Profiling-by-Association: A Resilient Traffic Profiling Solution for the Internet Backbone

Marios Iliofotou
UC Riverside

Brian Gallagher
Lawrence Livermore Lab

Tina Eliassi-Rad
Rutgers University

Guowu Xie
UC Riverside

Michalis Faloutsos
UC Riverside

ABSTRACT

Profiling Internet backbone traffic is becoming an increasingly hard problem since users and applications are avoiding detection using traffic obfuscation and encryption. The key question addressed here is: *Is it possible to profile traffic at the backbone without relying on its packet and flow level information, which can be obfuscated?* We propose a novel approach, called Profiling-By-Association (PBA), that uses only the IP-to-IP communication graph and information about some applications used by few IP-hosts (a.k.a. seeds). The key insight is that IP-hosts tend to communicate more frequently with hosts involved in the same application forming communities (or clusters). Profiling few members within a cluster can "give away" the whole community. Following our approach, we develop different algorithms to profile Internet traffic and evaluate them on real-traces from four large backbone networks. We show that PBA's accuracy is on average around 90% with knowledge of only 1% of all the hosts in a given data set and its runtime is on the order of minutes (≈ 5).

1. INTRODUCTION

Traffic profiling is essential in provisioning, managing and securing a network. It allows network administrators to evaluate what-if questions, such as: *"Is P2P traffic increasing? Is it worth the money to install an expensive P2P classifier to reduce traffic in my network?"*, *"Half of my traffic is encrypted; is that traffic regular web traffic or something else (and potentially malicious)?"*, *"Will it pay-off to add web-caches to reduce incoming traffic to my providers?"* In today's networks, an effective traffic-profiler needs to prevail over the following challenges. **C1:** Overcome application obfuscation (e.g., operate despite encryption). **C2:** Operate at the backbone where only partial view of traffic is typically observed.

ACM CoNEXT 2010, November 30 – December 3, Philadelphia, US.

Our goal is to provide a profiling solution that overcomes both aforementioned challenges. With respect to C1, obfuscation is particularly challenging, since it may appear at multiple levels: (i) port level, with randomization or encryption (i.e., IPsec), (ii) payload level, with encryption, (iii) flow level, with timing variations and packet padding, and (iv) connectivity level, with the randomization of connection patterns. We use the term **multi-level obfuscation** to describe this problem, since obfuscation may appear at multiple levels. Previous methods are not **robust** to multi-level obfuscation. Simply put, when a large portion of traffic is obfuscated, many obfuscated flows will end up unlabeled or mislabeled by most current methods. With respect to C2, past research has shown that traffic profiling approaches which perform well on enterprise traffic often struggle at the backbone [13, 7, 19]. A promising method uses Google to profile IPs (a.k.a. UEP) [19] in an effort to overcome C1 and C2. However, as reported by its authors, this method leaves large portions of P2P traffic unclassified ($\approx$40%) and requires long execution time. Since P2P traffic is the main traffic class that uses multi-level obfuscation, the problem still remains open. We provide an extensive literature survey and a discussion on how our work differs from it in §3.

The motivating insight for this work lies in the following two key observations. **O1:** Several applications tend to form **communities** (or clusters). In this work, we use the term community as defined by *modularity* [5], to be a set of nodes with higher intra-community edge-density than a random graph. **O2:** For a few hosts it is easy to identify what applications they use; we will refer to these hosts as the initial **seeds**. With respect to O1, we are the first to observe that application flows in the backbone divide into tightly-knit communities of IP-hosts, as shown in the *connectivity graph* of Figure 1. In the rest of this paper, we use the term connectivity to refer to this IP-to-IP graph of interactions: nodes are IP addresses and edges correspond to flows between the two nodes. For O2, from our analysis of four large backbone networks, we observe that some application (e.g., Bittorrent) users choose to encrypt their traffic, whereas some others do not enable encryption. Similarly, few hosts still use the default ports for some applications

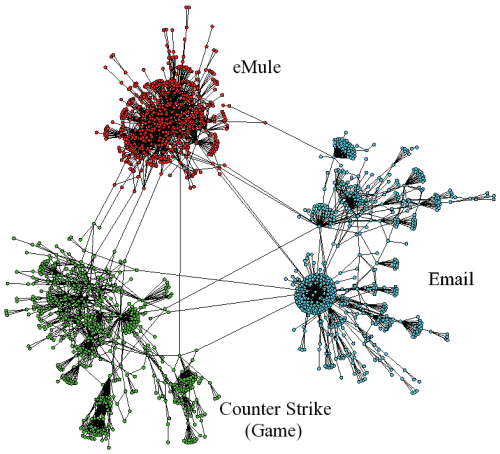

Figure 1: Applications form communities that are visible from the Internet backbone. In the graph visualization, nodes are IP addresses and edges represent flows between IPs. In this work, we take advantage of these communities in order to profile all the flows (edges in the graph) in a trace. For visualization purposes, only traffic from three applications is visible. To highlight the three communities we color the nodes based on their dominant application (better viewed in a color print-out or a monitor screen). The data is from a peering link of a large ISP in the US.

(e.g., 6881 for Bittorrent). Therefore, using a payload- or port-level classifier, we may still classify a small portion of the traffic. Moreover, other profilers that do not use ports or payload, such as BLINC [12] and UEP [19], can also successfully profile small portions of backbone traffic.

This work introduces the **Profiling-By-Associations (PBA)** framework to address the multi-level obfuscation problem. Our framework exploits the above observations (O1 and O2) regarding backbone traffic. In a nutshell, PBA takes as input an IP-to-IP connectivity graph and information about a small subset of IP-hosts and produces a prediction about the class of all the flows (edges) in the graph. Because network hosts use multiple applications, all our methods profile flows and not hosts. In this paper, we develop four novel algorithms based on the PBA framework. Our best PBA-based algorithm, called HYP, first uses a community discovery algorithm to identify different application clusters (O1), as observed in Figure 1. Then, it uses information about few members from each community (O2) to profile an entire cluster. With this two-step approach, we introduce a profiling (guilt) by association solution to the traffic profiling problem. In other words, with PBA, we stop looking at "how the flows of an IP look like" (e.g. number of packets, or port number used), but instead we focus on "which other IPs it interacts with." Our evaluation on four large backbone networks shows that our algorithms achieve: (a) high classification accuracy, and (b) high robustness to different levels of obfuscation.

The highlights of our work can be summarized in the following:

- We introduce four algorithms based on the PBA framework and highlight one, the HYP algorithm, with the highest robustness to obfuscation.
- Using only 1% of hosts as seeds, the HYP algorithm labels all flows in our trace without using payload, port, or flow statistic information and achieves above 90% accuracy as compared to a payload-based classifier (for details see §5.2.2).
- HYP can achieve above 80% accuracy with only 10% initial seed, even when 40% of the seeds are erroneous (for details see §5.2.4).
- The HYP algorithm is robust to connectivity-level obfuscation, where hosts open fake connections to other nodes in order to avoid detection. Even if hosts open 20 times more connections, HYP performance is reduced by only 9% (for details see §5.2.6).
- Our algorithms can work together with existing solutions (e.g., CoralReef [3] or BLINC [12]) and improve their profiling accuracy by up to 50% (for details see §5.2.5).

The rest of this paper is structured as follows. In §2 we define the problem. We describe the limitations of current methods in addressing the multi-level obfuscation problem in §3. In §4 we present the details of our algorithms. We evaluate our algorithms using four real-world backbone traces in §5. We provide a discussion of various practical and subtle issues of our approach in §6. Finally, we conclude the paper in §7.

2. PROBLEM DEFINITION

The goal of a traffic profiler is to assign application labels (e.g., Web, DNS) to all the flows in a packet trace. We define the *multi-level obfuscation* traffic profiling problem as follows. The inputs are:

- A set of hosts H.
- A set of flows F, where each flow $f \in F$ is a pair (a, b) s.t. $a, b \in H$. Flow f represents a network communication between hosts a and b.
- A set of flows $F_L \subset F$ or a set of hosts $H_L \subset H$, for whom an application class (e.g., "DNS" or "Web") is provided [1]. We interpret an application class c for a host h as evidence that h has either an incoming or outgoing flow of type c.

Given the above, the goal is to produce a function $pba(f)$ such that $\forall f \in F : pba(f) = tc(f)$, where $tc(f)$ returns the true application class of flow f.

Robustness to four levels of obfuscation. In this work, we considers four levels of obfuscation. The port-level (L1), payload-level (L2), flow-level (L3), and then connectivity-level (L4). Because traffic can be obfuscated, the $pba(f)$ function should not rely on any port-, payload-, or flow-level information. It should also be robust to hosts that try to evade detection by

[1] In this work, we consider H_L seeds.

Method	Robustness Port Obfuscation (Level 1)	Robustness Payload Encryption (Level 2)	Robustness Flow Stats Obfuscation (Level 3)	Robustness Conn. Obfuscation (Level 4)	High Accuracy at Backbone	High Accuracy on P2P	Low Training & Tuning Effort	Fast Execution Time
Coral Reef [3]	No	-	-	-	-	-	-	-
Payload [16]	-	No	-	-	-	-	-	-
Flow-based [17] (Mach. Learn.)	-	-	No	-	-	-	-	No
Our NLC [8]	-	-	-	No	-	-	-	No
BLINC [12]	-	-	-	-	No	-	No	-
Googling [19]	-	-	-	-	-	No	No	No
Our HYP⋆	-	-	-	-	-	-	-	-

Table 1: Comparison of key traffic profiling methods with our HYP algorithm. The table highlight the pros and cons of each method and their robustness to various levels of obfuscation. HYP is the only profiler that covers all requirements in the table. ⋆The HYP method is a contribution of this paper.

opening random connections in order to confuse the profilers (L4). For example, in Figure 1 we see that eMule hosts have more flows (edges) between them than with other applications. Our algorithms should be robust to eMule users that try to evade by opening connections to Email and Game servers in order to interfere with the formation of communities. More details on such experiments are found in §5.2.6.

In this work, we are interested in profiling traffic at the backbone, which is a more challenging problem. Moreover, other methods such as BLINC [12], are known to perform good at the edge but poor at the backbone. As with the vast majority of traffic profiling solution, results do not have to be generated in real-time. Typically, a network administrator will first collect packets for some time (e.g., five minutes) and then analyze the data off-line. As we show next, for analyzing five minutes of traffic our fastest algorithm takes less than five minutes to give predictions.

Q: How is this problem different from the "classic" traffic classification problem? Most previous profiling methods aim to learn the descriptive features (e.g., payload signatures) or behaviors (e.g., number of open TCP connections) of different traffic classes from a training set and use that knowledge to classify future traffic. For example, a classic profiler might observe (during training) that all App-X flows use port number 6881 and start with three packets less than 200 bytes. If no other traffic classes have these features, the classifier will use this description to detect future App-X traffic. This process assumes that the future traffic will look like the training traffic. However, in our problem, the future traffic might be obfuscated at multiple levels (e.g., using random port and random packet sizes) and thus look very different than the training traffic.

Profiling granularity: We want to identify applications at high granularity. We use the following set of applications: (i) Web, (ii) P2P, (iii) DNS, (iv) Email, (v) Games, and (vi) Chat.[2] The most important class is P2P traffic which is the hardest to detect [13, 19, 12]. Therefore, a method with high P2P accuracy is highly desirable in practice [13].

[2]The number of classes can vary depending on the goal of a measurement study, as we review in §3.

3. RELATED WORK

Due to its importance, the traffic profiling problem has attracted significant attention in the past. However, the problem is far from being solved. Table 1 summarizes the current key traffic profiling solutions, highlights their main limitations, and compares them to our PBA-based algorithms (NLC and HYP). We also note that other work uses graphs to analyze traffic [11], but they do not target traffic classification.

We present three factors that affect current traffic profiling methods at the backbone:

1) The obfuscation factor: In Table 1, the CoralReef [3] profiler relies only on well-known port numbers (e.g., port 53 denotes DNS). It performs well with legacy applications, but not with applications that use random or dynamically assign ports, such as online games and P2Ps [13]. Similarly, all methods that use payload and deep packet inspection (DPI) [16, 18, 15] will fail to classify encrypted traffic. Some profiling methods can detect traffic with encryption by relying on flow-level features and statistics (e.g., packet sizing information, packet inter-arrival times) and advanced Machine Learning methods [17, 1, 6, 10]. However, flow-features can be obfuscated through packet padding and randomization of inter-packet gap timing. From our study, we observed that popular applications, such as the uTorrent client for Bittorrent, currently enables obfuscation at the port (with randomization), payload (with encryption), and flow level (with packet padding). Connectivity based approaches like the NLC algorithm (see §4.2 and [8]) is sensitive to **homophily**[3], which makes it less robust to connectivity obfuscation (as we show next in our evaluation §5.2.6).

2) The location of observation factor: Profiling backbone traffic provides additional challenges compared to enterprise and edge networks [13, 7]. Host-based behavioral approaches, such as BLINC [12] and [21], profile end-hosts using their connections patters, e.g., how many distinct ports a host uses and how many distinct IP it communicates with. This approach makes

[3]Homophily in trace graphs refers to the tendency of neighboring edges having similar applications. It is computed as the probability of two neighboring edges having the same application. Two edges are considered neighbors if they share a common node. See [8] for details.

profiling robust to various levels of obfuscation, as we show in Table 1. At the backbone, however, we only observe a small fraction of the connections from each host (i.e., some of its flows do not pass by the monitoring link; for example, because of multi-homing). This is why BLINC [12] shows better performance at the edge, where all the flows of a host are observed, than at the backbone [13].

3) Availability of information factor: The most recent paper on host-profiling (Googling in Table 1) [19] uses readily available information from the Web in order to classify end-hosts based on results returned from the Google search engine. As reported in [19], this approach does not perform well for applications that have dynamic behavior, such as P2P protocols. In addition, this method can not work with anonymized traffic (randomized IP addresses), and for traffic that was collected in the past (since the information about the IPs might not still be available online). Having said that, we find that this last method is a very promising seeding process for our needs, especially if the code becomes publicly available in the future.

Tuning effort and execution time: In Table 1, training and tuning effort highlights practical challenges. The BLINC [12] profiler has been reported by others [13] to be very hard to fine-tune, because it requires manual adjustment of 28 parameters for each backbone link. Also, the Googling approach [19], requires careful parsing of Web pages in order to derive the keywords used for inference, which makes re-writing the tool from scratch hard. Regarding execution time, all flow-based machine learning algorithms with high classification accuracy are reported to require hours of training and inference [13]. Similarly, NLC also takes tens of minutes to produce results for some of our traces (§5). The Googling approach requires issuing queries to a search engine for each single IP in a trace, which can take from several minutes up to hours [19]. As we show in §5, our PBA-based HYP algorithm takes less than five minutes to run in all our traces. More on the deployment of our algorithms in practice is discussed in §6.

4. PROFILING-BY-ASSOCIATION (PBA)

PBA provides a general framework for profiling network traffic based on a two-step process: (a) *seeding* and (b) *inference*. The basic steps of this processes are shown in Figure 2. In a nutshell, with PBA we first extract connectivity information from a packet trace and use an external means (e.g., a payload classifier) to derive initial seeds. Next, we use a graph-based algorithm and profiling-by-association to classify all the flows in the packet trace.

4.1 Step A: Seeding

At this step, flows and/or hosts are labeled by some external means according to their known or surmised application class (e.g., "DNS" or "Web"). At the level of flows, it means that the flow is believed to have been produced by the specified application. At the host level, it means that the host is believed to have some amount of traffic of the specified type. We refer to these initial labels as *seeds*. Seeds may be obtained by: (a) information from a system administrator (e.g., using known IP addresses of server), (b) running port or payload based traffic classifiers on non-obfuscated portions of a trace, or (c) using techniques such as "Googling the Internet" [19]. More on seeding sources is discussed in §6. We believe using seeds on host to be more realistic since several host-profiling solution (see §3), report information at the host-level. Moreover, it is more likely for a host to be persistent over time (e.g., a server) than any of its flows.

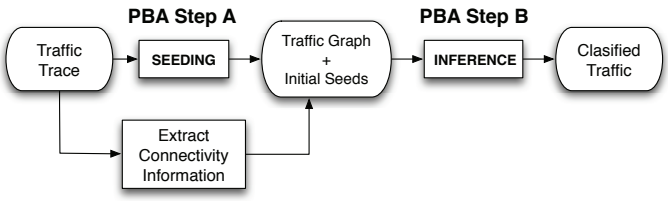

Figure 2: Overview of PBA framework for traffic profiling.

Our PBA based algorithms are indifferent to the source of seed information; and as we show in §5, are robust to both random errors and systematic bias in seed labels. We emphasize that in order for PBA to be effective, seeds need not be 100% correct nor widely available (e.g., 1% of hosts or flows is generally sufficient). Seeding is not unique to PBA; all machine-learning based methods require seeds (in form of training sets). However, PBA has a big advantage over traditional machine-learning based methods in that PBA's performance is not dependent on the training and testing sets having the same feature characteristics (e.g. packet sizes).

W.r.t. seeding, in our experiments (§5) we evaluate the following two cases. Recall that the seed we use is the information that a given IP-host is "known" to participate in a particular application.

(a) The base case: Agnostic to the real traffic mix. Unless stated otherwise, in all our experiments, we randomly select an equal number of initial hosts for each application class. In other words, we do not assume that our seeds accurately represent the traffic mix in the trace. With this seeding process, each application class starts with an equal chance to be the most popular class in the connectivity graph.[4]

(b) The practical case: Using other profiling methods. We conduct experiments where the seed is obtained by other existing or combinations of existing methods.

[4]In some instances, the base-case may require a certain number of seeds for an application, which does not appear enough in the trace. This is only an issue when we experiment with larger seed sizes (e.g., 5% and 10%). In such instances, we label the maximum number of seeds we can for that class and distribute the remaining seed slots equally to the remaining classes. Note that our seed methodology guarantees that the largest classes in a trace such as DNS, Web, P2P, and Email get equal share of initial seeds for each experiment.

Algo.	Step 1	Step 2	Step 3
NLC	Inference NLC	-	-
CLUS	Inference CLUS	-	-
C+NLC	Inference CLUS	Inference NLC	Merge results from Steps 1&2
CSEED	Inference CLUS	Seeding from Step 1, then Inference NLC	-
HYP	Inference CLUS	NLC on Hypergraph	Inter-cluster edge labeling heuristic

Table 2: Summary of the algorithms used for the inference step (step B) of the PBA framework.

Our vision for a practical deployment is as follows: Existing methods can be used to provide only the information for which they have high confidence. PBA takes that information as seeds and performs inference (see §4.2).

4.2 Step B: Inference

For the inference step, we present five PBA algorithms that make use of network structure in two distinct ways. First, by exploiting the **local structure** of the trace graph, where neighboring edges (flows) are likely to share the same application class i.e., graphs exhibit *edge homophily* [8]. We see this visually in Figure 1, where if we isolate any single node and inspect its edges and the edges of its neighbors, they are all likely to be of the same application (e.g., eMule). Second, they utilize the **global structure** of the graph to detect communities of hosts. As we see in Figure 1, all edges in such communities tend to share common application. There are various community discovery algorithms [2, 9, 5, 4, 20]. As we show in §5, the best one in terms of accuracy and speed for our problem is Louvain [2]. PBA's inference step can use local structure only, global structure only, or a combination of local and global structures. In this paper, we consider five PBA-based algorithms as listed in Table 2. The NLC algorithm was introduced in an earlier 5-page paper [8]. We include a short description of NLC for completeness and report it in our experiments for comparison. All the other algorithms are contributions of this work. Next, we provide details for all five algorithms.

Algorithm using local structure.

Neighboring Link Classifier (NLC): Flows are iteratively labeled according to the labels of adjacent flows (i.e., flows that share a common IP). In more details, at first all edges of the seeds are labeled to be of the same class. Next, at each iteration step, for each unlabeled edge, the output of NLC is a probability distribution over the application classes – i.e., $\forall u \in U$, $\forall c \in C : P(label(u) \equiv c|u)$, where U is the set of unlabeled edges $u = (s, d)$ in the trace graph and C is the set of application classes that we want to classify (e.g., C = {P2P, DNS, Web}). Where, $P(label(u) \equiv c|u) = (p_s + p_d)/2$, and p_s and p_d denote the ratio of edges of the end-nodes s and d of edge u, that are of class c. To obtain a classification for an unlabeled edge, we select the application with the highest probability on that unlabeled edge: $\forall u \in U : label(u) = argmax_{c \in C}(P(label(u) \equiv c|u))$. The iteration process is repeated until all edges are labeled. Here, we use 10 iterations which are enough for convergence. This approach is described in detail in our previous work [8].

Algorithm using global structure.

Clustering (CLUS): We cluster hosts using LUV and include in each cluster all flows for which both endpoints fall in the cluster (a.k.a. intra-cluster edges). How we label each cluster will be explained in the next paragraph. After a cluster is labeled, we assign the same label to every flow in the cluster. Flows between two clusters (a.k.a. inter-cluster edges) where both clusters share the same label are also assigned that label. Edges in clusters with no seeds, and inter-cluster edges between clusters that have different labels are left unlabeled.

Our cluster labeling process is as follows. For each cluster, we calculate the normalized frequency $nFreq(c)$ of each class by dividing the seeds of that class in the cluster by the total number of seeds of the class. If a cluster has seeds of only one application it takes the label of that application class. Else, if the cluster contains seeds from more than one class, we re-apply the clustering step to that cluster alone. The process is recursively repeated until all sub-clusters have no more than one class with $nFreq(c) > 0.01$, or if the clustering algorithm does not further divide the sub-cluster. We observed that any small value, other than 0.01, is sufficient for the recursion to terminate in a reasonable time. At the end of the recursion, each cluster takes the label of the class with the highest $nFreq(c)$ in the cluster. The normalized frequency guarantees that no class will be favored because it is easier to find seeds. This makes our classification processes more realistic in practice. For example, if payload is used for seeding, P2P will typically have far fewer seeds than DNS, since DNS is easier to be classified. In a mixed cluster of P2P and DNS, DNS might be the smaller class, but have more seeds because of its higher total number of seeds.

Algorithms combining local and global structures.

Clustering+NLC (C+NLC): As we show in Table 2, the inference part of this algorithm has three steps. At the first step, we apply CLUS to label as many flows possible. At the second step, we independently run NLC and get a new prediction for all the flows. At the last step we merge the two predictions by keeping all the labels by CLUS and use the predictions from NLC to classify the remaining flows. This algorithm is based on our observation that CLUS alone gives predictions with high accuracy, but leaves several edges without any prediction, i.e., it has low coverage. The coverage problem is here solved by using the prediction from NLC.

Cluster-Seeded NLC (CSEED): The inference process has two steps using CLUS and NLC. Unlike

C+NLC, the two algorithms are not executed independently. First, we apply CLUS and label as many flows possible. Next, we use all the predictions from step 1 and use them to seed NLC. This algorithm is always slower than C+NLC since the two inferences need to be executed sequentially. The motivation behind this method is similar to the algorithm above (C+NLC).

Hypergraph-based NLC (HYP): We apply CLUS and then apply NLC to the inter-cluster graph (i.e., the *hypergraph*) to label all clusters for which we do not have seeds. The inter-cluster graph is created by collapsing each cluster to a single node and preserving only inter-cluster flows. Inter-cluster flows $f = (u, v)$ are assigned the inferred class of the endpoint (i.e., u or v) with the largest intra-cluster degree (i.e., the number of flows for a host within its cluster).

The heuristic for labeling inter-cluster edges is based on the observation that servers have high degree. We give a toy example to explain this. We assume a host u is involved in a P2P application and also contacts a Web server v. This edge (u, v) will likely be between the P2P cluster of node u and the Web cluster of node v. Since Web servers are typically of high degree, the degree of v will be higher than the degree of u. Therefore, the prediction for edge u, v will be Web, which is the correct in this example. As we show next, HYP gives high classification accuracy and shows robustness to all four levels of obfuscation!

5. EVALUATION

This section presents our experiments on (a) the robustness of our techniques to **obfuscation** and (b) the impact of practical factors. For the former, we look at error/noise in seeding and perturbations to connectivity structure of the trace graph. For the latter, we look at location of observation (backbone link) and specific implementation choices of our techniques.

All our traces have both encrypted (up to 40%) and unencrypted traffic. To extract ground truth for our evaluation, we manually classify all the unencrypted traffic using payload signatures. In our experiments, we emulate high obfuscation setting by "hiding" (setting as unknown) and introducing intentional errors to parts of the known traffic. Next, we observe how our methods can both correct these errors as well as successfully predict the labels for all the "hidden" portions of the known traffic.

In §5.1, we describe the four real-world backbone datasets used in our experiments, the derivation of ground-truth for evaluation purposes, the metrics used to quantitatively evaluate our approach, and our experimental methodology. All our experiments are described in §5.2 and summarized in Table 4.

5.1 Experimental Setup

5.1.1 Backbone data sets

We applied our methods to four different backbone links. Table 3 presents a summary of trace characteristics, calculated over a five minute slice of each trace.

Name:	**MFN**	**WIDE**	**PAIX**	**BRAZ**
Year	2003	2006	2004	2007
Mbps	213.27	26.21	788.62	304.47
# Flows	909,684	157,090	2,671,885	787,783
# IPs	263865	92239	531057	402309
%UDP	32.89	72.46	33.42	49.13
%TCP	67.11	27.54	66.58	50.87
LCC	87.14	89.71	92.10	99.98
Avg. Degree	3.17	2.33	4.50	2.67
Application Breakdown				
P2P	11.93	0.84	20.52	24.88
Web	51.71	13.62	54.33	25.86
DNS	17.85	37.24	14.83	3.47
Email	0.88	2.57	3.64	1.67
Games	0.83	0.02	1.02	0.55
Other	0.80	1.57	2.76	2.28
Unknown	16.00	44.14	2.90	41.29
Total	100.00	100.00	100.00	100.00

Table 3: Summary of backbone traces. All traffic percentages refer to flows.

The traces are collected over different geographic locations (South America, US, and Japan) with diversity in the application mixing (see Table 3). We define a flow using the well-known 5-tuple: ⟨`srcIP`, `srcPort`, `dstIP`, `dstPort`, `protocol`⟩. In all traces, there exists a large connected component (LCC) that contains the vast majority of flows, ranging from 87.1% to 99.9% depending on the trace. We describe each data set in more detail below.

PAIX: This data set was collected from an OC48 link of a commercial large US Tier-1 ISP at the Palo Alto Internet eXchange (PAIX), connecting San-Jose with Seattle. This is the largest trace in our data set in terms of traffic volume, observed flows, and distinct IP addresses. The monitor captured traffic from both directions of the traffic link. In addition, the data set contains up to 16 bytes of payload from each packet.

MFN: This trace contains traffic from a peering link of a large ISP in the west-coast US. The monitor captured traffic from both directions of the traffic link. In addition, the data set contains up to 16 bytes of payload from each packet. The MFN and PAIX are kindly provided by CAIDA (`www.caida.org`).

WIDE: This trace is collected from a low bandwidth (100 Mbps Ethernet) transpacific backbone link connecting the US with WIDE (Japan) and carries commodity traffic of the WIDE member organization. The trace contains traffic from both directions of the link. For each packet, it contains full packet header and 40 bytes of payload. This trace is kindly provided by MAWI (`mawi.wide.ad.jp/mawi/`).

BRAZ: This data set represents a smaller backbone link. The captured traffic is from a 1 Gbps Ethernet link that connects a small stub ISP (residential users) to a larger provider. This trace is closer to the edge of the network and captures all the public traffic of the small ISP. This trace is kindly provided by Narus (`www.narus.com`).

Connectivity Information: Our connectivity information is an IP-to-IP interaction graph (a.k.a. a trace graph or connectivity graph). We form links between IP hosts (i.e., nodes) that share at least one flow (i.e.,

ID	Description	Seeding Source	Seed Size	Seed Errors	Connectivity Graph	Obfuscation Levels	Section
1.	Finding a good community discovery algorithm	Payload	1%	None	Unmodified	-	§5.2.1
2.	Comparing the different PBA algorithms over four different locations of observation	Payload	1%	None	Unmodified	-	§5.2.2
3.	Effect of seed size on classification accuracy	Payload	0.01%-50%	None	Unmodified	L1, L2, L3	§5.2.3
4.	Effect of random errors in the initial seeds	Payload	1%, 10%	Random	Unmodified	L1, L2, L3	§5.2.4
5.	Seeding from existing traffic classifiers: BLINC [12], CoralReef [3]	BLINC, CoralReef	20%-80%	No	Unmodified	L1	§5.2.5
6.	Robustness to connectivity-level obfuscation	Payload	1%	No	Edges are randomly added	L4	§5.2.6
7.	Robustness to blending by P2P applications	Payload	55%-95%	$\{p2p \rightarrow web\}$, $\{p2p \rightarrow mail\}$	Unmodified	L1, L2, L3	§5.2.7

Table 4: Details of all the experiments in the Evaluation Section (§5.2).

edge) between them. If multiple flows of the same application exist between two IP nodes, they are merged into a single link, which is then assigned that application label. In our traces, 99.9% of all parallel flows are classified to be of the same application by our payload-classifiers. As expected, the majority of parallel edges belong to Web. Our final graph is undirected and contains no self-loops.

5.1.2 Obtaining ground-truth

For evaluation purposes, we need to obtain ground-truth (namely, application information on flows). All our traces contain payload information, thereby enabling us to label flows with their applications using. In this paper, we used an existing payload-based classifier from CAIDA (`www.caida.org`), that was also used in other traffic profiling work [12, 13].

Payload classifiers report as "Unknown" flows that do not match any of their known signatures (e.g., due to encryption). Also, payload classifiers cannot classify flows that do not carry any payload (e.g. because of worm scanning activity and other failed TCP connections). Since we are using payload classifiers as our ground-truth providers, we do not use these unknown and unclassified flows in our evaluation.

For each trace, our chosen payload-based classifiers label flows according to the following main classes: *P2P*, *DNS*, *Email*, *Web*, *Games*, *Chat*, *Rest*, and *Unknown*. In our experiments, we consider all application classes that contribute to more than 0.1% of all the flows in the trace. We made this choice because sparse applications produce results that are statistically difficult to interpret.

5.1.3 Evaluation metrics

We compare the flow-labeling of our methods to the ground-truth (i.e., the set of flows that are "known" to our payload-classifiers). Our performance metrics are as follows:

Coverage: Percentage of flows being labeled.

Overall accuracy: Percentage of accurate labels over all flows. This metric gives the probability of a correct prediction to any randomly selected flow.

Precision, Recall, and F1-score per application class: Precision is the ratio of true-positives to the sum of true- and false-positives. Recall is the ratio of true-positives to the sum of true-positives and false-negatives. F1-score is the harmonic mean of Precision and Recall.

5.1.4 Experimental methodology

Unless otherwise stated, all our experiments follow the same methodology. A portion of the IP hosts with labeled flows are selected to be the seeds. We apply our PBA algorithms on each trace and classify all the flows. Given that seeds can have errors, in all our experiments we allow the initial labels on flows to change. For each experiment, we run 20 independent trials each time selecting a different set of initial seeds. We then report the average, minimum, and maximum performance metrics computed over all the flows with ground-truth.

Baseline classifier. In some experiments, we compared our techniques with a "baseline" classifier. This classifier does not perform any inference, but assumes perfect knowledge about the seeds used in the experiment. As discussed in §4.1, none of our techniques assumes perfect knowledge about the seeds, since in our evaluation typically only a fraction of a seed's flows is known. Since seeds are on IP hosts and results are reported on flows, the baseline classifier is a good indicator of how challenging the problem at hand really is. For example, in some traces, the 1% of hosts selected as seeds contribute to 2% or more of the total flows. The baseline results capture exactly this.

5.2 Experimental Results

Table 4 summarizes the various experiments that we conducted. The first and second columns provide an ID and a short description of each experiment, respectively. The third column, titled "Seeding Source," lists the specific method used to extract seeds; a "Payload" entry here means the source was the same as the one used for our ground-truth labels. The fourth column, "Seed Size," indicates the portion of hosts for which we had seeds. The fifth column, "Seed Errors," lists

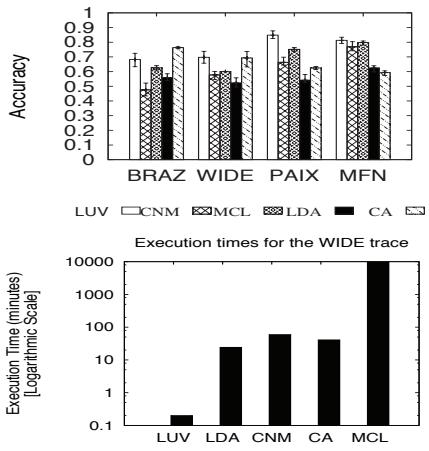

Figure 3: Performance for various community discovery algorithms on trace graphs. The top plot shows the accuracy of PBA's CLUS algorithm. The bottom plot shows execution time for the WIDE trace. Execution times for other traces were similar.

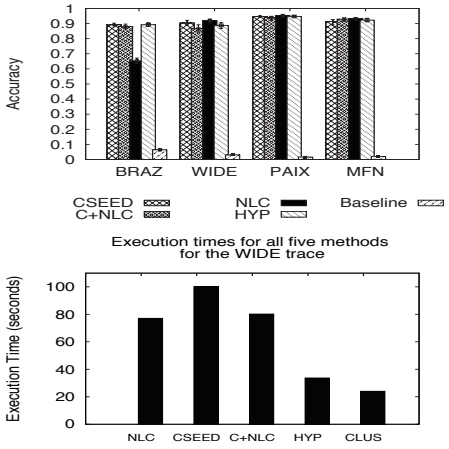

Figure 4: Classification results for various PBA algorithms using 1% seed size.

whether errors were intentionally introduced in the seeds; and if so, whether they were random (where a seed was reassigned another class at random) or targeted (where seeds from a particular class were changed to a specific class). The sixth column, "Connectivity Graph," indicates if the initial IP-to-IP graph is affected by the experiment (e.g., whether additional connections were inserted intentionally to obfuscate other traffic). The seventh column indicates whether the experiment dealt with obfuscation and at which levels (see §2 for the different levels). The last column lists the section where the experiment is described.

5.2.1 Comparing community discovery algorithms

Most PBA's algorithm use a community discovery algorithm. Here we evaluate the CLUS algorithm (§4.2), using five different community discovery algorithms: *Louvain* (LUV) [2], *Fast Modulatiry* (CNM) [5], *Markov CLustering* (MCL) [20], *LDA-G* (LDA) [9], and *Cross Associations* (CA) [4]. The LUV and CNM algorithms have been successfully used in a recent work on social networks [14]. All five algorithms are among the most popular techniques used for clustering graphs. We refer the reader to the cited papers for details. For PBA, a good algorithm is one that has high classification accuracy (where homogeneous communities are formed) and low computational time. Figure 3 shows the accuracy and runtime results of the aforementioned five community discovery algorithms on our trace graphs. The error bars show the maximum and minimum accuracy value over 20 trials. All experiments are executed on a Quad-Core AMD Opteron(tm) Processor 2350 at 2GHz with 8GB or RAM. In our experiments the memory requirements never exceeded 4GB.

Takeaway: Louvain (LUV) is an order of magnitude faster than CNM, MCL, LDA, and CA. It also produces communities with some of the best accuracies compared to others and is parameter-free. For the remainder of the paper, we use LUV for all of PBA's clustering-based algorithms.

5.2.2 Comparing different PBA-based algorithms

The HYP algorithm shows similar robustness to obfuscation as CLUS, but always produce higher accuracy. For brevity, for the remaining of the paper we do not explicitly report results for the CLUS algorithm. Figure 4 reports on classification performances of various PBA algorithms when only 1% of the IP hosts in each trace are used as seeds. The error bars show the maximum and minimum value over all trials. Details on the baseline classifier are given in §5.1.4.

Takeaway: All PBA methods perform well with even only 1% seed size. NLC has lower accuracy for the BRAZ trace due to lower homophily (defined in §3) in this trace. In the BRAZ trace, we have an order of magnitude more hosts involved in multiple applications compared to the other three traces. More on hosts with multiple application is discussed in §6. The fastest method is HYP because the clustering step reduces the computational cost compared to running NLC on all the edges of the graph. In all traces, HYP produces results in less than five minutes. Given that each trace is five minute long, HYP can produce near real-time classification results.

5.2.3 Effect of seed size

There are several factors that can affect seed size in a practical setting. First is the seeding method used and the actual application breakdown on the link (e.g., if most of the traffic is DNS then seeding using well-known ports is easier). Second is the portion of traffic being obfuscated (e.g., the percentage of encrypted traffic when the seeding is based on payload). Figure 5 shows the results for the MFN trace. The other traces give similar results so for brevity we have omitted them. In Figure 5, we also show the standard error from the different trials of the experiment. The baseline classifier shows the accuracy we would have had if an oracle-classifier gave the correct labels for all flows of

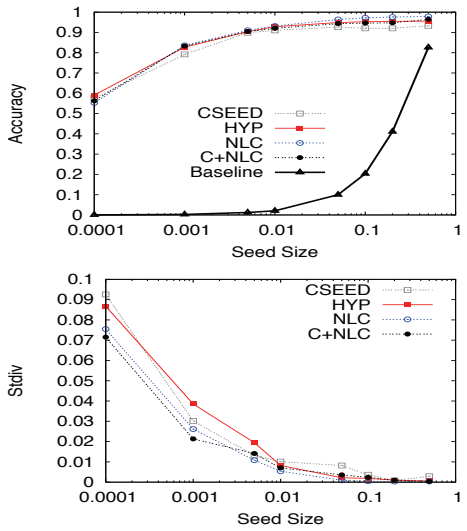

Figure 5: Effect of seed size for the MFN trace. The top plot shows overall accuracy. The bottom plot depicts standard error over various seed sizes (across 20 trials).

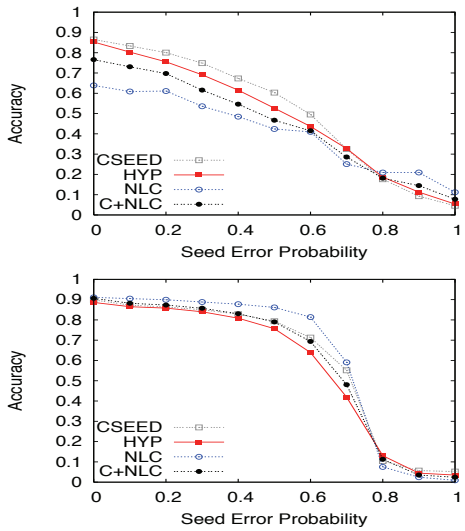

Figure 6: Effect of random errors in the seeds for the BRAZ trace. Other traces are qualitatively similar. The top plot shows results using 1% seed size; the bottom plot depicts results using 10% seed size.

the hosts selected as seeds (see more in §5.1.4).

Takeaway: As expected, increasing seed size improves the classification accuracy. It also reduces the standard error in the classification results. That is, more seeds eliminate the random effect of selecting different seeds. Moreover, we observe that NLC utilizes the high number of initial seeds better than the other methods, which saturate with approximately 10% seed size. With more seeds, the percentage of neighboring IP-hosts that are seeds increases. NLC is capable of utilizing these extra seeds since it operates at the neighbor-level. In contrast, having more known IP-hosts in a cluster is less important given that few seeds are enough to label each cluster.

5.2.4 *Effect of random errors in the seeds*

Errors in the seeds can occur from both obfuscation as well as from non-intentional sources (e.g., similarity between traffic of two different classes). For this experiment, we randomly select a seed and reassign an incorrect label to it by choosing a random application class label. From all four traces, the BRAZ trace is the most challenging. Figure 6 summarizes the results for the BRAZ trace using 1% seed size (top plot) and 10% seed size (bottom plot). For both seed sizes, we increase the probability of a seed to be incorrect from 0 (no intentional errors) to 1 (all seeds have errors) and report the overall accuracy. Compared to the BRAZ trace, our other traces have similar or better robustness to random errors.

Takeaway: All PBA algorithms are robust to random errors. For example, in Figure 6 (bottom plot) with 10% seeds and 40% of those seeds incorrect, the overall accuracy for all algorithms is above 80%. The higher the number of initial seeds, the more robust the PBA methods. This is because few incorrect seeds can have a significant impact in a low-seeded trace graph. We see an example of this behavior in Figure 6 (top plot) for the BRAZ trace.

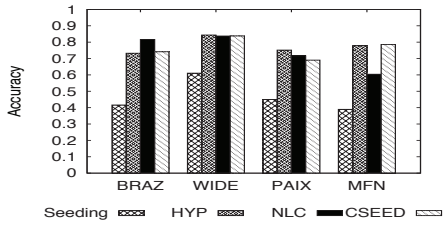

Figure 7: Seeding using seeds from a host-based and port-based classifier. The seeds are from the combination of BLINC and CoralReef (called ENSEMBLE).

5.2.5 *Seeding using existing classifiers*

Our algorithms can also be used to increase the accuracy of existing classifiers. For these experiments, we label all traces using: (a) BLINC [12], (b) Coral Reef [3], and (c) an ensemble result of BLINC and Coral (*ENSEMBLE*). In ENSEMBLE, we keep only the (flow-label) predictions for which the two methods agree; all other flows are left unknown. Intuitively, with ENSEMBLE we have less classified flows, but higher prediction accuracy on the flows for which we have predictions. For this experiment, we use all classified traffic from each method (BLINC, Coral, or ENSEMBLE) to seed our PBA algorithms. Figure 7 illustrates the results using ENSEMBLE for all four traces. Seeding with the other methods gives qualitatively similar results. Figure 7 also reports the

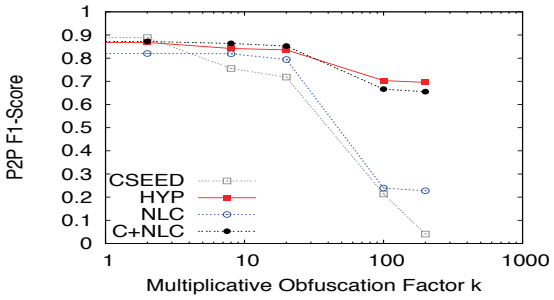

Figure 8: Effect of connectivity obfuscation for the BRAZ trace. This is the most challenging trace since it has the highest number of hosts with multiple applications.

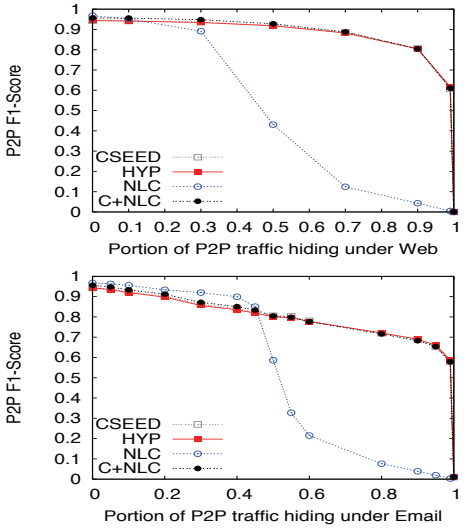

Figure 9: Obfuscation by blending for the BRAZ trace. P2P traffic is trying to tunnel its traffic using on of the following two applications: WEB (top plot) and EMAIL (bottom plot).

overall accuracy of the seeding method (refereed to as "Seeding") for comparison.

Takeaway: Our PBA algorithms take the results (seeds) from the initial classifier and increase the overall accuracy. All PBA algorithms perform very similar in our experiments. NLC is the one having the highest variation in these experiments (e.g., compare the results for the MFN trace in Figure 7). Looking closer at the seeds from ENSEMBLE, we observe that the classified traffic has very different application breakdown than with ground truth. Using the noisy seeds from Coral Reef and BLINC, we observe that HYP is more robust, which agrees with the rest of our experiments.

5.2.6 *Robustness to connectivity (L4) obfuscation*

All PBA algorithms are based on the connectivity properties of hosts in the trace graphs. If P2P hosts want to evade PBA-based detection, they can choose to increase their connections to non-P2P destinations. This reduces homophily (defined in §3) and makes P2P communities harder to detect. To evaluate the robustness to this level of obfuscation, we conduct the following experiment. We increase the number of flows between P2P hosts by adding a constant factor k of other applications. That is, we first measure the number of non-P2P flows from all the hosts that have at least one P2P flow; let that number be m. We then add $m \times k$ edges to the trace graph by randomly selecting P2P hosts and a random server of a different class (e.g., DNS, Web, etc.). Figure 8 shows the P2P classification accuracy (as measured by the F1-Score) over different k values for the BRAZ trace. The BRAZ trace is the most challenging for this test since it has the highest number of P2P hosts with multiple application. In other words, it has the highest m. We also experimented with opening connections to only a single application, such as Web. The results are either qualitatively similar, or much better than these shown here.

Takeaway: Even though all PBA algorithms are affected by connectivity-level obfuscation, none of them breaks especially for small values of factor k. In all our traces, NLC is the most sensitive because homophily significantly decreases as k increases. On the other hand, clustering-based algorithms are more robust. This is because P2P clusters are very dense and in order for them to be harder to identify, a P2P host needs to open a very high number of non-P2P connections. We stop our experiments at $k = 200$ since this is a very high number (200 times more non-P2P flows than currently used) and it makes the experiments much slower since the actual size of the graph increases by a factor of 50 in the BRAZ trace. Overall, HYP is the more robust to connectivity-level obfuscation than other PBA-based algorithms.

5.2.7 *Robustness to "blending" obfuscations*

To evade detection, a P2P application can make its traffic look like another application (e.g., Web). In other words, it can try to blend-in and evade being detected. This is different from random seed errors (§5.2.4) because here the seeder repeats the same error every time. To emulate this, we randomly take P2P seeds and reassign them to always be on one of the following applications: Web, DNS, Email, or FTP. The percentage of P2P seeds that are reassigned is called *blending intensity*. Figure 9 shows the P2P F1-Score for the BRAZ trace as we increase blending intensity. The top plot shows the results for Web applications; the bottom shows them for Email applications.

Takeaway: All PBA algorithms are robust to low levels of blending obfuscation. NLC shows a higher sensitivity to blending. If P2P blends in a large class such as Web, then all clustering methods show high robustness and achieve above 0.9 F1-Score even if 50% of P2P hosts are changed. This is related to the high population for Web in traces. Consider for example a P2P cluster in which a portion of the seeds are reassigned to be Web. As long as the normalized frequency of P2P hosts in the cluster (see definition of CLUS in

§4.2) is higher than Web, the cluster will continue to be classified as P2P. That is, even though the absolute number of seeds being Web can get higher than P2P, the normalized frequency of P2P would be higher as long as the overall P2P seeds in the trace are less than Web (which is something typical in most networks today). Moreover, the more P2P seeds are reassigned to Web, the higher the number of Web seeds will become; thereby increasing the gap between the two applications. This explains why we observe the nearly horizontal line in F1-Score in Figure 9 (top plot). Clearly, if P2P chooses to blend in a smaller class, the classification becomes more challenging for clustering methods. In such cases, NLC shows high robustness. This is because NLC relies on homophily (defined in §3) and smaller classes typically play less important role in the classification. We see this in Figure 9 (bottom plot). If the selected class contributes a very small portion of the traffic, such as FTP, then clustering-based methods suffer more and tend to report large portions of P2P as for example FTP. This will overall raise suspicions, given than FTP traffic is never expected to be that high (see Table 3). Moreover, to cleverly select the right class to blend-in requires knowledge of the application breakdown of the backbone link. This information is not available to the end users, and tends to be very different from link to link, as we see in Table 3. Given that the easier scenario is for a user to hide under Web, we conclude that our HYP algorithm shows high robustness to this type of traffic obfuscation.

6. DISCUSSION

How do we find seeds in practice? Our approach does not depend on any specific method for obtaining seeds, as we discussed in §4.1 and showed in our evaluation (§5). In short, we can use external knowledge such as information from sys-admins, legacy IPs, and well-known servers. In addition, we saw that "Googling the Internet" [19] is another method that harnesses the power of the Web. Seeds can also come from existing classifiers, as we show in §5.2.5. As a last resort, we can also use an active approach for the seeding process. For example, tools could periodically join P2P applications and online games to collect IPs of potential servers and users. Although these are typically more labor-intensive some of these measurements could be automated.

What if all traffic is obfuscated? If the obfuscations is at the port-, payload-, and flow-level, our approach is one of the best choices available. It should be clear by now that our method can work, and work well, as long as we can obtain seed information. In an extreme case where all traffic is obfuscated, we return to the question of how we find seeds. This is a topic we covered earlier in the section. Note that most known methods fail, especially if we also consider encryption or anonymization of IP addresses: Other methods do not use associations and cannot take advantage of seed information in this way. If obfuscation is at the connectivity level, we have seen in §5.2.6 that our HYP algorithm is robust. In particular, our approach provides an additional hurdle for an application that wants to evade detection. Namely, apart from obfuscating packet and flow level information, it must establish more connections in a strategic way.

Isolated groups of flows. A potential limitation of our methods is the labeling of isolated groups of flows. What can we do for such flows besides reporting them as "unknown?" We can address this by increasing the interval of observation in an effort to create larger connected components and repeat our analysis. In our traces, by increasing the interval of observation to ten minutes, the largest connected component includes more than 95% of all the flows. Using longer intervals introduces additional overhead (memory and CPU) to our algorithms, which is the main reason why we used five-minute traces. We also contacted extensive experiments with traces up to 15 minutes, but we do not include detailed results due to space limitations. As a takeaway, with longer traces we get either similar or considerably better results in all four traces when the same percentage of initial seeds is used.

Are our traces representative? This is a question that can haunt any trace-driven study. We find that the use of four different traces at significantly different locations provides a reasonable sample space. Our traces cover two Tier-1 backbone links (MFN, PAIX), one link from a single-home stub AS (BRAZ), and another link currying trasn-pasific traffic from a large organization (WIDE). Moreover, their traffic mix varies (see Table 3), with some links having Web, and others DNS, as their most popular application. To gain access to these backbone data with real-IPs and payload, required several months and signing privacy agreements with three different organizations. We will continue to look for new backbone traces to run our algorithms, but this is clearly not a trivial task.

How easy is it to deploy the algorithms in practice? All our algorithms operate with minimal parameters. The Louvain community discovery algorithm in HYP is parameter-free and for NLC we only select the number of iterations, where ten iterations suffice for convergence in all our traces over a large set of configurations (seeds sizes, intervals of observation, etc.). For seeding, we used the same payload signatures, the same port numbers in Coral Reef [3], and the same default parameters in BLINC [12], across all four backbone traces. We expect the algorithms and the parameters we use here to be reasonable starting points for deployment in other backbone locations as well.

Host with multiple applications. Despite receiving seeds on the hosts, all our algorithms profile traffic at the flow-level and not at the host-level. This allows different edges (flows) of a single host to have different neighborhoods and therefore be associated with different applications. In our experiments, the resulting percentage of hosts with multiple applications remain very close as compared to the results from ground truth. For example, in the PAIX trace where we have the fewer unknown flows, the initial percentage of hosts with multiple applications is 1.97% and the

amount reported by NLC and HYP is 1.92% and 1.84%, respectively. One should not be surprised by the small number of hosts with multiple applications on backbone links. Consider the following toy example. The monitoring point is behind a large Google server but not behind any large DNS server. Even though the browser of all Google users will first use DNS, the actual DNS flows might never cross our monitoring point. Therefore, even though all hosts use both Web and DNS, in our view of the traffic, the majority of the hosts are only observed to use Web.

Network Address Translation (NAT). Overall, our approach treats NAT-ed hosts the same way as hosts with multiple applications. In fact, how a NAT will appear in data trace has to do with: (a) how homogenous are the users behind the NAT regarding applications, and (b) what classes of traffic from the NAT will go through the observation point. For example, if all the hosts behind a NAT use the same one or the same mix of applications, this problem is analogous to that of one user with a mix of applications. The big difference is in the case where the hosts have widely different sets of applications, and all these types of flows pass through the link of observation. In that case, the NAT appears as a user with an unprecedented application mix.

7. SUMMARY AND CONCLUSIONS

In this paper, we address the problem of traffic profiling at the backbone in the presence of multi-level obfuscation. We propose **Profiling-by-Associations (PBA)** as a novel approach to this problem.

The key novelty is that our approach uses only connectivity information ("who-talks-to-whom") in a packet trace and some initial seeds. The seeds provide information about the applications used by few of the hosts in a trace. We develop four new algorithms following the PBA approach, and show empirically that they can effectively profile backbone traffic in the presence of obfuscation with as little as 1% seeding information. In particular, we highlight one of our algorithms, called HYP, which combines both local and global structures in the IP-to-IP graph, and shows high robustness to all levels of obfuscation.

In terms of practical deployment, our methods are not tied to a particular clustering algorithm or a particular source of seeds. Our experiments show that they are quite robust to the properties of the seed information, to connectivity obfuscation techniques, and the specifics of our data traces.

To conclude, we believe that our approach can lead to novel traffic analysis tools that overcome multi-level obfuscation. In fact, our approach can be seen as: (a) a standalone tool given seeding information, and (b) an add-on method to increase the accuracy of an existing profiling method (as we showed in §5.2.5).

8. ACKNOWLEDGMENTS

This work was supported partly by NSF NECO 0832069, a CISCO URP grant, and ARL CTA W911NF-09-2-0053. This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-461043. Special thanks to Thomas Karagiannis, Kenjiro Cho (WIDE), Ram Keralapura (Narus), and Antonio Nucci (Narus) for sharing their codes and datasets. Support for CAIDA's Internet traces is provided by the National Science Foundation, the US Department of Homeland Security, and CAIDA members.

9. REFERENCES

[1] L. Bernaille, R. Teixeira, and K. Salamatian. Early Application Identification. In *ACM CoNEXT*, 2006.
[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *J. Stat. Mech.*, page 10008, 2008.
[3] CAIDA Org. The CoralReef Project, http://www.caida.org/tools/measurement/coralreef/.
[4] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *SIGKDD*, 2004.
[5] A. Clauset, M. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70:066111, 2004.
[6] M. Dusi, A. Este, F. Gringoli, and L. Salgarelli. Using GMM and SVM-based techniques for the classification of SSH-Encrypted traffic. In *IEEE ICC*, 2009.
[7] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson. Identifying and Discriminating Between Web and Peer-to-peer Traffic in the Network Core. In *WWW*, 2007.
[8] B. Gallagher, M. Iliofotou, T. Eliassi-Rad, and M. Faloutsos. Homophily in application layer and its usage in traffic classification. In *IEEE INFOCOM (mini-conference)*, 2010.
[9] K. Henderson and T. Eliassi-Rad. Applying latent Dirichlet allocation to group discovery in large graphs. In *ACM SAC*, 2009.
[10] M. Iliofotou, H. Kim, P. Pappu, M. Faloutsos, M. Mitzenmacher, and G. Varghese. Graph-based P2P Traffic Classification at the Internet Backbone. In *IEEE GI*, 2009.
[11] Y. Jin, S. Esam, and Z. L. Zhang. Unveiling Core Network-Wide Communication Patterns through Application Traffic Activity Graph Decomposition. In *ACM SIGMETRICS*, 2009.
[12] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multi-level Traffic Classification in the Dark. In *ACM SIGCOMM*, 2005.
[13] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In *ACM CoNEXT*, 2008.
[14] H. Kwak, Y. Choi, Y.-H. Eom, H. Jeong, and S. Moon. Mining communities in networks: a solution for consistency and its evaluation. In *IMC*. ACM, 2009.
[15] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected Means of Protocol Inference. In *ACM IMC*, 2006.
[16] A. Moore and K. Papagiannaki. Toward the accurate identification of network applications. In *PAM*, 2005.
[17] A. Moore and D. Zuev. Internet Traffic Classification Using Bayesian Analysis Techniques. In *ACM SIGMETRICS*, 2005.
[18] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *WWW*, 2004.
[19] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci. Unconstrained endpoint profiling (Googling the Internet). In *ACM SIGCOMM*, 2008.
[20] S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000. http://www.micans.org/mcl/.
[21] K. Xu, Z. Zhang, and S. Bhattacharyya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *ACM SIGCOMM*, 2005.