

URCA: Pulling out Anomalies by their Root Causes

Fernando Silveira
Thomson and
UPMC Paris Universit as

Christophe Diot
Thomson

Abstract—Traffic anomaly detection has received a lot of attention over recent years, but understanding the nature of these anomalies and identifying the flows involved is still a manual task, in most cases. We introduce Unsupervised Root Cause Analysis (URCA) which isolates anomalous traffic and classifies alarms with minimal manual assistance and high accuracy. URCA proceeds by successive reduction of the anomalous space, eliminating normal traffic based on feedback from the anomaly detection method. Classification is done by clustering a new anomaly with previously labeled events. We validate URCA using manually analyzed real anomalies as well as synthetic anomaly injection. Our validation shows that URCA can accurately diagnose a large range of anomaly types, including network scans, DDoS attacks, and major routing changes.

I. INTRODUCTION

Several traffic anomaly detection methods have been proposed [1]–[8], and some techniques are now part of commercial products^{1,2}. These methods have in common the ability to flag alarms for a variety of events that may be important to a network operations center (NOC) including abuse traffic, flash crowds, and routing outages. However, once an alarm is raised, a root cause analysis needs to be performed in order to know how to address the problem. Root cause analysis is usually left to network operators, who use their knowledge and intuition to analyze the traffic trace where the anomaly was flagged in search of events that can explain the anomaly. This manual process is both time-consuming and error prone. In a large ISP network with hundreds of links, the number of events that can trigger alarms may easily overload the NOC. Under such circumstances, the operator is likely to ignore alarms or never even deploy the detection system in the first place.

In this paper, we introduce URCA (Unsupervised Root Cause Analysis), a tool that automates traffic anomaly root cause analysis. URCA operates in two steps. First, it *identifies the root cause traffic* by analyzing the traffic in the anomalous time bins and iteratively removing flows that seem normal. The key insight behind our algorithm is to use feedback from the anomaly detector in order to determine which flows are more likely to be unaffected by the anomaly. While our identification method relies on input from the anomaly detector, it does not make any assumptions on the detection algorithm, and thus can be used with existing techniques such as PCA [3], Kalman [5], ARIMA [2], and Wavelets [1].

Having isolated the anomalous traffic, we build a graph representation of these flows which characterizes the type of

root cause event. Our classification algorithm then clusters the graph representations of previously labeled events, and builds a taxonomy tree where similar anomalies are nested in subtrees. We use this fact to develop a scheme that classifies new anomalies without knowing the number of clusters in the dataset. This is important since operators cannot tell in advance how many types of anomalies happen in their networks.

We study URCA using anomalies found by the ASTUTE anomaly detector [8] on traces from six different links from a large backbone (Section III-A). We have chosen to work with ASTUTE because it detects a class of anomalies that are more difficult to analyze by hand. Namely, Silveira et al. [8] showed that ASTUTE finds events that involve several flows at once, at the expense of anomalies caused by a few large flows (i.e., heavy hitters). Since heavy hitters can be easily found by checking the largest flows in a link (a standard feature in today’s routers), we believe that anomalies involving many flows pose a harder challenge from an operator’s point of view.

Our results show that URCA accurately identifies the causes of most anomalies (Section V). Using manually labeled anomalies from previous work on ASTUTE [8], we show that URCA’s identification step misses less than 11% of the anomalous flows in over 97% of the anomalies (Figure 5) while not including any normal flows. Classification results show that we can correctly label the causes of over 80% of the alarms (Figure 6). We also perform controlled anomaly injection experiments in five traces. The accuracy of our algorithms in these experiments is similar to that of the manually labeled anomalies, but we are able to identify which types of anomalies URCA handles less precisely. We find that small routing changes are the main source of errors in URCA.

The main novelty in our approach is that although our flow identification step uses the anomaly detector in its loop, URCA can be used with different anomaly detectors without requiring changes to their algorithms. Previous works have tried to address the root cause analysis problem by designing new detection methods with features that facilitate either identification [6], [8] or classification [4]. We review these approaches in Section VI. URCA only requires a couple of inputs from the detector, namely the set of flows that the detector uses to compute its alarm function and the alarm function itself. We also show that different detection methods from the literature can provide these inputs to URCA (Section II-B).

¹Peakflow - <http://www.arbornetworks.com>

²NetReflex - <http://www.guavus.com>

II. URCA AND ANOMALY DETECTORS

We analyze how events trigger anomalies and we specify the inputs required by URCA from detectors. We also define notation that we use to present our algorithms (Section IV).

A. The Root Cause Analysis Problem

The diagram in Figure 1 represents the relationship between anomalies and their causes. First, a root cause event impacts a subset of the traffic flows. The corresponding change in the total traffic then triggers an alarm in a given anomaly detector. Root cause analysis is the process of trying to revert the causal chain, going from the alarm to the anomalous traffic and from this traffic to the root cause event. URCA has an algorithm for each step of root cause analysis process: (1) *identifying* the flows whose change has triggered the anomaly, and (2) *classifying* the root cause event from the characteristics of the identified flows.

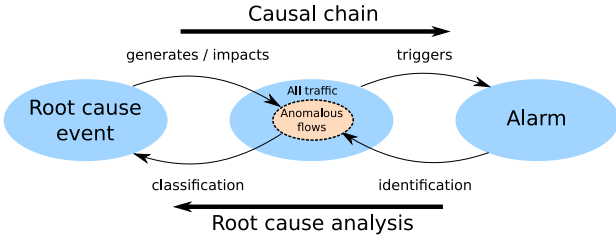


Fig. 1. Link between anomalies to their root cause events.

In the case of identification, the challenge is to efficiently narrow down the flows that could have triggered the anomaly. Suppose that, when a given detector flags an alarm, there are F traffic flows in the link. Without any assumptions on what could have caused the anomaly, we have to inspect 2^F subsets of flows, which is infeasible even for a few thousands flows. On the other hand, the challenge with classification is the lack of information about events. While in identification we still have a trace of flows that is used to compute the alarm, in classification, there are no records of the events that generated these flows. Although some events may have been recorded in some places (e.g., routing changes in BGP/IGP traces) this information is scattered across different networks, and it is not possible to collect it every time an alarm is flagged.

B. URCA’s Input from the Anomaly Detection Method

We consider that traffic traces are binned in fixed-size time intervals. Within each bin, packets that share the same 5-tuple information (i.e., IPs, ports, and transport protocol) are grouped into flows. In the n -th time bin, let $\mathcal{F}_1, \dots, \mathcal{F}_n$ denote the sets of flows in each of the bins observed so far. We represent the anomaly detector as a function $K(\cdot)$ that receives as input the flows in the $k + 1$ most recent bins, $\mathcal{F} = \{\mathcal{F}_{n-k}, \dots, \mathcal{F}_n\}$, and outputs a non-negative score denoting how anomalous is the n -th bin. Finally, we let K' be the detection threshold used to flag a bin as anomalous. Namely, an alarm is flagged if and only if $K(\mathcal{F}) > K'$.

In order to make the identification problem tractable, URCA uses two inputs from the detector: (1) the set of flows \mathcal{F} used by the detector to compute the alarm, and (2) the detector function $K(\cdot)$. Since the alarm is a function of \mathcal{F} , this set must contain the flows involved in the root cause event. This is important for our identification algorithm (Section IV-A) which starts from \mathcal{F} and iteratively narrows it down by eliminating normal flows. URCA uses the detector function $K(\cdot)$ to decide which flows seem normal. However, we treat this function as a “black box”, making no assumptions about its form. This is important because different anomaly detectors use different functions $K(\cdot)$.

For the ASTUTE anomaly detector [8], \mathcal{F} corresponds to the flows in the two most recent bins in the link being measured (i.e., $k = 2$). In the PCA method [3], the Q-statistic (PCA’s correspondent of the $K(\cdot)$ function) can be computed independently for each individual time bin (i.e., $k = 1$), but taking the flows from all links in the network, since PCA is a network-wide method. Detectors based on linear models (ARIMA/EWMA [2], and Kalman [5]) have infinite memory, but the contribution of the k -th most recent bin to the anomaly decays exponentially with k [9]. Thus, the flows in most recent bins (small k) are more likely to have caused the alarm.

Finally, for detectors that require a training phase (all of the above except for ASTUTE), we also need to assume that the flows involved in the anomaly were not active in the training phase; otherwise, our identification algorithm would have to recalibrate the detector’s model in each of its iterations. In this paper, we study URCA with anomalies found by ASTUTE for reasons described in the introduction. We plan to study URCA with other detectors in future work.

III. EXPERIMENTAL DATA

A. Anomaly Datasets

We analyze URCA using six backbone traffic traces summarized in Table I. These traces have been collected on routers in the GEANT2 backbone³. GEANT2 interconnects European NRENs (National Research and Education Networks) and provides them access to the commercial Internet as well as to other research networks worldwide. Each trace measures one direction of a link and contains flow records collected with Juniper’s J-Flow⁴ with 1/1000 random packet sampling. Trace A has been collected in November 2007 on a link connecting a Tier-1 ISP to GEANT2. We include trace A in our study because its anomalies have been manually labeled in previous work [8] (independently of this work). Traces B-F span the first quarter of 2009 and they include three links from NRENs and two links from a Tier-1 ISP connecting the commercial Internet to GEANT2.

We detect anomalies in each trace using the ASTUTE detector [8]. We divide the traffic in fixed sized time bins and compute the number of packets per 5-tuple flow. ASTUTE

³<http://www.geant2.net>

⁴<http://junos.juniper.net>

TABLE I
ANOMALY DATASETS USED TO EVALUATE URCA.

Trace ID	Collection period	Link type	Binning interval	# of anomalies $K'=6$
GEANT2-2007				
A	Nov 2007	Internet	5 min	93
GEANT2-2009				
B	Jan-Mar 2009	NREN	1 min	519
C	Jan-Mar 2009	NREN	1 min	329
D	Jan-Mar 2009	Internet	1 min	386
E	Jan-Mar 2009	Internet	1 min	1099
F	Jan-Mar 2009	NREN	1 min	680

receives a pair of consecutive bins as input and decides if these bins contain an anomaly.

ASTUTE has two parameters: (1) the binning interval, and (2) the detection threshold. Table I shows the binning interval for each trace. In trace A, we use a binning period of five minutes to obtain the same set of anomalies analyzed in [8]. In all other traces, we use a binning period of one minute to look for events at finer time scales. Note from Table I that using smaller time bins increases the number of alarms triggered per day. Still, our rationale is that in a real deployment, URCA would enable operators to run anomaly detectors at such fine time scales, since most of the alarms are going to be analyzed automatically. We flag the anomalous bins using a threshold value $K' = 6$ which is the same threshold value used in [8].

B. Flow Features

URCA uses flow features to identify and classify the anomalous traffic. Specifically, we consider the following features: (1) source and destination IP addresses; (2) source and destination ports; (3) input and output router interfaces; (4) previous-hop and next-hop AS numbers; and (5) source and destination AS numbers. Note that, while features (1) and (2) describe a flow’s *end hosts*, features (3), (4), and (5) describe the *network path* taken by the flow. We use this distinction in our flow identification algorithm in Section IV-A.

Flow features (1)-(4) can be obtained directly from our flow traces. To obtain the source and destination AS numbers, we need a map between IP prefixes and the ASes that own those prefixes. We build this IP-to-ASN map using publicly available data from UCLA’s IRL⁵ and the *whois* service maintained by Team Cymru⁶. These databases tell us the ASes where a given IP is located. Note that, once bootstrapped, this mapping should remain largely consistent for long periods of time, since we do not expect large and sudden changes in IP space allocation [10]. In this paper we take a single snapshot of the prefix-to-AS mapping and assume it does not change over the duration of our traces. In a real deployment of URCA, we could drop this assumption and keep the mapping up-to-date by receiving updates from systems such as Cyclops⁷, which tracks changes in prefix advertisements seen by hundreds of vantage points across the Internet.

⁵<http://irl.cs.ucla.edu/topology/>

⁶<http://www.team-cymru.org/Services/ip-to-asn.html>

⁷<http://cyclops.cs.ucla.edu/>

IV. ALGORITHMS FOR URCA

In this Section we present our algorithms to identify flows involved in an anomaly and then classify the anomaly’s root cause event.

A. Identification

Let \mathcal{F} be the set of flows in the link between two consecutive bins. When an alarm is triggered, we have $K(\mathcal{F}) > K'$ and our task is to identify the subset of flows $\mathcal{A} \subseteq \mathcal{F}$ that are involved in the anomaly.

Our identification algorithm starts from the set of flows \mathcal{F} used by the detector to compute the alarm and iteratively reduces it to smaller subsets, by discarding flows that seem normal to the detector function $K(\cdot)$. Our key insight is to consider only subsets where all flows share a value for some flow feature. Recall from Section III-B that each flow in \mathcal{F} is characterized by features. We denote by v_{fx} the value of feature x in flow f (e.g., a specific flow might have the value “80/TCP” for the feature “source port”).

Each iteration of our algorithm executes a procedure called *partition-reduce*. In short, this procedure (1) inputs a flow feature x (Section III-B) and a candidate set of flows \mathcal{A} which contains the anomaly, (2) partitions \mathcal{A} according to the different values of v_{fx} , and (3) returns a subset of \mathcal{A} which is likely to still contain the whole anomaly. Figure 2 shows four iterations of the partition-reduce procedure using different flow properties to identify the flows causing an anomaly from trace B. Each plot shows the number of flows per minute for a 30-minute window around the anomaly. In each plot, we also highlight the flows in the candidate set. Note that, although we show a 30-minute window, URCA uses only the flows in the two bins around the anomaly (highlighted with vertical bar). Plot 2(a) shows the initial state of the algorithm when all flows are candidates. Plot 2(b) shows that, after identifying the output router interface, the set of candidates is considerably reduced. Next, URCA reduces the destination IPs to a /12 prefix (Figure 2(c)). After reducing the destination port (Figure 2(d)), the spike caused by the anomaly is clearly isolated from the normal flows.

Put formally, given a flow feature x and a set of flows \mathcal{A} which is assumed to contain the anomaly, we partition \mathcal{A} into subsets \mathcal{A}_v containing the flows f whose feature value v_{fx} equals v . Then, we need to determine if one of the \mathcal{A}_v appears to contain the whole anomaly. We re-compute the anomaly detection function, $K(\cdot)$ on the *complement* of each \mathcal{A}_v with respect to \mathcal{F} . We denote this value by K_v^c , for feature value v , and we assume without loss of generality that the n feature values are sorted such that $K_{v_1}^c \leq K_{v_2}^c \leq \dots \leq K_{v_n}^c$. Finally, we check if (a) $K_{v_1}^c \leq K''$, for a chosen $K'' \leq K'$, and (b) $K_{v_2}^c > K'$. Note that K'' is a parameter introduced by our algorithm. We call it the *normality threshold*, and it represents how low the anomaly score should be so that a set of flows can be considered normal. If K'' is set too high (close to K'), our algorithm becomes aggressive in reducing the candidate set, and can wrongly discard some of the anomalous flows. If K'' is very low (close to zero), the algorithm is conservative

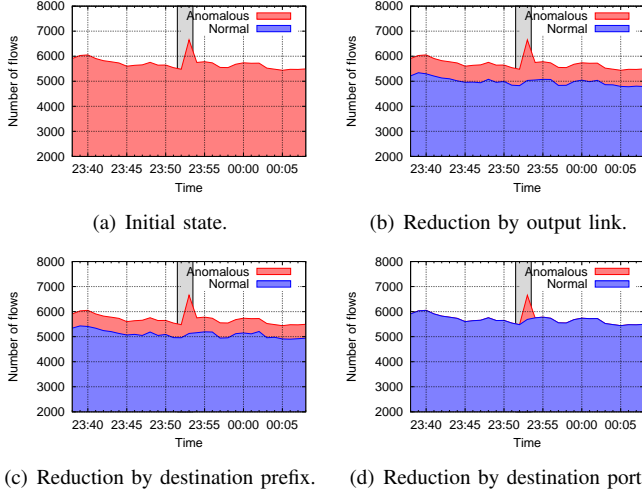


Fig. 2. Example of identification using the partition-reduce procedure.

and may output more flows than only the anomalous ones. In our experiments, we set $K'' = 3$, i.e., 50% of the value of K' . We found this threshold value to work well in practice, by running URCA with several values of K'' on trace A, for which we have manually identified flows and labels. We leave the automation of this calibration method for our future work.

Condition (a) above means that if we remove \mathcal{A}_{v_1} from the link, the remaining traffic seems normal to the anomaly detector. Conversely, condition (b) implies that removing the flows of any other individual feature value does not affect the anomaly. When these two conditions are met, we have a strong indication that the anomalous flows share the value v_1 for feature x . Thus, we can reduce the original candidate set \mathcal{A} to its subset \mathcal{A}_{v_1} . If either condition (a) or condition (b) does not hold, the partition-reduce procedure skips the reduction step and simply keeps \mathcal{A} as the candidate set. In that case, we try to partition the candidate set using other flow features until all features have been analyzed. Procedure 1 shows the pseudo-code description of the partition-reduce steps.

Procedure 1 PARTITION-REDUCE($\mathcal{F}, \mathcal{A}, p$)

Input: \mathcal{F} : the full set of flows

\mathcal{A} : the candidate set of flows

x : a flow feature

Output: a new candidate set that is a subset of \mathcal{A}

$\mathcal{V} \leftarrow \{v_{fx} \mid f \in \mathcal{A}\}$

for all $v \in \mathcal{V}$ **do**

$\mathcal{A}_v \leftarrow \{f \mid f \in \mathcal{A} \text{ and } v_{fx} = v\}$

$K_v^c \leftarrow K(\mathcal{F} \setminus \mathcal{A}_v)$

end for

{Let $K_{v_1}^c \leq K_{v_2}^c \leq \dots \leq K_{v_n}^c$ }

if $K_{v_1}^c \leq K''$ **and** $K_{v_2}^c > K'$ **then**

return \mathcal{A}_{v_1}

else

return \mathcal{A}

end if

The identification algorithm consists of several iterations of the partition-reduce procedure, each with a different flow feature. The initial candidate set is \mathcal{F} , and the output from each iteration becomes the candidate set for the next one. As a first phase, we run the partition-reduce procedure with the network path features ordered by decreasing aggregation level: input and output router interfaces, previous-hop and next-hop AS, and source and destination AS. Note that the more a feature aggregates traffic, the fewer subsets it partitions the candidate set and, consequently, the faster is the partition-reduce iteration. Moreover, if in one iteration we narrow down, e.g., the output interface, we also reduce the number of next-hop ASes in the candidate set to only ASes that are reachable through this interface. Thus, by analyzing the features in decreasing order of aggregation, we remove larger amounts of traffic in early iterations and speed up the algorithm. At the end of this first phase, if the candidate flows come from more than one source AS, we do not try to identify the source host features, since the anomaly likely involves multiple hosts in different networks. Likewise, we do not try to identify destination host features if the candidate flows go to more than one destination AS.

To identify end host features (i.e., IPs and ports) we also exploit the fact that their values are hierarchically structured. For instance, consecutive IP addresses can be aggregated by their longest prefix. By treating each *bit* of an IP address as a flow feature, we can use the partition-reduce procedure to discover, one bit at a time, the network prefix that contains all the anomalous hosts. Likewise, application ports can be grouped in ranges of consecutive port numbers, e.g., ports in the range 0-1023 are usually assigned to well-known services. Again, we can use the partition-reduce procedure with each bit of the port numbers to narrow down on the precise range of anomalous ports. Note that this makes identification of IP addresses and ports faster since each bit partitions the candidate flows in at most two sets.

We formally define the *hierarchical partition-reduce* procedure as follows. Given a candidate set \mathcal{A}_0 and a flow feature h with b bits, denoted h_1, \dots, h_b , we execute the default (non-hierarchical) partition-reduce procedure sequentially for each bit h_i , reducing the previous set \mathcal{A}_{i-1} to a subset $\mathcal{A}_i \subseteq \mathcal{A}_{i-1}$. We finish the loop after all bits of h have been discovered or, prematurely, if we find a bit h_i for which the partition-reduce procedure must skip the reduction step. When identifying source or destination IPs with this procedure, we only update the candidate set if we find an IP prefix longer than $/8$.

Note that, for a given anomaly, our identification algorithm runs at most 10 iterations of the partition-reduce procedure, i.e., one for each flow feature. In our traces, our identification algorithm analyzes over 50% of the anomalies in 5 iterations or less, and over 98% in 7 or less iterations.

B. Classification

After identifying the anomalous flows, we have to infer the event that caused it. Our approach is to look for patterns in the set of anomalous flows. This is a challenge since

anomalies can typically involve thousands of flows. We draw inspiration from the traffic classification literature, namely BLINC [11] which uses graphs to represent traffic flows generated by a given host. However, while BLINC tries to classify individual *hosts*, URCA deals with many sources and destinations simultaneously. In addition, URCA builds graphs that also contain network path features such as AS numbers and router interfaces.

Figure 3 shows examples of graphs representing three anomalies identified by URCA. We lay the 10 flow features (Section III-B) horizontally in an order that reflects the end-to-end path that flows follow through the network. Each node in the graph corresponds to the value of a flow feature. For each flow, we draw edges connecting its feature values in two consecutive layers, e.g., each source port value is connected to the source IPs that use that port. The weight of an edge is the volume change, across all flows in that edge, between the two bins where the anomaly happened. In Figure 3, we color edges in dark or light depending on whether their weight is respectively positive or negative. The thickness of edges is also proportional to their weights. Note that, to avoid cluttering of nodes in Figure 3, we have collapsed features with more than two values, and show only the number of distinct values in parenthesis.

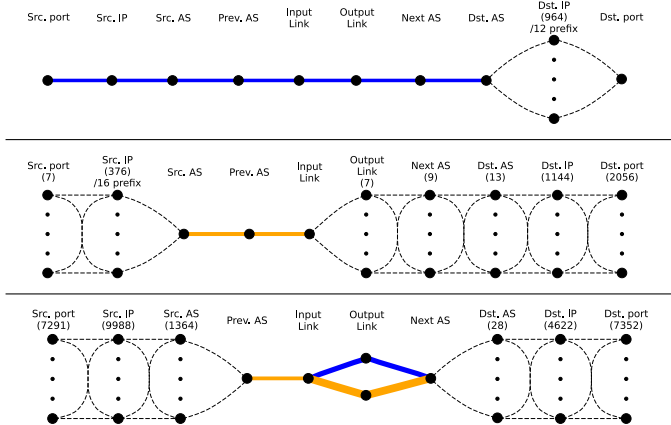


Fig. 3. Examples of root causes found by URCA. From top-down: network scan from Figure 2, upstream outage, egress change.

The graphs in Figure 3 tell a lot about the type of event triggering the anomaly. Note we omit the feature values from the nodes in the figure to avoid cluttering the diagrams. When we analyze these graphs in URCA, we include that information to help diagnose some of the events. The topmost graph corresponds to a network scan from a unique source to almost a thousand hosts within a /12 prefix in a single AS. Note that all the scan flows have a single destination port (found to be 22/TCP, used by SSH) suggesting the scanner is looking for vulnerable machines. The middle graph corresponds to an outage of an entire /16 network in a single AS. We verified that this AS is owned by Google, and the source hosts were essentially web and e-mail servers. The bottom anomaly is caused by an egress change inside GEANT2. This

is characterized by the different edge colors showing the traffic shifted from one output link to another.

We expect that flows impacted by similar types of events will produce similar graph structures. Under that assumption, we can build a taxonomy of anomaly root causes using hierarchical clustering [12]. Note that we could have used other clustering techniques such as k -means, or a different classification approach like k -nearest neighbors. However, these techniques would introduce additional parameters into URCA, i.e., the number of clusters in k -means or the neighborhood size in k -nearest neighbors. Hierarchical clustering allows us to develop a simple classification scheme which does not depend on extra parameters, as we show next.

In order to cluster anomaly root causes, we first need to map each root cause as a point in a space where the distance between two points reflects their dissimilarity. We use four types of coordinates: (1) the average flow size in packets; (2) the average packet size in bytes; (3) for each of the 10 flow features, the entropy of the distribution of packets per feature value [4]; and (4) for each of the 10 flow features, the fraction of feature values in the full link traffic that also appear in the root cause traffic. Coordinates of types (1)-(3) represent the structure of the root cause flows, while coordinates of type four (4) reflects how much of the total link traffic is impacted by the root cause event. Note that each of the 22 coordinates described above may have completely different scales. Given a set of root causes we want to cluster, we first normalize all the coordinate values by their corresponding standard deviations. This puts all the coordinates in equal scales and avoids a distance metric that gives more importance to the coordinates with higher variance.

Our classification algorithm works as follows. Given a set of labeled root causes \mathcal{L} , and an unknown anomaly u , we cluster the points in $\mathcal{L} \cup \{u\}$ using hierarchical clustering [12]. The clustering method inputs the coordinates of the anomalies and outputs a taxonomy tree \mathcal{T} where each anomaly corresponds to a leaf. Then, we find the subtree \mathcal{T}' that contains the siblings of u . If all the nodes in \mathcal{T}' have a single label, we classify u with that same label. If there is more than one label in \mathcal{T}' we consider that there is uncertainty about the true class of u . In that case, we leave it to the operator to choose which label in \mathcal{T}' best describes the type of u .

Figure 4 shows an example with the three possible outcomes of a classification. Consider we have 10 labeled anomalies of three known types (labeled a , b and c in the Figure), and we cluster them together with a new anomaly (shown as the dotted branch in the tree). Suppose that the new anomaly is of type b , but that is not known in advance. Figure 4(a) shows a *correct* classification, i.e., the subtree \mathcal{T}' of siblings of the new anomaly contains only nodes of type b . In Figure 4(b), since all nodes in \mathcal{T}' have type c , our algorithm would *misclassify* the new anomaly as type c . Figure 4(c) shows an example of *ambiguous* classification because \mathcal{T}' contains both a and b nodes. In the next section, we evaluate our classification algorithm by measuring how often a classification results in each of these outcomes.

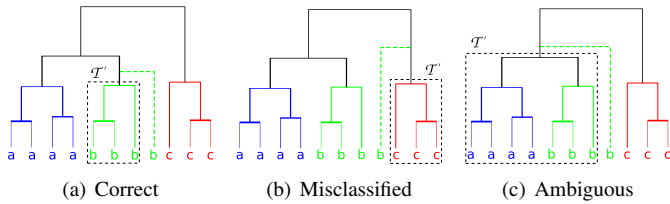


Fig. 4. Possible outcomes of our classification scheme.

Note that, although we use a clustering method, our classification algorithm does not require the user to know in advance the number of clusters in which the data is divided. This an important feature of our algorithm because, in practice, the operator cannot know *a priori* how many types of anomalies will happen on a link. Previous work that used clustering for classifying anomalies had to empirically study how to choose of the best number of clusters for their algorithms [3].

V. PERFORMANCE EVALUATION

We measure the accuracy of both the identification and classification algorithms using manually labeled anomalies (available for trace A only) and anomaly injection (traces B-F).

A. Evaluation with manually labeled anomalies

We compare URCA’s identification and classification algorithms to the manual anomaly characterization found in [8], which we treat as “ground truth”. For each anomalous time bin in trace A, [8] provides both the set of anomalous flows and the nature of the event that triggered the alarm.

1) *Identification*: We compare our identification algorithm’s output to the flows in our ground truth. For each anomaly, we measure (1) the fraction of ground truth flows (or packets) that are missed by URCA, and (2) the fraction of traffic found by URCA which is not in the ground truth. We call these metrics *missed traffic* and *extra traffic* respectively. Figure 5 shows CDFs of missed and extra traffic (in flows and packets) across all anomalies in trace A. URCA does not miss any flow or packet for nearly 36% of the anomalies. In addition, URCA misses less than 11% of the flows in the ground truth for 97% of the anomalies. Note also that URCA does not introduce extra flows to the anomalies. Among the anomalies where at least one flow is missing, 94% are caused by short measurement gaps.

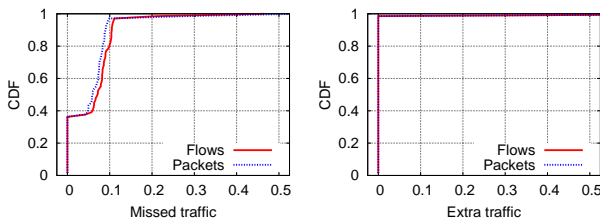


Fig. 5. Identification performance in trace A.

2) *Classification*: We evaluate our classification algorithm as follows. Given an anomaly flagged at time t , we consider that all previous anomalies have been correctly classified with their ground truth labels. We compute the coordinates of each anomaly using the flows found in the identification step. We then query the classification algorithm and compare its output to the corresponding ground truth label.

Figure 6 shows the cumulative number of correct, ambiguous, and misclassified anomalies across time. By the end of the month, URCA classifies over 80% of the anomalies correctly. Another 5% of the anomalies produce ambiguous labels, meaning that URCA cannot narrow down to a single anomaly type. We verified that in all of these ambiguous classifications, the correct label is among the labels proposed by our algorithm. Interestingly, we verified that even among the anomalies where the identification algorithm has missed some flows (Figure 5) over 90% are correctly classified, showing that our classification scheme is robust to errors in the identification step.

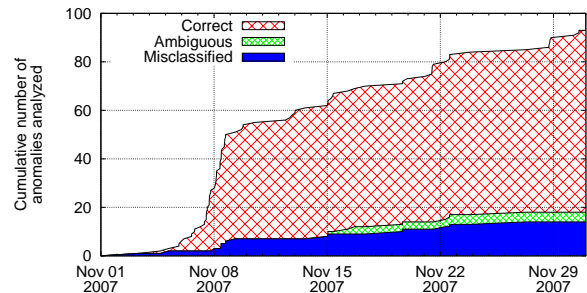


Fig. 6. Classification performance across time in trace A.

Among the 15% misclassified anomalies, 33% happen because the corresponding event type happens for the first time in the trace, and thus URCA has no chance of guessing the correct type. The remaining 66% are gaps in the trace being classified as routing changes or vice-versa. The main reason for this is that both these types of events impact traffic in similar ways.

B. Analysis of Unlabeled Traces

We run URCA on traces B-F, whose anomalies have not been analyzed before. Despite the absence of ground truth, this evaluation puts us in the same situation as an operator. We expect to observe results similar to the ones in trace A.

We run URCA for each anomaly in each trace. Using the identification output for an anomaly, we plot the graph representation of its anomalous flows (as in Figure 3), and the time series of flows per time bin, highlighting the anomalous flows (as in Figure 2(d)). Then, we check whether the label suggested by URCA matches how we would classify the anomaly by visual inspection of the graph and time series plots for that anomaly. We tag the anomaly as *correctly classified* if our visual inspection agrees with the classification output, and we tag it as *misclassified* otherwise. If URCA

outputs more than one label for the anomaly, we consider the classification *ambiguous*. If we are not able to confirm whether the classification output is correct through our visual inspection, we mark the anomaly as *unlabeled*.

Figure 7 shows how classification performance evolves across time, aggregated for traces B-F. We have been able to label 76% of the anomalies across the five traces. Among the anomalies we have labeled, 90% are correctly classified by URCA, 5% are misclassified, and the remaining 5% are ambiguous. Most of the misclassified anomalies (nearly 67%) are routing changes that are classified as gaps or vice-versa. Note that this result is similar to the one observed for trace A in the previous section. In the next section, we explore this issue in more detail using anomaly injection. Among the anomalies with ambiguous classifications, the correct class is among the labels suggested by URCA in 96% of the cases.

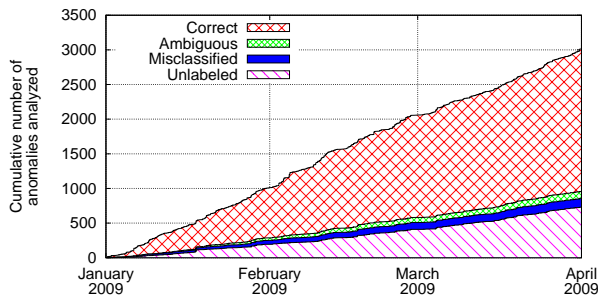


Fig. 7. Classification performance across time in traces B-F.

To understand the anomalies we could not label, we plot in Figure 8 the fraction of unknown events as we increase ASTUTE’s detection threshold, K' . The plot shows that only 5% of the most significant anomalies (high threshold values) have not been labeled. This shows that the events we have not been able to label are most likely small anomalies.

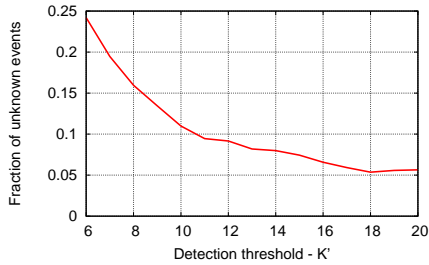


Fig. 8. Most unknown events correspond to small anomalies.

Figure 9 summarizes the types of events we have labeled in each of the five traces. We could identify three classes of malicious traffic: DoS attacks, port scans, and network scans. Note that in all these malicious anomalies, URCA’s identification algorithm gives us the IP addresses of both attackers and victims which can be used to block the unwanted traffic. Anomalies caused by large file transfers involve pairs of end hosts sending large packets through many parallel

TCP connections. We verified by the IPs and port numbers involved that most of these transfers are using GridFTP⁸ between research institutions. We also note that many of the alarms are triggered by gaps in the traces. While some of these gaps are legitimate link outages, most of them are caused by measurement artifacts in the J-Flow implementation that is used in GEANT2 routers [13]. Finally, routing changes are outages that impact only specific networks (BGP changes) or links inside GEANT2 (IGP changes).

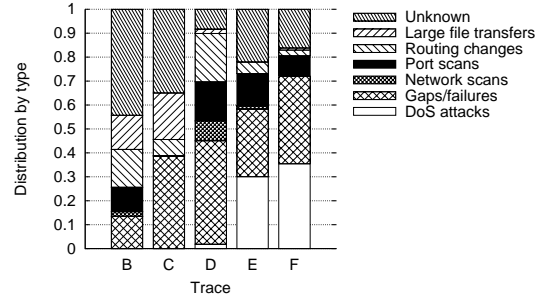


Fig. 9. Summary of classification results in traces B-F.

C. Identification Accuracy through Anomaly Injection

In order to evaluate flow identification accuracy in traces B-F, we conduct controlled experiments with anomaly injection. The advantage of using injection is that we can study our algorithm’s sensitivity to variables such as background traffic and the intensity of the anomalies injected.

We inject six types of anomalies summarized in Table II in the last day of traces B-F. For each 1-minute bin where there is not already an anomaly in the trace, we simulate the impact of each anomaly on the flows in that bin. If ASTUTE detects the injected anomaly, we run URCA’s identification algorithm.

The first three types of anomalies are network outage events which remove traffic from the link. We simulate link failures by removing all traffic flows that cross the link for a given duration. We inject failures of 20, 40, and 60 seconds. We simulate upstream (and downstream) routing changes by removing flows from a given source AS (resp. to a given destination AS) for an interval of 60 seconds. Given a time bin, we simulate a routing change involving each of the three source (or destination) ASes with the most packets sent (resp. received) during that bin.

The remaining three types of anomalies are events initiated by end hosts. We focus on three cases of malicious traffic: D-DoS attacks, network scans and port scans. To make these end hosts anomalies more realistic, we generate flows using features (IP prefixes, ports, AS numbers, etc) from similar anomalies found in these same links. This makes identification harder, since these feature values are also used by other (normal) flows in the link.

Figure 10 shows CDFs of missed and extra flows for failures and routing changes. In link failures of 20 seconds (Figure

⁸http://www.globus.org/grid_software/data/gridftp.php

TABLE II
TYPES OF ANOMALIES INJECTED.

Anomaly type	Impact description
Link failure	Erases all flows crossing the measured link during an interval.
Upstream routing change	Erases all flows coming from a source AS during an interval.
Downstream routing change	Erases all flows going towards a destination AS during an interval.
D-DoS attack	Adds flows from multiple sources in different networks to a single destination IP and port.
Network scan	Adds flows from a single source to several destination IPs with a single destination port.
Port scan	Adds flows from a source to a destination to several destination ports.

10(a)) our algorithm makes errors less than 1% of the time. Failures of 40 and 60 seconds (not shown in the plots) are easier to identify and have practically no identification errors. Although URCA misses some of the flows involved in the routing changes (left plots of Figures 10(b) and 10(c)), there are no extra flows for practically all of these anomalies (right plots in the same figures). This implies that URCA identifies the correct ASes impacted by each routing change, although it may not output all the traffic from (or to) that particular AS. Figure 11 shows the fraction of missed flows when during upstream routing changes for different source ASes. Note that the AS with most traffic (rank #1) is more easily identified, and performance degrades as we go to smaller ASes.

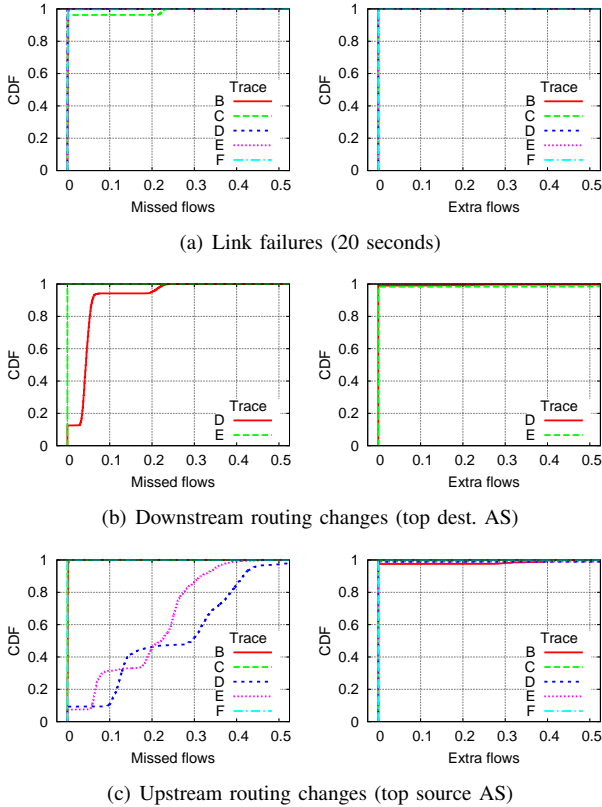


Fig. 10. Identification performance of failure events in traces B-F.

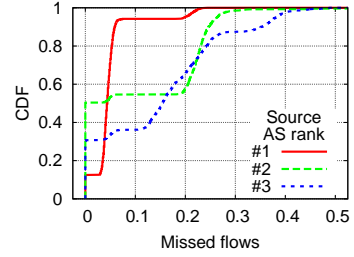


Fig. 11. Missed flows in routing changes for largest ASes in trace D.

For D-DoS attacks and both types of scans (Figure 12), our algorithm outputs the exact ground truth set of flows in over 80% of the bins where these anomalies are detected. We can explain this by the fact that these anomalies have very simple structure, i.e., at least one of their end host features has a single value. In that case, once our algorithm identifies a unique victim (e.g., for D-DoS and port scans) or a single vulnerable port being exploited (e.g., for networks scans), it is likely that most of the flows in the bin that have these specific feature values are part of the anomaly.

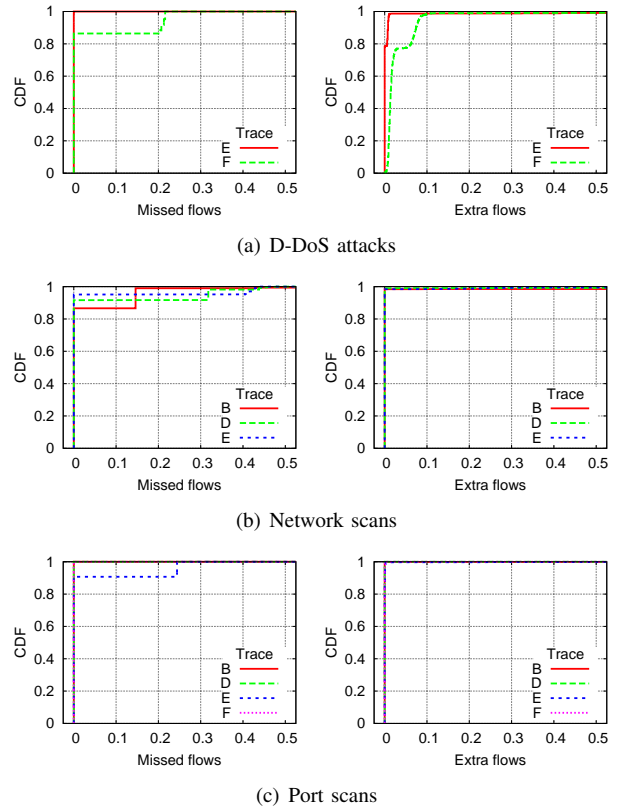


Fig. 12. Identification performance of malicious traffic in traces B-F.

VI. RELATED WORK

Few works have tried to address the problem of automating root cause analysis of traffic anomaly detectors. Lakhina et al. [4] have proposed a method based on clustering of

entropy residuals to classify the anomalies found by their PCA anomaly detector [3]. Since these entropy residuals are an internal variable of the PCA detector, the main limitation of this approach is that it only classifies anomalies that are visible by PCA on entropy. On the other hand, URCA classifies anomalies using metrics of the traffic they impact (as seen in Section IV-B).

Li et al. [6] combined PCA with traffic sketches to develop *Defeat*, a detector that can also identify the traffic involved in the anomalies. While that paper takes a concrete step towards automated root cause analysis, their solution is restricted to the PCA detector, and it requires a modification to the original detector's algorithm, i.e., the aggregation of traffic into k -ary sketches. URCA does not require changes to current anomaly detectors, nor is it a new detection method itself.

Although generic root cause analysis approaches still require manual investigation, there are traffic analysis tools that can help an operator to discover what is going on in the traffic when an anomaly happens. Several methods can identify high volume traffic clusters (i.e., heavy-hitters) [14]–[16], AutoFocus [14] being the most well-known example. The disadvantages of relying purely on tools like AutoFocus for root cause analysis are: (1) some anomalies (e.g., scans) involve low traffic volumes [4], [7], [8], and (2) none of these tools is explicitly trying to correlate their output with the alarms triggered by anomaly detectors, and thus still require manual intervention from operators. URCA overcomes these limitations by including the anomaly detector in its loop.

Several works have studied root cause analysis of routing changes [17]–[19]. It is important to highlight the difference between our problem (*traffic anomaly* root cause analysis) and the problem of *routing change* root cause analysis. In the former case, we use traffic data combined with anomaly detectors to infer the root causes of the alarms output by those detectors (some of which may even be related to routing changes). In the latter problem, one starts from a routing change observed, e.g., through BGP updates and tries to discover the events (e.g., specific link failures or policy changes) that triggered those updates in the first place.

VII. CONCLUSIONS

We have introduced URCA, a tool that explains the causes of traffic anomalies using little assistance from human operators. Our experimental evaluation has shown that URCA accurately isolates the anomalous traffic, with only few or zero missed flows in over 95% of the cases. URCA can considerably reduce the effort of network operators, by correctly classifying the root causes in 80% of the alarms. We have shown that most of the errors in URCA are caused by small routing events. In our future work, we plan to study ways to improve our algorithms for those particular types of anomalies, as well as study URCA with other anomaly detectors.

We note that even though we have conducted experiments using off-line traces, URCA can process these traces fast enough to allow an on-line deployment. For instance, traces B-F, which are each three months long, are fully processed by URCA in less than five hours using commodity hardware like current PCs.

REFERENCES

- [1] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proceedings of IMW*, November 2002, pp. 71–82.
- [2] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: methods, evaluation, and applications," in *Proceedings of IMC*, October 2003, pp. 234–247.
- [3] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *Proceedings of SIGCOMM*, August 2004, pp. 219–230.
- [4] —, "Mining anomalies using traffic feature distributions," in *Proceedings of SIGCOMM*, August 2005, pp. 217–228.
- [5] A. Soule, K. Salamatian, and N. Taft, "Combining filtering and statistical methods for anomaly detection," in *Proceedings of IMC*, October 2005, pp. 331–344.
- [6] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina, "Detection and identification of network anomalies using sketch subspaces," in *Proceedings of IMC*, October 2006, pp. 147–152.
- [7] G. Nychis, V. Sekar, D. G. Andersen, H. Kim, and H. Zhang, "An empirical evaluation of entropy-based anomaly detection," in *Proceedings of IMC*, October 2008, pp. 151–156.
- [8] F. Silveira, C. Diot, N. Taft, and R. Govindan, "Detecting correlated anomalous flows," Thomson, Tech. Rep. CR-PRL-2009-02-0001, 2009.
- [9] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. Springer, 2002.
- [10] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "An analysis of BGP multiple origin AS (MOAS) conflicts," in *Proceedings of IMW*, 2001, pp. 31–35.
- [11] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: multilevel traffic classification in the dark," in *Proceedings of SIGCOMM*, August 2005, pp. 229–240.
- [12] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, 1999.
- [13] I. Cunha, F. Silveira, R. Oliveira, R. Teixeira, and C. Diot, "Uncovering artifacts of flow measurement tools," in *Proceedings of PAM*, April 2009.
- [14] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *Proceedings of SIGCOMM*, August 2003, pp. 137–148.
- [15] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications," in *Proceedings of IMC*, October 2004, pp. 101–114.
- [16] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data," in *Proceedings of SIGMOD*, 2004, pp. 155–166.
- [17] M. Caesar, L. Subramanian, and R. H. Katz, "Towards localizing root causes of BGP dynamics," UC Berkeley, Tech. Rep., 2003.
- [18] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs, "Locating Internet routing instabilities," *SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 205–218, 2004.
- [19] R. Teixeira and J. Rexford, "A measurement framework for pin-pointing routing changes," in *Proceedings of NetT*, September 2004, pp. 313–318.