reduction in prediction accuracy, whereas increasing this percentage improves prediction results, since there is more information available.

The plot of AUC against percentage learning duration for various values of the learning duration t_l is shown in Figure 6. The results show that the prediction performance gracefully degrades as the amount of data available for learning is reduced. The decrease is much more steep when the learning duration t_l is low, e.g., 1 day, and this effect almost disappears when t_l reaches 30 days. This result implies that one can predict long-term availability by learning from only a short learning period, as long as the period spans a few days, e.g., a week. This gives further credence to the feasibility of availability prediction. The bagged decision tree model performs the best for all learning duration percentages when the learning duration t_l is less than around 3 weeks. Beyond that value of t_l, Naïve Bayes performs best.

We also plot the change in the AUC for each of the models when increasing the learning duration t_l, keeping the percentage learning duration $t_l/(t_l+t_p)$ constant. Two typical plots are shown in Figure 7. The plots show that for the same percentage learning duration, as more learning data is available (higher t_l), the performance of all the models



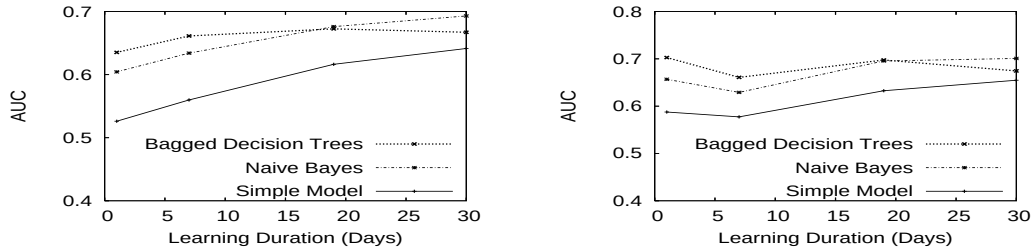
(a) t_l = 1 day (b) t_l = 7 days



(c) t_l = 19 days (d) t_l = 30 days

Figure 6: Effect of percentage learning duration $t_l/(t_l + t_p)$ on prediction performance for different values of t_l

improves, except when t_l = 30 days. The plots also show that the crossover point between the performance of bagged decision trees and Naïve Bayes is about three weeks, as indicated above.



(a) $t_l/(t_l + t_p)$ = 0.1 (b) $t_l/(t_l + t_p)$ = 0.25

Figure 7: Effect of learning duration t_l on prediction performance for different values of $t_l/(t_l + t_p)$

Based on the results, we can conclude that an availability prediction system using bagged decision trees can learn from a few days of routing data logs. Our system can be adapted to *real time deployment* by sliding the time window of the learning period to always learn from the most recent data. For example, if we learn from a week of data, we can slide our learning window by a day at a time to always learn from the most recent past week. Predicting the availability for about three times the learning duration gives accuracy and AUC of around 75% and 0.66 respectively.

If these performance measures are acceptable, one can triple the prediction duration at every stage. If we are learning our prediction models from the most recent week of data, we can predict the availability for three weeks into the future maintaining this level of performance. If one desires higher performance, one should reduce the prediction duration for the same learning duration, i.e., increase the percentage learning duration. Our prediction framework allows the system administrator to trade off prediction performance and prediction duration.

6.5. Classification Attributes

We now study the importance of attributes in the prediction process, by studying the effect of using different sets of attributes on the output metrics of Naïve Bayes and bagged decision trees. We start with the bagged decision tree results from Table 8, and remove certain attributes of the combinations, so that less data is available to the prediction model. The degradation in various performance metrics is studied; as degradation increases, the importance of the removed attribute subset increases. We present typical results of removal of some of the attributes for t_l = 30 days, $t_l/(t_l + t_p)$ = 0.1 in Table 9. We choose these values of t_l and t_p as this represents a long enough learning period and a hard prediction problem: predicting availability for 9 times the learning duration. The results for the other values of t_l and t_p were similar albeit with different values. We choose AUC for comparison because of its strength as a performance metric as described earlier. The first column of the table indicates which attributes of the combinations were used for prediction. Along with using subsets of the four attributes from Section 5.2, we also use the the attribute of *past availability* to build prediction models. This attribute is used in the simple model and we seek to study the performance of machine learning-based prediction models which use this attribute to predict availability.

Table 9: Percentage change in performance metrics with subsets of attributes for t_l = 30 days, $t_l/(t_l + t_p)$ = 0.1. All percentage changes are w.r.t. results of the corresponding models from Table 6 and Table 8

Attributes Used for Prediction	% Change in AUC for Naïve Bayes	% Change in AUC for Bagged Decision Trees
Past Availability	-12.03	-9.8
MTTF	-4.43	-1.97
MTTR	-14.67	-1.93
Prefix Length	-15.22	-24.96
Update Frequency	-18.18	-7.12
Prefix Length and Update Frequency	-9.94	-3.13
MTTF and MTTR	-3.75	-1.65
MTTR, Prefix Length and Update Frequency	-4.00	-1.33
MTTF, Prefix Length and Update Frequency	-0.03	0.25

These results lead us to the following conclusions. Performance significantly degrades (AUC is 10-12% lower) when only past availability is used. Combining this with the results of the simple model, we conclude that past availability is not an adequate metric for prediction of future availability. Prefix length and update frequency are weak attributes, with prefix length being the weakest since using it alone causes the AUC to decline by 7-25%. MTTF is the most important attribute since using it alone causes the least drop in AUC among any single item attribute set. Using either MTTF or MTTR with prefix length or update frequency, or MTTF and MTTR together, causes the

AUC and accuracy to be within 4% of their values when no attribute is removed. MTTF combined with the prefix length and update frequency give very close results to those obtained when MTTR was also added to the set, further confirming that MTTF is the strongest attribute (complemented by the use of prefix length and update frequency). We also experimented with adding *past availability* to these attribute subsets and found that the performance did not change significantly.

It is intuitive that MTTF is the most important prediction attribute and MTTR is the next important, since the time to fail or recovery will characterize the availability of a combination; a *high* availability combination should have a high MTTF and low MTTR. There exists no subset or superset of the four attributes used that would cause significantly better results than the four attribute set we have chosen.

It is also interesting to note that the Naïve Bayes model is much more sensitive to the removal of attributes than bagged decision trees. This is because the intrinsic assumption used in this model is that attributes are conditionally independent given the class label. This assumption cannot be used when there is only one attribute. Among multiple attributes, the results will depend upon the degree of conditional independence between the attributes. The bagged decision tree model, in contrast, builds decision trees based on various attribute values. While attribute removal does hurt its performance, the trees formed based on other attributes are still reasonably accurate, unless the prediction attribute is weak, e.g., prefix length.

6.5.1. Additional Attributes

We now investigate whether the prediction accuracy can be improved if we add additional attributes that we have not considered in this work so far. In Section 5.2, we give the rationale for the selection of attributes to be the relation between these attributes and the availability of the prefix. There are other attributes of BGP updates [41, 42] such as AS path, community, MED, and aggregation, which we have not considered in this work. This is because we believe that these attributes are not significantly related to availability as much as MTTF, MTTR, update frequency, and prefix length. For example, repeated announcements with different AS path do not change the Announced or Withdrawn status of prefixes. If the prefix flaps frequently with announcements and withdrawals, affecting availability, this will be captured by our update frequency metric.

The additional attributes that we consider in this section are: (1) Average AS path length percentage change of the changed AS path w.r.t. the old AS path, averaged over all announcements, (2) Fraction of times AS path length changes over all announcements, (3) Fraction of time the aggregator attribute changes over all announcements, (4) Fraction of time the community attribute changes over all announcements, and (5) Fraction of time MED attribute changes over all announcements. As usual, we compute these attributes for each combination and use them for availability prediction. We consider these attributes one at a time, and all these five together for availability prediction using bagged decision trees. We find that the AUC results are 16% poorer on the average across these six prediction cases w.r.t. the results in Table 8 . The average AUC of prediction comes out to be only 0.55. Although better than a random classifier, these results are poor compared to the prediction performance achieved previously. This is

explained by the fact that these attribute changes are due to AS policies for diverting traffic to the inbound prefixes by modifying existing announcements, and are less correlated with changes in the announced or withdrawn state of prefixes, which affects availability.

6.6. Predictability of Prefixes

Thus far, we have used a random set of (peer, prefix) combinations for training the prediction models and for testing the effectiveness of the prediction techniques. We now investigate whether certain combinations are more predictable than others. The intuition behind this is that the availability of a combination is more predictable from the attributes chosen in our work for certain kinds of prefixes than for others. There can be several causes of BGP routing dynamics [41], and some causes are likely to be more correlated with availability, making a particular prefix group more predictable. For example, a prefix can be withdrawn and announced with a specific pattern (e.g., dependent on time of day) for traffic engineering purposes, and all prefixes which are announced according to similar policies will exhibit more predictable availability. The authors of [41] discovered both daily and weekly patterns in prefix announcements, attributed to several known and unknown causes. Y. Zhang et. al. [13] predicted data plane failures using control plane updates and also observed that certain prefixes are more predictable than others. While we leave detailed investigation of exact predictability classes of prefixes to future work, we investigate whether there are more predictable combinations in our dataset.

Our methodology is motivated by [13]. Out of all the prediction models considered in this work, only Naïve Bayes (Section 6.2) gives a probability of prediction of prefix availability as *high* or *low* based on its attributes. We use an option in Weka [22] to output the class prediction probabilities for each of the instances along with the true availability class. For each of the 20 sets of results from Table 6, we investigate the instances which were classified incorrectly. We note the probability of incorrect classification P_{inc} as the P(predicted class label - *low* or *high*) for the incorrectly classified instances output from the Naïve Bayes model. P_{inc} can never be less than 0.5 since a label is only predicted if its probability is greater than the other class label. The CDF of P_{inc} is shown in Figure 8. The plot shows that about 91% of the incorrectly classified instances have a class prediction probability above 0.93 when they are incorrectly classified. This implies that when a prediction error is made, the case is *not* borderline – the model almost surely predicts the incorrect class label. This gives credence to the fact that some prefixes in combinations are very poor in predictability compared to others.

We now seek to isolate the combinations which have poor prediction performance. We look at the instances incorrectly classified by Naïve Bayes for all the 20 cases of Table 6 and isolate the combinations which have a probability of prediction of the (incorrect) class label exceeding 0.75. We chose this threshold of 0.75 since it is midway between 0.5 and 1 and we want to ignore combinations for which a slight prediction error is made. Across all the 20 cases, this gives 15,722 “poorly predicted” combinations, which is about 39.33% of the total number of unique combinations for the 20 cases.

To evaluate the prediction performance of the poorly predictable combinations versus the predictable ones, we

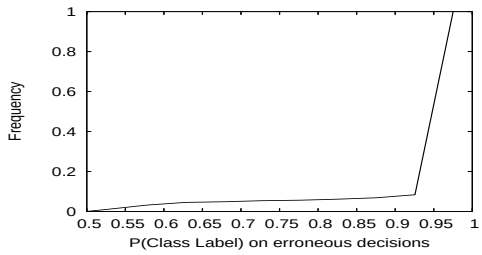


Figure 8: CDF of class label prediction probability for incorrectly classified instances using Naïve Bayes

run the bagged decision tree model from Section 6.3 on both sets of combinations. We show the performance as indicated by AUC for both the predictable and poorly predictable combinations in Table 10. The results indicate a large difference in predictability between the two types of combinations. On the average, the predictable combinations have 40.95% higher prediction performance (measured in terms of AUC) than the poorly predictable combinations.

Table 10: Results for predictable and poorly predictable combinations obtained from bagged decision tree model

t_l	$t_l/(t_l+t_p)$	AUC for predictable combinations	AUC for poorly predictable combinations	% difference in AUC of Col. 4 from Col. 3
1 day	0.1	1	0.4581	54.19
	0.25	1	0.5144	48.56
	0.5	1	0.5128	48.72
	0.75	1	0.5226	47.74
	0.9	1	0.4893	51.08
7 days	0.1	0.6624	0.4058	38.73
	0.25	0.7046	0.5311	24.62
	0.5	0.7807	0.5321	31.84
	0.75	0.8565	0.4783	44.16
	0.9	0.9188	0.3766	59.01
19 days	0.1	0.8384	0.3236	61.40
	0.25	0.8469	0.5589	34.01
	0.5	0.8608	0.6074	29.44
	0.75	0.8626	0.593	31.25
	0.9	0.8837	0.4033	54.36
30 days	0.1	0.6971	0.4848	30.45
	0.25	0.8122	0.5501	32.27
	0.5	0.8269	0.5777	30.14
	0.75	0.8558	0.6113	28.57
	0.9	0.8751	0.5394	38.36

These results indicate a close to bimodal distribution of predictability of combinations. There are some combinations which are highly predictable (having an average AUC of 0.864) and some which are poorly predictable (average AUC of around 0.5), and, on the average, a 40.95% difference in AUC exists between the two prefix sets. We conjecture that this due to the two types of reasons behind BGP dynamics: planned prefix traffic engineering leading to specific update patterns, and the non-stationary nature of link failures [13]. Understanding the reasons behind varying prefix predictability has been shown to be a difficult problem [13] because of lack of information about AS policies

and limited visibility to BGP updates from vantage points. This is similar to root cause identification for BGP updates, which is a hard problem as well [43, 44, 27, 45]. We leave detailed investigation of the causes behind prefix predictability to future work.

7. Larger Test Datasets

So far in this paper, we have used training and test sets which are constructed out of a sample of 10,000 combinations using 10-fold cross-validation. We now investigate the scalability of our models, where we apply the learned models to a large number of combinations. This may be required of a typical prediction application, if one is interested in predicting the availability of a set of prefixes from a large number of vantage points in the Internet.

To evaluate scalability, we learn Naïve Bayes and bagged decision trees from 10,000 combinations, but predict the availability of all the remaining combinations in each month (about 11.5 million). The prediction takes only about 2 minutes to complete for each of the models on a 3.6 GHz single-core machine. The prediction results for a typical case (t_l = 30 days, $t_l/(t_l + t_p)$ = 0.1) show about a 1-2% difference from the results in Table 6 and Table 8, as illustrated in Table 11. We therefore conclude that our models are scalable for availability prediction of a large number of combinations, without significant degradation in prediction quality. These results also show that the 10-fold cross-validation methodology does not suffer because of using a relatively low number (1000) of combinations in the test set.

Table 11: Percentage change in performance metrics of a large prediction dataset from Table 6 and Table 8 for t_l = 30 days, $t_l/(t_l + t_p)$ = 0.1.

Performance Metric	% Change for Naïve Bayes	% Change for Bagged Decision Trees
AUC	1.45	-1.52
Accuracy	2.01	-0.12

8. Conclusions and Future Work

In this paper, we have developed a long-term availability prediction framework that uses mean time to recovery, mean time to failure, prefix length, and update frequency as attributes. These attributes are easily computable from public RouteViews data observed for a short period of time. Our framework learns a prediction model from a set of Internet prefixes, and uses that model to predict availability of other prefixes. To the best of our knowledge, this is the first work that uses the similarity of prefix behavior in the Internet to predict properties such as availability.

Our simple prediction model is a good baseline with high true positive rate and accuracy. The model, however, has a high false positive rate and low AUC. Naïve Bayes and bagged decision trees improve on these metrics, and the latter performs best especially when the learning period t_l is shorter than about 3 weeks. For longer learning periods, Naïve Bayes performs best. The Naïve Bayes model, however, is highly susceptible to a change in attributes. We recommend the use of bagged decision trees, learned from a moderate learning period of a week or two, to predict

availability for longer future durations. The learning period can be a sliding window which slides with a granularity of a few days so as to feed the model with the most recent data for learning.

We also find that mean time to failure is the most important attribute for prediction followed by mean time to recovery. Past availability is inadequate to predict future availability, which is also a reason for the worse performance of the simple model. We quantify how prefix availability is related to prefix length and update frequency. Our results show that future availability is indeed predictable. The results are promising given that we are using only public information about prefixes and that we are building our model using a random set of prefixes.

We can extend our framework to predict availability of an arbitrary end point as viewed by an arbitrary vantage point by using techniques similar to those used in [19]. If the vantage point contributes to RouteViews, and the prefix, for which availability prediction is desired, is unknown to RouteViews, we can use any other known prefix in the same BGP atom [46] as the original one. This assumes that updates for two prefixes with the same AS path to a vantage point will be the same, which should be true unless there is a different BGP policy. If the vantage point is also arbitrary, one can predict AS path *ASP* between the vantage point and the prefix *P* using iPlane [19] and then choose a RouteViews vantage point that has sufficient overlap with *ASP* in AS path to *P* or a prefix in the same BGP atom as *P*. This extension (similar to the one used in [19]) may be inaccurate because of iPlane's inaccuracy and the incomplete overlap of AS paths, but it provides a rough availability estimate.

Other future work plans include considering availability of a prefix from multiple peers, as opposed to considering a single (peer, prefix) combination at a time. We will also investigate additional prefix attributes in our prediction model, such as the ASes to which the prefixes belong, and the AS paths to the prefixes. We hope to uncover interesting patterns, improve the prediction quality, and possibly exploit locality for feature prediction or for prefix clustering. We also aim to study availability across prefixes which are subprefixes of other prefixes, modifying the long-term availability metric to incorporate the time varying nature of announcement of these prefixes. Studying the inherent causes of predictability of prefixes is another topic of future work, since it gives a deeper insight into why some prefixes are more predictable than others. Finally, a study on control plane availability is not complete without studying the data plane. We plan to understand the correlation between the control and data planes, leveraging previous work in this area.

References

[1] John Shepler, The Holy Grail of five-nines reliability, `http://searchnetworking.techtarget.com/generic/0,295582,sid7_gci1064318,00.html`, 2005.

[2] M. Dahlin, B. B. V. Chandra, L. Gao, A. Nayate, End-to-end WAN service availability, IEEE/ACM Transactions on Networking 11 (2003) 300–313.

[3] V. Paxson, End-to-end routing behavior in the Internet, IEEE/ACM Transactions on Networking 5 (1997) 601–615.

[4] AT&T, AT&T High Speed Internet Business Edition Service Level Agreements, `http://www.att.com/gen/general?pid=6622`, URL Accessed April 2010.

[5] Sprint, Sprint service level agreements, `http://www.sprintworldwide.com/english/solutions/sla/`, URL Accessed April 2010.

[6] P. Pongpaibool, H. S. Kim, Providing end-to-end service level agreements across multiple ISP networks, Computer Networks 46 (2004) 3–18.

[7] R. Keralapura, C.-N. Chuah, G. Iannaccone, S. Bhattacharyya, Service availability: a new approach to characterize IP backbone topologies, Twelfth IEEE International Workshop on Quality of Service (IWQOS) (2004) 232–241.

[8] E. Zmijewski, Threats to Internet routing and global connectivity, in: Proc. of 20th Annual FIRST Conference 2008.

[9] Cable News Network (CNN), Internet failure hits two continents, http://edition.cnn.com/2008/WORLD/meast/02/01/internet.outage/index.html, 2008.

[10] Fox News, Severed Cables Cut Egypt's Internet Access Once Again, http://www.foxnews.com/story/0,2933,470224,00.html, 2008.

[11] Akamai, Mideast outage, http://www.akamai.com/mideast-outage, 2008.

[12] R. Bush, O. Maennel, M. Roughan, S. Uhlig, Internet optometry: assessing the broken glasses in internet reachability, in: IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, ACM, New York, NY, USA, 2009, pp. 242–253.

[13] Y. Zhang, Z. M. Mao, J. Wang, A Framework for Measuring and Predicting the Impact of Routing Changes, in: INFOCOM 2007, pp. 339–347.

[14] F. Wang, Z. M. Mao, J. Wang, L. Gao, R. Bush, A measurement study on the impact of routing events on end-to-end internet path performance, in: SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, ACM, New York, NY, USA, 2006, pp. 375–386.

[15] N. Feamster, D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, Measuring the effects of internet path faults on reactive routing, in: SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, ACM, New York, NY, USA, 2003, pp. 126–137.

[16] University of Oregon, Route Views Project, http://www.routeviews.org/, URL Accessed April 2010.

[17] M. Caesar, J. Rexford, BGP routing policies in ISP networks, IEEE Network Magazine 19 (2005) 5–11.

[18] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, Studying Blackholes in the Internet with Hubble, in: Proc. of NSDI 2008.

[19] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, A. Venkataramani, iPlane: An information plane for distributed services, in: Proc. of OSDI 2006, pp. 367–380.

[20] B. Cohen, Incentives Build Robustness in BitTorrent, http://www.bittorrent.org/bittorrentecon.pdf, 2003.

[21] R. Khosla, S. Fahmy, Y. C. Hu, J. Neville, Predicting Prefix Availability in the Internet, in: INFOCOM 2010.

[22] I. H. Witten, E. Frank, Data Mining: Practical machine learning tools and techniques, Morgan Kaufmann, San Francisco, 2nd edition, 2005.

[23] J. Rexford, J. Wang, Z. Xiao, Y. Zhang, BGP routing stability of popular destinations, in: Proc. of ACM IMW 2002.

[24] A. Broido, E. Nemeth, K. Claffy, Internet expansion, refinement and churn, in: European Transactions on Telecommunications 2002.

[25] X. Zhao, D. Massey, S. F. Wu, M. Lad, D. Pei, L. Wang, L. Zhang, Understanding BGP Behavior through a Study of DoD Prefixes, in: Proc. of DARPA Information Survivability Conference and Exposition 2003.

[26] R. V. Oliveira, R. Izhak-Ratzin, B. Zhang, L. Zhang, Measurement of highly active prefixes in BGP, in: Proc. of GLOBECOM 2005, volume 2.

[27] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, B. Maggs, Locating Internet Routing Instabilities, in: Proc. of ACM SIGCOMM 2004.

[28] D.-F. Chang, R. Govindan, J. Heidemann, The temporal and topological characteristics of BGP path changes, in: Proc of IEEE ICNP 2003, pp. 190–199.

[29] Y. Zhang, N. Duffield, V. Paxson, S. Shenker, On the constancy of Internet path properties, in: Proceedings of the Internet Measurement Workshop 2001, pp. 197–211.

[30] H. V. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, A. Venkataramani, iPlane Nano: path prediction for peer-to-peer applications, in: NSDI'09, pp. 137–152.

[31] RIPE, RIPE Network Coordination Centre, http://www.ris.ripe.net/source/, URL Accessed April 2010.

[32] B. Zhang, V. Kambhampati, M. Lad, D. Massey, L. Zhang, Identifying BGP routing table transfers, in: Proc. of ACM MineNet workshop 2005.

[33] Network Working Group , RFC 2519: A Framework for Inter-Domain Route Aggregation, http://www.ietf.org/rfc/rfc2519.txt, February 1999.

[34] E. L. Lehmann, J. P. Romano, Testing Statistical Hypotheses, Springer, New York, 3rd edition, 2005.

[35] S. S. Sawilowsky, Fermat, Schubert, Einstein, and Behrens-Fisher: The probable difference between two means when $\sigma_1 \neq \sigma_2$, in: Journal of Modern Applied Statistical Methods 2002, volume 1.

[36] T. Fawcett, ROC Graphs: Notes and Practical Considerations for Researchers, Tech Report HPL-2003-4, HP Laboratories, 2003. Available: http://home.comcast.net/~tom.fawcett/public_html/papers/ROC101.pdf.

[37] L. Hamel, Model assessment with ROC curves, in: The Encyclopedia of Data Warehousing and Mining, Idea Group Publishers, 2nd edition, 2008.

[38] R. Webster West, T distribution calculator, http://www.stat.tamu.edu/~west/applets/tdemo.html, URL accessed April 2010.

[39] J. R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, 1993.

[40] J. R. Quinlan, P. J. Compton, K. A. Horn, L. Lazarus, Inductive knowledge acquisition: a case study, in: Proceedings of the Second Australian Conference on Applications of expert systems 1987, pp. 137–156.

[41] J. Li, M. Guidero, Z. Wu, E. Purpus, T. Ehrenkranz, BGP routing dynamics revisited, SIGCOMM Computer Communication Review 37 (2007) 5–16.

[42] Second PacNOG Meeting, Conference and Educational Workshop, BGP Attributes and Policy Control, http://www.pacnog.org/pacnog2/track2/routing/b1-1up.pdf, February 1999.

[43] M. Caesar, L. Subramanian, R. H. Katz, Towards Localizing Root Causes of BGP Dynamics, Technical Report UCB/CSD-03-1292, EECS Department, University of California, Berkeley, 2003.

[44] J. Wu, Z. M. Mao, Finding a needle in a haystack: Pinpointing significant bgp routing changes in an ip network, in: In NSDI.

[45] R. Teixeira, J. Rexford, A measurement framework for pin-pointing routing changes, in: SIGCOMM 04 Workshops.

[46] A. Broido, K. Claffy, Analysis of RouteViews BGP data:policy atoms, in: Network Resource Data Management Workshop 2001.

Author Biographies

Ravish Khosla is a graduate student pursuing a Ph.D. in Electrical and Computer Engineering Department at Purdue University under the supervision of Prof Sonia Fahmy and Prof Y. Charlie Hu. His research interests lie in routing protocols in the Internet, specifically Border Gateway Protocol (BGP). He is currently working on evaluating BGP resilience by studying availability of Internet prefixes. He is a student member of IEEE. He has a MS in ECE from Purdue University with thesis titled "Reliable Data Dissemination in Energy Constrained Sensor Networks" and a B.Tech (H) in Electrical Engineering from IIT Kharagpur, India.

Sonia Fahmy is an Associate Professor of Computer Science department at Purdue University. She received her Ph.D. degree from the Ohio State University in 1999. Her current research interests lie in Internet measurement and tomography, network testbeds, network security, and wireless sensor networks. She received the National Science Foundation CAREER award in 2003, and the Schlumberger technical merit award in 2000. She is a member of the ACM and a senior member of the IEEE. For more information, please see: http://www.cs.purdue.edu/∼fahmy/

Y. Charlie Hu is an Associate Professor of Electrical and Computer Engineering at Purdue University. He received his Ph.D. degree in Computer Science from Harvard University in 1997 and was a research scientist at Rice University from 1997 to 2001. His research interests include wireless networking, overlay networking, operating systems, and distributed systems. He has published more than 120 papers in these areas. Dr. Hu received the NSF CAREER Award in 2003. He served as a TPC vice chair for ICPP 2004, ICDCS 2007, and SBAC PAD 2009. He is a senior member of ACM and IEEE.

Jennifer Neville is an Assistant Professor of Computer Science and Statistics at Purdue University. She received her Ph.D. degree from University of Massachusetts, Amherst, in 2006. Her research interests lie in data mining and machine learning techniques for relational data. She focuses on the development and analysis of relational learning algorithms and the application of those algorithms to real-world tasks. For more information, please see: http://www.cs.purdue.edu/∼neville/