Offline/Realtime Traffic Classification Using Semi-Supervised Learning

Jeffrey Erman [a] Anirban Mahanti [b] Martin Arlitt [a,c] Ira Cohen [c] Carey Williamson [a]

[a] *Department of Computer Science, University of Calgary, Canada*
[b] *Department of Computer Science & Engineering, Indian Institute of Technology, Delhi, India*
[c] *Enterprise Systems & Software Lab, HP Labs, Palo Alto, USA*

Abstract

Identifying and categorizing network traffic by application type is challenging because of the continued evolution of applications, especially of those with a desire to be undetectable. The diminished effectiveness of port-based identification and the overheads of deep packet inspection approaches motivate us to classify traffic by exploiting distinctive flow characteristics of applications when they communicate on a network. In this paper, we explore this latter approach and propose a *semi-supervised* classification method that can accommodate both known and unknown applications. To the best of our knowledge, this is the first work to use semi-supervised learning techniques for the traffic classification problem. Our approach allows classifiers to be designed from training data that consists of only a few labeled and many unlabeled flows. We consider pragmatic classification issues such as longevity of classifiers and the need for retraining of classifiers. Our performance evaluation using empirical Internet traffic traces that span a 6-month period shows that: 1) high flow and byte classification accuracy (i.e., greater than 90%) can be achieved using training data that consists of a small number of labeled and a large number of unlabeled flows; 2) presence of "mice" and "elephant" flows in the Internet complicates the design of classifiers, especially of those with high byte accuracy, and necessities use of weighted sampling techniques to obtain training flows; and 3) retraining of classifiers is necessary only when there are non-transient changes in the network usage characteristics. As a proof of concept, we implement prototype *offline* and *realtime* classification systems to demonstrate the feasibility of our approach.

Key words: Internet Traffic Classification, Realtime Classification, Machine Learning, Semi-supervised Learning

1. Introduction

The demand for bandwidth management tools that optimize network performance and provide quality-of-service guarantees has increased substantially in recent years, in part, due to the phenomenal growth of bandwidth-hungry Peer-to-Peer (P2P) applications. Going by recent measurement studies in the literature and estimates by industry pundits, P2P now accounts for $50 - 70\%$ of the Internet traffic [3, 29]. It is, therefore, not surprising that many network operators are interested in tools to manage traffic such that traffic critical to business or traffic with realtime constraints is given higher priority service on their network. Critical for the success of any such tool is its ability to accurately, and in realtime, identify and categorize each network flow by the application responsible for the flow.

Identifying network traffic using port numbers was the norm in the recent past. This approach was successful because many traditional applications use port numbers assigned by or registered with the Internet Assigned Numbers Authority. The accuracy of this approach, however, has been seriously dented because

of the evolution of applications that do not communicate on standardized ports [3, 17, 28]. Many current generation P2P applications use ephemeral ports, and in some cases, use ports of well-known services such as Web and FTP to make them indistinguishable to the port-based classifier.

Techniques that rely on inspection of packet contents [15, 20, 22, 28] have been proposed to address the diminished effectiveness of port-based classification. These approaches attempt to determine whether or not a flow contains a characteristic signature of a known application. Studies show that these approaches work very well for today's Internet traffic, including P2P flows [15,28]. In fact, commercial bandwidth management tools use application signature matching to enhance robustness of classification.

Nevertheless, packet inspection approaches pose several limitations. First, these techniques only identify traffic for which signatures are available. Maintaining an up-to-date list of signatures is a daunting task. Recent work on automatic detection of application signatures partially addresses this concern [15,20]. Second, these techniques typically employ "deep" packet inspection because solutions such as capturing only a few payload bytes are insufficient or easily defeated (See Section 4.5 for empirical evidence of this.). Deep packet inspection places significant processing and/or memory constraints on the bandwidth management tool. On our network, for example, we have observed that during peak hours, effective bandwidth is often limited by the ability of the deployed commercial packet shaping tool to process network flows. Finally, packet inspection techniques fail if the application uses encryption. Many BitTorrent clients such as Azureus, μtorrent, and BitComet already allow use of encryption.

In this paper, we propose a methodology that *classifies* (or equivalently, identifies) network flows by application using *only* flow statistics. Our methodology, based on machine learning principles, consists of two components: a *learner* and a *classifier*. The goal of the learner is to discern a mapping between flows and applications from a training data set. Subsequently, this learned mapping is used to obtain a classifier. Traditionally, learning is accomplished using a fully labeled training data set, as has been previously considered in the traffic classification context [23, 27]. Obtaining a large, representative, training data set that is fully labeled is difficult, time consuming, and expensive. On the contrary, obtaining unlabeled training flows is inexpensive.

In our work, we develop and evaluate a technique that enables us to build a traffic classifier using flow statistics from both *labeled* and *unlabeled* flows. Specifically, we build the learner using both labeled and unlabeled flows and show how unlabeled flows can be leveraged to make the traffic classification problem manageable. This *semi-supervised* [1, 6] approach to learning a network traffic classifier is one key contribution of this work. There are three main advantages to our proposed semi-supervised approach. First, fast and accurate classifiers can be obtained by training with a small number of labeled flows mixed with a large number of unlabeled flows. Second, our approach is robust and can handle both previously unseen applications and changed behaviour of existing applications. Furthermore, our approach allows iterative development of the classifier by allowing network operators the flexibility of adding unlabeled flows to enhance the classifier's performance. Third, our approach can be integrated with solutions that collect flow statistics such as Bro [25]. It may also be possible to integrate our approach with flow management solutions such as Cisco's NetFlow [5].

Our work is guided by prototype implementations of offline and realtime classification systems. Several practical considerations are pertinent in the design of our classification system including size and composition of the training data set, labeling of flows in the training data set, choice of flow statistics, expected longevity of the classifiers, and ability to detect when the system needs to relearn the flow characteristics to application mapping. Our paper provides insights to each of the above-mentioned design considerations. The remainder of this section presents a summary of our key results.

Using our prototype classification systems, we find that flow statistics can indeed be leveraged to identify, with high accuracy, a variety of different applications, including Web, P2P file sharing, email, and FTP. We obtained flow accuracies as high as 98% and byte accuracies as high as 93%. In addition, our results show that the presence of "mice" and "elephant" flows in the Internet complicates the design of classifiers, especially of those with high byte accuracy, and necessitates the use of weighted sampling techniques to obtain training flows. We find that larger training data sets consistently achieve higher classification accuracies. While larger training data sets may appear to make the task of labeling the training data set time expensive and difficult, in practice, a priori labeling of only a fraction of the training flows is sufficient.

Another distinguishing aspect of our work is the implementation of a realtime classifier in the Bro [25] Intrusion Detection System (IDS). Note that determining the application type while a flow is in progress is harder than offline identification because only partial information is available. We address this problem by designing a layered classifier that classifies flows at specific packet milestones using flow statistics that are available then.

We also consider the longevity of classifiers (i.e., how long they remain accurate in an operational network). Our experiments with long-term Internet packet traces suggests that classifiers are generally applicable over reasonably long periods of time (e.g., on the order of weeks) with retraining necessary when there are significant changes in the network usage patterns including introduction of new applications. To facilitate retraining, we present a technique for detecting retraining points. We expect this retraining point detection technique to be used with the realtime classifier such that once retraining is deemed necessary, collection of additional flows for use as training data can be automatically initiated.

The remainder of this paper is structured as follows. Related work is presented in Section 2. Section 3 presents our proposed semi-supervised classification method. To evaluate our proposal, we needed Internet packet traces for which base truth (i.e., the actual flow to application mapping) was available. Because such traces were not publicly available, we collected about a terabyte of full-packet traces that span a 6-month period and established their base truth ourselves; Section 4 describes these data sets. Section 5 evaluates the design alternatives for offline classification. Section 6 introduces and evaluates our realtime classifier. Section 7 discusses the history of the traffic classification problem, longevity, and retraining point detection. Section 8 presents conclusions and future work directions.

2. Related Work

This paper complements our recent work on traffic classification at the network core [11]. In the network core, typically only unidirectional traces are available. We developed a method to estimate the flow statistics of the unseen packets and used these estimates in conjunction with statistics from the direction seen to classify traffic. The classification framework in [11] assumed availability of a fully labeled training data set. In the current paper, we enhance the classification framework by leveraging unlabeled training data to improve the precision of our classifier, and by allowing classifiers to be incrementally built over time. In addition, we consider pragmatic issues such as selecting training data sets to obtain high byte accuracy, realtime classification, classifier longevity, and retraining of classifiers.

The work of Bernaille *et al.* [2] and Crotti *et al.* [7] use a classification framework that is similar to our previous work [11]. In [2], Bernaille *et al.* explored the potential of classifying traffic by using the sizes of the first P packets of a TCP session. Conceptually, their approach is similar to payload-based approaches that look for characteristic signatures during protocol handshakes to identify the application and is unsuccessful classifying application types with variable-length packets in their protocol handshakes such as Gnutella. Neither of these studies access the byte accuracy of their approaches which makes direct comparisons to our work difficult. Our evaluation suggests that achieving a high flow accuracy is relatively easy. The more difficult problem is obtaining a high byte accuracy as well. Our work concentrates on achieving both high flow and byte accuracy [12].

Prior studies focused on traces with limited number of applications: 10 in [2] and only 3 in [7]. In this paper, we use traces that span several months, and furthermore, we try to classify all traffic in our traces because we find that some applications with only a few flows can still contribute substantially to the amount of bytes transferred in the network (See section 5.5).

Traffic identification approaches that rely on heuristics derived from analysis of communication patterns of hosts have also been proposed [17, 18, 31]. For example, Karagiannis *et al.* developed a method that leverages the social, functional, and application behaviors of hosts to identify traffic classes [18]. Concurrent to [18], Xu *et al.* [31] developed a methodology, based on data mining and information-theoretic techniques, to discover functional and application behavioral patterns of hosts and the services used by the hosts. They subsequently use these patterns to build general traffic profiles, for example, "servers or services", "heavy hitter hosts", and "scans or exploits". In contrast, our approach uses only the characteristics of a single flow

to classify network traffic, and achieves comparable or better accuracies when classifying traffic, including traffic originating from P2P applications.

3. Classification Method

Network monitoring solutions operate on the notion of network *flows*. A flow is defined here to be as a series of packet exchanges between two hosts, identifiable by the 5-tuple (source address, source port, destination address, destination port, transport protocol), with flow termination determined by an assumed timeout or by distinct flow termination semantics. For each flow, network monitors can record statistics such as duration, bytes transferred, mean packet interarrival time, and mean packet size. This section outlines our classification method that can map flows (characterized by a vector of flow statistics) to applications (or traffic classes), with high accuracy and in realtime.

We now introduce notations and terminology to describe the problem formally. Let $\mathbf{X} = \{X_1, \cdots, X_N\}$ be a set of flows. A flow instance X_i is characterized by a vector of attribute values, $\mathbf{X_i} = \{X_{ij} | 1 \leq j \leq m\}$, where m is the number of attributes, and X_{ij} is the value of the j^{th} attribute of the i^{th} flow. In the traffic classification context, examples of attributes include flow statistics such as duration, bytes transferred, and total number of packets. The terms attributes and features are used interchangeably in the machine learning literature, and often $\mathbf{X_i}$ is referred to as a feature vector. Also, let $\mathbf{Y} = \{Y_1, \cdots, Y_q\}$ be the set of traffic classes, where q is the number of classes of interest. The Y_i's can be classes such as "HTTP", "Streaming", and "Peer-to-Peer". Our goal, therefore, is to learn a mapping from a m-dimensional variable X to Y. This mapping forms the basis for classification models, also referred to as classifiers in the machine learning literature.

Traditional learning methods of classifiers use a training data set that consists of N tuples $(\mathbf{X_i}, \mathbf{Y_i})$ and learn a mapping $f(\mathbf{X}) \rightarrow \mathbf{Y}$. The goal is to find a mapping that (correctly) generalizes to previously unseen examples. Such learning methods are referred to as *supervised* learning methods [8]. Supervised machine learning techniques have previously been applied for classifying network flows. Roughan *et al.* [27] classified flows into four predetermined traffic classes (interactive, bulk data transfer, streaming, and transactional) using the Nearest Neighbor and the Linear Discriminate Analysis classification techniques. Moore *et al.* [23] evaluated the suitability of a Naïve Bayes classifier for the Internet traffic classification problem. Recently, Williams *et al.* [30] presented a preliminary comparison of five supervised learning algorithms.

In designing our classification method, we are interested in overcoming two main challenges faced by supervised techniques:

(i) Labeled examples are scarce and difficult to obtain. With few labeled examples, traditional supervised learning methods often produce classifiers that do not generalize well to previously unseen flows.

(ii) Not all types of applications generating flows are known *a priori*, and new ones may appear over time. Traditional supervised methods force a mapping of each flow into one of q known classes, without the ability to detect new types of flows.

To address these challenges, we designed a method that combines unsupervised and supervised methods. Our classification method consists of two steps. We first employ a machine learning approach called *clustering* [8] to partition a training data set that consists of scarce labeled flows combined with abundant unlabeled flows. Clustering partitions the training data set into disjoint groups ("clusters") such that flows within a group are similar to each other whereas flows in different groups are as different as possible. Second, we use the available labeled flows to obtain a mapping from the clusters to the different known q classes (Y). This step also allows some clusters to remain unmapped, accounting for possible flows that have no known labels. The result of the learning is a set of clusters, some mapped to the different flow types. This method, referred to as *semi-supervised* learning [1, 4, 6], has received considerable attention, recently, in the machine learning community.

We note that our application of semi-supervised learning is novel in that we leave some of the clusters unlabeled. This is different from the traditional application of semi-supervised learning; in the traditional application of this approach, all classes are known a priori, and unlabeled flows are used to improve precision of the classifier. In the traffic classification problem, however, not all classes are known a priori, and thus,

we use the unlabeled clusters to represent new or unknown applications. In effect, unlabeled flows are used to improve precision and handle unknown applications. The remainder of this section discusses the details of the classification method.

3.1. *Step 1: Clustering*

The first step in training our classifier is to leverage all available training flows and group them into clusters. In the machine learning paradigm, clustering is an example of an *unsupervised* learning algorithm [8] because the partitioning of the flows in the training data is guided only by the similarity between the flows and not by any predetermined labeling of the flows. A key benefit of the unsupervised learning approach is the ability to identify hidden patterns. For example, new applications as well as changed behavior of existing applications can be identified by examining flows that form a new cluster.

Clustering algorithms use a measure $d(\mathbf{x_i}, \mathbf{x_j})$ of similarity between feature vectors $\mathbf{x_i}$ and $\mathbf{x_j}$, and find a partition that attempts to place similar examples in the same clusters, and dissimilar examples in different clusters. There are various similarity metrics that can be used. Without loss of generality, in this paper we use the Euclidean distance as the similarity measure:

$$d(\mathbf{x_i}, \mathbf{x_j}) = \left[\sum_{k=1}^{m} (x_{ik} - x_{jk})^2 \right]^{1/2}. \tag{1}$$

Clustering of Internet traffic using flow statistics has received some attention in the literature. As such, the focus of prior work [21, 32] and our own work [10] has been on demonstrating the ability of clustering algorithms to group together flows according to application type using only flow statistics and not on classifying new traffic.

There are many different clustering algorithms in the machine learning literature. We emphasize that our approach is *not* specific to any particular clustering algorithm. In our earlier work, we investigated three different clusterings algorithms K-Means, DBSCAN, and EM Clustering [10]. Although we have successfully developed classifiers based on each of these algorithms, in this paper we focus on the K-Means algorithm [8]. Several factors prompted us to select the K-Means algorithm over the other more sophisticated clustering algorithms. First, it is simple and easy to implement. This allows the classifier to have the least amount of computational overhead because the data structures representing the clusters allow fast computations of $d(\mathbf{x_i}, \mathbf{x_j})$. Second, our prior work suggests that K-Means can generate clusters that largely consist of a single application type [10]. The other clustering algorithms investigated in some cases provided more pure clusters, however, once converted into classifiers the difference in classification accuracy was negligible. Third, the more complex clustering algorithms required significantly longer learning time than K-Means (e.g., hours versus minutes). Finally, the K-Means algorithm converges to a well-understood probabilistic model: the Gauss Mixture Model [8].

The K-Means algorithm partitions the feature vectors in the training data set into a fixed number of spherical-shaped clusters by minimizing the total mean square error between feature vectors and the cluster centroids. Starting with an initial partition (random or other), the algorithm iteratively assigns each vector to the cluster whose centroid is nearest, and recalculates the centroids based on the new assignments. This process continues until membership within clusters stabilizes. The complexity of the algorithm is $O(lKn)$ where l is the number of iterations and n is the number of vectors in the training data set [8]. For the data sets tested in this paper, the algorithm converges within a few iterations.

3.2. *Step 2: Mapping Clusters to Applications*

The output of the K-Means clustering algorithm is a set of clusters, represented by their centroids, γ_k. Given a flow feature vector $\mathbf{x}$, we assign it to one of the clusters by finding the nearest centroid to $\mathbf{x}$, using:

$$C_k = \arg\min_k d(\mathbf{x}, \gamma_\mathbf{k}), \tag{2}$$

where $d(\cdot,\cdot)$ is the distance metric chosen in the clustering step. For K-Means with Euclidean distance, this step amounts to the maximum likelihood cluster assignment solution.

However, knowing to which cluster a flow feature vector most likely belongs does not provide the actual classification to one of the application types. Therefore, we need a mechanism to map the clusters found by the clustering algorithm to the different application types.

We use a probabilistic assignment to find the mapping from clusters to labels: $P(Y = y_j|C_k)$, where $j = 1, ..., q$ (q being number of application types) and $k = 1, ..., K$ (K being the number of clusters). To estimate these probabilities, we use the set of flows in our training data that are labeled to different applications $(\mathbf{x_i}, \mathbf{y_i})$, $i = 1, ..., L$, where L is the total number of different labeled applications. $P(Y = y_j|C_k)$ is then estimated by the maximum likelihood estimate, $\frac{n_{jk}}{n_k}$, where n_{jk} is the number of flows that were assigned to cluster k with label j, and n_k is the total number of (labeled) flows that were assigned to cluster k. To complete the mapping, clusters that do not have any labeled examples assigned to them are defined as "Unknown" application types, thus allowing the representation of previously unidentified application types.

Finally, the decision function for classifying a flow feature vector $\mathbf{x}$ is the maximum *a posterior* decision function:

$$y = \arg \max_{y_1,...,y_q} (P(y_j|C_k)), \tag{3}$$

where C_k is the nearest cluster to $\mathbf{x}$, as obtained from Eq. 2. Our approach uses *hard* clustering. However, labelling using *soft* clusters can easily be accommodated into our framework. For instance, the confidence of a flow's label could be based on $P(y_j|C_k)$ and labels below a certain threshold could be considered "Unknown". Exploration of soft clustering and its potential benefits is left for future work.

4. Data Sets

This section describes the data sets used in this work. Section 4.1 outlines our trace collection methodology. Section 4.2 presents high-level summary statistics of our traces. Section 4.3 describes the method used to establish the base truth of the flow to application mappings for collected traces. An overview of the data sets is provided in Section 4.4. Section 4.5 presents some empirical observations as additional motivation for our work.

4.1. *Traces and Collection Methodology*

To facilitate our work, we collected traces from the Internet link of a large university. Depending on the specific subnets traced, the collected traces are categorized as Campus, Residential, and Wireless LAN (WLAN).

Although our classification approach uses only flow statistics, application-layer information is helpful for training the classifiers and required for validating the results. Thus, we decided to collect full packet traces. An additional goal of our work is to examine traffic over an extended period of time, to assess the longevity of the classifiers.

The limited disk space available at the network monitor used for trace collection precludes the possibility of continuous full packet tracing. Thus, we collected forty-eight 1-hour traces, over a span of six months, of traffic to and from all academic units and laboratories on the campus. Specifically, we collected eight 1-hour traces each week, for five consecutive weeks in the spring of 2006 (April 6 to May 7) and also an additional week in the fall of 2006 (September 28). The traces were collected Thursdays, Fridays, Saturdays, and Sundays from 9-10 am and 9-10 pm on each of these days. Our reasoning for this collection scheme is as follows. First, we expected there to be noticeable differences in usage between the morning and evening hours. Second, we expected there to be noticeable differences in usage between work days and non-work days. Third, the collection period spanned several important transitions in the academic year: the busy final week of the semester (April 6-9), a break before final examinations (April 11-16), the final examination period

(April 17-28), the start of summer break for students (April 29-May 7), and a week in the fall semester (September 28-October 1).

Based on the above observations, we expected that our traces would capture any substantial changes that occurred in the traffic during the collection period, while substantially reducing the volume of data we needed to collect. We call this set of forty-eight traces the Campus traces; these contain Internet traffic from faculty, staff, and students. The network infrastructure uses a signature-based bandwidth management tool to actively limit all identifiable P2P traffic. In addition, user accounts are actively policed for the presence of non-academic content.

The Residential trace was collected on October 20, 2006 from midnight to 10 am from a specific set of subnets corresponding to the student residence network of the university. The student residence network is of interest because it is not actively policed. Instead, there is a "soft" limit on the bandwidth available to each user, and in addition, the total bandwidth usage of this network is limited during work hours.

The WLAN trace is a 1-hour trace, collected from the campus WLAN from 9 am to 10 am on September 28, 2006. The WLAN covers many of the buildings on campus, and is open to faculty, staff, and students.

4.2. *High-level Statistics of the Traces*

In total, 1.39 billion IP packets containing 909.2 GB of data were collected. Of this, 89.0% of the packets and 95.1% of the bytes were transferred using TCP and 10.2% of the packets and 4.7% of the bytes were transferred using UDP.

The 10-hour Residential trace contains 97.5 million IP packets and 58.3 GB of data. Of this, 85.1% of the packets and 83.2% of the bytes are TCP and 14.5% of the packets and 16.6% of the bytes are UDP. The WLAN trace contains 18.4 million IP packets and 11.6 GB of data. Of this, 95.7% of the packets and 98.3% of the bytes are TCP and 1.8% of the packets and 3.6% of the bytes are UDP.

This paper focuses exclusively on classifying TCP traffic. As discussed above, our traces also had non-TCP traffic (e.g., UDP and ICMP). There are two main reasons for our focus on TCP flows. First, TCP traffic accounts for a significant fraction of the overall traffic. Classifying this traffic accurately allows evaluation of the robustness of our approach. Second, if flow characteristics from other protocols were collected, it would likely be advantageous to have a separate classifier for the non-TCP traffic. We leave this as a useful direction for future work.

4.3. *Methodology for Establishing Base Truth*

We established base truth for the traces using an automated process that consists of payload-based signature matching, heuristics, and HTTPS identification. We used hand classification as a validation method.

The payload-based classification step uses Bro [25], which has a signature matching engine that generates a match event when the packet payload matches a regular expression specified for a particular rule. We used many of the same methods and signatures described by Sen *et al.* [28] and Karagiannis *et al.* [17], but augmented some of their P2P signatures to account for protocol changes and some new P2P applications.

Some P2P applications are now using encryption. For example, BitTorrent is using a technique called Message Stream Encryption and Protocol Encryption. To identify some of this encrypted P2P traffic, we used the following heuristic. Specifically, we maintain a lookup table of (IP address, port number) tuples from flows that have recently (i.e., within 1-hour) been identified as using P2P. If a flow is unlabeled and there is a match in our P2P lookup table, we label it as possible P2P. This mechanism works on the basis that some P2P clients use both encryption and plaintext.

We also analyzed unlabeled traffic on port 443 to establish whether or not this traffic is indeed HTTPS. This verification was done using an experimental version of Bro that has this detection capability. In addition, automated random checks were performed to determine whether or not flows labeled as HTTPS involved at least one host that was a Web server.

Table 1
Application Breakdown (Campus traces)

Class	Flows	% Flows	Bytes	% Bytes
HTTP	9,213,424	39.5%	334.4 GB	38.7%
P2P	620,692	2.7%	310.9 GB	36.0%
EMAIL	1,123,987	4.8%	42.5 GB	4.9%
FTP	23,571	0.1%	20.3 GB	2.3%
P2P Encrypted	35,620	0.2%	12.3 GB	1.4%
STREAMING	3,396	0.0%	7.4 GB	0.9%
DATABASE	3,057,362	13.1%	3.0 GB	0.3%
CHAT	26,869	0.1%	1.0 GB	0.1%
OTHER	51,298	0.2%	32.1 GB	3.7%
UNKNOWN	990,492	4.2%	70.1 GB	8.1%
UNKNOWN (443)	1,409,707	6.0%	29.7 GB	3.4%
UNKNOWN (NP)	6,765,214	29.0%	1.0 GB	0.1%
Total	23,321,632	100.0%	864.6 GB	100.0%

Table 2
Application Breakdown (Residential trace)

Class	Flows	% Flows	Bytes	% Bytes
P2P	297,781	17.6%	38.52 GB	79.3%
HTTP	118,485	7.0%	3.37 GB	6.9%
P2P Encrypted	39,943	2.4%	0.34 GB	0.7%
EMAIL	1,159	0.1%	0.12 GB	0.2%
STREAMING	29	0.0%	0.07 GB	0.1%
CHAT	1,207	0.1%	0.05 GB	0.1%
OTHER	190	0.0%	0.03 GB	0.1%
UNKNOWN	91,275	5.4%	5.88 GB	12.1%
UNKNOWN (NP)	1,135,242	67.2%	0.13 GB	0.3%
UNKNOWN (443)	4,833	0.3%	0.06 GB	0.1%
Total	1,690,144	100.0%	48.56 GB	100.0%

4.4. *Overview of the Data Sets*

Table 1 summarizes the applications found in the forty-eight 1-hour Campus traces. Application breakdowns for the 10-hour Residential trace and the 1-hour WLAN trace are shown in Table 2 and Table 3, respectively.

Over 29 different applications [1] were identified. To simplify the presentation, we group the applications by category. For example, the P2P category includes all identified P2P traffic from protocols including BitTorrent, Gnutella, and KaZaA. P2P flows identified using heuristics are labeled P2P Possible. The OTHER category constitutes various applications that were identified but did not belong to a larger group and did not account for a significant proportion of flows. The tables also list three categories of UNKNOWN flows. There are UNKNOWN (NP) flows that have no payloads. Most of these are failed TCP connections, while some are port scans. The UNKNOWNS (443) are flows on port 443; these are likely to be HTTPS traffic. The third category is simply labeled as UNKNOWN to reflect the fact that we have not identified the applications that generated this traffic. The unknown flows are not used in our analysis. General observations from these data sets follow.

On the campus network (Table 1), HTTP, DATABASE, and EMAIL traffic contribute a significant portion of the total flows. On this network, P2P contributes only 2.7% of the flows. However, P2P still accounts for a considerable portion, approximately 36%, of the bytes. In contrast, the traffic from the residential network (Table 2) exhibits comparatively less diversity in the usage of applications, with HTTP and P2P being the dominant applications. In the 10-hour Residential trace, P2P has a significant presence, both in terms of number of flows and number of bytes. We attribute this difference, somewhat speculatively, to the network use policies in place and the profile of the network users. As mentioned earlier, the campus network is used by faculty, staff, and students, and is actively regulated for non-academic content. Furthermore, the network infrastructure uses signature-based identification to severely throttle P2P traffic. In contrast, the residential network is used exclusively by students, is not actively policed, and only applies a soft limit on the bandwidth available to each user.

Table 4 shows that BitTorrent and Gnutella-based P2P applications such as BearShare, LimeWire, Morpheus, and Gnucleus are prevalent on the residential network. KaZaA was hardly seen in the traces.

[1] These application include: BB, BitTorrent, DirectConnect, eDonkey, FTP, Gnutella-based P2P programs (e.g., LimeWire, BearShare, Gnucleus, Morpheus, FreeWire), GoToMyPC, HTTP, ICQ, IDENT, IMAP, IMAP SSL, JetDirect, KaZaA, MySQL, MSSQL, MSN Messenger, MSN Web Cam, NNTP, POP3, POP3 SSL, RTSP, Samba, SIP, SMTP, SOAP, SpamAssassin, SSH, SSL, VNC, and Z3950 Client.

Table 3
Application Breakdown (WLAN trace)

Class	Flows	% Flows	Bytes	% Bytes
P2P	61,603	15.9%	6.90 GB	60.3%
HTTP	145,177	37.5%	2.94 GB	25.7%
P2P Encrypted	7,842	2.0%	0.13 GB	1.2%
CHAT	2,928	0.8%	0.05 GB	0.5%
EMAIL	695	0.2%	0.02 GB	0.1%
FTP	157	0.0%	0.00 GB	0.0%
STREAMING	13	0.0%	0.00 GB	0.0%
OTHER	374	0.1%	0.01 GB	0.1%
UNKNOWN	16,100	4.2%	1.16 GB	10.1%
UNKNOWN (443)	8,581	2.2%	0.22 GB	2.0%
UNKNOWN (NP)	143,631	37.1%	0.02 GB	0.1%
Total	387,101	100.0%	11.4 GB	100.0%

Table 4
P2P Breakdown (Residential Trace)

Application	Flows	% Flows	Bytes	% Bytes
BitTorrent	286,794	96.3%	22.00 GB	57.1%
Gnutella-based	10,066	3.4%	16.47 GB	42.7%
eDonkey	921	0.3%	0.05 GB	1.4%
Other	161	0.1%	0.01 GB	0.4%
Total	297,942	100.0%	38.5 GB	100.0%

Table 5
P2P Port Usage (Residential Trace)

Application	Non-Standard Port (% Flows)	Non-Standard Port (% Bytes)
BitTorrent	91.7%	84.0%
Gnutella-based	82.1%	99.1%
eDonkey/eMule	89.1%	99.0%

4.5. *Empirical Motivation for Our Work*

We supplement our trace data analysis with three empirical observations that further motivate our traffic classification work. These observations concern port numbers, pad bytes, and encryption.

Table 5 shows that use of non-standard ports is prevalent [2]. Approximately 92% of the BitTorrent flows used non-standard ports. This contrasts starkly with the study by Sen *et al.* [28] in 2004 where they found only 1% of the BitTorrent flows used non-standard ports. This provides further evidence on the declining effectiveness of port-based classification.

Figure 1 shows the empirical distribution of variable-length padding observed in Gnutella before the characteristic payload signature was found. We found that collecting only the initial 64 bytes of the payload bytes will allow approximately only 25% of the Gnutella flows to be identified using payload signatures. Over 400 payload bytes of each packet would need to be captured to increase the number identified to 90%. Furthermore, an application could easily make the length greater than 400 bytes if it helped avoid detection.

Finally, our base truth establishment process indicates the presence of encrypted traffic, most of which is likely to be from P2P applications. We have labeled these as P2P Encrypted in Tables 1-3. We believe that as P2P applications evolve, encryption will become the norm, and in that case, packet inspection techniques will likely fail.

5. Offline Classification

5.1. *Design Considerations*

We implemented a prototype offline classification system, incorporating both steps of the classification methodology, in approximately 3,000 lines of C++ code. In this section, we discuss the design considerations that affect the performance of the classifier. The design considerations are:

(i) Composition of the training data set: There are two related considerations, the fraction of the training flows that are labeled, and the methodology used to select flows for the training which are discussed in Sections 5.2 and 5.3, respectively. Unless stated otherwise, we assume that all training flows are labeled.

(ii) The features used to characterize the flows: Feature selection is discussed in Section 5.4.

[2] Default port numbers used were BitTorrent (6881-6889,32459), Gnutella-based (6346), eDonkey/eMule (4661,4662,4711,4712).

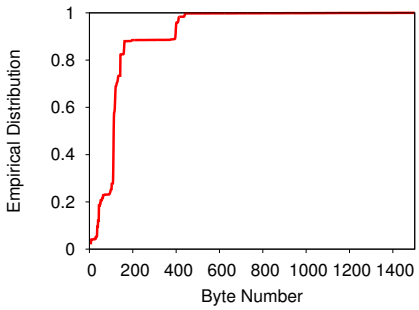


Fig. 1. Variable-length Padding in Gnutella

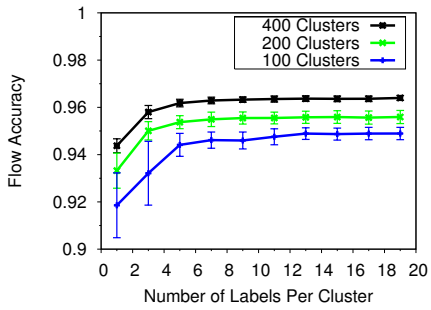


Fig. 2. Impact of Selective Labeling of Flows after Clustering

(iii) The number of clusters K generated in the clustering step of the classification method: This parameter can be used to tune our classifier to achieve better accuracy, however, at the cost of additional computation for the classifier. Unless stated otherwise, we assume $K = 400$. We explore this factor in Section 5.5.

Our primary performance metrics are flow and byte accuracy. Flow accuracy is the number of correctly classified flows to the total number of flows in a trace. Byte accuracy is the number of correctly classified bytes to the total number of bytes in the trace. In our results, we report for a given test data set the average results and the 95% confidence interval from 10 runs each with a different training set of feature vectors. Unless stated otherwise, the training data set is selected from the test data set used for evaluation. In all our experiments the test data set is a factor of 10 to 100 larger than the training data set.

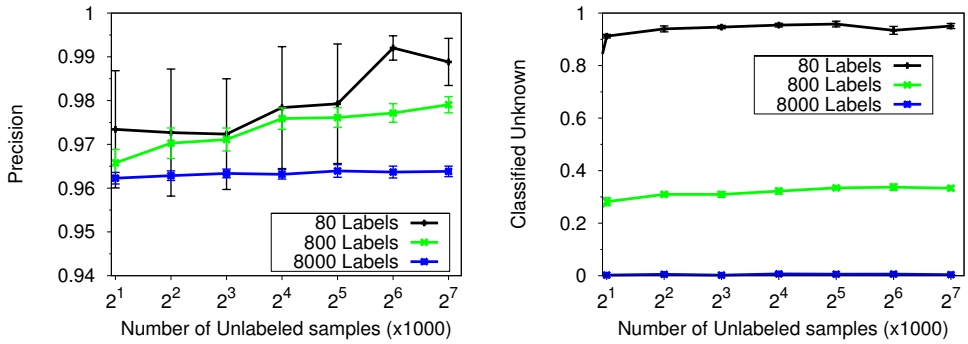
5.2. *Semi-Supervised Learning*

Labeling of training feature vectors is one of the most time-consuming steps of any machine-learned classification process, especially because many Internet application purposefully try to circumvent detection. We expect a vendor to achieve labeling of flows using a variety of orthogonal approaches, including payload analysis, port-based analysis, experimentation, expert knowledge, or a combination thereof. Clearly, it is an advantage if high classification accuracy is achieved by labeling only a small number of flows.

Recall that our approach allows clustering to use both labeled and unlabeled training flows, and then relies on only the labeled flows to map clusters to applications. This semi-supervised approach to training the classifier leverages the fact that clustering attempts to form disjoint groups, wherein each group consists of objects that bear a strong similarity to each other. Thus, the hypothesis is that if a few flows are labeled in each cluster, we have a reasonable basis for creating the cluster to application type mapping.

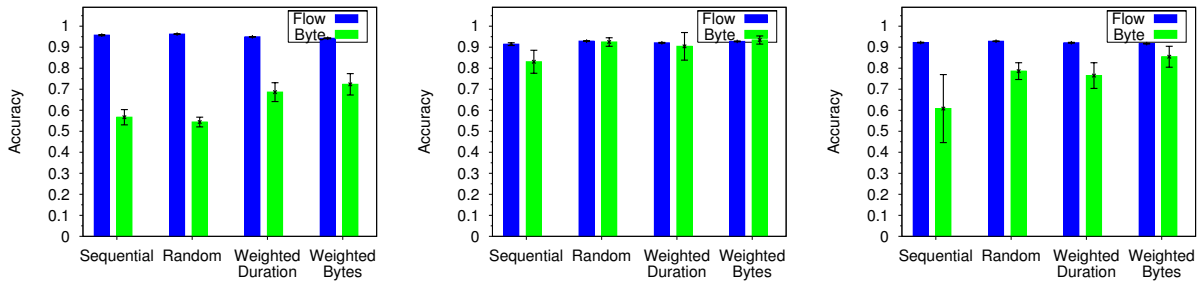
To test the aforementioned hypothesis, we conducted a number of experiments. The first set of experiment considers the possibility of the entire training data set being unlabeled. In this case, we can selectively label a few flows from each cluster and use these labeled flows as the basis for mapping clusters to applications. The hypothesis here is that the clustering step produces "pure" (in the sense of application types) clusters; in our earlier work [10], we provided empirical evidence of this hypothesis. Figure 2 presents results from this experiment. We assume that we are provided with 64,000 unlabeled flows. Once these flows are clustered we randomly label a fixed number of flows in each cluster. Interestingly, the results show that with as few as two labeled flows per cluster and $K = 400$, we can attain 94% flow accuracy. The increase in classification accuracy is marginal once five or more flows are labeled per cluster.

For the second set of experiments, results of which are shown in Figure 3, we utilized 80, 800, and 8,000 labeled flows, and mixed these labeled flows with varying numbers of unlabeled flows to generate the training data set. Both labeled and unlabeled flows were randomly chosen from the April 6, 9 am Campus trace. These training flows were used to learn the flow to application mapping, with $K = 400$ in the clustering step, and we tested the resulting classifier on the same Campus trace. Note that there are 966,000 flows in this trace.



(a) Precision (b) Fraction labeled as Unknown

Fig. 3. Impact of Training with a Mix of Labeled and Unlabeled Flows



(a) April 6, 9 am Campus trace (b) Residential trace (c) WLAN trace

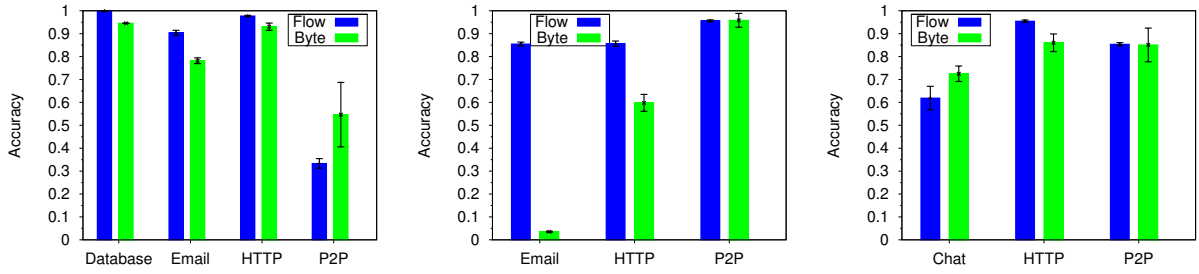
Fig. 4. Impact of Sampling Methodology on Classification Accuracy

Figure 3 reports the precision of the classifier. Precision is defined as the number of correctly labeled flows to the total number of labeled flows, with those labeled "unknown" excluded from the calculation. We observe that for a fixed number of labeled training flows, increasing the number of unlabeled training flows increases our precision. This is an important empirical result because unlabeled flows are relatively inexpensive to obtain and the penalty for incorrect labeling of a flow might be high (e.g., assigning lower priority to business critical traffic). Thus, by simply using a large sample of unlabeled flows, the precision rate can be substantially increased. This experiment further demonstrates the potential of the semi-supervised learning method.

The semi-supervised classifier makes it possible to start with a few labeled flows, and over time incrementally label more training flows so as to improve the classification performance. The results in Figure 3 show that even when a very small fraction of flows are labeled the precision of the classifier remains high. As additional labels are added, the precision remains high, albeit decreasing slightly, but has the accompanying effect of significantly reducing the amount classified as unknown. Further reductions in "unknown" classifications can be hastened by "cherry picking" which flows to label; specifically, obtaining a few labels corresponding to highly used clusters can substantially reduce the number of unknowns.

5.3. *The Dichotomy of Elephant and Mice Flows*

The presence of elephant and mice flows in Internet traffic is well documented (see [24] and the references therein). According to this phenomena, a majority of the Internet flows are small-size "mice" flows and only a small fraction are large-sized "elephant" flows; however, the mice flows account for only a small percentage of the total number of packets and bytes transmitted on the Internet. Without proper representation of both types of flows in the training data set, we run the risk of producing a classifier that may, for example, have a



(a) Campus trace (b) Residential trace (c) WLAN trace

Fig. 5. Classification Accuracy by Application

high flow accuracy but a low byte accuracy [12]. In the machine learning literature, this problem is referred to as the *class imbalnce* problem. In this section, we investigate how sampling methodology influences the selection of both elephant and mice flows in the training data set and helps address the class imbalance problem. Other potential techniques for addressing this problem are discussed elsewhere [12].

We considered both sequential and random sampling techniques. All experiements in this section are with training data set of 8,000 labeled flows. For sequential sampling, we generated each of the ten training data sets needed for the experiments by randomly picking a point to begin sequential selection of flows. Along with simple random sampling, we also considered weighted random sampling techniques that bias selection of samples according to the transfer size of a flow or according to the duration of a flow. Our weighted sampling policy takes 50% of the flows from below and 50% of the flows from above the 95^{th} percentile of the flow transfer sizes or of the flow durations for the weighted bytes and duration policies, respectively. We believe this weighted scheme allows additional clusters to be formed to better represent elephant flows.

Figure 4 shows classification results from a single Campus trace (April 6, 9 am Campus trace), the Residential trace, and the WLAN trace. We observe very high flow accuracies, in excess of 95% with the Campus traces and around 90% with the Residential and WLAN traces, irrespective of the sampling technique. However, the corresponding byte accuracies are lower and they vary across the traces and with the sampling strategy. Depending on the sampling strategy, byte accuracies between 50% and 85% are attained with the Campus traces, whereas byte accuracies between 80% and 93% and between 60% and 85% are obtained for the Residential and WLAN traces, respectively.

Our experiments and the results in Figure 4 also show that sequential sampling for selecting training flows performs poorly in comparison to the random and weighted random sampling techniques. For example, in the WLAN trace, on average, byte accuracy of 61% is achieved with sequential sampling whereas byte accuracy of 86% is achieved with weighted byte sampling. The weighted byte sampling technique results in a 41% improvement of the byte accuracy compared to that with sequential sampling. Similar improvements in byte accuracies are observed in experiments with the remaining forty-seven Campus traces. The byte accuracies with the Residential trace are generally higher; yet, a modest improvement of 13% can be achieved by switching from sequential to weighted byte sampling. In general, the weighted bytes sampling technique achieves the best byte accuracies when classifying traffic. We attribute this improved classification performance to the increased probability of forming more representative clusters for infrequently occurring elephant flows. Finally, it is worth noting that the large improvement in byte accuracy is possible with only a marginal reduction in flow accuracy.

We conclude this section with a discussion of classification accuracy by application type. Figure 5 shows the classification accuracies for applications that contribute at least 0.5% of the flows or bytes in the traces. The results are from the weighted byte sampling experiments shown in Figures 4. Overall, our approach is able to classify any type of traffic, including P2P traffic, provided there are enough samples in the training data set from which the mapping between flows and applications may be learned. For the Campus trace considered (Figure 5(a)), we find that the classification accuracy for P2P traffic is lower than that for other traffic because P2P flows account for only a small percentage, typically less than 3% of the total flows, and therefore, our sampling techniques are unable to capture enough of the P2P dynamics to learn the

flow to application mapping. It is the misclassification of P2P flows that results in the overall lower byte accuracy seen in Figure 4(a). As can be seen in Table 1, P2P accounts for a small fraction of the flows but a large fraction of the total bytes. When P2P is prominent (Figures 5(b) and (c)), as in the WLAN and the Residential traces, we achieve flow and byte accuracies near 90% for this type of traffic.

5.4. *Feature Selection*

Another important design choice in training our classifiers is the set of features used in the classifier. Many flow statistics (or features) can be calculated from a flow; however, not all features provide good discrimination between the classes. Using such features can decrease the accuracy of the classifier. We started with 25 candidate features. To find a subset of discriminating features we employ a feature selection method. In general, the feature selection task is exponentially hard; however, efficient methods for feature selection are widely used [14].

In our work, we employ the backward greedy feature selection method [14]. The method works as follows. Given n features, we train a classifier with all features and compute its accuracy. We then find the single feature to remove such that the classifier with $n-1$ features has the highest accuracy. This process is continued until we find the maximum number of features to remove such that the resultant classifier has the best accuracy.

To choose a subset of features to use in all of our experiments, we perform the backward greedy search with the various data sets. We then find which subset of the features were chosen most often in the different experiments. The eleven flow features that were chosen are: total number of packets, average packet size, total bytes, total header (transport plus network layer) bytes, number of caller to callee packets, total caller to callee bytes, total caller to callee payload bytes, total caller to callee header bytes, number of callee to caller packets, total callee to caller payload bytes, and total callee to caller header bytes. In the rest of the paper we use this set of features as a basis for our classifiers [3] .

Interestingly, we found that flow features that have a time component such as duration, interarrival time, and flow throughput were found not to be useful by the feature selection algorithm. In general, selection of time-oriented features should be avoided as they are less likely to be invariant across different networks.

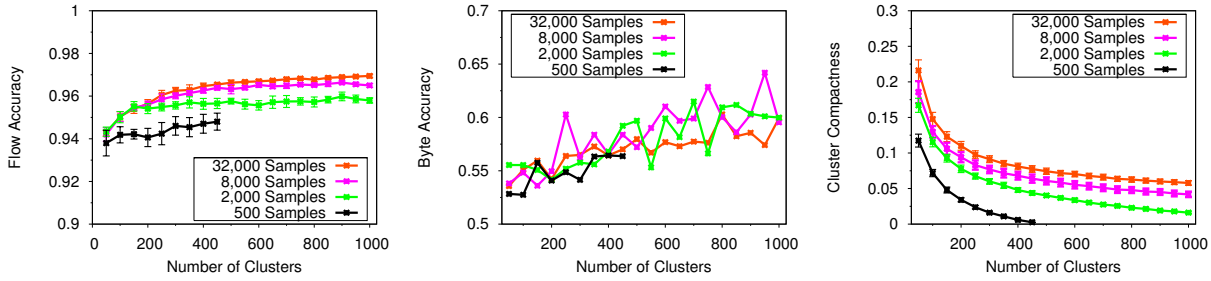
Internet flow features, in general, exhibit a high degree of skewness [26]. We found it necessary to transform the flow features to obtain higher classification accuracies. Experimentation with several commonly used transforms indicated that logarithmic transformations yield the best results. In general, transformation of features is often necessary in most machine learning applications.

5.5. *Tuning the Classifier*

The number of clusters (K) impacts the quality of clustering (and thus the quality of classification), the time complexity of building the classifier, and the runtime performance of the classifier. To determine a suitable K, we varied both the number of clusters and the number of labeled training flows. Figure 6 shows the results from experiments where we varied K from 50 to 1,000, and varied the number of vectors in the training data sets from 500 to 32,000 flows. The training flows were selected using a simple random sampling (See Section 5.4.).

Several observations can be made from the flow accuracy results in Figure 6(a). First, flow accuracies in excess of 95% are achieved when using training data sets with 2,000 or more labeled flows. Second, although having more flows in the training data set improves flow accuracy, the percentage improvement shows diminishing returns. Third, as K increases, we observe that the flow accuracy also increases. For example, for training data sets with 8,000 or more flows, a large K ($\geq 4,000$) can facilitate flow accuracies

[3] Caller is the host that initiates a flow (e.g., the host that sends the SYN packet during TCP connection establishment); callee is the host that reacts to the initiation request (e.g., the host that responds with a SYNACK packet during TCP connection establishment).



(a) Flow Accuracy (b) Byte Accuracy (c) Cluster Compactness

Fig. 6. Parameterizing the Classification System (April 6, 9 am Campus trace)

around 97.5%. However, having such large values for K is not practical as this increases the time complexity of the classification step.

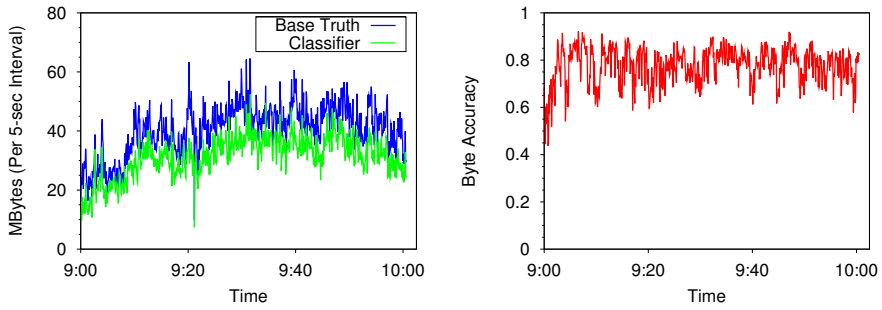
Figure 6(b) shows the byte accuracy results. The byte accuracies, on average, ranged from 52% to 62%. We did not find any clear relationship between number of flows in the training data set and the corresponding byte accuracy. Byte accuracy is very sensitive to a few large "elephant" flows in network traffic. In general, a simple random selection of training flows from the traces is unlikely to capture enough elephant flows in the training data sets, especially because the training data sets consist only of a few thousand flows. For example, there are 58 FTP data transfers that account for 6.5% of the bytes in the April 6, 9 am Campus trace, and these are rarely captured in the (randomly chosen) training data set. Thus, these large FTP flows are typically misclassified. Increasing the number of clusters K typically improves byte accuracy, albeit marginally, because the likelihood of forming clusters for the elephant flows when they are selected in the training data set increases. The use of more sophisticated sampling techniques such as weighted bytes policy (discussed in Section 5.3) can substantially improve the byte accuracies. Another solution we found for classifying "rare" applications of interest is to specifically add flows of this type to the training data set. This makes it possible for the classifier to have a clusters representing this application as well.

Figure 6(c) shows *cluster compactness* [16]. Cluster compactness measures the degree of homogeneity within the clusters formed; a low compactness measure indicates more homogeneity among flows in the clusters. Clearly, if each flow in the training set is assigned its own independent cluster, then cluster compactness will reach zero. We see this trend in the graph wherein the larger K becomes, the lower compactness becomes. However, we also see a plateau effect for $K \geq 400$, wherein compactness decreases slowly with increases in K.

Choosing parameter values for the clustering step presents a tradeoff between accuracy and classification overhead. Our results show that a larger training data set improves the flow accuracy, and a larger K improves flow accuracy, byte accuracy, and cluster compactness. A large value for K, however, increases the classification overhead and some caution must be emphasized when choosing K. Because our semi-supervised learning does not require all flows to be labeled, we advocate using a large training data set with as many labeled flows as possible, and a K value that achieves the desired tradeoff between accuracy and computation overhead. Essentially, the size of the training data set and the value for K are tuning parameters that can be adjusted depending upon the application.

6. Realtime Classification

In this section we discuss the design, implementation, and performance of a prototype realtime classification system we developed using our proposed classification framework.



(a) Bytes Classified (b) Byte Accuracy

Fig. 7. Performance of Realtime Classifier

6.1. *Design Considerations*

A fundamental challenge in the design of the realtime classification system is to classify a flow as soon as possible. Unlike offline classification where all discriminating flow statistics are available *a priori*, in the realtime context we only have partial information on the flow statistics.

We address this challenge by designing a layered classification system. Our layers are based upon the idea of *packet milestones*. A packet milestone is reached when the count of the total number of packets a flow has sent or received reaches a specific value. We include the SYN/SYNACK packets in the count. Each layer is an independent model that classifies ongoing flows into one of the many class types using the flow statistics available at the chosen milestone. Each milestone's classification model is trained using flows that have reached each specific packet milestone.

To classify flows in realtime we track the flow statistics of each ongoing flow. When a flow reaches the first packet milestone, it is classified using the first layer's classification model. When the flow reaches further packet milestones it is then reclassified using the appropriate layer's model. When a flow is reclassified, any previously assigned labels are disregarded.

This layered approach allows us to revise and potentially improve the classification of flows. The memory overhead of our approach is linear with respect to the number of flows because we use the same feature set at all layers.

An alternative approach would be to classify at points that are significant in the transport-layer protocol. For example, the first layer could classify with just the transport protocol and port number when the very first packet is seen. For TCP connections, the next layer could be when the first data packet is seen (i.e., following the connection establishment phase). We defer this approach for future work.

Our prototype system was built using an existing IDS system called Bro [25]. Bro is an ideal candidate for our prototyping effort because by design it performs the realtime analysis of network traffic. We added two scripts to Bro 0.9a (unmodified) to enable our realtime classifier. The first script tracks the flow feature set. When a flow reaches a specific packet milestone, the script calls a classification function in our second Bro script. The second Bro script contains a classification function for each specific milestone at which we reclassify our flows. This second Bro script was generated by a C++ program that reads in the training flows and generates the mapping from flows to applications. We use the same features as in Section 5 with one obvious exception; we do not use *total number of packets*.

6.2. *Classification Results*

For these experiments, we trained the classifier using flows from the April 6, 9 am trace with 966,000 flows. For each of N layers we created models using 8,000 training flows, using $K = 400$. In our implementation, we use thirteen layers and separate our packet milestones exponentially $(8, 16, 32, \cdots)$. For layers eleven and higher (packet milestones greater than 4,096), fewer than 5% of flows in the trace reached these milestones.

Table 6
Realtime byte accuracy with number of layers varied

Layer	Packet Milestone	Byte Accuracy
1	8	40.0 %
2	16	45.8 %
3	32	48.9 %
5	128	49.5 %
10	4096	49.7 %
13	16384	77.5 %

Therefore, for these layers we trained with all available flows in the trace (always more than 500). We do not test our model on the same trace from which we generated the training data to avoid biasing our results.

We calculated realtime byte accuracy as follows. When a packet arrives for a given flow we use the current label assigned by our classifier to determine if the bytes for this packet have been correctly classified. Byte accuracy in a given time interval is simply the fraction of bytes that were assigned the correct labels. Note that the system may reclassify a flow several times and could therefore assign multiple labels to the flow during its lifetime. Thus, we report only byte accuracy in a moving time window.

Figure 7 presents example results by using the April 13, 9 am trace (our largest 1-hour Campus trace). We see that the classifier performs well with byte accuracies typically in the 70 to 90% range. Quantitatively similar results were obtained when tested on the other traces.

Another aspect we considered was the effect of adding additional layers to our classification system. For the April 13, 9 am trace shown in Table 6, 78% of the flows had correct labels after classification at the first layer (8 packets). If this were the only layer used in our system, this would result in 40% of the bytes being correctly classified. This low value occurs because many of the elephant flows are incorrectly classified at the early stage. Using five layers improves the byte accuracy to 50%. Finally, with thirteen layers, byte accuracy reaches 78% as we are correctly classifying the elephant flows. We also note that the last label given to a flow is correct 82% of the time.

Some of the intermediate layers appear to provide little or no improvement in byte accuracy. These additional layers can be removed and still allow our classification system to achieve similar byte accuracies while reducing overhead.

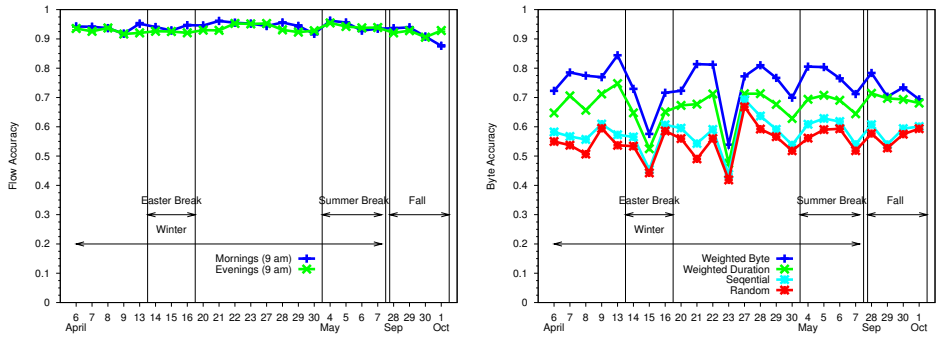
7. Discussion

In this section we discuss three topics: the *arms race* occurring between network operators and users/application developers (Section 7.1), the longevity of our classifier (Section 7.2), and the ability of our methodology to determine when retraining is required (Section 7.3).

7.1. *The Classification Arms Race*

To fully comprehend the traffic classification problem, one needs to understand its history. For many years, traffic classification was trivial, as applications tended to abide by well-known port numbers. Application developers had little motivation to deviate from this. Over time though, things changed; network bandwidths increased, new applications emerged, and the Internet became available to a much larger audience. In the late 1990's, the exchange of high fidelity music (and later video) became feasible and accessible to a large audience. The increased bandwidth consumption contributed to the creation of the traffic classification problem.

What ensued can best be described as an arms race involving at least four parties - content owners, ISPs, users, and application developers. The race started slowly. Initially ISPs could identify these file sharing applications using well known ports. The ISPs could then control or block the offending applications. In



(a) Flow accuracy for all Campus traces (b) Byte accuracy for 9 am Campus traces

Fig. 8. Longevity of Classifier

September 2002 KaZaA escalated the race by introducing dynamic ports, effectively bypassing blocked ports. Since that time, the two sides have gone back and forth numerous times.

One important observation is that file sharing users have little loyalty to the applications. If an application is blocked or impeded by an ISP, users will quickly migrate to an application that can provide them with access to the content they want. It is, therefore, important for a traffic classifier to overcome current countermeasures, and also be able to function with the countermeasures that may come in the future. For example, encryption is currently not widely used by file sharing applications, even though some of these applications already support it. If required, users could easily start encrypting their traffic. This would immediately prevent content-based classifiers from properly identifying file-sharing traffic.

We believe our classifier based on flow statistics will be difficult to circumvent. This is because it is very hard for applications to disguise their behavior without adding large amounts of overhead. Consider the *average packet size* feature. To disguise this feature, an application would need to modify flows so the *average packet size* across all its flows appear random. This would involve adding significant overhead because sometimes either padding would need to be added to packets to increase the packet size or full packets broken up into several smaller packets when sent. Similarly, changing the ratio of data sent between hosts could also require substantial amounts of extra data transfer. Ultimately, to defeat the classifier the overhead required would be crippling. Nevertheless, if new applications originate or old applications change behaviour, we would like the classification system to adapt accordingly.

In light of the above discussion, we can identify (at least, to first order) two important considerations. One, a classification system should be robust and be able to maintain high classification accuracy in the presence of transient changes in network/application usage patterns; our hope would be that classifiers have a reasonably long shelf life. Two, when there are substantial changes, for example, owing to introduction of new applications, or owing to behavioral changes of existing applications, the classifier should automatically detect the need for retraining; our intent in this case is to keep up with the arms race. These two issues are further discussed in Sections 7.2 and 7.3, respectively.

7.2. *Longevity*

To experimentally evaluate the long-term predictive value of classifiers, we tested the classifiers that were built by sampling from the April 6, 9 am Campus trace (see Section 5.3) across the forty-eight Campus traces. Figure 8 presents sample results from our experiments.

Figure 8 (a) shows the classification accuracy as a function of time. The results shown are for classifiers trained using labeled flows sampled by the weighted bytes technique. Qualitatively similar results were obtained for other sampling techniques (we do not show them on the graph to avoid line "crowding"). Our results show that the classifier retained a high flow accuracy throughout the 6-month period. Flow accuracies close to 95% are consistently achieved in the traces we tested, including major transitions such as end of winter semester, summer break, and beginning of fall semester. For example, the student population

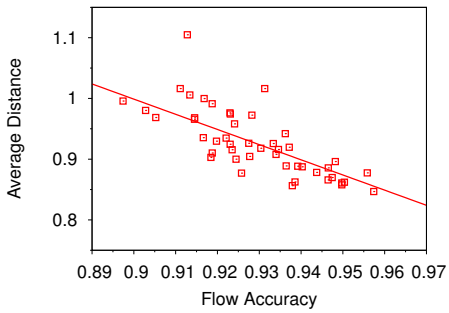


Fig. 9. Correlation between Average Distance and Flow Accuracy

substantially dwindles during the summer months. Also, during the summer months, the number of Database flows (MSSQL) substantially increased from the 5% originally seen in the training data sets to over 25% during during this period. However, our classifier is still able to classify the new database traffic correctly. There is no substantial loss in classification accuracy.

In Figure 8 (b), we present the byte classification accuracies for the 9 am Campus traces. The results for the 9 pm Campus traces are qualitatively similar. The byte accuracy trend is similar to the flow accuracy trend but shows more variability. We also find that the weighted bytes approach for selecting training flows consistently achieves higher accuracies than the random and sequential selection techniques because more P2P traffic is successfully classified by the former. We further investigated why the byte accuracy drops significantly on April 15 and April 23. The drop in byte accuracy was due to misclassification of FTP flows as either P2P or HTTP. In general, FTP is not captured well by any of the sampling techniques because it accounts for only a small fraction ($< 0.01\%$) of the flows, and thus, is unlikely to be captured in a small-sized training data set. Typically, FTP accounts for less than 5% of the bytes but on those days it accounted for 21.6% and 26.6% of the bytes, respectively.

7.3. *Retraining*

The results above show that our classifiers remained fairly robust over time and for different traces. While encouraging, a mechanism for updating the classifiers is still required. An update of the classifier can be in the form of re-clustering, re-labeling of clusters, or both. The ideal way to determine if an update is required is to track and measure the classification accuracy as new flows are classified. However, measuring the accuracy is not possible, as the flow labels are not known. There are, however, two indirect measures for measuring reliability of the classifiers. The first is to build classifiers using a mix of labeled and unlabeled flows, as discussed in Section 5.2. Then, we can track the number of flows that are not assigned any label. If this number increases, it indicates the need for labeling some of those unknown flows so that their corresponding clusters are also labeled. The semi-supervised approach makes it possible over time that this type of an update would capture under-represented flow types and allow the accuracy of the classifier to improve.

Alternatively, a statistical measure could be used to detect changes in the quality of the clustering model. We propose using the average distance of new flows to their nearest cluster mean; a significant increase in the average distance indicates the need for an update. Formally, this measure corresponds to the likelihood function of the clustering model in representing the new flows. The measure is easy to compute and track, as it does not require knowledge of the flow labels. While an indirect measure of accuracy, the *clustering likelihood* measure is correlated to the accuracy of the classifier. Recall from Section 3 that new flows are mapped to clusters using the distance metric as a measure of similarity. Thus, it is expected that average distance between flows and cluster centers is negatively correlated with accuracy.

Figure 9 shows a scatter plot of flow accuracy and average distance for all forty-eight Campus traces for one of the classifiers used in Figure 8. These sample results show that when the average distance to the cluster centers is higher, the flow accuracies are typically lower, and vice versa. We repeated the above test for the 9 remaining weighted bytes classifiers we built by sampling from the April 6, 9 am Campus trace

and found similar results. The correlation between average distance and accuracy ranged from -0.57 to -0.75 in the models we tested; the average correlation was -0.69.

In practice, the clustering likelihood can be easily used as an indicator of when our classification models need to be retrained. As previously demonstrated, the classification model is fairly robust and would not need to be retrained frequently. The average distance could be recorded for large intervals such as on an hourly or a daily basis. The average distance obtained during the interval just after retraining could be used as a baseline as this most likely is when the model is most accurate. If the hourly or daily average distance increases, and stays generally above a certain threshold (e.g., 50% above the baseline), then this may be treated as an indicator for retraining. The detection threshold can be adjusted to accommodate different amounts of variation in flow accuracy.

Once the need for retraining is detected there are various approaches to retraining that can be employed to update the classification model besides the simple and extreme one of retraining the models completely from "scratch" using new labeled and unlabeled flows. While we do not evaluate these approaches, we note some approaches to retraining that do not require completely rebuilding the model. One approach is to create new clusters using new flows that were far from their means. This would be followed by selectively querying the labels of flows from these uncertain clusters. In the machine learning literature, this is known as *active learning*. Another approach is to sample new flows and randomly replace only a fraction of the existing flows in the training data set and then rebuild the classifier.

8. Conclusions and Future Work

This paper proposed and evaluated a semi-supervised framework for classifying network traffic using only flow statistics. A key advantage of the semi-supervised approach is the ability to accommodate both known and unknown flows during development of the classifier. We show that our technique can achieve high classification accuracy by using only a few labeled examples in combination with many unlabeled examples. Our results show that: 1) both high flow and byte accuracy can be achieved; 2) a variety of applications, including P2P, HTTP, FTP, and email can be successfully classified; and, 3) robust classifiers can be built that are immune to transient changes in network conditions. Furthermore, to facilitate automatic detection of non-transient changes such as introduction of new applications or behavioral changes to existing applications, we propose a retraining point detection mechanism. A salient feature of our work is the development of working prototypes.

We believe that generic classifiers based on flow statistics can be developed. A vendor may train the classifier using a mix of labeled and unlabeled flows, where labeled flows may be obtained from operational or test networks. Our retraining point detection enables the possibility of discovery of network specific unknowns; these may be, at the discretion of the operator, delivered to the vendor for labeling. As and when required, vendors may distribute new classifiers along with a set of labeled flow feature vectors.

Several opportunities exist for future work. We outlined several retraining approaches. Evaluation of these can also be considered in future work. Furthermore, our approach can leverage recent work on techniques for flow sampling and estimation (e.g., [9, 13, 19]). The evaluation of the effectiveness of sampling and estimation techniques can also be interesting avenues for future work. Finally, we hope our research will stimulate researchers to consider other clustering and classification approaches.

9. Acknowledgements

Financial support for this research was provided by Informatics Circle of Research Excellence (*i*CORE) in the Province of Alberta, as well as by Canada's Natural Sciences and Engineering Research Council (NSERC).

References

[1] S. Basu, M. Bilenko, and R. Mooney. A Probabilistic Framework for Semi-Supervised Clustering. In *Proc. KDD '04*, Seattle, USA, August 2004.

[2] L. Bernaille, R. Teixeira, and K. Salamatian. Early Application Identification. In *CoNEXT'06*, Lisboa, Portugal, December 2006.

[3] Cache Logic. Peer-to-Peer in 2005, http://www.cachelogic.com/home/pages/research/p2p2005.php, 2005.

[4] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, USA, 2006.

[5] Cisco NetFlow. http://www.cisco.com/warp/public/732/tech/netflow.

[6] F. Cozman, I. Cohen, and M. Cirelo. Semi-Supervised Learning of Mixture Models. In *ICML'03*, Washington, USA, August 2003.

[7] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic Classification through Simple Statistical Fingerprinting. *Computer Communications Review*, 37(1):7–16, 2007.

[8] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, second edition, 2001.

[9] N. Duffield, C. Lund, and M. Thorup. Flow Sampling Under Hard Resource Constraints. In *SIGMETRICS'04*, New York, USA, June 2004.

[10] J. Erman, M. Arlitt, and A. Mahanti. Traffic Classification using Clustering Algorithms. In *SIGCOMM'06 MineNet Workshop*, Pisa, Italy, September 2006.

[11] J. Erman, M. Arlitt, A. Mahanti, and C. Williamson. Identifying and Discriminating Between Web and Peer-to-Peer Traffic in the Network Core. In *WWW'07*, Banff, Canada, May 2007.

[12] J. Erman, A. Mahanti, and M. Arlitt. Byte Me: The Case for Byte Accuracy in Traffic Classification. In *SIGMETRICS'07 MineNet Workshop*, San Diego, USA, June 2007.

[13] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a Better NetFlow. In *SIGCOMM '04*, Portland, USA, August 2004.

[14] I. Guyon and A. Elisseef. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, pages 1157–1182, 2003.

[15] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. ACAS: Automated Construction of Application Signatures. In *SIGCOMM'05 MineNet Workshop*, Philadelphia, USA, August 2005.

[16] J. He, A.-H. Tan, C. L. Tan, and S. Y. Sung. *Clustering and Information Retrieval*, chapter On Quantitative Evaluation of Clustering Systems, pages 105–134. Kluwer, 2003.

[17] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy. Transport Layer Identification of P2P Traffic. In *IMC'04*, Taormina, Italy, October 2004.

[18] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. In *SIGCOMM'05*, Philadelphia, USA, August 2005.

[19] R. Kompella and C. Estan. The Power of Slicing in Internet Flow Measurement. In *IMC'05*, Berkeley, USA, October 2005.

[20] J. Ma, K. Levchenko, C. Krebich, S. Savage, and G. Voelker. Unexpected Means of Protocol Inference. In *IMC'06*, Rio de Janeiro, Brasil, October 2006.

[21] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow Clustering Using Machine Learning Techniques. In *PAM'04*, Antibes Juan-les-Pins, France, April 2004.

[22] A. Moore and K. Papagiannaki. Toward the Accurate Identification of Network Applications. In *PAM'05*, Boston, USA, March 2005.

[23] A. W. Moore and D. Zuev. Internet Traffic Classification Using Bayesian Analysis Techniques. In *SIGMETRIC'05*, Banff, Canada, June 2005.

[24] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto. Identifying Elephant Flows through Periodically Sampled Packets. In *IMC '04*, Taormina, Italy, October 2004.

[25] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Comput. Networks*, 31(23-24):2435–2463, 1999.

[26] V. Paxson. Empirically-Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, August 1998.

[27] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-Service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification. In *IMC'04*, Taormina, Italy, October 2004.

[28] S. Sen, O. Spatscheck, and D. Wang. Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures. In *WWW'04*, New York, USA, May 2004.

[29] S. Sen and J. Wang. Analyzing Peer-to-Peer Traffic across Large Networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232, 2004.

[30] N. Williams, S. Zander, and G. Armitrage. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *Computer Communication Review*, 30:5–16, October 2006.

[31] K. Xu, Z.-L. Zhang, and S. Bhattacharyya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *SIGCOMM '05*, Philadelphia, USA, August 2005.

[32] S. Zander, T. Nguyen, and G. Armitage. Automated Traffic Classification and Application Identification using Machine Learning. In *LCN'05*, Sydney, Australia, November 2005.