Inferring Applications at the Network Layer using Collective Traffic Statistics

Yu Jin*, Nick Duffield†, Patrick Haffner†, Subhabrata Sen†, Zhi-Li Zhang*
*Computer Science Dept., University of Minnesota †AT&T Research

***Abstract*—Operating, managing and securing networks require a thorough understanding of the demands placed on the network by the endpoints it interconnects, the characteristics of the traffic the endpoints generate, and the distribution of that traffic over the resources of the network infrastructure. A major differentiator in the types of resource required by traffic is the class of endpoint application that generates it. Service providers determine the application mix present in traffic via measurements, e.g., flow measurements furnished by routers. Previous work has shown that a fairly accurate determination of application type can be made from this data. However, protocol level information, such as TCP/UDP ports and other parts of the transport header, and also parts of the network header in some cases, may not be accessible due to the use of encryption or tunneling protocols by endpoints or gateways. Furthermore, the utility of ports as signifiers of application type has some limitations due to abuse and non-standard usage, amongst other reasons. These factors reduce the classification accuracy. In this paper, we propose a novel technique for inferring the distribution of application classes present in the aggregated traffic flows between endpoints, that exploits both the measured statistics of the traffic flows, and the spatial distribution of those flows across the network. Our method employs a two-step supervised model, where the *bootstrapping* step provides initial (inaccurate) inference on the traffic application classes, and the *graph-based calibration* step adjusts the initial inference through the collective spatial traffic distribution. In evaluations using real traffic flow measurements from a large ISP, we show how our method can accurately classify application types within aggregate traffic between endpoints, even without knowledge of ports and other traffic features. While the bootstrap estimate classifies the aggregates with 80% accuracy, incorporating spatial distributions through calibration increases the accuracy to 92%, i.e., roughly halving the number of errors.**

I. Introduction

Today's Internet connects hundreds of millions of endpoints such as computer servers, desktops, laptops, smartphones, and so forth. The management of communication networks that interconnect these hosts requires a detailed understanding of the traffic demands, both their spatial distribution over the network and the manner in which they use network resources. A major differentiator amongst the type of resources used comes from the class of applications that generates the traffic, for example, web access, VoIP or teleconferencing, email, online gaming, multimedia downloads or streaming, and peer-to-peer file sharing. Resource demands typically differ between application classes, e.g., VoIP sessions are low bandwidth but delay sensitive vs. high bandwidth downloading that is delay-elastic. Providers need to understand the traffic mix in order that the appropriate resource can be allocated to each application class. Service providers also need to characterize traffic presenting security threats (such as malware propagation and network attacks) in order that resources can be denied. Thus we can summarize the demand characterization problem as being to determine the spatial disposition and mix of resource demands across different application classes.

Service providers commonly characterize demands by means of traffic flow measurements provided by routers. These produce flow records, i.e., summaries of flows of packets with common header properties (IP source and destination addresses, TCP/UDP ports if visible, IP protocol) together with count and total bytes of the flows packets, and timing information. The network path taken by such traffic can be determined either directly by correlation of multiple measurements at separate locations, or indirectly by combining with network routing state [1].

Application class is commonly attributed to flow records on the basis of the reported TCP/UDP port numbers, sometimes in conjunction with traffic features such as total packets, bytes, duration, or combinations thereof. Attribution through port numbers has several shortcomings. First, this approach would only work for applications with *well-known* reserved ports [2], such as HTTP, Email, DNS; many applications either do not have well-known reserved ports (e.g., some chat or gaming applications), or may select ports arbitrarily (e.g., some peer-to-peer file sharing applications) or vary port usage in order to hinder detection (e.g., some network attacks). Second, even well-known ports may be mis-used or abused by other applications (e.g., some peer-to-peer applications use TCP port 80 to circumvent firewalls). Third, we would like to be able to attribute applications more finely that is generally possible by ports alone. For example, we would like to be able to separate multimedia streaming or downloads from general web accesses, both of which are typically performed through TCP port 80. Fourth, in some cases, UDP/TCP ports and some or all other transport layer header information may be absent from flow records, either because they are offset within the packet by encapsulation protocols and thus not reported (for example with GRE [3]), or because they are obscured by encryption (such as by IPSec [4]). Network-level header fields may also be absent (e.g., the IP protocol field with IPSec). Together, these factors motivate us to develop a worst-case approach to application identification that can function with only what we term the *basic flow features* in flow records: source and destination IP address, together with aggregate flow statistics of number of packets, their total bytes, and flow duration.

To solve the problem of application attribution knowing only

basic flow features, we are inspired by two strands of recent related work. First, the accuracy of port-based application classification is increased by including precisely the basic aggregate flow statistics as features on which to classify: a number of recent works have use machine learning techniques to construct flow based classifiers from training datasets of application-labeled flows through supervised learning [5], [6], or semi-supervised learning [7]. Second, the statistics of spatial features of the network flows of a given application, as characterized e.g., by the distribution of the number of different endpoints exchanging traffic with a given endpoint, depend strongly on the application class [8]. Therefore, in the case that port information is not available, we propose to *classify application traffic based upon the both traffic features and the spatial features of the traffic disposition*. Specifically, we propose a *machine learning approach to derive a set of rules to classify from these features*.

We now set up our framework to discuss the spatial properties of application traffic. First we adapt from [8] the notion of a *traffic activity graph* (TAG for short). The nodes of a TAG correspond to network endpoints, with two endpoints joined by an (undirected) edge if there is any traffic between them. To each edge we wish to associate a label that encodes the application class of the traffic. In the context of a set of flow measurement data, the existence of a flow between two endpoint implies the existence of an edge between the corresponding nodes; we will also associate with each edge a set of traffic features derived from all the flows between those nodes. Our problem then is to derive the set of edge labels from the TAG topology and the traffic features of the edges.

We propose a novel two-step approach to solving the edge label inference problem. In the first *bootstrapping* step, we apply a standard supervised machine learning algorithm to *classify* the edges of the TAG *based on solely the traffic attributes associated with each edge*, without using any structural properties of the TAG. The results of the bootstrapping step therefore gives us an initial edge labeling of the TAG. In the second *graph-based calibration* step, we then incorporate the inherent neighborhood and local properties of the edges in the TAG to calibrate (re-enforce or re-label) the edge labeling. Both two steps can be formulated as classical *multi-class* classification problems. We break down each multi-classification problem into a series of binary classification sub-problems. The decomposition into multiple binary subproblems that runs through our method has been used in [9] to make learning scale to very large data volumes, in the sense of achieving realistic learning times and classification throughput.

We validate the proposed approach through the application of traffic classification using flow records gathered over a period of a year from a large ISP. For the purpose of establishing ground truth, the flow records are annotated with application labels derived by an operational packet-level classification system that utilizes application protocol level information. As test data for classification we used the same flow records, stripped off application labels and all information relating to protocols above the network layer; hence all TCP/UDP protocol information, such as port numbers, is stripped. The evaluation result shows that our method can reduce 50% of the errors from the best results in the bootstrapping step, and hence increase the overall accuracy from 80% to 92%. More importantly, the accuracy is improved for all application classes. We also study the temporal persistence of the accuracy enhancement over the year. Our proposed inference method exhibits strong temporal stability. When the training time for the collaborative prediction model and the testing time is one month apart, we can achieve 40% reduction in the errors from the bootstrapping step. Even when the time gap extends to 1 year, a noticeable error reduction of 23% is still observed.

The remainder of the paper is organized as follows. Section II introduces the datasets and the notion of colored TAGs for visualizing the spatial distribution of traffic classes in the network. The TAG edge label inference problem is defined formally in Section III and the related work is discussed at the end of Section III. We then propose a two-stage model for solving the TAG edge label inference problem in Section IV, and evaluate the performance our method in Section V. Finally, Section VI concludes the paper.

II. Datasets, Application Classes and *Colored* TAGs

In this section we first describe the network datasets, and present the application classes and traffic statistics that will be used for our study. We then formally introduce the new notion of *colored* TAGs (traffic activity graphs). The colored TAGs provide us a tool to visualize the spatial distribution of applications in the network. This motivates our work of inferring network applications using collective traffic statistics.

TABLE I
TCP/UDP Broad Application Classes

ID	TCP/UDP	Class/Label	Example Applications
1	TCP/UDP	Business	Middleware, VPN, etc.
2	TCP/UDP	Chat	Messengers, IRC, etc.
3	TCP/UDP	DNS	DNS application
4	TCP/UDP	FileSharing	P2P applications
5	TCP	FTP	FTP application
6	TCP/UDP	Games	Everquest, Xbox, etc.
7	TCP	Mail	SMTP and POP
8	TCP/UDP	Multimedia	RTSP, MS-Streaming, etc.
9	TCP/UDP	NetNews	News
10	TCP	SecurityThreat	Worms and trojans
11	TCP/UDP	VoIP	SIP application
12	TCP	Web	HTTP application

A. Datasets and Application Classes

The datasets used for our study are network flow records from a large ISP over the time period of a year. A flow is a sequence of packets with a common key – namely, the standard 5-tuple of IP protocol, source and destination IP addresses, and TCP/UDP ports – that are localized in time. Flow measurements comprise summary statistics that aggregate information derived from a flow's packet headers (including the key, aggregate packet and byte counts for the flow, and timing information) that are exported as IP flow records to a collector. The flow records are collected by special purpose traffic measurement devices operating at two

geographically dispersed sites of the ISP. Due to the huge traffic volume, sampling is employed in the creation of flow records, with 1 out of 20 flows reported on, sampling over the standard flow level 5-tuples. However, for each sampled flow, the flow record aggregates header information from all its packets, without further sampling. The datasets contain flow records from approximately 40,000 ISP network endpoints gathered at two sites, representing several hundred Terabytes of network traffic. No endpoint is represented at both sites.

Serving as the *ground truth* for both training and testing purposes, the flow records in the data set are annotated with a number of broad "application class" *labels*, which are then used to define *edge types* (or *edge labels*), representing the dominant application between two endpoints. Similar to [10], [11], the labels are generated in an automated way by the measurement devices, using a set of packet-level rules based on combinations of packet signatures that operate on layer-4 packet header information, and layer-7 application protocol signatures. The flow records do not include any application data; neither do they report any user identity information. Motivated by the network management tasks of the large ISP, we define twelve (12) broad application class labels, as shown in Table I. We note that these 12 application classes are *not* defined uniquely by transport protocols and port numbers; i.e., there is no one-to-one correspondence between application labels and port numbers. For instance, while HTTP and TCP port 80 are often used by the four classes, we separate the more specific `NetNews`, `Multimedia` (as well as some `Business`) applications from general `Web` accesses. Furthermore, `Multimedia` and `Business` may use port numbers other than TCP port 80.

The distribution of flows over the application classes of Table I is highly unbalanced. The largest two classes, `Web` and `FileSharing`, account for 60% to 80% of the total flows in different weeks, while the smallest classes (e.g., `NetNews` and `SecurityThreat`) contain only a few thousand flows out of millions. In addition, a portion of the flows (29.4% of total flows representing 19.9% of total bytes) *cannot be classified* using the packet-based classifier, i.e., they do not match any rule. This can be caused by encryption of application level information, or the presence of new applications or security threats for which signatures are not yet developed. We call these flows `Unknown`, and exclude them from the datasets used in our study, for lack of ground truth.

Using the application class labels of the flows, we now examine how many pairs of endpoints generate only one type of application traffic, i.e., whether all flows between them fall within a single application class. We vary the time windows for constructing the TAG from 1 hour to 1 day. We observe that in all cases, even when extending the time window to an entire day, for a predominant majority (nearly 99.5% or more) of the edges, all flows between the two endpoints of an edge fall within a single application class. Thus the edge can be assigned a single label. Further analysis shows that even among the remaining edges with flows belonging to multiple classes, one class dominates, and thus the edge can be labeled by the dominant application class. Therefore in the remainder of this paper we will often use the term "application class/label" and "edge type/label" interchangeably, choosing the term that best suits the context. We note that although in terms of the number of flows in the corresponding dataset, the `FileSharing` and `Web` flows are of roughly similar proportions, there are far more `FileSharing` *edges* (75%) than `Web` (20%) *edges*. This in fact is not surprising, as a) there are far more client endpoints than server endpoints; *and* b) a `FileSharing` flow typically involve two client endpoints, while a `Web` flow typically involves a client and a server.

TABLE II
FLOW-LEVEL FEATURES

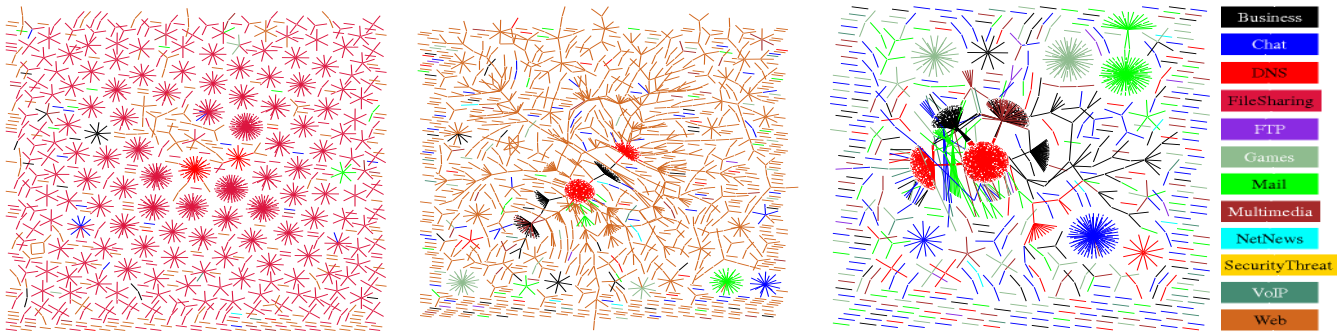
Name	Type	Name	Type
duration	numeric (*)	packet	numeric
mean_packet_size (mps)	numeric (*)	byte	numeric
mean_packet_rate (mpr)	numeric (*)	tos	numeric
toscount	numeric	numtosbytes	numeric
tcpflags	text	srcinnet	{0,1}
dstinnet	{0,1}		

Finally, as part of the edge color inference problem, we augment each edge with an attribute set. The attribute set used is a set of *flow-level* traffic statistics derived from flows between the two endpoints of an edge. These attributes are listed in Table II[1]. *Duration*, *packet* and *byte* represent the length of the flow, number of packets and bytes in the flow, respectively. *Mean_packet_size* is the average bytes per packet, and *mean_packet_rate* is the average packet interarrival time in seconds. The *tcpflag* feature contains all possible TCP flags in the packets. The TOS (type of service) related features *tos*, *toscount* and *numtosbytes* are the predominant TOS byte, the number of packets that were marked with tos, and the number of different tos bytes seen in a flow, respectively. The last two features *srcinnet/dstinnet* equals 1 if the source/destination address belongs to the ISP network, and 0 otherwise.

B. Colored TAGs: Definition and Properties

Given the datasets, we now introduce the notion of *colored traffic activity graph* (or *colored* TAG in short) which embodies both the spatial disposition of traffic as well as the applications used. As in [8], a TAG (traffic activity graph) is a bi-partite graph defined using the flows (with known class labels) from a specific time window T (e.g., 1 hour or 1 day), and describes endpoint pairs represented in the flows. Formally, let $\mathcal{H} = \mathcal{IH} \cup \mathcal{OH}$ denote the set of observed endpoints, where $\mathcal{IH}$ is the set of all endpoints (hosts) *internal* to the ISP network, and $\mathcal{OH}$ is the set of endpoints (hosts) *external* to the ISP that exchange traffic with those in $\mathcal{IH}$. We first construct the (*uncolored*) TAG, $\mathcal{G} = (\mathcal{H}, \mathcal{E})$, as follows: we include an edge e_{ij} in the edge set $\mathcal{E}$ if and only if we observe at least one flow between an internal and external endpoints pair, $h_i \in \mathcal{IH}$ and $h_j \in \mathcal{OH}$. (For topological reasons the dataset does not include any flows exchange between pairs of internal nodes). We then define

[1]The features marked (*) are not reported directly in the flow record, but computed from quantities thereof.



(a) all applications (b) `FileSharing` removed (c) `FileSharing` and `Web` removed

Fig. 1. TAGs containing 2000 edges, where different applications are represented with different colors.

the *colored* TAG by coloring each edge of the TAG using the (dominant) application class label of the flows between the two endpoints of the edge. Formally, for each edge $e_{ij} \in \mathcal{E}$, we define $L(e_{ij})$ as the (dominant) application class label associated with the edge.

As an example, Fig. 1[a] displays a (small) TAG constructed using the first 2000 edges starting at 10AM on 05/03/2008. The dominance of `FileSharing` edges obscures most Web and other edges in Fig. 1[a]. In Fig. 1[b], we illustrate the resulting colored TAG *after removing* `FileSharing` *traffic*. We now see that the Web edges now dominate. In Fig. 1[c], we remove both the `FileSharing` and `Web` traffic (the two dominant application classes) to better visualize the spatial distribution of traffic for less used application classes.

This small example helps illustrate several salient *local* properties of a *colored* TAG, which motivates the *TAG edge label inference* problem addressed in this paper. We see that the edge labels tend to be clustered together – where edges incident on some nodes are all of the same color – and hence regions of the TAG seem to have the same color. This seems to suggest that certain groups of hosts tend to generate application traffic in a similar way (e.g., exchanging traffic with the same set of web servers), thereby showing up with the same color on the TAG. On the other hand, *local* graph structures do not appear to be indicative of the color of the edge clusters. For instance, many edge clusters have a similar "star-like" structure, but with *different* colors. We also see many edges of different colors incident on the same nodes. These observations indicate that the spatial distribution of application classes can provide useful information for inferring the edge labels. However, to utilize such information is a non-trivial task. In the next section, we formally define the edge label inference problem and present our solution in Section IV.

III. Inferring TAG Edge Label

In this section we provide a mathematical formulation of the application inference problem, given an augmented (and initially *unlabeled*) TAG, with a set of traffic statistics attributes associated with each edge.

A. Mathematical Formulation

Let $\mathcal{G} := \{\mathcal{H}, \mathcal{E}\}$ denote a particular TAG constructed over a specific time period T, where $\mathcal{H}$ denote the set of all hosts in the network and each edge $e_{ij} \in \mathcal{E}$ represents the aggregation of all traffic between the endpoints h_i and h_j. In our edge label inference problem, we assume that each edge $e_{ij} \in \mathcal{E}$ belongs to one of K pre-defined application classes, C_k, $1 \leq k \leq K$ (with $K = 12$). However, what class e_{ij} belongs to is *unknown and to be determined*. Let $L : \mathcal{E} \rightarrow \{C_k, 1 \leq k \leq K\}$ denote the edge class mapping, $L(e_{ij}) = C_k$ for some k. Our problem is to infer this edge class mapping L, given the unlabeled $\mathcal{G}$ and the collection of the edge attribute sets, $\{\mathbf{x}_{ij} : e_{ij} \in \mathcal{E}\}$. To solve this problem, we assume a *supervised* machine learning environment, where we are given a training dataset, i.e., a labeled $\mathcal{G}$ (constructed from the traffic within a certain time period) where the class of each edge is given. The inference problem becomes the following learning problem: can one learn a function f which returns an estimate of the edge class mapping each edge e_{ij} (Eq. 1)?

$$\tilde{L}(e_{ij}) = f\left(\mathbf{x}_{ij}, L(e_{i\cdot}), L(e_{\cdot j})\right), \tag{1}$$

where the *traffic features* $\mathbf{x}_{ij}$ contain the traffic statistics for edge e_{ij}; $e_{i\cdot}, e_{\cdot j} \subset \mathcal{E}$ represent the edges incident on h_i and those incident on h_j, respectively; the *neighborhood features* $L(e_{i\cdot})$ and $L(e_{\cdot j})$ are obtained through aggregation of the corresponding edge classes.

Eq.(1) indicates that the edge label inference problem depends not only on the traffic statistics on each edge e_{ij}, but also on the collective distribution of all traffic exchanged with the two endpoints h_i and h_j, as reflected by the edge labels within a neighborhood of these two endpoints on $\mathcal{G}$. Without the knowledge of $\mathcal{G}$, the problem reduces to a classic *multi-class classification* problem, where one learns f that returns an estimate of the edge label based purely on the traffic statistics attributes, i.e., $\tilde{L}(e_{ij}) = f(x_{ij})$. The main question we are interested in exploring in this paper is whether–and *how*–the spatial disposition of traffic embodied by the TAG can be exploited in inferring the dominant application classes between two endpoints. Before we introduce the proposed two-step method in Section IV, we first discuss the related work.

B. Related Work

The problem of network traffic classification has been widely studied. Most solutions fall into one of the two general approaches (or a combination thereof): the signature-based

approach using deep packet inspection, or statistical/machine-learning based approach that utilizes only traffic statistics derived from packet/flow-level header fields and timing information [5]–[7], [12]. All these solutions focus almost exclusively on classification of packet- or flow-level traffic. At the other end of the spectrum, several recent studies [11], [13]–[15] have examined the problems of endpoint traffic characterization from on network traffic data as well as other information. Of particular interest is the multi-level traffic classification scheme (called BLINC) proposed in [11]. Nonetheless, the goal of BLINC is primarily on characterizing the traffic of endpoints. To the best of our knowledge, our work is the first to advocate the analysis of "edge relations" between endpoints, and show it can be applied to the traffic classification problem.

In terms of analyzing spatial patterns from network traffic data, [16] uses host-level "communities-of-interest" (COIs) as reference profiles for detecting propagation of malware. The notion of traffic activity graphs (TAGs) come from the two recent studies [8], [17] which investigate the properties of various *application-specific* TAGs. In particular, [8] proposes a novel nonnegative matrix tri-factorization (tNMF) method for decomposing spatial interaction patterns, and illustrates how these interaction patterns discerned from application-specific TAGs can be interpreted. Inspired by the findings in [8], in this paper we study *generic* TAGs with mixed application classes, introduce the notion of *colored* TAGs, and formulate the TAG edge label inference problem and solve it effectively with the proposed two-step model.

In the machine learning domain, the TAG edge label inference problem has been recently studied under the term *collective classification*, with techniques ranging from iterative classification to Gibbs sampling (see [18] for an overview). In particular, iterative classification consists of applying Eq.1 several times, using the previously computed $\tilde{L}(e_{ij})$ as an approximate input. Compared to iterative classification, our proposed two-step method is significantly simpler and works very well for our problem, as we shall see in Section V.

IV. Methodology

In this section, we propose a novel two-step approach which employs the state-of-the-art machine learning algorithms for solving the edge label inference problem using both traffic attributes and neighborhood information in TAGs[2].

A. Two-step Model

Recall that for the edge label inference problem, our purpose is to find the edge label mapping: $\tilde{L}(e_{ij}) = f(\mathbf{x}_{ij}, L(e_{i\cdot}), L(e_{\cdot j}))$, where $\mathbf{x}_{ij}$ is the vector of traffic attributes associated with edge e_{ij}, and $L(e_{i\cdot})$ and $L(e_{\cdot j})$ are the labels for the edges connected to both endpoints h_i and h_j of edge e_{ij}, respectively.

Direct learning of $\tilde{L}(e_{ij})$ is difficult since it requires the knowledge of all the labels of the neighborhood edges, which are obtainable only when the model $\tilde{L}(e_{ij})$ is known. In this paper, we propose a streamlined variant of the classical collective classification algorithm [18], which we call the two-step model. Although the traditional collective classification algorithms are known to suffer from serious instability and overfitting issues, in the application scenarios of this paper, we found that our two-step model offers excellent results. The schematic view of the training phase and the testing phase for the proposed model is depicted in Fig. 2.

The proposed model consists of two components. The first step, which we refer to as *bootstrapping*, treats structural properties of the TAG (the neighborhood information of edges) as unknown and infers edge labels according to only the traffic attributes $\mathbf{x}_{ij}$ associated with each edge. The initial classification from the bootstrapping step is in Eq. 2.

$$\tilde{L}_0(e_{ij}) := f_0(\mathbf{x}_{ij}) \tag{2}$$

Bootstrapping provides us with the initial labels for all edges, though the accuracy of these labels depend on the available traffic information in different application scenarios and hence can be inaccurate in certain situations. For example, as we shall see in Section V, in the application of traffic classification at the network layer where most traffic attributes (e.g., port numbers, protocol number, etc.) are absent, the accuracy after bootstrapping can only reach around 80%. In a study where all flow level attributes are accessible, the bootstrapping step can achieve an accuracy of over 96% [9].

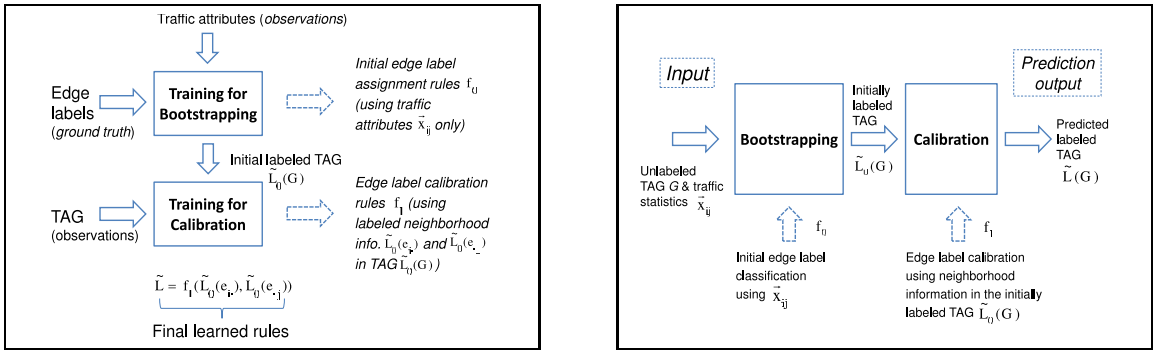
The second step, referred to as *graph-based calibration* or *calibration* in short, incorporates the inherent neighborhood and local properties of the edges in the TAG to re-enforce or re-label the initial edge labeling provided by the bootstrapping step. For example, given an edge labeled as `Business` and all the neighborhood edges labeled as `Web`, the calibration step may follow the edge clustering rule and change the edge label into `Web`. The calibration process is expressed in Eq. 3:

$$\tilde{L}(e_{ij}) := f_1\left(\tilde{L}_0(e_{i\cdot}), \tilde{L}_0(e_{\cdot j})\right) \tag{3}$$

Why do we deprive classifier f_1 from traffic features $\mathbf{x}_{ij}$? The explanation is that we want the classifier to focus on the neighborhood features, which, for a given endpoint, only change slowly over time. This means that if we use the neighborhood features only, test data that has been collected from the same graph as the training data (but later in time) may still have a distribution that is close to the training data. On the other hand, the traffic features suffer from a much greater time variability, and can become undesirable noise when one has access to neighborhood features. In a preliminary study, we always obtain a higher error rate by incorporating traffic features in the calibration step.

In addition, we found that a single calibration step was enough to obtain the best performance, hence the simplification of the algorithm into a two-step approach. Therefore, from Eq. 3, the edge classification from the proposed model is expressed as a combination of the bootstrapping step and the calibration step. We note that the inference on the class

[2]We note that the model defined in this section is corresponding to the situation where only one label is associated with each edge. However, the proposed model can be easily extended for inference on multi-labeled edges.



(a) Training phase (b) Testing/operating phase

Fig. 2. Architectures for training and operating of the proposed two-step model.

of a particular edge e_{ij} is based on the initial (inaccurate) classification of the neighborhood edges $(\tilde{L}_0(e_{i\cdot}), \tilde{L}_0(e_{\cdot j}))$ from the bootstrapping step. Therefore, the training of the calibration function f_1 also depends on the initial classification provided by the function f_0 in the bootstrapping step, instead of depending on the ground truth. In the following, we discuss the training and operating of the proposed two-step model.

B. Training and Operating the Two-step Model

Taking advantage of the ground truth that we have for the network flow data, we formulate both the bootstrapping step and the calibration step as classical multi-class classification problems. Hence, the bootstrapping function f_0 and the calibration function f_1 correspond to two multi-class classifiers, and we apply the state-of-the-art machine learning techniques to learn these two classifiers and hence to solve the edge label inference problem.

Overall Training and Operating Architecture. The training architecture is presented in Fig. 2[a]. Given the ground truth of edge labels in the training data, we first learn a multi-class classifier f_0, which maps traffic features $\mathbf{x}_{ij}$ corresponding to each edge e_{ij} to the initial labeling $\tilde{L}_0(e_{ij})$. We then generate initial labeling for the entire TAG, $\tilde{L}_0(\mathcal{G})$ and learn the classifier f_1 for the calibration step, which maps initial labeling to the true labeling based on the labels of the neighbors of individual edges.

After learning two classifiers f_0 and f_1, at the operating time (Fig. 2[b]), given a TAG $\mathcal{G}$ created from the test dataset, we first apply f_0 to obtain the initial labeling for all the edges in the TAG, namely, $\tilde{L}_0(\mathcal{G})$. We then encode the neighborhood information of all the edges into histograms and apply the classifier f_1 for the calibration purpose, which will produce the final prediction $\tilde{L}(\mathcal{G})$ after calibration.

We note that these two classifiers (f_0 and f_1) only differ in the feature sets. f_0 uses traffic features associated with individual edges. The available traffic features depend on specific applications scenarios. We next explain how to encode the neighborhood information as features for constructing f_1.

Encoding Neighborhood Information for Graph-based Calibration. Given the fact that an edge may have an unbounded number of neighborhood edges connected to the end nodes, we encode the neighborhood information as histograms. More specifically, for an edge e_{ij}, let $|C_k|$ denote the number of edges connected to h_i which are labeled as C_k, $1 \leq k \leq K$. We then define K features corresponding to the neighborhood edges connected to the endpoint h_i as $|C_k| / \sum_j |C_j|$, representing the percentage of edges connected to h_i that are labeled as C_k. Similarly, we define K features to encode the neighborhood edges connected to h_j. In addition, we include the degrees of h_i and h_j as two additional features. Therefore, for $K = 12$ (the number of predefined application classes in Table I), we create a total of 26 features to encode the neighborhood information of individual edges. Despite the loss of structural information, encoding objects as histograms has enabled a fast deployment of machine learning solutions to many real world problems, with surprisingly good results.

C. Implementing the Two-step Model

In this section, we discuss the details of implementation and training of the two classifiers, f_0 and f_1. We use f_0 as an example for illustration, since f_1 is implemented in the same way and only differs in the selected feature set.

Due to the huge amount of data during both training and testing in the application of network traffic classification, compared to a direct training of a K-class classifier, the decoupled approach, i.e., training K binary classifiers and then assembling them for the K-class classification, has shown to be superior in both accuracy and scalability [9]. In this paper, we adopt the same decoupled approach for building f_0. In particular, we train K binary classifiers, corresponding to K posterior probabilities, $P(C_k|\mathbf{x}_{ij})$, where $1 \leq k \leq K$. Given such a model, we then compare the K posterior probabilities, and assign the example to the class (label) $f_0(e_{ij}) = \text{argmax}_{C_k} P(C_k|\mathbf{x}_{ij})$. In the ideal case, this assignment exactly corresponds to the Bayes optimum for the multi-class classification problem [19].

We implement the K ($K = 12$) binary classifiers using the *Adaboost* [20] algorithm, which applies a greedy incremental approach that can be restricted to learn a limited number of features (with implicit L_1 regularization). The output of *Adaboost* classifiers are further remapped to approximate $P(C_k|\mathbf{x}_{ij})$, using univariate logistic regression [9]. To strike a balance between accuracy and scalability, we choose the decision stump (the simplest decision tree, having one level)

as the weak learner. In the previous studies, this classifier has proved to attain the best scalability while having an accuracy comparable to sophisticated non-linear classifiers [9].

V. Experimental Results

In this section, we evaluate the proposed two-step model for classifying traffic at the network layer. All the metrics for evaluation are based on counting edges. For example, the accuracy is defined as the number of correctly classified edges divided by the total number of edges.

TABLE III
EDGE TRAFFIC FEATURES DERIVED FROM BASIC FLOW FEATURES.

Name	Type	Name	Type
min_duration	numeric	max_duration	numeric
min_pkt_size	numeric	max_pkt_size	numeric
min_pkt_rate	numeric (*)	max_pkt_rate	numeric
symmetry	numeric		

Bootstrapping. As motivated in Section I, our evaluation focuses on the worst-case scenario where both the network-level header and transport layer header are absent (e.g., the IPSec traffic where the entire IP packet is encrypted). To simulate this application scenario with existing flow data, this evaluation uses only basic features from the flow records, namely, IP addresses (which are attributed as internal or external), flow packets, bytes and duration. In particular, there are no fields that related to the transport layer, such as protocol, port numbers or ToS bytes. However, other flow features are either unchanged, like *packets*, or can be derived by adding a constant length of the outer header, like *bytes*.

Prior to bootstrapping, we aggregate the traffic flows associated with each direction in an edge between two endpoints into a set of edge level traffic features, listed in Table III. In order to simulate the effect of flow records formed when UDP/TCP ports are not available for the flow key, we first temporally aggregate flow records of the same source and destination IP address, using a 30 second inactive timeout. This done, we then compute the traffic duration (i.e. sum of flow duration), the average packet size, the average packet inter-arrival rate for the bi-directional traffic on each edge separately. The *min_duration* and *max_duration* represent the minimum and the maximum traffic durations of the bi-directional traffic. Similarly, the *min_pkt_size* and *max_pkt_size* are the minimum and the maximum average packet sizes; the *min_pkt_rate* and *max_pkt_rate* are the minimum and the maximum average packet inter-arrival rates. We define the *symmetry* of an edge as the minimum number of bytes divided by the maximum number of bytes of the bidirectional traffic.

Overall Accuracy. Our first evaluation focuses on the overall classification accuracy. We use a one-hour data set (from 05/03/2008 10-11AM) to train both the bootstrapping step and the calibration step. We then evaluate the accuracy of the two-step model at different times of the day, using three one-hour data sets from 05/05/2008 (2-3AM, 11-12AM and 6-7PM). For a larger scale evaluation, we also use one *whole-day* training data from (05/03/2008) and use the entire day data from (05/05/2008) for testing.

Fig. 3 displays the overall accuracy for the four experiments mentioned above. We observe that the classification accuracy using one-hour time window varies between 79.5% and 83.3% after bootstrapping possibly due to the fluctuation of traffic mix due to the time-of-the-day effect. However, in all cases, the accuracy increases substantially to around 90% after the graph-based calibration. When the length of the time window extends to one day, we have access to both more edges, and more accurate traffic statistics (especially for smaller traffic classes). In this case, the accuracy for the bootstrapping step increase to 84%, and, again, the calibration step improves the accuracy to 91.3%. This means that by applying the graph information, we can reduce close to 50% of all the classification errors made by the bootstrapping step!

Per-Class F1 Score. We next zero-in on individual traffic classes to evaluate how the calibration step improves the per-class classification accuracy. Due to the highly unbalanced traffic class distribution, the error rate for smaller traffic classes are generally small and hence hard to compare. We instead use the F1 score which is defined as the harmonic mean of precision and recall. The F1 score ranges between 0 and 1, and a higher F1 score indicates a better classification result.

Fig. 4 displays the F1 scores for different traffic classes on the whole-day data set. The traffic classes are ordered decreasingly by the F1 scores before calibration. Obviously, the F1 scores are increased for all the traffic classes after calibration. This indicates that the enhancement of the overall accuracy is not an artifact of the accuracy improvement of a few large traffic classes, like `Web` and `FileSharing`, instead, it is the result of a universal accuracy increase for all traffic classes. Even when the F1 scores are low or close to zero for certain classes before calibration, such as `Chat` and `FTP`, the calibration step can still significantly improve the per-class F1 scores. This demonstrates the effectiveness of graph-based calibration which infers the colors of such edges solely based on the structural properties in TAGs.

Temporal and Spatial Stability. In practice, temporal and spatial stability is an important requirement for a traffic classification system. Temporal stability means the system can be trained once and run without human intervention for a long time period. Spatial stability means the system can be trained at one site and run at a second site without re-training.

We evaluate the stability of the proposed method using data sets collected at two geographically separated sites for one year. The training data is from 05/03/2008 10-11AM and four one-hour datasets for the temporal stability test are used, which are 2 days, 1 week, 1 month and 1 year later from the time when the training data is collected, respectively. In addition, we use a one-hour dataset collected at the second site where there is also a one-month gap between the training data and the test data. We note that all the test datasets are from 6-7PM of the corresponding day.

We first observe that the bootstrapping step is very stable across time and space. Within one month time period at the same site, the calibration process consistently reduces the error rate by 50% and boosts the accuracy from around 80% to

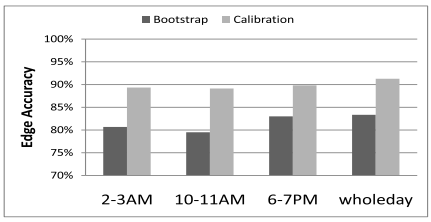


Fig. 3. DNS failure graph properties

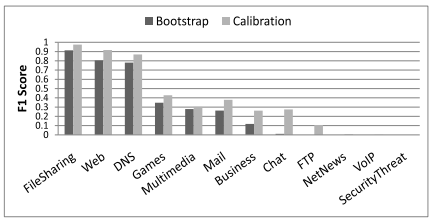


Fig. 4. F1 scores for different traffic classes

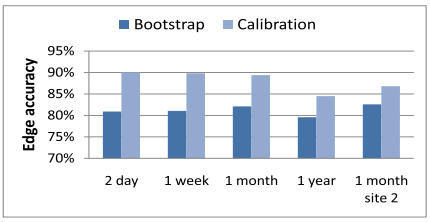


Fig. 5. Stability of the calibration results

around 90%. The calibration step still reduces significantly the error rate (by around 30%) after one year time period or at the second site.

Real-time Classification. So far, our evaluation is in an offline manner. However, the proposed method can also be implemented as a real-time system. The basic idea is to maintain a historical TAG for the calibration step. More precisely, let T be the time window length to construct the TAG for the calibration step. To classify an edge e_{ij} in real-time at t_0, we need to maintain the traffic statistics and neighborhood edges of e_{ij} in a past time window from $t_0 - T$ to t_0.

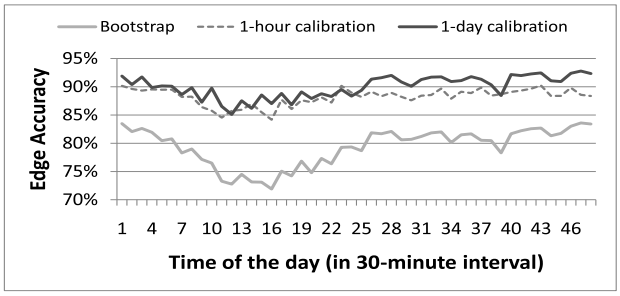


Fig. 6. Real-time classification results

Fig. 6 shows the real-time classification results, where the x-axis represents the index of 30-minute time intervals of the day (05/05/2008) and the y-axis is the edge accuracy for the corresponding 30-minute time interval. We use a one-hour dataset from (05/03/2008 10-11AM) to train the system and apply $T = 1$ hour and $T = 1$ day to illustrate the impact of the history length on the calibration performance.

From Fig. 6, the calibration performance is persistent and reduces the error rate by at least 50% throughout the day. A low accuracy after bootstrapping usually leads to a low accuracy after calibration, however, the fluctuation of accuracy after calibration is not as significant as the one after bootstrapping. This indicates the calibration also helps the whole system to achieve a more persistent classification accuracy. In addition, a longer history for calibration increases the overall accuracy by 3% in general.

VI. Conclusions

In this paper, we proposed a two-step supervised model which utilizes collective traffic statistics in the traffic activity graphs (TAGs) for solving the application inference problem at the network layer, where the available traffic features are limited. The bootstrapping step provides initial labels (traffic class assignments) of the edges purely based on available traffic statistics and the graph-based calibration step utilizes inherent neighborhood and local properties of edges in the TAG to re-label or re-enforce the initial labels to achieve much better accuracy. Using flow records from a large ISP network, our evaluation results showed that the calibration step consistently reduced the error rate from the bootstrapping step by 50% and improved the accuracy for all traffic classes.

References

[1] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational ip networks: Methodology and experience. *IEEE/ACM Trans. Netw.*, 9(3):265–280, 2001.

[2] Port numbers, http://www.iana.org/assignments/port-numbers.

[3] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). RFC 2784, 2000.

[4] S. Kent and K. Seo. Security architecture for the internet protocol. *RFC 4310*, December 2005.

[5] A. W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In *Proc. of ACM SIGMETRICS'05*, 2005.

[6] H. Jiang, A. W. Moore, Z. Ge, S. Jin, and J. Wang. Lightweight application classification for network management. In *Proc. of INM '07*, 2007.

[7] J. Erman, A. Mahanti, M. F. Arlitt, I. Cohen, and C. L. Williamson. Offline/Realtime traffic classification using semi-supervised learning. *Perform. Eval.*, 64(9–12):1194–1213, 2007.

[8] Y. Jin, E. Sharafuddin, and Z-L. Zhang. Unveiling core network-wide communication patterns through application traffic activity graph decomposition. In *Proc. of SIGMETRICS '09*, pages 49–60, 2009.

[9] Y. Jin, N. Duffield, J. Erman, P. Haffner, S. Sen, and Z.-L. Zhang. A modular machine learning system for flow-level traffic classification in large networks. Technical report, 2009.

[10] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. ACAS: Automated construction of application signatures. In *Proc. of MineNet*, 2005.

[11] T. Karagiannis, K. Papagiannaki and M. Faloutsos. BLINC: Multilevel traffic classification in the dark. In *Proc. of ACM SIGCOMM*, 2005.

[12] T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4), 2008.

[13] K. Xu, Z.-L. Zhang and S. Bhattacharyya. Profiling Internet backbone traffic: behavior models and applications. In *Proc. of ACM SIGCOMM*, August 2005.

[14] I. Trestian, S. Ranjan, A. Kuzmanovi, and A. Nucci. Unconstrained endpoint profiling (Googling the Internet). In *Proc. of ACM SIGCOMM '08*, Seattle, USA, 2008.

[15] M. Iliofotou, M. Faloutsos, and M. Mitzenmacher. Exploiting dynamicity in graph-based traffic analysis: techniques and applications. In *Proc. of CoNext'09*, 2009.

[16] P. McDaniel, S. Sen, O. Spatscheck, J. Van der Merwe, B. Aiello, and C. Kalmanek. Enterprise security: a community of interest based approach. In *Proc. NDSS*, 2006.

[17] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *Proc. of ACM IMC*, 2007.

[18] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 2008.

[19] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

[20] R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Mach. Learn.*, 39(2-3):135–168, 2000.