SubFlow: Towards Practical Flow-Level Traffic Classification

Guowu Xie†, Marios Iliofotou*, Ram Keralapura*, Michalis Faloutsos† and Antonio Nucci*
†University of California Riverside, {xieg, michalis}@cs.ucr.edu
*Narus, Inc., {miliofotou, rkeralapura, anucci}@narus.com

***Abstract*—Many research efforts propose the use of flow-level features (e.g., packet sizes and inter-arrival times) and machine learning algorithms to solve the traffic classification problem. However, these statistical methods have not made the anticipated impact in the real world. We attribute this to two main reasons: (a) training the classifiers and bootstrapping the system is cumbersome, (b) the resulting classifiers have limited ability to adapt gracefully as the traffic behavior changes. In this paper, we propose an approach that is easy to bootstrap and deploy, as well as robust to changes in the traffic, such as the emergence of new applications. The key novelty of our classifier is that it learns to identify the traffic of each application in isolation, instead of trying to distinguish one application from another. This is a very challenging task that hides many caveats and subtleties. To make this possible, we adapt and use subspace clustering, a powerful technique that has not been used before in this context. Subspace clustering allows the profiling of applications to be more precise by automatically eliminating irrelevant features. We show that our approach exhibits very high accuracy in classifying each application on five traces from different ISPs captured between 2005 and 2011. This new way of looking at application classification could generate powerful and practical solutions in the space of traffic monitoring and network management.**

I. Introduction

Identifying the flows generated by different applications is of major interest for network operators. For Internet Service Providers (ISPs), identifying traffic allows them to differentiate the QoS for different types of applications, such as voice and video. Moreover, it enables them to control high-bandwidth and non-interactive applications, such as peer-to-peer (P2P). For enterprise networks, it is very important for administrators to know what happens on their network: what services users are running, which application is dominating their traffic, etc. Traffic classification is also important for securing the network. In fact, even traditional protocols are often used as means to control attacks, such as the use of IRC as C&C for botnets. Overall, traffic classification is the first step in building network intelligence.

In this paper, our goal is to develop a practical traffic classification approach, which should be: (a) easy to use, and (b) effective, in terms of accuracy. To the best of our knowledge, no such solution exists today. First, most deployed solutions rely heavily on payload-based or deep packet inspection (DPI) techniques. However, these techniques have several limitations, as they fail to classify encrypted traffic and raise privacy concerns. In addition, it is often desirable to classify traffic that is summarized in the form of flow records or packet headers. Second, port-based classification has severe limitations, as many applications randomize their ports and cannot detect new applications. Third, flow-level machine learning (ML) approaches have been proposed as an alternative to the expensive and cumbersome DPI methods. These methods use flow-level properties, such as packet sizes and timing information, to classify traffic. However, despite significant research efforts [14], [1], [12], these methods have not made the anticipated impact in the real world.

The contribution of our work is twofold:

(a) We study and document the limitations of existing flow-level ML solutions. ML approaches are grouped into two categories: supervised and semi-supervised solutions. Methods from both groups require a set of known flows to train/bootstrap the classifier. Typically, the training data are provided by a DPI classifier. In a nutshell, supervised methods train a ML algorithm to distinguish flows among a predefined set of applications. This has two key disadvantages: (i) it is often very hard to know all applications in the network a priori and train for them; and (ii) the classification performance degrades significantly when new applications emerge. Intuitively, if an algorithm learns to distinguish A from B, it is very hard to deal with a new class C. Semi-supervised solutions overcome this limitation by utilizing clustering techniques and do not explicitly learn to distinguish A from B. Unfortunately, these methods suffer from what is known as the curse of dimensionality. Standard feature selection methods used before [12], [9], are supervised and, therefore, have the same limitations explained before. As we show in §II in more detail, these challenges significantly affect the accuracy of existing solutions, both supervised and semi-supervised.

(b) We propose a new approach, **SubFlow**, that operates unlike any previous ML classifier: It learns the intrinsic statistical fingerprint of each application in isolation. In other words, our method learns to identify class A and then class B, instead of trying to distinguish A from B. Second, in order to address the curse of dimensionality, we utilize subspace clustering, which has never been used in the context of traffic classification before. Using our technique, our classifier extracts the key features of each application and ignores the features that are not useful. This is a very attractive property given the fact that one feature can be great for identifying application A, but be useless in identifying application B. Our approach has the following key advantages, which effectively address the limitations of previous methods: (i) bootstrapping is easier and practical, as we demonstrate in the rest of this paper; (ii) our approach is robust and adaptable to *change*, such as the appearance of new protocols, or the evolving behavior of existing applications.

978-1-4673-0775-8/12/$26.00 ©2012 IEEE

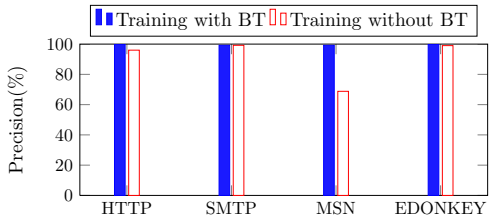

Fig. 1. The per-protocol precision when (a) BT is used in training, and (b) BT is not used for training, for a traditional multi-class classifier.

II. Problem Definition and Limitations of Previous Methods

Traffic classification problem: We define an application class (referred to as *application* throughout the paper) as an application-level protocol with a distinct documented behavior in terms of communication exchanges, control packets, etc. With this definition, our application classes distinguish between: SMTP, MSN, BitTorrent, Gnutella, POP3, etc. Moreover, the application is often referred to as the label or the class of the flow. The task of a flow-level traffic classifier is to identify the applications that generate flows in a network, without relying on payload information. In the paper we consider uni-directional flows defined by the five-tuple: {*Source IP, Source Port, Destination IP, Destination Port, and Protocol*}. Given that in the backbone one of the two directions is often missing (e.g., due to routing asymmetry), using uni-directional flows allows the classifier to be deployed anywhere in the network.

Statistical flow-level features: Flow-level classifiers utilize statistical information of a flow to make a prediction. Each feature represents one dimension in the feature vector of a flow. Throughout this paper, we use the terms feature and dimension in the same context. For this work, we have collected a number of key features that are also used by others [14], [12]. We used the exact size and the inter-arrival time (IAT) of the first six packets of the flow. In addition, we use the average, max, min, and standard deviation values of packet sizes and IATs over the entire flow. Intuitively, the exact size of the first packets captures the protocol behaviors during the initiation of the protocol interaction. The IAT is a good indicator of real-time applications such as VoIP or Video.

The performance of previous supervised ML flow-level classifiers degrades when new applications appear: We verified this observation using a large set of real-world data and different supervised classifiers proposed in the literature [9], such as nearest neighbors, logistic regression, decision trees, support vector machines (SVM), and Bayesian networks. We used the WEKA implementation of the algorithms with their default parameters as in [9]. Here, we show an example using data from Asia-3[1], where we try to classify five protocols, namely: HTTP, SMTP, EDONKEY, BitTorrent (BT), and MSN. From all the supervised algorithms we used, the Bayesian Networks gave the best results; therefore, here we only present results from this algorithm. Our experimental methodology is the same as in all the experiments in the paper and is explained in detail in §IV-A.

[1]More details on the specific trace is given in §IV.

Our goal is to compare the performance of the Bayesian Network classifier when: (a) all five protocols are included during training, and (b) when one or more of the protocols is not used for training. With (b), we simulate the senario where a new application appears (i.e., the one not used for training). For testing the classifier, we always use all the flows from all protocols, even from those excluded from training. In Figure 1, we compare the percentage of correctly classified flows (a.k.a. precision) of HTTP, SMTP, eDonkey, and MSN when (a) BT is included in training and (b) when BT is not used for training. From the figure, we see that when the training set consists of all five protocols, the precision approximates 100%. Unfortunately, when the classifier does not take BT into consideration, it mistakenly reports the traffic of BT as MSN. In fact, 35% of MSN traffic is now erroneous. We repeated the same experiment excluding protocols other than BT from training with qualitatively similar results. Moreover, we observed that the more protocols we exclude from training, the worse these classifiers performs.

Semi-supervised solutions perform less accurate than their supervised counterparts. We repeated the same experiment with BitTorrent as explained above, using two semi-supervised algorithms, namely k-means and DBSCAN. Both algorithms were used before for traffic classification [3]. Overall, we observed that clustering algorithms are less accurate than their supervised counterparts (approximately by $10-20\%$ in our experiment). In addition, their performance also degrades when applications are not used during training. In fact, if BT is excluded from training, the precisions of MSN and eDonkey drop by 20% and 10%, respectively.

Conclusions: Previous supervised ML flow-level algorithms are designed to effectively distinguish between a predefined set of classes. Unfortunately, when a new class appears the trained decision boundaries do not perform as well. This is what we have seen happening in the example of Figure 1. Regarding semi-supervised algorithms, even though they do not suffer from this limitation, we found them to perform less accurate than their supervised counterparts. An in-depth explanation of the differences between supervised and semi-supervised solutions is an interesting topic on its own, but is out of the scope of this work. For the purpose of our study, we identify one key limitations of semi-supervised solutions. The presence of irrelevant features leads to what is known as the curse of dimensionality. Simply put, the dilemma is that each application is captured best by different features, but if we use all these features together in a single feature space, they are too many, and the classification performance degrades. As we explain in §V, popular feature selection techniques are "multi-class" and therefore inherit the same limitation as traditional multi-class supervised classifiers.

III. SubFlow: Traffic Classification using Subspace Clustering

In this section, we present our SubFlow classifier. First, we give a basic overview in §III-A. Then, we explain the details of our subspace clustering method in §III-B.

A. Overview of SubFlow

Our classifier executes two different processes: (a) the generation of signatures, and (b) the classification of incoming flows. Here we give an overview of the two processes.

1) Signature generation process: During training, our classifier generates a set of signatures for each application. In what follows, we only provide the basic operations of the subspace clustering. All the algorithmic details are described in the next subsection §III-B.

Input: A set of flows F that belong to the same application and a full set of features S. Each flow $f \in F$ is represented as an $|S|$-dimensional vector of numerical values.

Output: One or more pairs of flows and subspaces (F_i, S_i), such that $F_i \subset F$ and $S_i \subset S$. S_i reports relevant features for the flows in F_i. This essentially projects the initial $|S|$-dimensional flows, to an $|S_i|$-dimensional subspace with $|S_i| \in (1...|S|)$. A feature subspace may contain more than one cluster comprising of different flows. That is, when $S_x = S_y$, $F_x \cap F_y = \emptyset$. Also, a flow can belong to clusters in different subspaces. That is, $f \in F_x$, $f \in F_y$, but $S_x \neq S_y$.

A signature is in fact a set of flows F_i and a corresponding feature space S_i returned by the subspace clustering algorithm. For each signature (F_i, S_i), the flows in F_i meet a given cluster criterion (i.e., they are very close to each other) when projected into the feature subspace S_i.

2) Classification process: During classification, a new flow is tested for a match over all the signatures for all applications. Essentially, each signature is a binary classifier that reports either match or not. We next present the process in more detail.

Testing a signature for a match: When a new flow is tested over a specific signature (F_i, S_i), it is first project to S_i and then compared with the flows in F_i. The distance between flows is calculated using the standard Euclidean distance. The distance to the closest flow is set to be the distance of the test flow to signature (F_i, S_i). If the test flow is within a predefined radius (r) is reported as a match. Essentially, each signature is a fixed-radius nearest neighbor classifier with $|S_i|$ dimensions and $|F_i|$ points. The fixed radius guarantees the signatures are specific enough to match the flows of the application without matching flows of other applications. It is not fair to use the same fixed radius for all subspaces, since from the Euclidean distance formula $d = \sqrt{\sum_{k=1}^{k=|S_i|} (x_k - y_k)^2}$ we see that the larger $|S_i|$ is, the larger the distance will become, even if the distance of each individual dimension remains small. We use the basic scaling factor of $\sqrt{|S_i|}$ to remedy this. Therefore, if the one dimensional radius we use in our classifier is r it means that the value becomes $r_i = \sqrt{|S_i|} \cdot r$ for signature i. We refer to the region covered by the radius of all the points of the signature as its *region of interest*. We evaluate our algorithm over different radius (r) values in the next section.

Classifying a flow: Our SubFlow classifier contains a number of binary classifiers. Each binary classifier corresponds to one application signature (F_i, S_i). Assume that at a specific point in time we have n binary classifiers, $X = \{x_1, x_2, ..., x_n\}$. Any new flow that reaches the SubFlow classifier is processed by each of the n binary classifiers. Each binary classifier replies with a `true` or `false` and the distance (d). Therefore, the outcome is: (a) an n-dimensional boolean vector $L = \{l_1, l_2, ..., l_n\}$ where the variable l_i captures the label given by the binary classifier i, i.e., where $l_i = 1$ iff x_i labels the flow `true`, otherwise $l_i = 0$; and (b) an n-dimensional vector $D = \{d_1, d_2, ..., d_n\}$, where the variable d_i captures the distance of the test flow to signature i. Now, since an application can be associated to multiple signatures, it may be mapped to more than one binary classifier. To keep track of the mapping between binary classifiers and applications, we introduce a new vector called M, where $M(i) = App$ if the binary classifier i is from the application *App*.

The final decision on whether the flow should be labelled as *App* or *Unknown* is made by executing the following algorithm that use vectors L, D, and M:

1) Add all i, where $L(i) =$ `true`, to the response set R.
2) If $|R| = 0$, reply as "Unknown" since no binary classifier gave a label for the flow. Else if $|R| = 1$, reply as $M(k)$, where $R = \{k\}$ since only one binary classifier labeled the flow. Else if $|R| > 1$, reply as $M(k)$, where $k \in R$ and $\forall i \in R, i \neq k, |S_j| < |S_k|$. That is, if more than one classifier labels the flows, the classifier from higher dimensional signature is chosen because it is more specific.

B. Details of subspace clustering

As we mentioned before, the input to our algorithm is a set of flows F that belong to a single application, over a feature set S. Our subspace clustering algorithm takes the following basic steps:

1) A standard clustering algorithm is used to find the so called base clusters in each dimension $s \in S$. Essentially, we cluster all the flows in F in each 1D-space $s \in S$.
2) Base clusters from different dimensions are then merged together to form higher dimensional subspaces. In order for base clusters to be merged, we require them to have a large ratio of common flows. Merging base clusters results in subspaces of dimensionality $d \in (2, ..., |S|)$.
3) To find the clusters in each subspace, we project all the flow in F to each subspace S_i. Then, we use a standard clustering algorithm to find the clusters in each subspace.

Additional implementation details: All the features are normalized using the standard z-score algorithm, defined as $x_{norm} = (x - \mu)/\sigma$, where μ and σ are the mean and standard deviation of the feature calculated over the training data. During the cluster merging phase, in order for two bass clusters to be merged at step 2, we required that the number of common flows between them is at least 50% of the number flows of the smaller of the two base clusters (we try different ratios and find the results are similar). Finally, it is often the case that a large cluster, say, C_1 has high overlap to two (or more) other clusters, say, C_2 and C_3, but the overlap of $C_2 \cap C_3$ is very small. The design choice of FIRES is to always split the larger base cluster into two smaller disjoint base clusters (C_{11}, C_{12}). In our implementation, we split a large cluster if and only if the new base clusters are larger than the average size of all base clusters. In our current implementation, we identify base clusters using DBSCAN [4] as FIRES.

IV. Evaluation

Datasets: We use five full packet traces (headers and payload) from different ISPs collected between 2005 to 2011. The ISPs are distributed across different geographic locations: three are in Asia, one in South America, and one in North America. We will refer to them them as Asia-1, Asia-2, Asia-3, SouthA, and NorthA in the paper. The traces Asia-1, Asia-2, Asia-3, and SouthA capture residential traffic from the customers of the ISPs as well as transient traffic. The trace from NorthA is from a cellular service provider and contains traffic from only mobile devices, such as laptops and smart phones with high speed data plans. Overall, our data traces are collected from a diverse set of network links, over different time periods, with different users, applications, and characteristics.

Extracting the ground truth labels of flows: The ground truth of a flow refers to its generating application. We extract the ground truth for our experiments, using a DPI classification techniques similar to those used in [8], [17]. Our traces contain traffic from the following applications: HTTP, SMTP, POP3, MSN, BitTorrent, eDonkey, Gnutella, Telnet, Samba (SMB), IMAP, XMPP, Yahoo IM, SSH, and FTP. The total number of unlabeled flows corresponds to 20% of the traffic. Typically, DPI labels a large number of flows as unknown because of either encrypted payload or incomplete signatures. This highlights the benefit of our algorithm, where its performance is not affected by the presence of unknown applications.

A. Experimental methodology and evaluation metrics

For evaluating our classifier, we split each trace into two parts. we use a fraction of the flows for training and the other disjoint fraction for testing. For training the classifier, we consider applications with more than 1,000 flows. We believe the number is small enough for collecting training data and large enough for extracting good signatures. The applications which have no enough flows (<1,000) are not included in training data, but included for testing the classifier. In what follows, we repeat all experiments ten times and just report the average values over all runs since the variations in different runs is very small (<1%). In all our experiments, we generate signatures for uni-directional flows. This allows us to extract different signatures from the client to server interaction, as well as from the server to the client. When we compare our classification predictions with the ground truth, we look at the labels we gave for both directions of the flow. If one direction was found to be unknown, we give it the label of its reverse direction if it exists. If the labels from two directions are different, we report the flow as unknown.

Evaluating the new application detection rate: With these experiments we aim to evaluate the accuracy of our classifier in detecting novel traffic. That is, when we classify the traffic of an application that we did not include in training, we want its traffic to be reported as unknown. For example, if the classifier does not have a signature for, say, BitTorrent (BT) the classifier should report all BT traffic as "unknown." Since BT was not known to the classifier before, it effectively represents a new application. Therefore, $NewApp_i = \frac{Unknown_i}{Total_i}$. We use the following methodology. We exclude from training one application at a time. Then, we observe how the classifier reports the flows of the "new" application (i.e., the $NewApp$ rate for that application). We summarize this behavior by averaging the $NewApp_i$ over all the applications i in the trace.

In addition to $NewApp$ rate we use the following performance metrics:

Coverage is the number of flows that were actually given a prediction (i.e., not labeled as "unknown"), divided by the total number of flows in the trace. It is also known as "completeness."

Accuracy on covered set is the number of correctly classified flows divided by the total number of predicted flows. That is, accuracy on covered $= \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i)}$. True positives for an application i (TP_i), is the number of flows correctly classified as i. False positives for i (FP_i), is the number of flows from other applications misclassified as i.

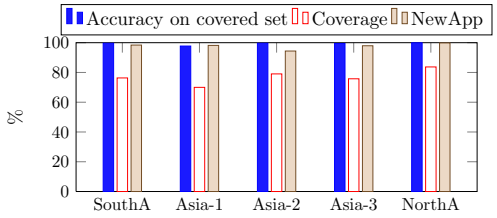

Fig. 2. The overall accuracy and coverage and NewApp rate for all five traces using the default configuration.

B. Experimental results

We evaluate the performance of our algorithm under different configurations. We used one trace to explore the different configurations and then finalize these parameters for all the other traces in all our experiments. Our subspace algorithm uses DBSCAN to generate the base clusters (see §III). The DBSCAN algorithm has two parameters, the distance (ε) and minimum points (*minPts*) [4]. Since we already have ε to denote the maximum allowed distance between flows, we also assign the radius r of the region of interest to have the same value. Therefore, in all our experiments $r = \varepsilon$. In addition, our classifier allows us to control its precisions by excluding signature of low dimensionality. Intuitively, using only one or two features to make a signature, increasing the probability of matching flows from other applications. We refer to the minimum allowed dimension of a signature as *minDim*.

Parameter sensitivity: To understand the the parameter sensitivity, we study the overall accuracy of our classifier by varying ε, *minPts*, and *minDim* over a large range of values. We observed that our classification is significantly affected by small variations of these parameters. More results are omitted due to space limitations. For the remaining of the paper, we use the following as our **default configuration**: $minPts = 10$, $\varepsilon = 10^{-4}$, and $minDim = 4$. We use the Asia-3 trace to study these parameters and to select a good configuration for our classifier. We then apply the same configuration to the remaining four traces.

Classification performance: We show the overall performance of SubFlow over all five traces using the default configuration in Figure 2. Event though we choose our parameters using the Asia-3 trace, we see from the figure that

the performance is very good over all five traces. Specifically, we see that SubFlow achieves high accuracy ($> 98\%$), high NewApp rate ($> 97\%$), and good coverage ($> 75\%$) over traces with a diverse traffic composition. The high NewApp rate shows that our method can successfully report traffic from applications not included in training as unknown in all traces.

V. Related Work

Over the last years, there have been many paper presenting different machine learning algorithms for solving the traffic classification problem. We refer the interested reader to a survey [14] that covers the majority of popular techniques. In what follows, we present the most representative supervised and semi-unsupervised ML methods proposed in the literature.

Supervised ML algorithms include naive Bayes classifier, nearest neighbors, decision trees, logistic regression, neural network and support vector machines [12], [9], [16], [2]. Features are selected manually or using supervised feature reduction algorithms like Correlation-Based Filter in [12], and regression techniques [2]. Such supervised feature selection techniques inherit the same limitations as supervised learning algorithms in handling new applications.

Semi-supervised algorithms explored in traffic classification include AutoClass, Expectation Maximization (EM), K-means, DBSCAN and more [3], [1]. Semi-supervised algorithms group flows into different clusters according to statistical information without a priori information about traffic mix. Moreover, as we explain in §II, semi-supervised techniques do not perform as well as their supervised counterparts. This is because uncovering clusters in high dimensional data is challenging because of the curse of dimensionality.

Other classification approaches are based on the behavior of end-hosts. BLINC [8], [18] classifies flows based on the number of connections created by different hosts. The label granularity from some of these approaches [18] are coarser than ours. In addition, host-based approaches, such as BLINC [8] do not perform well at the backbone [9]. All the traces we used here are from backbone networks and SubFlow performed very well. In [7], the proposed algorithm identifies traffic from known applications that try to evade detection. The algorithm uses a graph-based solutions and does not report unknown traffic. It therefore suffers from the same limitation as the other supervised approaches. Finally, the novel solution proposed in [17] utilizes information from Internet to profile IP addresses. As reported in [17], this approach gives very small recall for P2P applications.

Subspace clustering and one-class classification: The FIRES [11] algorithm is one of the many existing subspace clustering techniques. such as, SUBCLU, FIRES, and INSCY [13]. At the same time, the problem of building a classifier using only positive samples is not new in the data mining community. The problem is often refer to as one-class classification, or novelty detection [6]. We believe that how we define the traffic classification problem here, will open the way for new research efforts to investigate the effectiveness of different, subspace clustering and one-class classifiers in addressing the problem.

VI. Summary and Conclusions

The goal of this work is to develop an application classifier that would be relevant in practice. As our first contribution, we identify the factors that limit the practicality of the majority of the existing methods, especially focusing on Machine Learning techniques. For example, we see that the introduction of BitTorrent without prior training, decreases the accuracy of detecting MSN traffic by more than 30%. As our second contribution, we propose SubFlow, a different approach to application classification leveraging a subspace clustering technique. The key novelty is that our approach learns the intrinsic statistical fingerprint of each application in isolation, which deviates from typical supervised classification approaches. We show that SubFlow has a lot of promise. We stress-test the capabilities of our approach in a series of experiments with five different backbone traces. SubFlow performs very well with minimal manual intervention: it identifies traffic of an application with very high accuracy, on average higher than 95%, and can detect new applications successfully.

References

[1] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *2006 ACM CoNEXT conference*. ACM, 2006.

[2] T. En-Najjary, G. Urvoy-Keller, M. Pietrzyk, and J. Costeux. Application-based feature selection for internet traffic classification. In *22nd International Teletraffic Congress (ITC)*. IEEE, 2010.

[3] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. Semi-supervised network traffic classification. In *ACM SIGMETRICS*. ACM, 2007.

[4] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining*, volume 1996, pages 226–231. Portland: AAAI Press, 1996.

[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[6] K. Hempstalk and E. Frank. Discriminating against new classes: One-class versus multi-class classification. In *Advances in Artificial Intelligence*, pages 325–336. Springer, 2008.

[7] M. Iliofotou, B. Gallagher, T. Eliassi-Rad, G. Xie, and M. Faloutsos. Profiling-By-Association: a resilient traffic profiling solution for the Internet backbone. In *ACM CoNEXT*, 2010.

[8] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multi-level Traffic Classification in the Dark. In *ACM SIGCOMM*, 2005.

[9] 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.

[10] R. Kohavi and G. John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324, 1997.

[11] H. Kriegel, P. Kroger, M. Renz, and S. Wurst. A generic framework for efficient subspace clustering of high-dimensional data. In *Fifth IEEE International Conference on Data Mining,*. IEEE, 2005.

[12] A. Moore and D. Zuev. Internet Traffic Classification Using Bayesian Analysis Techniques. In *ACM SIGMETRICS*, 2005.

[13] E. Muller, S. Gnnemann, I. Assent, and T. Seidl. Evaluating clustering in subspace projections of high dimensional data. *Proceedings of the VLDB Endowment*, 2(1):1270–1281, 2009.

[14] T. T. T. Nguyen and G. Armitage. A Survey of Techniques for Internet Traffic Classification using Machine Learning. *IEEE Communications Surveys and Tutorials*, 4th edition, Mar. 2008.

[15] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, 2004.

[16] M. Pietrzyk, J. Costeux, G. Urvoy-Keller, and T. En-Najjary. Challenging statistical classification for operational usage: the adsl case. In *Proceedings of the 9th ACM IMC*, pages 122–135. ACM, 2009.

[17] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci. Unconstrained endpoint profiling (Googling the Internet). In *ACM SIGCOMM*, 2008.

[18] K. Xu, Z. Zhang, and S. Bhattacharyya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *ACM SIGCOMM*, 2005.