Separating Wheat from Chaff: Winnowing Unintended Prefixes using Machine Learning

Andra Lutu*†, Marcelo Bagnulo†, Jesus Cid-Sueiro† and Olaf Maennel‡
*Institute IMDEA Networks, Spain
†University Carlos III of Madrid, Spain
‡Loughborough University, Loughborough, UK

***Abstract*—In this paper, we propose the use of *prefix visibility* at the interdomain level as an early symptom of anomalous events in the Internet. We focus on detecting anomalies which, despite their significant impact on the routing system, remain concealed from state of the art tools. We design a machine learning system to winnow the prefixes with *unintended* limited visibility – symptomatic of anomalous events – from the prefixes with *intended* limited visibility – resulting from legitimate routing operations. We train a *winnowing algorithm* with ground-truth data on 20,000 operational *limited visibility prefixes (LVPs)* already classified by the operators of the origin networks. The ground-truth was collected using the *BGP Visibility Scanner*, a tool we developed to provide operators with a multi-angle view on the efficacy of their routing policies. We build a dataset with the pre-classified prefixes and the features describing their visibility status dynamics. We further use this dataset to derive a boosted decision tree which winnows unintended *LVPs* with an accuracy of 95%.**

I. Introduction

The performance of the global routing system is vital to thousands of entities operating the Autonomous Systems (ASes) which make up the Internet. The Border Gateway Protocol (BGP) is currently responsible for the exchange of reachability information and the selection of paths according to their specified routing policies. By tweaking the BGP configurations, the network operators are able to express their interdomain routing preferences, designed to accommodate myriad economic and technical goals. The implementation of routing policies is a complicated process in itself, involving fine-tuning operations. Thus, it is an error-prone task and operators might end up with faulty configurations that impact the efficacy of their strategies and, most important, their revenues. Flawed routing policies cause for anomalies to emerge in the Internet, including interdomain prefix leaks or prefix hijacks. Over the last years, a lot of effort has gone in the direction of identifying, classifying and eliminating some of these anomalies [1].

Withal, even when correctly defining legitimate routing policies, unforeseen interactions between ASes have been observed to cause important disruptions that heavily affect the global routing system [2]. The main reason behind this resides in the fact that the actual inter-domain routing is the result of the interplay of many routing policies from ASes across the Internet, possibly bringing about a different outcome than the one expected [3]. Consequently, in order to ensure the efficacy of their routing policies, ASes need to periodically control how their preferences resonate in the external routing system.

In light of the perpetuity of the above-mentioned causes of anomalous interdomain events, there is an overall acute need for a simple warning system for faulty configurations and/or problematic external routing conditions to assist operators in optimizing the performance of their routing policies. In this paper, we argue that monitoring the *prefix visibility* at the interdomain level can be used to detect a subset of anomalous events that still remain hidden from state of the art tools [4], [5], despite their important impact on the routing system.

Using the interdomain route propagation process reflected in the global routing tables as an expression of routing policy interaction, we introduce the concept of **Limited-Visibility Prefix (LVP)**. We define *LVPs* as stable long-lived Internet routes that are visible in the routing tables of at least two different ASes and in *at most* 95% of all the global routing tables from the ASes analyzed. The choice of the 95% visibility threshold allows for a 5% error in the routing tables sampling process, also accommodating possible glitches that may appear in the data. Contrariwise, we define the **High-Visibility Prefixes (HVPs)** as the set of prefixes that are propagated in *at least* 95% of all the available global routing tables. We note that *the limited visibility does not imply limited reachability*[1]. There could be a less-specific *HV* covering prefix that provides reachability. Though some legitimate routing policies of an AS constrain the visibility of its prefixes in the Internet, the limited visibility can, more than often, stem from human operator errors or unpredicted interplay with the external netting of otherwise correctly defined routing policies.

The problem we challenge in this paper is **distinguishing the *unintended LVPs*, caused by misconfigurations or unforeseen routing policies interactions, from the *intended LVPs*, which are natural expressions of intentional routing policies in the Internet**. We design a machine learning *Winnowing Algorithm* able to predict with 95% accuracy if a LVP is intended or unintended. We rely on the robust machine learning concept of *boosted classification trees* [7] to train the system on ground-truth operational *LVPs* and thus enable it to

[1]In this sense, we also identify the **Dark Prefixes (DPs)** [6], which represent the subset of *LVPs* that are not covered by any *HV* less-specific prefix. These prefixes represent address space that, in the absence of a default route, may not be globally reachable.

978-14799-3360-0/14/$31.00 ©2014 IEEE

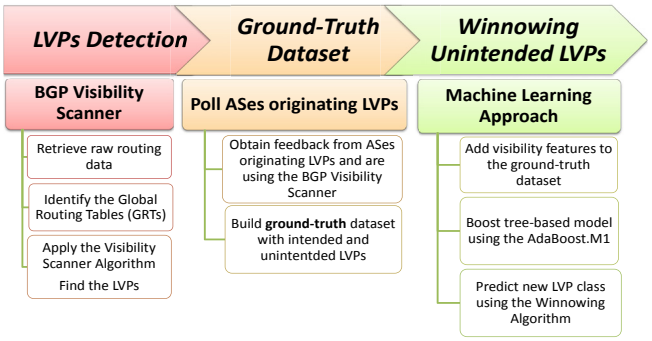

Fig. 1. Winnowing system flowchart.

learn the patterns of misconfigurations and backfired routing policies which are normally hard to detect. The classification model uses only visibility-related per-prefix features in order to predict the type of the *LVPs*.

Figure 1 shows the flowchart of the work we propose in this paper. Each processing block depicted summarizes one of the three main steps for building the winnowing system.

First, **we use the BGP Visibility Scanner as a data-mining engine for retrieving stable prefixes with limited visibility at the interdomain level**. The tool corroborates BGP routing information from more than 150 vantage points in the Internet, each within a different AS. Thus, it enables operators to achieve a multi-angled perspective on how their routing policies reflect at the interdomain level. Using the routing data retrieved from the widely-popular RIPE RIS[2] and RouteViews[3] projects, the tool analyses the public routing tables to retrieve *LVPs* on a daily basis. We have publicly released the BGP Visibility Scanner[4] in November 2012, allowing any network operator to check if the AS originates *LVPs*. Since the moment we have first made it available, the tool has been well received by the operational community[5] and it continues to attract a large amount of attention and feedback. In Section II, we provide a detailed description of the methodology employed by the scanner to detect the *LVPs*.

Second, using the survey integrated with the BGP Visibility Scanner, *we perform a poll on the ASes originating LVPs, inquiring if the identified prefixes are intended LVPs or not.* Thus, **we build a unique ground-truth dataset of 20,000 *LVPs* classified according to the direct feedback obtained from the ASes responding to the poll**. This dataset brings additional value to the system in that it accurately documents the distinct causes for *LVPs* and provides a deep understanding of the routing conditions which allows them to emerge. For example, we were able to identify more than 18,000 unintended *LVPs* and assist the origin networks in identifying their causes. Given that presently the visibility scanner detects, in average, 90,000 *LVPs* daily, the ground-truth dataset represents a statistically relevant sample of *LVPs*, recording multiple cases of misconfigurations, unforeseen interactions and intentional routing policies effects. In Section III we further expand on several of these operational examples.

Third, leveraging the machine-learning model of boosted classification trees, **we design the *Winnowing Algorithm* to distinguish *unintended LVPs* from the ones which are the stable expression of *intended* routing policies**. The main idea behind this algorithm is to combine many base learners, e.g., in our case the decision trees, to produce one robust classification algorithm. Using the exclusive ground-truth we have previously described, we derive a classification model to accurately winnow the identified *LVPs*. This brings real value to the operational community given that it has the capacity of revealing routing issues which are not easily discovered otherwise. We further expand on this approach in section IV.

II. Mining Limited Visibility Prefixes

The BGP Visibility Scanner is a *data mining*[6] *tool* for identifying stable prefixes with limited visibility at the interdomain level. In rough numbers, the visibility scanner detects on a daily basis around 90,000 *LVPs* and 430,000 *HVPs*. The daily set of prefixes with limited visibility can be queried using the BGP Visibility Scanner public web-page. Our goal is to collect feedback from the ASes querying for LVPs and to validate if the intention of the origin network is reflected in the visibility status of its prefixes. In this section we explain the methodology we use for parsing large datasets of raw BGP routing data to identify the visibility patterns of interest. An early version of this tool is documented in [8].

A. Refining the Raw Routing Data

We work with publicly available routing data, periodically retrieved from the RouteViews and RIPE RIS projects. These two repositories gather BGP data throughout the world, currently deploying 24 different collection points, which we further refer to as *collectors*. The collectors periodically receive over 400 BGP routing table *snapshots*, i.e., the content of the routing table at a certain moment in time from a router within an AS active as a monitor. A *monitor* represents a network peering with the public RIS/RouteViews repositories and willingly propagating its routing information to a collector. The monitors have different policies with respect to the public repositories, thus providing different types of routing feeds. We are able to identify three different types of feeds injected to collectors, namely *Partial Routing Tables*, *Global Routing Tables* and *Global Routing Table with internal routes*. However, only by comparing what we call *Global Routing Tables (GRTs)*, we can identify the sets of *HV* and *LV* prefixes which are of further interest. For the purpose of this paper, we loosely define the GRT as the entire routing table provided by a network to its customers requesting a full routing feed. This is not a formal definition, but it captures the main idea behind the type of data required by the system.

[2]RIPE RIS raw data: http://www.ripe.net/data-tools/stats/ris/ris-raw-data

[3]University of Oregon Route Views Project: http://www.routeviews.org/

[4]The BGP Visibility Scanner is publicly available at **visibility.it.uc3m.es**

[5]The BGP Visibility Scanner [8] has been presented in different network operators group meetings, including NANOG, LACNOG, UKNOF, EsNOG, and has also been announced on RIPE Labs [5].

[6]We loosely use the term data mining as the process of collecting, searching through, and analyzing a large amount of data in order to discover patterns or relationships.

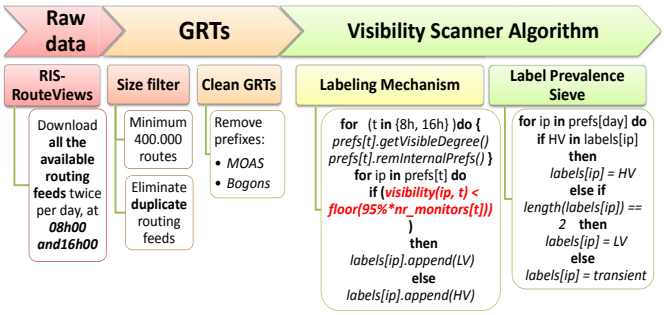

Fig. 2. LVPs Detection: Detailed methodology used for the BGP Visibility Scanner in Step 1 of the winnowing system flowchart depicted in Figure 1.

In Figure 2 we depict the main steps we follow for processing the raw data within the BGP Visibility Scanner. The first two processing blocks, namely **Raw data** and **GRTs**, focus on retrieving the routing data and parsing it in order to eliminate the unnecessary information. First, in order to exclude the Partial Routing Tables from the whole set of feeds, we verify the actual size of the routing table. We consider that *a global routing table from a monitor should have no less than 400,000 routing entries* [9]. Consequently, we keep for further study only the routing feeds that comply with the minimum limit on the number of prefixes. Second, we perform a couple of "sanitary" checks on the actual data contained in the identified GRTs, in order to further discard the information that is of no interest for the visibility scanner. Hence, using periodically updated filters from [10], we build the *bogon filter* which we apply on all the GRTs to eliminate any class of routes which should never appear in the Internet. At this point, it is also important to filter out the cases of prefixes emerging as *LV* prefixes which, in fact, are internal routes of the active monitors. In order to discard any internal paths, we remove all the routes learned only from one monitor, which is also the origin AS for the prefixes in question.

B. The Visibility Scanner Algorithm: Mining for LVPs

Having obtained the "clean" version of the GRTs, we further apply the **Visibility Scanner Algorithm** for identifying prefixes with *stable* limited visibility in the Internet, as depicted in Figure 2. Additionally, though converging routes also appear as limited visibility prefixes, they do not consist a routing policy expression. We avoid this type of false positive limited visibility prefixes in our results by analyzing two different samples of routing tables taken 8-hours apart. We define the *visibility degree* as the number of GRTs within the daily sample which contain (i.e., "see") a certain prefix, and the *visibility label* as the visibility status of each prefix, i.e. *LV* for Limited Visibility and *HV* for High Visibility.

We evaluate the *visibility degree* at every sampling moment and assign a *visibility label* at each time. *We define Limited Visibility prefixes as prefixes present in less than* 95% *of the active monitors at a sampling time.* Otherwise, the prefixes are defined as High Visibility prefixes. According to our threshold sensitivity analysis, we find that the set of *LVPs* is not sensitive to the value of the 95% visibility threshold [8]. Based on the visibility degree of the prefixes at each of the two sampling times (i.e. 08h00 and 16h00 UTC), we assign each prefix two corresponding *visibility labels*.

When deriving the final visibility label, we account for the dynamics of a prefix in time, as presented in the last block from Figure 2. The high visibility of a prefix in at least one monitor sample hints the fact that the route could reach all the observed ASes. Should this change during the analyzed time, it might be a cause of, for example, topology changes or failures. Therefore, we consider that *the HV label always prevails*, i.e. if a prefix is tagged as *HV* in one of the samples, it is tagged as *HV* in the final set. Otherwise, when no *HV* label is tagged, we analyze the cases of *LV* prefixes emerging in our results. If a prefix appears only at one sampling time and it is tagged as *LVP*, this might be a sign that the prefix is in the process of being withdrawn or, contrariwise, in the process of converging after just being injected. These particular routes cannot be qualified within the visibility scanner, thus we filter out any prefix with only one label in a day and that label being *LV*. The only case where we can say a prefix has limited visibility and mark it accordingly, is when the two labels assigned at each sampling time are both *LV*.

C. The LVPs in Rough Numbers

The set of LV prefixes identified using the BGP Visibility Scanner is made publicly available so that each network can check the status of its prefixes. The results are refreshed on a daily basis such that the networks can have an updated view on the efficacy of their routing policies. Every day we collect more than *500* routing feeds, for each of the two different sampling moments. After applying the *cleansing process*, we distinguish, in average, *150 GRTs* injected to the public repositories by unique ASes. We then compare the content of the 150 GRTs in order to identify the LVPs. In rough numbers, the daily overall total number of prefixes identified is around *550,000 prefixes*. Out of these, around *10,000 prefixes* are singled out as leaked internal routes and, consequently, discarded from our analysis. Furthermore, we remove the converging routes that may emerge as limited visibility in the visibility scanner. This incurs the elimination of about *8,000* additional prefixes in average. For the remaining prefixes we continue our visibility analysis and assign LV/HV visibility tags. We thus identify about *90,000 prefixes* in average that are tagged *LVP* and around *420,000* prefixes marked *HVP*. When checking how the two sets of prefixes overlap, we find that there are more than *2,500* LV prefixes without a covering high-visibility prefix, which we mark *DP*. We have observed more than 3,800 networks which inject limited visibility prefixes, out of which less than 1000 ASes originate DPs. These numbers may vary from day to day, given that neither the monitors providing their global routing tables, nor the actual content of the GRTs are the same in time.

III. Ground-Truth: Understanding *LVPs* through Operational Use Cases

The daily set of visibility data can be accessed by querying the BGP Visibility Scanner on a per-origin AS basis. Since

the tool first became publicly available, it gathered over 5,000 queries performed for more than 1,200 different origin ASes. Using the survey integrated with the Visibility Scanner, we have performed a poll on some of the ASes which were originating *LVPs*. Leveraging the feedback received, we build a unique ground-truth dataset including 20,000 *LVPs*. For each of these prefixes, the network operators reported which was the expected visibility status of the prefixes after defining their interdomain routing policies. We match the origin's intention with the observed visibility status of the prefixes identified with the BGP Visibility Scanner, and separate the *LVPs* in two pre-determined classes: *intended* and *unintended*. As a results, we identify 1,150 prefixes of the class *intended* and a staggering 18,850 *LVPs* of the class *unintended*. We note that the ground-truth dataset exhibits an important disproportion between the two defined classes. The *class imbalance problem* is a well-know issue in the machine-learning domain and it is characteristic to many other real-life applications.

After analyzing the feedback received on 20,000 operational LVPs out of the 90,000 identified on a daily average, we learn about a significant variety of factors which lead to prefixes with limited visibility. We develop next on a few relevant operational examples.

A. Intended LVPs

Some ASes create *LVPs* on purpose. There are several ways this can be done, including scoped advertisements (e.g. geographically scoped prefixes to offer connectivity only to networks located in a certain region) or advertisements only through (some) peering and not transit relationships. We next provide real cases of ASes deliberately restricting the propagation of their prefixes. For example, using the BGP Visibility Scanner, we were able to verify and validate the routing policies of two of the Internet DNS root-servers. For each root-server we have identified the presence of one more-specific *LV* prefix, which is meant for providing connectivity only to direct peers and, consequently, is tagged with the well-known *NO-EXPORT* community. The limited visibility of the more-specific prefix correctly reflects the impact of the *NO-EXPORT* community on the connectivity of the prefix. However, the *LV* prefix has global reachability due to the presence of *HV* less-specific prefixes, which is used by the root-servers in order to avoid connectivity issues.

In another case, the tool also validated the routing policy of a large content provider which deliberately limits the visibility of one of its prefixes in order to ensure that the incoming traffic is fed only through a geographically-specific local path.

B. Unintended LVPs

The second type of use cases we present captures unintended results of routing policies, i.e., accidental misconfigurations or unforeseen interactions between external routing policies at the interdomain level.

1) Misconfigurations/Accidental Errors:

In many cases, *LVPs* are the result of errors in the configuration of filters of the origin or other ASes that have received

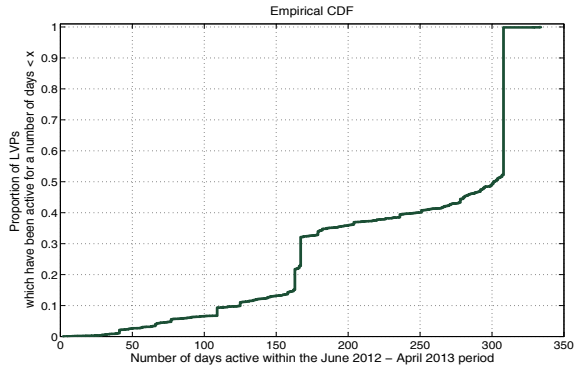

Fig. 3. Empirical CDF of *LVPs* known to be *unintended* on the number of days they were active from June 2012 until the end of April 2013.

the prefix announcement. For example, a large and widely-spread ISP learned that a large set of prefixes with limited visibility were leaking through some of its direct peers. After further investigation, the ISP was able to identify the misconfiguration of its outbound prefix-filters, which should have otherwise ensured that those prefixes were not being advertised to other networks. After correcting these issues, the origin AS successfully eliminated *4,000 unintended LVPs* of whose existence it was previously unaware even if they were affecting the receiver's routing policies for these prefixes. We note that these misconfigurations remained undetected for a very long time, given that we were able to detect the LVPs in question with more than 6 months before the BGP Visibility Scanner became operational.

2) Inflicted by Third-parties:

The interactions with legitimate and correctly defined routing policies of third-party ASes can limit the visibility of some prefixes at the interdomain level. For example, in the case of two different networks with correctly defined routing policies, the operators reported that the limited visibility of their prefixes is due to the impact of the filtering policies deployed by third-party ASes. More exactly, since the *LVPs* detected did not have an object defined in the Regional Registry's database, they were discarded by the ASes filtering based on the information retrieved from the registry databases. This, consequently, caused the prefix to suffer from unintended limited visibility at the interdomain level. Thanks to the BGP Visibility Scanner, the origin networks have discovered and solved the issue.

A clear example of the serious impact that these type of undetected mistakes might have on the origin networks is the case of an ISP whose prefixes were labeled by the Visibility Scanner as long-lived *dark prefixes*. This not only means that the prefixes were not globally propagated, but might have been suffering from limited reachability in the Internet. After investigating this issue, the origin AS found that, due to a mistake in the configurations of its transit provider, the prefixes were not being correctly advertised.

We have previously stated that the anomalous events included in the unintended *LVPs* dataset remain undetected for a long time. For example, for the majority of the operational cases of unintended *LVPs* above-explained, we have observed that the prefixes were originated long time before the BGP Visibility

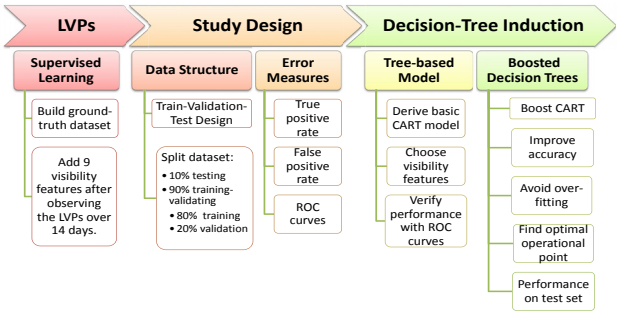

Fig. 4. Winnowing Unintended LVPs: detailed methodology used for step 3 of the winnowing system flowchart depicted in Figure 1.

Scanner tool was used to verify their existence. In order to support our statement on the average lifetime of a limited-visibility prefix, we use the BGP Visibility Scanner to generate the daily set of *LVPs* across a period of 11 months, from the beginning of June 2012 until the end of April 2013. This enables us to capture the dynamics of the *LVP* within this period. Figure 3 depicts the empirical CDF of the *LVPs* known to be unintended on the time they were active within this 11 months period.

IV. Winnowing Unintended LVPs: the Machine Learning Approach

In this section we propose a supervised learning approach for the classification of *LVPs* in two predetermined classes, namely *intended* expressions of routing policies or *unintended LVPs* generated by errors or by complex interaction between networks. Following the system flowchart depicted in Figure 1, this section corresponds to the last processing block in our work. Figure 4 shows an aggregated view on the different steps of this last part integrated in the winnowing system. We first describe the dataset we work with and the visibility features considered. We present the proposed machine learning study design and talk about the error measures we try to optimize. We advance a decision tree model using the optimal set of features, chosen according to the *information gain* measure. Leveraging the popular AdaBoost [7] algorithm, we boost the obtained basic model for achieving higher accuracy. We finally test the boosted tree-based model on the hold-out dataset.

A. Study Design

The ground-truth consists of 20,000 *LVPs*, each pre-classified to indicate if the prefix has *unintended* limited visibility or if it is the consequence of *intended* interdomain behaviour. In order to use it for building the machine learning Winnowing Algorithm, we first identify the full set of visibility features which we attach to each prefix. For every *LVP*, the origin AS is observed over a period of *14 days* prior to the feedback moment, to characterize the visibility dynamics captured in the BGP Visibility Scanner. All the possible visibility parameters are listed and explained in Table I.

We design the learning process in a training-validation-test format, as illustrated in the second processing block of the flowchart in Figure 4. The three datasets considered in the study design must be perfectly disjoint (i.e., we should observe no prefixes nor origin ASes in common between any two datasets of the three defined). We thus split the ground-truth such that all the LVPs generated by the same AS are included in one unique dataset. We impose these restrictions in order to ensure a correct estimation of the algorithm performance when predicting the class of LVPs originated by **new** ASes, on which we have no prior ground-truth. This is a challenge, since predicting the class of LVPs originated by a network on which we have previously trained is significantly easier.

1) Data Structure: We first create the hold-out test dataset, by randomly choosing 10% of all the ASes which provided feedback. This hold-out test set is under no circumstances to be used in the training-validation phase of the learning process. Its main purpose is to provide a realistic estimation of the performance of the optimal winnowing algorithm derived in the training-validation phase, by using independent data on which the algorithm was not previously trained nor validated.

We split the remaining data in two different sets, namely the training and validation datasets. We perform the separation such that the training dataset has approximatively 80% of the remaining ground-truth dataset, and the validation set, the rest of 20%. We require that this constraint is respected from the point of view of the total number of prefixes and also from the point of view of the number of different ASes, i.e., 80% of ASes must be in the training dataset and the rest of 20% in the validation dataset. Additionally, we require that the 80-20 split for the training-validation datasets is also respected for each of the two classes of prefixes. In other words, we must have 80% of the intended LVPs in the training and the rest 20% in the validation dataset and the same for the unintended LVPs. We impose these rules to ensure a similar distribution of prefixes and ASes in the training and in the validation datasets. We identify exactly 989 different ways in which the training-validation split can be done such that all the imposed constraints are met. In rough numbers, this means training on about 15,000 *LVPs*, validating on about 4,500 *LVPs* and, finally, testing on approximatively 100 independent *LVPs*.

We use the training dataset to derive the classification algorithm. To verify the performance of the model, we then perform an initial test of the classification model on the validation data. We repeat the training and initial testing for every of the 989 data splits and we choose an optimal overall algorithm. This optimal algorithm is thus finally evaluated by testing on new independent data, i.e., the hold-out data.

2) Error Measures: Before explaining the model selection and assessment, we first need to define the error measures which correctly describe the performance of the classification algorithm. Generally speaking, the *accuracy* of a classifier is defined as the percentage of ground-truth tuples which are correctly classified when tested on a set of data the model was not previously trained on. However, even when we obtain a very high value for the accuracy of the classifier, it may be the case that the model does not recognize very well the tuples of one of the two classes, especially when dealing with *unbalanced* classes in the data, which is our case. To evaluate the model performance, we define the following four concepts:

- *True Positive tuples [TP]*: number of tuples classified as

TABLE I
THE LIST OF PER-LVP VISIBILITY FEATURES. ALL THE VALUES ARE CALCULATED FOR AN OBSERVATION PERIOD OF **14 DAYS**. THE FEATURES ARE ORDERED IN DECREASING ORDER OF THEIR IMPORTANCE, ACCORDING TO THE INFORMATION GAIN.

Extracted per-LVP Feature	Explanation	Information Gain [weights]
`mean_nrPrefs`	Average number of LVPs generated by the same origin AS	0.319
`mean_MonitorsDetecting`	Average *proportion of active monitors* detecting the LVP	0.308
`std_MonitorsDetecting`	Standard deviation of the *proportion of active monitors* detecting the LVP	0.3068
`std_nrPrefs`	Standard deviation of the number of LVPs generated by the same origin AS	0.3060
`mean_VisibilityDegree`	Average *absolute visibility degree* for the LVP	0.244
`std_VisibilityDegree`	Standard deviation of the *absolute visibility degree* for the LVP	0.234
`length`	Prefix length of the LVP detected by the BGP Visibility Scanner	0.183
`TimeActive`	Proportion of time the LVP remained with limited visibility	0.153
`VisibilityLabel`	Visibility label assigned by the BGP Visibility Scanner [*LVP*/*DP*]	8.61e-05

unintended, which really are of unintended class.

- *False Positive tuples [FP]*: number of tuples classified as unintended, which really are of intended class.
- *True Negative tuples [TN]*: number of tuples classified as intended, which really are of intended class.
- *False Negative tuples [FN]*: number of tuples classified as intended, which really are of unintended class.

We further can define the two error metrics which allow us to correctly evaluate the performance of the classification by capturing the per-class classification accuracy. Namely, we use **True Positive rate** [TP_{rate}] and **False Positive rate** [FP_{rate}], defined as follows:

$$TP_{rate} = P(unintended \mid unintended\ ) \sim \frac{TP}{TP + FN},$$
$$FP_{rate} = P(unintended \mid intended) \sim \frac{FP}{TN + FP}.$$

Intuitively, the TP_{rate} represents the probability of predicting a tuple as unintended, conditioned by the fact that the tuple is indeed unintended. Similarly, the FP_{rate} represents the probability of classifying a tuple as unintended, conditioned by the fact that the tuple is actually intended.

We use *Receiver Operating Characteristic (ROC) curves* to visualize the performance of a classifier. Given a binary classification problem, like the one we are currently addressing, ROC curves allow us to analyze the trade-off between the true positive rate and the false positive rate. Many classification models, including decision trees, assign a probability to every tuple, expressing the degree to which the tuple is considered to belong to a certain class. By setting a decision threshold on these probabilities, we obtain a categorical classifier, i.e., the tuples are classified as *unintended* if their probability is higher than the fixed threshold, and "intended" otherwise. The performance of such a model is characterized by a single (TP_{rate}, FP_{rate}) pair of values which can be plotted in the ROC space. When considering different values of the decision threshold, we obtain a set of points capturing the TP_{rate} to FP_{rate} trade-off which can be plotted in the ROC space.Together, these points form the ROC curve of the decision model. Overall, the ROC curve gives an aggregated view on the performance of the model, without reference to a specific threshold value.

To asses the general performance of various models using the ROC space, we can measure the area under the curve (AUC). The ROC space usually shows an ascending diagonal line, corresponding to the ROC curve of a non-informative classifier (i.e. one making stochastic decisions independent on data). As the ROC curve goes closer to this line, the AUC goes closer to 0.5 and the model becomes less accurate, up to the point of random or even worse than random. Consequently, an AUC closer to 1 shows high performance for the model.

The ROC curve can be used to determine the operating point for the classification model. Note that, since each point in the curve corresponds to a decision threshold, selecting the operating point is equivalent to selecting a decision threshold. This selection may depend on the design considerations. For instance, if positive and negative examples are equally likely, the operating point maximizing the sum between the TP_{rate} and $1-\text{FP}_{\text{rate}}$ could be a good choice, because this is equivalent to maximizing the number of correctly detected tuples. However, the decision model operating with this threshold may not provide good results on new datasets with different distributions of tuples per class. To this end, a robust choice of the operating point is the *break even point*. The latter represents the value of the threshold where FP is equal to FN, and it can be shown to optimize the performance of the classifier under worst case conditions, i.e. under adversarial choices of the class distributions.

B. Decision Tree Induction

After previously defining the data structure, error measures and tools for assessing the performance of a classification model, we next explain the constructive process we follow in order to derive the tree-based Winnowing Algorithm. Following the flowchart depicted in Figure 4, we thus proceed to the last step, namely the decision tree induction. In the model, we choose decision trees as base learners which are boosted to create a robust classification model.

Decision tree induction is the process of deriving decision trees from the ground-truth datasets [11]. We use the training datasets to build the decision model, whose performance we initially evaluate using the validation datasets. We determine the overall optimal decision threshold and we further test the calibrated boosted decision tree on the hold-out dataset.

1) Decision Tree Model: In our work, we use the extensively tested and popular machine learning method called Classification and Regression Trees (CART) [12] for deriving and fine-tuning an anomaly decision tree model. Using all the 989 different data splits, we determine the optimal decision tree to be used in the following step, where by boosting we combine multiple such base learners to form a robust classification model. We derive the decision trees using the standard library *tree* for R [13].

Tree-based learning methods rely on iteratively partitioning the data into smaller groups of similar elements [11]. The splitting of the data is done using the features that best separate the outcomes. The key idea is to chose the splits which maximize the group homogeneity, i.e., how similar are the elements within the same group, or until the small groups are sufficiently *pure*. Choosing the right number of splits is a challenge, since we can easily overfit the model by considering splits that are very specific to the training data, or, contrariwise, underfit it by considering shallow general splits. Finding the correct balance is conditioned by finding the optimal set of features used to partition the data.

We next perform the feature selection in accordance with the *information gain* for each of the visibility features. The information gain is a widely accepted method for evaluating the capacity of a feature to distinguish between tuples of different classes. In the third column of Table I we show the weights associated to each of the 9 different visibility parameters after evaluating the information gain. In order to select the subset of features which ensures the optimal performance of the base learner for any training-validation data configuration, we adopt a progressive approach. We first verify the performance in every of the 989 training-validation splits of a tree model using only the highest weighted feature to classify the samples in the validation dataset. In a nutshell, we grow a different decision tree on each of the 989 training datasets, using as class-discriminating feature on the *mean_nrPrefs*. We then validate each of these 989 trees on the validation dataset of the considered data split and derive one ROC curve. Once having obtained 989 different ROC curves, we calculate the overall average performance of the decision tree by evaluating the average true positive rate and false positive rate at different decision thresholds. For every value of the threshold, the averaging algorithm selects from each ROC curve the corresponding point. These points are then averaged and produce a point for every value of the threshold, thus generating the threshold average ROC curve.

Next, in decreasing order of the per-feature Information Gain weight values, we progressively add one new feature to the tree classification model and evaluate its performance, e.g., if we were initially classifying only with *mean_nrPrefs*, in the next model we add *mean_MonitorsDetecting*, and so on. In total, we derive 989 x 9 ROC curves corresponding to each of the [feature subset - data split] combination. We repeat the process explained above for deriving the threshold averaged ROC curve per feature subset. We depict in Figure 5 the threshold-averaged ROC curve in each of the 9 cases.

To further identify the optimal set of features, we compare the AUC for the 9 different ROC curves in Figure 5. Consequently, we observe that the classification tree using the first 7 most-important features has the highest performance, with an average AUC equal to 0.94. In the overall best operating point for all the 989 data splits, the decision tree has an average TP_{rate} equal to 0.99 and an average FP_{rate} equal to 0.1.

2) Boosting for Improved Accuracy: We previously determined that the optimal decision tree uses only the first 7 most-important visibility features (as explained in Table I) to classify the *LVPs*. In order to improve the classification performance across all the possible data splits, we further combine 50 such base learners using the ensemble technique called *boosting* [14]. Boosting is one the most powerful learning mechanisms proposed in the last 20 years, used to improve the accuracy of a classification algorithm. The main idea behind this algorithm is to combine many base classifiers (e.g., in our case, CART models built with 7 features) to produce one robust classification algorithm. Unlike other boosting algorithms, AdaBoost [7] adjusts adaptively to the errors of the base learners derived at each iteration. For the purpose of this paper, we use the AdaBoost.M1 algorithm implemented in the publicly available package *adabag* [15] for R. In order to assure good general performance of the boosted tree-based model with 7 features, we determine next the overall optimal decision threshold in every of the 989 splits.

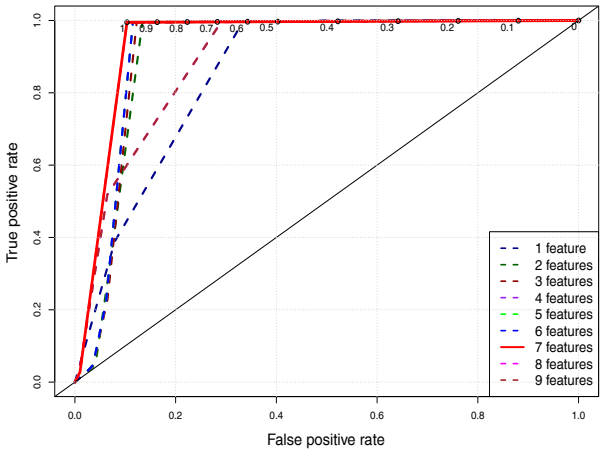

Fig. 5. Threshold-average ROC curves for performance estimation of the decision tree built with the 9 feature-sets. The red continuous curve for the model using the 7 most important features has the highest AUC and, thus, is the optimal model.

3) Learning the Optimal Decision Threshold: We boost the optimal decision tree for 50 times in the case of each of the 989 possible training-validation data configurations. Consequently, we train the classification model on the corresponding training dataset and perform an initial performance evaluation on the validation dataset. For each of the 989 boosted decision trees obtained, we derive the associated ROC curve, to obtain an aggregated view of the performance of each classifier. In order to determine the overall optimal operating point for the 989 models, we then calculate the *threshold averaged ROC curve* using all the 989 ROC curves.

Next, using the threshold-average ROC curve in Figure 6, we determine the optimal decision threshold for the boosted classification tree. We first note that, independently of the threshold value, the classification model is generally very accurate for any of the training-validation splits, with an AUC equal to 0.997. Moreover, we observe that in the best operating point, the decision algorithm has an average true positive rate equal to 0.98, and an average false positive rate of 0.05. The average accuracy of the decision model is 98%.

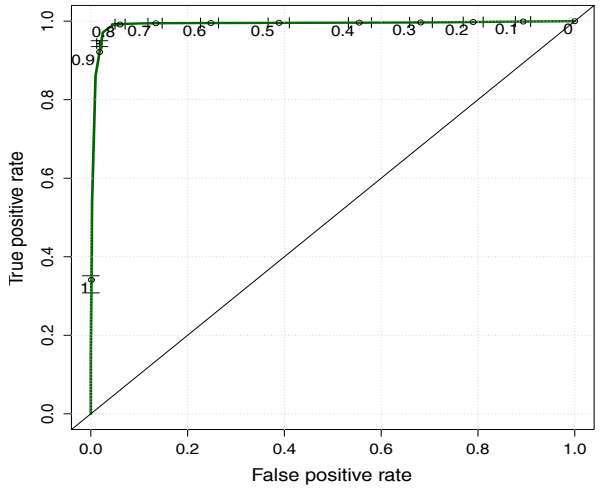

Fig. 6. Threshold-average ROC curve of the boosted decision trees derived using each of the 989 possible data splits.

Though this is a very positive result, our aim is to design a classification algorithm which generalizes well by accurately predicting for *any* previously unknown case of AS originating *LVPs*. Given that for a new AS we do not have ways to learn the distribution of intended and uninteded prefixes, we choose as optimal operating point the value of the threshold where the performance of the algorithm is the highest for any possible distribution of prefixes per class. In other words, we choose the value of the threshold which gives the best performance under the worst known conditions. This point is the break even point, where the threshold value is equal to 0.6. In this operating point, the decision algorithm has an average true positive rate equal to 0.99, and an average false positive rate of 0.24. The average accuracy of the tree-based model at the break-even point is 95%. Though we observe a slightly weaker performance than in the best operating point, we ensure that the decision algorithm at the break even point achieves optimal performance for new cases of ASes originating *LVPs*. We further refer to the boosted tree-based classification model using the 7 most-important features and operating with a decision threshold of 0.6 as the *Winnowing Algorithm*.

C. Performance on the Hold-Out Dataset

To further asses the Winnowing Algorithm performance, we test the model on the held-out independent dataset, which has not been previously used for training or for validation. We train the prediction model on all the available ground-truth data, encompassing both validation and training datasets. We then test the boosted decision tree on the tuples in the held-out dataset. The performance of the winnowing algorithm is characterized by an average true positive rate of 0.951, with a 95% confidence interval of [0.87, 0.99] and an average false positive rate of 0.01, with 95% confidence intervals of [0, 0.02]. We further calculate the accuracy of the Winnowing Algorithm, by evaluating the overall proportion of tuples correctly identified. We obtain an average accuracy on the held-out test set equal to 97.2%.

V. Discussion

In the previous section, we have built the Winnowing Algorithm, a boosted decision tree which can classify with 95% accuracy new *LVPs* detected with the BGP Visibility Scanner. Though the machine learning approach is gaining popularity for Internet-oriented applications, it is sometimes hard to understand its underlying. In this section, we provide the intuition behind the decision rules implemented in the winnowing system.

A. On the Visibility Features

One particularity of the Winnowing Algorithm is that, for classifying, it builds upon the 7 most important visibility features of the available ground-truth *LVPs*. We further observe that the set of features is consistent with the operational status of the routing system. For example, it has been long observed that accidental leaks usually generate a large number of prefixes at once. This explains why, in the context of the Winnowing Algorithm, the most important feature used to pinpoint *unintended LVPs* is the average number of *LVPs* injected by the same origin AS. Also, a high variation in the total number of *LVPs* from the same origin AS hints that the prefixes may not be expressions of stable long-lived routing expressions, but merely a side-effect of unexpected routing events. Additionally, we use the visibility degree and the proportion of active monitors from the daily sample detecting *LVPs*. This features capture the prefix visibility dynamics caused by the variations in the daily set of active monitors used. For example, we have observed that majority of unintended prefixes have a stable visibility degree of about 3, which is consistent with the fact that misconfigurations usually affect the routing policies of the ASes in the direct vicinity of the origin. Furthermore, discarding the last two features can be rationalized using the ground-truth data. For instance, as previously depicted in Figure 3, the lifetime of *unintended LVPs* is much longer than the expected lifetime of easily-noticeable anomalous events, which are quickly fixed by the origin. Thus, the parameter *time_active* does not discriminate well between the two classes of *LVPs*.

B. On the Data Structure

One of the restrictions we impose in the data structure proposed in the machine learning study design is that the *LVPs* originated by the same AS be all included in the same dataset, namely training, validation or testing. This restriction ensures that we are correctly using our winnowing mechanism to distinguish between *LVPs* from **new ASes** that might be suffering from unforeseen events. However, it is also important to accurately classify **new *LVPs*** from a network which already provided feedback used for deriving the Winnowing Algorithm. In order to verify the performance of the classification model on new *LVPs* originated by ASes used in the training phase, we perform a very simple experiment. Namely, we split the dataset independently of the origin AS. We withhold 100 random instances of each class for testing and use the rest for training. We find that the Winnowing Algorithm derived

in section IV-B2 performs a highly accurate classification of the test samples, only misclassifying one out of the 200 test tuples. In other words, when training on *LVPs* originated from one particular origin AS, the algorithm has a fairly easy task in classifying the new LVPs originated by the same AS.

VI. Related Work

Machine learning in the context of the interdomain routing has been already proven to be a successful approach. Using traffic feature distributions, Lakhina et al. [16] show that the existence of some anomalies can be detected from traffic flows. In [17], the authors advance a Bayesian framework for detecting mistakes in the router configuration files using statistical anomalies. Relying on network data, the authors propose in [18] the usage of statistical algorithms to detect deviations from the long-term profile of BGP routing updates. Similarly, in [19] the authors propose an instance-learning framework to identify deviations from the normal defined state of BGP routing dynamics. Likewise, in [20] Li et. al advance a rule-based framework for the detection of abnormal routing behaviour caused by a major worm or a blackout.

In this paper we focus on detecting anomalous events which, despite their high impact on the routing policies, emerge as legitimate expressions of routing policies. We use the BGP Visibility Scanner [8] as a tool for mining *LVPs* and capture their visibility dynamics. Despite that many other similar tools [4], [5] leverage the massive amount of available routing data, the visibility scanner is, to the best of our knowledge, the only tool offering specific information on global prefix visibility.

VII. Conclusions and Future Work

In this paper, we build a system for identifying cases of prefixes which emerge in the Internet as side-effect of routing misconfiguration or complex policy interactions. Leveraging the popular method of boosted classification trees, we use a unique ground-truth dataset in order to derive a classification model for the *LVPs* detected by the BGP Visibility Scanner. After extensive testing, we conclude that the proposed system winnows unintended *LVPs* with 95% accuracy. This further proves that visibility features are generally powerful to detect anomalies which, despite their impact on the routing system, are hard to single out. In order to develop the machine learning algorithms, we have used standard packages for R [13], thus reinforcing the robustness and generality of the system. Additionally, we make available an anonymized version of the ground-truth dataset upon request, allowing for the reproducibility of the machine learning analysis.

Using the Winnowing Algorithm, we can further classify new *LVPs* identified by the BGP Visibility Scanner. For example, for the set of *LVPs* retrieved on the 14^{th} of July, 2013, counting exactly 84,982 *LVPs*, the boosted decision tree classifies 25,860 *LVPs* as intended ($\sim 30\%$), and the rest of 59,117 *LVPs* as unintended ($\sim 70\%$). We are currently in the process of validating these results with the origin ASes.

The ground-truth dataset of 20,000 *LVPs* documents a wide range of cases of previously undetected anomalous events, affecting the interdomain entities. Though the root causes of the anomalies we detect are recurring in the Internet, their appearance in the BGP Visibility Scanner might change in time. We leave for future work the analysis on the lifetime of the ground-truth knowledge we have accumulated and the stability of the accuracy of the winnowing system in time.

Acknowledgements

This work was partially supported by the European Communitys Seventh Framework Programme (FP7/2007-2013) grant no. 317647 (Leone). We would like to thank Arturo Azcorra, Alberto Garcia-Martinez, Cristel Pelsser and Randy Bush for the discussions which helped improve this work.

References

[1] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, 2002.

[2] L. Quan, J. Heidemann, and Y. Pradkin, "Trinocular: Understanding Internet Reliability Through Adaptive Probing," in *Proceedings of the ACM SIGCOMM Conference*, Hong Kong, China, August 2013, p. to appear.

[3] T. Griffin and G. Huston, "BGP Wedgies," 2005, RFC 4264.

[4] Y.-J. Chi, R. Oliveira, and L. Zhang, "Cyclops: the AS-level connectivity observatory," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 5, 2008.

[5] "RIPE Labs." [Online]. Available: https://labs.ripe.net/

[6] C. Labovitz, A. Ahuja, and M. Bailey, "Shining Light on Dark Address Space," Arbor Netwoks, Ann Arbor, Michigan, USA, Tech. Rep. TR-2001-01, November 2001.

[7] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Proceedings of the Second European Conference on Computational Learning Theory*, ser. EuroCOLT '95, 1995.

[8] A. Lutu, M. Bagnulo, and O. Maennel, "The BGP Visibility Scanner," in *IEEE Global Internet Symposium (GI 2013)*, April 2013.

[9] "BGP Routing Table Analysis Report." [Online]. Available: http://bgp.potaroo.net/

[10] "Team Cymru - The Bogon Reference." [Online]. Available: http://www.cymru.com/BGP/bogons.html

[11] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[12] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[13] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org/

[14] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. New York: Springer-Verlag, 2001.

[15] E. Alfaro-Cortes, M. Gamez-Martinez, and N. Garcia-Rubio, *adabag: Applies multiclass AdaBoost.M1, AdaBoost-SAMME and Bagging*, 2012.

[16] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *Proceedings of SIGCOMM '05*, 2005.

[17] K. El-Arini and K. Killourhy, "Bayesian Detection of Router Configuration Anomalies," in *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, ser. MineNet '05, 2005.

[18] K. Zhang, A. Yen, X. Zhao, D. Massey, S. F. Wu, and L. Zhang, "On Detection of Anomalous Routing Dynamics in BGP," in *NETWORKING*, 2004.

[19] J. Zhang, J. Rexford, and J. Feigenbaum, "Learning-based anomaly detection in BGP updates," in *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, ser. MineNet '05, 2005, pp. 219–220.

[20] J. Li, D. Dou, Z. Wu, S. Kim, and V. Agarwal, "An internet routing forensics framework for discovering rules of abnormal BGP events," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, October 2005.