Identifying and Discriminating Between Web and Peer-to-Peer Traffic in the Network Core

Jeffrey Erman Anirban Mahanti Martin Arlitt Carey Williamson
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, Canada T2N 1N4
{erman, mahanti, arlitt, carey}@cpsc.ucalgary.ca

ABSTRACT

Traffic classification is the ability to identify and categorize network traffic by application type. In this paper, we consider the problem of traffic classification in the network core. Classification at the core is challenging because only partial information about the flows and their contributors is available. We address this problem by developing a framework that can classify a flow using only unidirectional flow information. We evaluated this approach using recent packet traces that we collected and pre-classified to establish a "base truth". From our evaluation, we find that flow statistics for the server-to-client direction of a TCP connection provide greater classification accuracy than the flow statistics for the client-to-server direction. Because collection of the server-to-client flow statistics may not always be feasible, we developed and validated an algorithm that can estimate the missing statistics from a unidirectional packet trace.

Categories and Subject Descriptors

C.2.2 [**Computer-Communications Networks**]: Network Protocols; C.4 [**Computer Systems Organization**]: Performance of Systems

General Terms

Algorithm, Measurement, Performance

Keywords

Traffic classification, Machine learning, Clustering

1. INTRODUCTION

In recent years, Peer-to-Peer (P2P) file-exchange applications have overtaken Web applications as the major contributor of traffic on the Internet. Recent estimates put the volume of P2P traffic at 70% of the total broadband traffic [3,22]. P2P is often used for illegally sharing copyrighted music, video, games, and software; P2P traffic can cause network congestion and performance degradation of traditional client-server applications such as the Web. The legal ramifications of this traffic combined with its aggressive use of network resources has necessitated a strong need for identification of network traffic by application type. This task, referred to as traffic classification, is a pre-requisite to many network management and traffic engineering problems.

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

The classical traffic classification approach of mapping traffic to applications based on port numbers is now ineffective [18, 19, 22, 30]. This ineffectiveness arises because applications such as network games, multimedia streaming, and Peer-to-Peer file sharing use dynamic ports for communication. Some P2P applications are also masking their identity by using port numbers reserved for other applications. For example, KaZaA is known to use port 80, which is reserved for Web traffic.

An alternative approach is payload-based analysis where packet payloads are searched for characteristic signatures of known applications [4, 15, 24, 30]. Application-layer analysis of packet contents is employed by some commercial bandwidth management tools [2,26]. This general approach, however, poses several technical challenges. First, these techniques identify only traffic for which signatures are available; maintaining an up-to-date list of signatures is a daunting task. Second, these techniques typically require increased processing and storage capacity. Solutions such as capturing only a few payload bytes are not as effective because many applications intentionally use variable-length padding to obscure application signatures. Finally, these techniques fail to detect encrypted traffic; many P2P applications are now moving towards using encryption.

The diminished effectiveness of the aforementioned techniques motivate use of flow statistics for classifying network traffic [1, 25, 29]. There are at least three reasons why this approach is recommended. First, different applications manifest dissimilar behaviors and thus exhibit different flow statistics. For instance, a large file transfer using FTP would have higher average packet size and smaller mean packet interarrival time than an instant messaging client sending short, occasional, messages to other clients. Second, although obfuscation of flow statistics is also possible, it is generally much harder to implement. Third, classification based on flow statistics can benefit from the large body of work on scalable flow sampling/estimation techniques [5, 7, 8, 13, 21].

In this paper, we propose and evaluate a machine learning approach for identifying and grouping network traffic according to traffic classes (e.g., Web, P2P, FTP, Others) at egress and ingress points of core networks. Recent traffic classification efforts, including those that leverage flow statistics, are developed and evaluated assuming that the observation point is the network edge, where packet transmissions in both directions of a flow can possibly be observed.

At egress/ingress points of a network core, observing both directions of a flow may not be possible because of routing asymmetries. This poses two challenges. First, important statistics for the satisfactory classification of a flow may not be available. Second, classification can only use per-flow information and cannot rely on additional information such as communication pattern between hosts.

In light of the above, our goal is to achieve rich traffic classification using only *unidirectional* flow records. Specifically, we propose and evaluate a *clustering* based framework for classifying network traffic using only unidirectional flow statistics. Our work is facilitated by recent full-payload Internet packet traces. We identified the applications corresponding to individual flows in the traces and used these pre-classified traces as the "base truth" to evaluate the classification accuracy of our approach.

One of the objectives of this work is to study the influence of directionality of flow statistics in classifying traffic. From our performance evaluation, we find that flow statistics for the server-to-client direction of TCP connections achieve, on average, classification accuracies of 95% for flows and 80% for bytes; in contrast, using the client-to-server flow statistics yields, on average, classification accuracies of 93% for flows and 60% for bytes. Based on our results, we hypothesize that statistics for the server-to-client direction can better discriminate between flows than statistics for the client-to-server direction, since for many common network applications the flow of application payload data is greater in the server-to-client direction.

The server-to-client statistics of a flow may not always be available at the network core. Motivated by our observations regarding the predictive power of server-to-client statistics, we developed and verified an algorithm that uses the packets seen along one direction of a flow to estimate statistics for the direction that is not observed.

To the best of our knowledge, this is the first work to explore traffic/flow classification using unidirectional flows (as typically seen at the network core). In this paper, we summarize our experience with a clustering-based classification framework. Because server-to-client statistics are typically available at the network edge, our approach can also be applied to classify traffic at the network edge. The contributions of this paper are summarized as follows:

- We propose and evaluate a machine learning based classification technique that only takes flow statistics as input.
- The comparison of predictive capability of different unidirectional flow statistics (e.g., packets originating only from the client, server, and combinations of both).
- We develop an algorithm capable of estimating statistics from an unidirectional traces such as number of bytes and the number of packets of the unseen portions of the flow.
- We briefly discuss the longevity of the models used in our classifier, and the possible modifications that can be made to use our classifier in real-time.

The remainder of this paper is structured as follows. Section 2 describes the classification framework considered in this paper. Section 3 describes our methodology. Results for classification with unidirectional statistics are provided in Section 4. The flow statistics estimation algorithm, its validation, and the classification results obtained with estimated statistics are presented in Section 5. Section 6 discusses the longitudinal accuracy of our classification models, and possible modification for real-time classification. Section 7 overviews related work in light of ours. Section 8 concludes the paper.

2. MACHINE-LEARNED CLASSIFICATION

The goal of traffic classification is to map network flows into predefined application types or traffic classes. In this paper, our goal is to achieve traffic classification using only *unidirectional* flow statistics as input.

Formally, the traffic classification problem can be defined as follows: Given a set of flows $X = \{f_1, f_2, \cdots, f_N\}$, where each flow vector f_i is characterized by a set of p attributes $\{x_{i1}, x_{i2}, \cdots, x_{ip}\}$, and a set of traffic classes $C = \{C_1, C_2, \cdots, C_m\}$, the goal of traffic classification is to define a mapping $f : X \rightarrow C$ such that each flow f_i is assigned to only one traffic class [9]. Examples of flow attributes include average packet size, average flow duration, and flow size, whereas examples of possible traffic classes include Web, Peer-to-Peer, and FTP.

Machine learning techniques can be used to solve the aforementioned traffic classification problem. In the model building step, *training* data (e.g., a collection of flow records) are used to learn the characteristics of the (desired) classes; this step provides the basis for designing a classifier. The remainder of this section discusses details of the model building and classifier design.

2.1 Model Building

Machine learning techniques for model building may be broadly categorized as either *supervised* or *unsupervised* [9]. Supervised learning produces a model that fits the training data, where the training data is labeled *a priori*. In contrast, unsupervised learning uses unlabeled training data to find similarities or patterns among objects in the data set.

Typically, model building is achieved using supervised learning techniques. In the traffic classification context, however, we believe that unsupervised learning can offer certain advantages compared to supervised learning approaches. A key benefit is that new applications can be identified by examining the flows that are grouped to form a new cluster. The supervised approach can only classify traffic for which it has labeled training data, and cannot discover new applications [11].

In this paper, we consider a specific type of unsupervised learning called *clustering* [9]. Clustering is the partitioning of previously unlabeled objects into disjoint groups, referred to as "clusters", such that objects within a group are similar according to chosen criteria. Formally, the clustering of training flows can be described as follows: Given a set of training flows $D = \{t_1, t_2, \cdots, t_n\}$ and the desired number of clusters, k, the task of clustering is to define a mapping $f : D \rightarrow \{1, 2, \cdots, k\}$ where each flow is assigned to only one cluster Y_i, $1 \leq i \leq k$, such that $D = \cup_{j=1}^{k} Y_j$ and $Y_i \cap Y_j = \emptyset, \forall i \neq j$ [9, 16, 17].

The goal of clustering is to group together objects that are similar. This grouping is achieved using a similarity metric. In the machine learning literature, several similarity metrics have been defined. Without loss of generality, we use the Euclidean distance to measure the similarity between two

flow vectors f_i and f_j:

$$sim(f_i, f_j) = \sqrt{\sum_{k=1}^{p}(x_{ik} - x_{jk})^2}.$$

The smaller the Euclidean distance between two flow vectors is, the greater is the similarity between them.

There are many different clustering algorithms in the literature [16, 17]. In this paper, we use the K-Means [9] algorithm. Our choice of K-Means is guided by our previous work [10] where we found that K-Means is one of the quickest and simplest algorithms for clustering of Internet flows; furthermore, our work also showed that K-Means can generate very "pure" clusters (i.e., clusters that consist largely of a single application type).

The K-Means algorithm belongs to the partition-based class of clustering algorithms. The algorithm begins by randomly choosing cluster centroids χ_i, $i = 1, 2, \cdots, k$, from within the training samples. The flows in the training data set are then partitioned into the nearest cluster centroids using the Euclidean distance metric. K-Means iteratively computes new centroids of the clusters that are formed and then repartitions the flows based on the new centroids. This process continues until a convergence criterion is met. In our implementation, the convergence criterion is to minimize the sum of the squared error for the clusters. The complexity of the clustering step is $O(knm)$ where k is the number of clusters, n is the number of training flows, and m is the number of iterations.

There are some known difficulties with K-Means. For instance, the algorithm often finds a local optimum instead of a global optimum. This necessitates running K-Means multiple times to obtain a reasonable partition of the training flows, as done in this work. In practice, partitioning of the training flows is expected to be undertaken infrequently, and we do not expect this problem to significantly add to the cost of model building. In our experiments, we find that the classification models generated by running K-Means multiple times have similar performance. We also note that our overall approach is not specific to K-Means. Other clustering algorithms may be used; investigation of classification performance with other algorithms is left for future work.

In most classification problems, selection of features (or attributes) plays an important role. Many statistics can be obtained from a flow. Following extensive experimentation with over 25 different statistics using standard feature selection algorithms [14], we reduced the set of flow features to the following: total number of packets, mean packet size, mean payload size excluding headers, number of bytes transferred, flow duration, and mean inter-arrival time of packets. Due to the heavy-tailed distribution of many of these features, we found it necessary to transform the flow features [28]. Our experiments with many commonly used transformations indicated that logarithmic transformations yield the best results.

2.2 Classifier

Our classifier is distance-based [9]. A new flow is assigned to the cluster to which it is most similar. The K-Means algorithm produces clusters that are spherical in shape and thus well-represented by the cluster centroids. Thus, a new flow f is assigned to cluster Y_j such that:

$$Y_j = \arg\min_j sim(f, \chi_j).$$

The above is equivalent to maximum likelihood cluster assignment for partitions generated by the K-Mean algorithm. The complexity of classification of each flow is $O(k)$, where k is the total number of clusters.

The aforementioned operation automatically assigns flows to the clusters. To assign traffic classes to flows, a mapping between clusters and traffic classes is required. Clearly, the task of mapping the clusters obtained in the model building step to the traffic classes requires identification of (some of the) individual flows in the training data set. Training flows may be labeled using techniques including payload analysis, port-based analysis, heuristics, expert knowledge, experimentation, manual classification, or a combination thereof.

Identification of training flows is expected to be time consuming; however, we also expect that once completed, the training data set can be used for a reasonably long period of time. Furthermore, it may not be necessary to identify each flow in the training data set. We have found that clustering generates partitions with high purity and thus identification of a small portion of the flows in a cluster may be sufficient to map a cluster to a traffic class with a high degree of confidence [10]. In this work, we assume that labels for all training flows are available, and map a cluster to the traffic class that makes up the majority of flows in that cluster. Our ongoing work is studying issues pertaining to labeling clusters and mapping clusters to traffic classes [12].

3. EXPERIMENTAL METHODOLOGY

This section outlines our experimental methodology. Section 3.1 describes the empirical traces used in this work. Section 3.2 discusses the process by which we established base truth for the traces. The testing scenarios are outlined in Section 3.3. Section 3.4 defines the performance metrics used in this study.

3.1 Empirical Traces

To facilitate our work, we required traces of recent Internet traffic. Although the classification framework requires only transport-layer information, application-layer information is required to validate the results. Thus, we decided to collect full packet traces from a monitor attached to our campus Internet link.

We collected eight 1-hour traces between April 6-9, 2006. Specifically, we collected traces on (Thursday) April 6 from 9-10 am and 9-10 pm, on (Friday) April 7 from 9-10 am and 9-10 pm, on (Saturday) April 8 from 9-10 am and 9-10 pm, and on (Sunday) April 9 from 9-10 am and 9-10 pm. We were limited to capturing only one hour of continuous full-packet traces owing to the disk capacity of our network monitor. Nevertheless, we expect our traces to cover some typical cases such as the noticeable differences in usage between the morning and evening hours, and the noticeable differences in usage between weekdays and weekends.

Of the total trace data collected, approximately 85% of the packets, and approximately 90% of the bytes were transferred using TCP. Thus, we focus exclusively on applications that use TCP for the remainder of the paper. Classification of UDP flows is left for future work.

A TCP flow, also referred to as a connection, consists of

a bi-directional exchange of packets between two hosts. The start of a flow is determined when SYN/SYNACK packets are received. Flows are (typically) terminated when either a FIN or RST packet is received; in addition, we assume that a flow terminates if it was idle for 900 seconds. For each flow, we designate the host that initiated the connection (i.e., sent the SYN packet) as the *client*, and the host that responds to the connection initiation (i.e., sends the SYNACK) as the *server*.

3.2 Establishing Base Truth

We classify the TCP flows in the traces using a three step process that consists of payload-based signature matching, heuristics, and HTTPS identification. The heuristics and HTTPS identification steps deal with encrypted traffic that cannot be identified using payload signature matching. Manual classification served as a validation tool for our classification process.

Our payload-based classification uses many of the same methods and signatures described by Sen *et al.* [30] and Karagiannis *et al.* [20]. We augmented some of their P2P signatures to account for protocol changes and some new P2P applications. This step uses Bro [27], whose signature matching engine generates a signature match event when the packet payload matches a regular expression that is specified for a particular rule.

Some P2P applications are now using encryption. For example, BitTorrent is using a technique called Message Stream Encryption (MSE) and Protocol Encryption (PE). The MSE/PE technique uses a Diffie-Hellman exchange that is combined with the *infohash* of the *torrent* to establish the key for the connection.[1] After this exchange has occurred, the clients use RC4 to encrypt the data packets.

We developed a heuristic to identify some of the aforementioned encrypted P2P traffic. Specifically, we maintain a lookup table of IP address and port number tuples from flows that have recently 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. In general, heuristics such as these may be used to assign labels to any encrypted P2P flow in the training data set.

We also analyzed unlabeled traffic on port 443, to determine whether or not this traffic is indeed HTTPS. This verification was done using an experimental version of Bro with 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. These tests, however, were not done in an exhaustive fashion.

Table 1 summarizes the classification results for our traces. Over 29 different protocols were identified; these 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. We grouped these protocols according to application categories. For example, the P2P category includes positively identified P2P traffic from protocols including BitTorrent, Gnutella, and KaZaA. Encrypted P2P flows identified using heuristics are labeled P2P (Encrypted). The OTHER category contains various identified applications that were not part of a larger group and did not account for many flows. The tables also list three categories of UNKNOWN flows. The UNKNOWN(NP) refers to flows with no payloads. Most of these are failed TCP connections, while some are port scans. The UNKNOWN(443) are flows on port 443; they are likely to be HTTPS traffic. The third category is simply labeled as UNKNOWN(Others) to reflect the fact that we were not able to determine the applications that generated this traffic.

[1]http://en.wikipedia.org/wiki/protocol_encryption

Table 1: Summary Statistics of the Empirical Traces

Traffic Class	Flows	% Flows	Bytes	% Bytes
Web	2,791,301	55.7%	59.6 GB	40.6%
EMAIL	299,736	6.0%	7.4 GB	5.0%
DATABASE	218,324	4.4%	0.1 GB	0.1%
P2P	201,231	4.0%	53.2 GB	36.2%
P2P (Encrypted)	11,746	0.2%	3.1 GB	2.1%
OTHER	12,130	0.2%	2.3 GB	1.6%
CHAT	10,680	0.2%	0.3 GB	0.2%
FTP	4,636	0.1%	4.3 GB	2.9%
STREAMING	1,107	0.0%	1.6 GB	1.1%
UNKNOWN (NP)	932,897	18.6%	0.1 GB	0.1%
UNKNOWN (443)	366,347	7.3%	4.5 GB	3.1%
UNKNOWN(Others)	161,492	3.2%	10.3 GB	7.0%
Total	5,011,627	100.0%	146.8 GB	100.0%

From the table, we see that Web and P2P traffic account for a majority of the campus Internet traffic. Note that although P2P accounts for only 4% of the flows, it still accounts for approximately 36% of the total bytes transferred.

In our classification experiments, we classify flows into one of the 8 traffic classes shown in Table 1. For the purpose of classification, P2P and P2P (Encrypted) are grouped together to form the P2P traffic class. We exclude all unknown flows as we do not have base truth for these.

3.3 Testing Scenarios and Data Sets

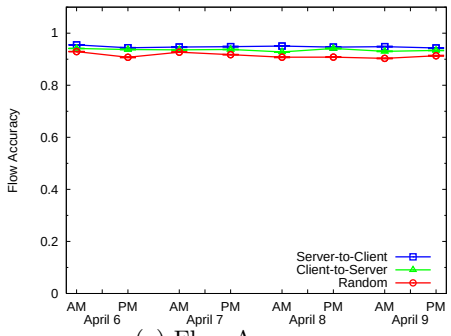
The empirical traces at our disposal have both directions of a flow. Our goal is to study how the directionality (i.e., client-to-server or server-to-client) of a flow impacts classification results. We generated from each empirical trace a "server-to-client" data set and a "client-to-server" data set that for each flow in the trace records only the packets seen in the server-to-client direction or the client-to-server direction, respectively. To represent the typical case of traffic seen at the network core, we selected for each flow in an empirical trace either the client-to-server direction packets or the server-to-client direction packets. We refer to this third category of data sets as "random directionality".

3.4 Performance Metrics

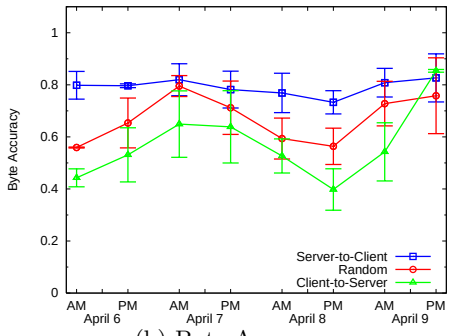
We use four metrics in our evaluation, namely flow accuracy, byte accuracy, precision, and recall.

Flow accuracy is the number of correctly classified flows divided by the total number of flows in the test data set. *Byte accuracy* is the amount of correctly classified bytes divided by the total number of bytes in the test trace. Both metrics are important to maximize because having a few flows misclassified could result in a many bytes being classified incorrectly.

While these traditional metrics are useful, the accuracy measures neglect the fact that there might be consequences associated with incorrect classification of a flow. For ex-



(a) Flow Accuracy



(b) Byte Accuracy

Figure 1: Classification accuracy (all traces).

ample, classification of Web flows as P2P might result in blocking or lower priority assignment for the Web flows. We use precision and recall to quantify the impacts of incorrect classification.

Precision of a traffic class is defined as the ratio of the number of true positives to the sum of true and false positives for the class under consideration. If there is risk associated with misclassification of flows, a higher precision value is desired. *Recall* measures the fraction of relevant flows in a traffic class that have been correctly classified. It is defined as the ratio of the number of true positives to the sum of true positives and false negatives for the class under consideration. A high recall value implies that there are very few relevant flows that are misclassified as another traffic type. For example, if we want to assign high priority to Web flows, we need to achieve high recall values for this traffic class.

4. CLASSIFICATION RESULTS USING UNIDIRECTIONAL FLOWS

This section evaluates the effectiveness of using different unidirectional data sets in our classification framework. As discussed in Section 3.3, we consider three test scenarios: data sets containing only client-to-server packets, data sets containing only server-to-client packets, and data sets that contain a random mixture of each directionality.

From each data set, we generated 10 different training data sets, each generated by selecting 64,000 random sample flows. A sample size of 64,000 represents a good compromise between the model's ability to represent different applications and the computational cost of building the model.[2] After the clustering was complete, we used each of these models in our classifier for classification of the entire respective data set. We report the average results and the 95% confidence intervals for the 10 models.

The K-Means algorithm requires the number of clusters as an input. Here we describe how we selected k; in general, k can be considered a tuning knob that needs adjustment based on the type of traffic we are trying to classify. In our experiments, we found that both flow and byte accuracies improved as we increased k from 25 to 400. The flow accuracy shows an incremental improvement of 5% to 8% for all three types of data sets. However, in terms of byte accuracy, the classification using server-to-client data sets shows the greatest improvement, from 59% accuracy to 80%. With the client-to-server data sets the improvement in byte accuracy is only 10%, and with the random data sets it is only 5%. The improvement witnessed with the server-to-client data sets is because P2P traffic is being correctly identified. When the value of k used is 25, P2P flows are almost always incorrectly classified as Web because there is only a single cluster representing P2P. However, when k increases to 400 there are normally 12 to 15 clusters representing P2P, and the byte accuracy, on average, improves to 80%. In general, we expect k to be larger than the number of traffic classes or applications we are trying to classify. For example, we expect to form more than one cluster for Web traffic to capture its various characteristics, including well-known heavy-tailed file transfer sizes. With larger values for k we are able to capture more of the application characteristics within our clusters in the model. For the remainder of the evaluations, we used k equal to 400.

Figure 1 shows the classification accuracy results for data sets derived from each trace. Overall, we found that the server-to-client data sets consistently give the best classification accuracy achieving, on average, 95% and 79% in terms of flows and bytes, respectively. With the random data sets, the average flow and byte accuracy was 91% and 67%, respectively. For the client-to-server data sets, 94% of the flows were correctly classified, on average, for an average byte accuracy of 57%. In general, using the client-to-server data sets resulted in the worst byte accuracies in all traces, except for the April 9, 9 pm trace.

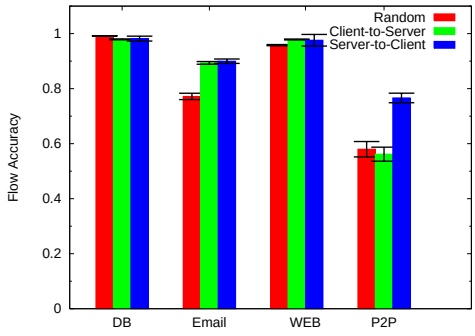
Figure 2 shows the flow and byte accuracies achieved for the four most significant applications (in terms of number of flows). We found that all three types of data sets have a high flow accuracy for the Database, Email, and Web traffic, with both client-to-server and server-to-client data sets achieving, on average, accuracies in excess of 90% for all three applications. The application type that proved the most difficult to classify was P2P. The server-to-client data sets achieved a 77% flow accuracy; this is 20% greater than the accuracies with client-to-server and random data sets.

Table 2: Confusion Matrix of Flows with Server-to-Client Data Sets (April 6, 9 am trace)

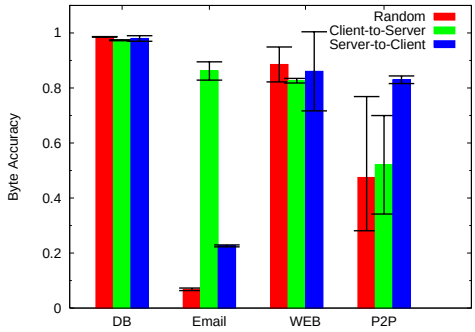
Actual Class	Classification				
	WEB	EMAIL	P2P	DB	OTHER
WEB	511375	5214	7284	520	1084
EMAIL	6620	64732	3066	88	631
P2P	6886	3620	47716	199	254
DB	1262	420	872	41262	166
OTHER	1904	232	1018	103	5336

To help illustrate the accuracy of the classification, Table 2 shows the "confusion matrix" [9] for the classifier when

[2]Note that the number of flows in a test data set typically ranges between 500,000 to 1,000,000.



(a) Flow Accuracy



(b) Byte Accuracy

Figure 2: Classification accuracy by application (April 6, 9 am trace).

using a server-to-client data set. In this $m \times m$ matrix, the data value $c_{i,j}$ indicates the number of flows from class i that were classified as class j. Obviously, we want values along the diagonal to be much larger than the others which is what we observe. By looking across the row of the confusion matrix at a given class i we can calculate the recall for that class. Likewise, by looking down a column at a given class j we can calculate the precision of that class.

The per-application byte accuracy for Database and Web is high with all three types of data sets. However, for Email and P2P flows the accuracies vary considerably between the different data sets. For Email flows, the client-to-server data sets provide 86% accuracy, but the random and server-to-client data sets have extremely low accuracies of 7% and 23%, respectively.

While it is difficult for us to make a generalization to encompass every model and trace, in the models where we did extensive analysis of the results, the reason why the client-to-server data sets classified Email so well was that SMTP flows were being correctly classified. In the client-to-server models, SMTP flows were put into a few large clusters that classified most of the SMTP traffic, with one of these clusters normally capturing most of the large SMTP flows (in terms of bytes). However, in the server-to-client models the SMTP clusters were more fragmented and generally formed many small clusters. The smaller clusters were generally for SMTP flows with few bytes transferred (less than 2000 bytes). The larger SMTP flows that accounted for most of the Email bytes generally did not form a cluster and were included in clusters labeled either as P2P or Web. The confusion matrix in Table 2 further confirms that these misclassifications with the server-to-client models are mostly P2P and Web. In the random directionality case, SMTP did not form many clusters, which resulted in SMTP being misclassified most of the time.

P2P flows are classified more effectively for the server-to-client data sets than the other data sets. With the server-to-client data sets, byte accuracy of approximately 83% is achieved, which represents a 30% increase over client-to-server and random data sets. This higher classification accuracy is because 20% more P2P flows are correctly classified using the server-to-client data sets. This marked difference from the other data sets is one of the main reasons why server-to-client data sets achieve the best flow and bytes accuracies in Figure 1.

If server-to-client flow statistics are used for discriminating between P2P and Web flows, encouraging results would be attained. Overall, in the server-to-client models Web flows have precision and recall values of 97%; P2P flows have precision of 82% and a recall of 77%. If a Quality-of-Service policy of assigning lower priority to P2P flows than to Web flows is implemented, 77% of the P2P would be correctly given a lower priority and at the same time less than 3% of the Web flows would be mistakenly given a lower priority. Such a deployed (real-time) system could ease the strain that P2P puts on many networks.

While we have advocated the discrimination of P2P and Web traffic in the above example we are, however, not limited to just these two types of applications. If reducing P2P was not the concern and instead prioritizing mission-critical business traffic was the focus then our classification system could be used just as successfully. Business-critical traffic from a Database achieves a high accuracy as well. The confusion matrix provides further evidence of this fact with a precision of almost 98% and a recall of 94% when classifying Database flows.

5. CLASSIFICATION RESULTS USING FLOW ESTIMATION

In this section we introduce and use our flow statistic estimation algorithm. This algorithm uses the packets of an unidirectional flow to estimate the flow statistics of the unobserved direction. The estimation algorithm is based on the syntax and semantics of the TCP protocol and thus, would not work for other transport protocols such as UDP. We first introduce the algorithm in Section 5.1. Section 5.2 discusses some of the assumptions. In Section 5.3, we empirically verify our estimation algorithm's predictions. Last, in Section 5.4 we test the classification accuracy using the estimated statistics.

5.1 Algorithm

The statistics of interest to us can be divided into three general categories: duration, number of bytes, and number of packets. After we obtain the data for these three general categories we can calculate other statistics such as average throughput, mean packet interarrival time, and packet average size.

The duration of a flow is the amount of time from when the first packet of a flow is sent until the last packet of the flow is sent. This statistic is fairly easy to calculate; we can use the first and last packet sent in the observed direction as a good estimate of the flow duration. This works well

because typically in a well-behaving TCP connection every packet that is sent receives a corresponding acknowledgment from the other host. The packet exchanges typically occurring at the beginning and at the end of a flow have the SYN and FIN packets, respectively. In cases where we did not see the SYN and/or FIN exchange, such as when the traffic monitor drops packets, we calculate the duration with the first or last exchange of data packet and acknowledgment packets, which may result in a less accurate estimate of the flow duration.

The second category of statistics is concerned with the number of bytes transmitted. Our approach for calculating the number of bytes is similar to the technique developed by Smith *et al.* [31]. In the TCP protocol, the host responds to reception of TCP segments (packets) by sending acknowledgments (ACKs) with the sequence number field (SEQ) in the TCP header set to indicate the next expected in-order byte. By using these ACK numbers we can estimate the amount of data that has been received by calculating the offset between the highest ACK number and the lowest ACK number seen. This works fairly well for most TCP connections. However, it does not work reliably for connections that were closed using TCP resets (RST). For TCP RST packets, the ACK number may not correspond to the in-order byte sequence received. Instead, some TCP implementations assign a random value. To combat this problem, we exclude the ACK numbers from RST packets when we calculate the highest and lowest ACK numbers.

The last category of statistics, the number of packets sent, is the most difficult to estimate. We derive a set of heuristics that estimate, for each TCP flow, the number of packets that could potentially be received in the other direction between transmission of two successive packets. We assume that if a SYN packet is seen, then we are seeing the client-to-server packets of a flow. Otherwise, we assume we are seeing the server-to-client packets. Algorithm 1 shows the rules that we defined. We track the last sequence number (*PrevSeq*) and acknowledgment number (*PrevAck*) seen in the flow; these values are initialized to zero before a flow starts. We also calculate the change in the sequence number (*SeqChg*) and acknowledgment number (*AckChg*) between the packets observed. In the event that we do not receive a SYN or a SYNACK packet at the beginning (or at all), our algorithm processes the first data packet with either our first (line 5) or second rule (line 7), and then works correctly afterward.

We explain the remainder of this algorithm using examples. Let us assume that we are seeing the client-to-server packets, that one packet (for the flow of interest) had a sequence number of 100 and an acknowledgment number of 200, and that the next packet has a sequence number of 1560 and an acknowledgment number of 200. The increase in sequence number indicates that the most recent packet carried some payload data. However, since the acknowledgment number has not increased we infer that the missing server-to-client packets for this interval had no payload data and would most likely be ACKs corresponding to the payload in the last packets sent. This case would be caught by our third rule (line 9) where we check to see if the sequence number has increased and the acknowledgment number has not. We calculate the number of ACKs as the sequence number change divided by the expected maximum segment size (MSS). Conversely, if the sequence number does not increase but the acknowledgment number does increase we infer that

Algorithm 1 Packet Estimation Algorithm

```
1:  PrevSeq ← 0, PrevAck ← 0
2:  MissedAcks ← 0, MissedData ← 0
3:  for each packet do
4:      CALCULATE(SeqChg, AckChg)
5:      if SeqChg > 0 and AckChg = 0 and PrevSeq = 0 then
6:          continue          ▷ SYN packet sent and nothing is missed
7:      else if SeqChg > 0 and AckChg > 0 and PrevAck = 0 then
8:          MissedAcks ← MissedAcks + 1      ▷ SYNACK or SYN missed
9:      else if SeqChg > 0 and AckChg = 0 then
10:         MissedAcks ← MissedAcks + ⌈SeqChange/MSS⌉
11:     else if SeqChg = 0 and AckChg > 0 then
12:         MissedData ← MissedData + ⌈AckChange/MSS⌉
13:     else if SeqChg > 0 and AckChg > 0 then
14:         MissedData ← MissedData + ⌈AckChange/MSS⌉
15:     else if SeqChg ≤ 0 or AckChg ≤ 0 then
16:         continue      ▷ Nothing has been missed from last packet seen
17:     end if
18: end for
```

in this interval packets that were sent in the other direction contained a total payload size directly proportional to the change in the acknowledgment numbers. To calculate the number of data packets, we divide the acknowledgment number change by the MSS. This case is handled by our fourth rule (line 11). The fifth rule (line 13) handles cases where data are being sent simultaneously in both directions. The sixth rule (line 15) handles retransmissions and packets that are received out of order.

5.2 Assumptions

In our rules we make three general assumptions, the first pertaining to the expected MSS of packets, the second pertaining to the ACK-ing policy of the TCP stacks, and the last in regards to retransmissions and packet loss.

We use MSS in our calculations for the number of packets sent. The MSS can be estimated from the options field in the SYN/SYNACK packets of a connection. A MSS announcement is made by each host at the beginning of a TCP connection with the lowest value typically being used. In a unidirectional trace it would be possible on a per-flow basis to estimate MSS based on any announcements seen. However, to be more computationally efficient to determine the expected MSS, we analyzed the empirical distribution of MSS in our traces. Our analysis showed that 95% of the connections had a MSS of 1460 bytes. Approximately 5% had a MSS of 1380 and some other minor groupings at 512 and 1260. Therefore, we used 1460 bytes as the expected MSS in our verification and results.

How TCP acknowledges segments depends on the TCP stacks of both the client and the server. In some cases, an ACK is sent for every packet, while in other cases an ACK is sent for every other packet. Our heuristics assume a simple acknowledgment strategy of an ACK (with 40 bytes of header data and no payload) for every data packet in the flow. We realize that this may over estimate the number of ACKs.

We also assume there are no packet losses, and therefore, our statistics do not take into account any retransmissions. We make this assumption because retransmissions reflect network congestion and transmission errors, rather than application-specific behaviour of the flows that we want to classify. However, this does make our estimations lower than what the actual numbers should be but this has the

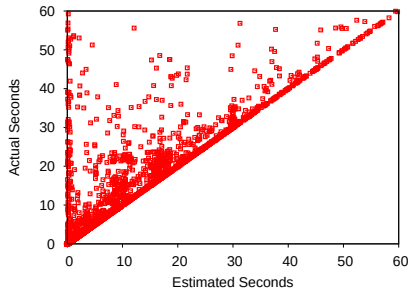


Figure 3: Estimation of the flow duration.

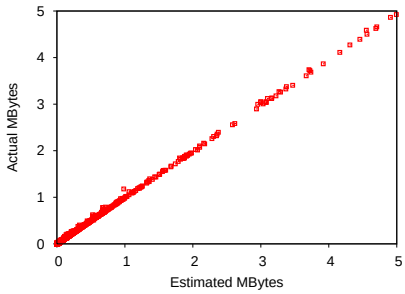


Figure 4: Estimation of the number of bytes.

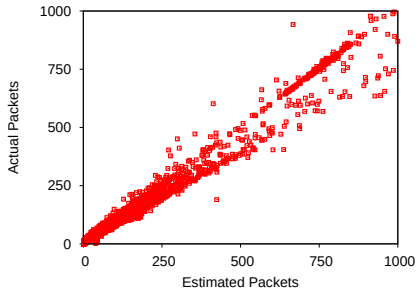


Figure 5: Estimation of the number of packets.

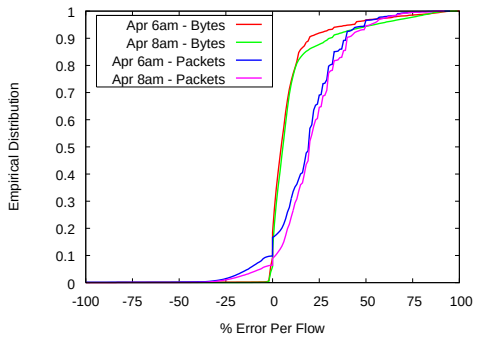


Figure 6: CDF of the per-flow percentage error.

positive effect of balancing the overestimation of the number of ACKs.

5.3 Validation

Estimating flow duration is easy, and overall the error in the duration estimation is low. The average flow duration was 27.5 seconds, with an error of 7.3%. In our estimation results shown in Figure 3, we found that normally 90% of the flows had duration errors less than 1 msec. In most cases where there was a high error in the duration, we found that the error was caused by a RST or FIN packet being sent well after the rest of the flow's packets were sent.

Figure 4 shows a scatter plot of the actual number of bytes versus the estimated number of bytes for the random data set generated from the April 6, 9 am trace. The scatter plot shows strong agreement between the actual and estimated amount of bytes. For our traces, the algorithm was always within within 0.4% and 1.4% of the actual number of bytes.

Figure 5 shows a scatter plot of the actual number of packets versus the estimated number of packets for the random data set generated from the April 6, 9 am trace. As seen in the scatter plot, the estimated number of packets closely follows the actual number of packets; the estimate inaccuracy appears to be somewhat larger when there are more packet transfers along the missing direction of the flow. Experiments with the remaining traces showed that the packet estimate was, on average, within -5.3% and 1.6% of the actual number of packets.

On a per flow basis, Figure 6 shows the distribution of the per-flow percentage error for both packet and byte estimation. It shows that our estimate is within 30% of the actual number of packets for 80% of the flows, and within 20% of the actual number of bytes, for 90% of the flows.

Looking solely at the percentage error is somewhat misleading, since the high error cases often correspond to flows with few packet transmissions (less than 10). The main source of inaccuracy are flows that after the TCP handshake had occurred saw a single reject or RST packet from the server. The client in such cases attempts to send the initial data packets several times. This typically occurred in P2P connections that were refused. If our algorithm sees the server side of such flows it estimates it missed either 0 or 1 packets because we ignore RST packets. Otherwise, if it sees the client side of the flow it thinks it missed several ACKs because we assume all packets are acknowledged. In both cases, the algorithm is off by a couple packets but the percentage error is large. We found that overall average error per flow is 2.4 packets, and that 87% of flows are within 5 packets of the actual number. In terms of bytes, the overall average error is 120 bytes, and 92% of the flows are within 500 bytes of the actual number.

5.4 Classification Using Estimated Statistics

We examine the classification accuracy of our classifier if we use the estimation algorithm described in the previous section to estimate server-to-client statistics for our traces when only the client-to-server or random directionalities are seen.

Figures 7 and 8 show the classification accuracy. These experiments are similar to those reported in Figure 1, except that both model building and the subsequent classifier use the estimated statistics when necessary. As seen in these figures, we find that when we use the estimation algorithm to estimate the server-to-client statistics the flow accuracy and byte accuracy achieved using the client-to-server and random directionality is very close to the actual accuracy we achieved when using the actual server-to-client statistics obtained from the empirical traces.

Interestingly, our classification accuracies are largely unaffected by the potential errors in our estimated flow statistics. We think this robustness is due to the fact we use the logarithm of the flow statistics (as mentioned in Section 2.1). The magnitude of difference of the flow statistics has a much greater impact in the classification than the small errors in the estimations. This makes us believe it is possible to use our estimation technique to calculate the different statistics that allow the best classification, even though only partial information is available.

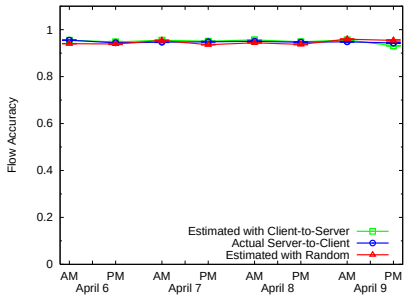


Figure 7: Flow accuracy with estimation algorithm estimating Server-to-Client statistics.

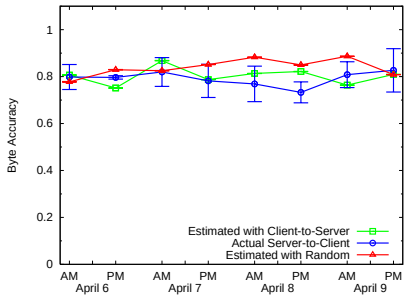


Figure 8: Byte accuracy with estimation algorithm to estimating Server-to-Client statistics.

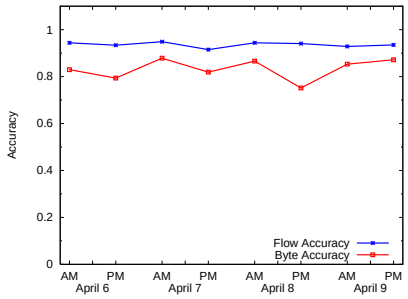


Figure 9: Longitudinal classification accuracy using a single model on all traces.

6. DISCUSSION

We found in our experimentation that increasing the sample size between 2000 and 128,000 does not significantly improve our classification results. However, we did find that a larger sample size had a higher degree of confidence of covering the sample space of traffic characteristics.

Figure 9 shows that a single model (built by drawing samples from one trace) is fairly resilient over a longer period of time. For this particular experiment, we used a model built from the (Friday) April 7, 9 am trace; our results show that this model is applicable on other days and times. This shows that a generic model could be built to classify traffic for long periods of time. This would reduce the frequency at which the expensive step of model building (i.e., clustering and labeling) would need to be repeated.

We think that the classification framework we use could be adapted in several ways. One example could be to build several models to reflect the characteristics of flows after certain *packet milestones* (e.g., 8, 32, and 64 packets). Then, a flow currently in-progress could be classified using the different models as it reaches the pre-set packet milestones. This approach can facilitate real-time classification of flows and is an idea that we are currently exploring [12].

7. RELATED WORK

The use of flow statistics for clustering network traffic has received some attention in the literature [10, 23, 33]. These prior works only considered the model building stage and did not evaluate the predictive power of classifiers obtained from clustering.

The classifier proposed by Bernaille *et al.* is the closest to our work [1]. The classifier similarly uses the K-Means algorithm and a minimum distance measure to assign flow to an application label. Their work uses *bi-directional* flow statistics derived from the first P packet exchanges, and is not applicable to the more challenging traffic classification setting considered in this paper.

Some non-clustering, machine learning-based, techniques also use flow statistics for classifying traffic [25,29]. Roughan *et al.* [29] use three algorithms from the data mining literature, namely Nearest Neighbour, Linear Discriminate Analysis, and Quadratic Discriminate Analysis, to classify flows into four predetermined traffic classes. They study the use of different statistics such as duration and average packet size of a flow for classifying traffic into four distinct classes. Moore *et al.* [25] use a supervised machine learning algorithm called Naïve Bayes as a classifier. Using a "hand classified" trace, they show that the Naïve Bayes approach has a high accuracy classifying traffic. These prior efforts did not consider classification at the network core. Our work on the use of clustering techniques for classification at the network core complements these prior efforts.

Traffic identification approaches that rely on application-level behaviors such as the number of concurrent TCP/UDP connections to an IP address have also been developed [6,19, 20,32]. Karagiannis *et al.* [19] proposed heuristics that capitalize on the unique behaviors (e.g., concurrent use of both TCP and UDP by a source/destination host pair) of P2P applications when they are transferring data or establishing connections to identify this traffic. Their results show that this approach is comparable in terms of accuracy to payload-based identification. In earlier work, a similar approach was used to identify chat traffic [6]. More recently, Karagiannis *et al.* [20] developed another method that uses the social, functional, and application behaviors to identify all types of traffic. Concurrent to [20], Xu *et al.* [32] develop 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 flow characteristics to classify network traffic, and achieves comparable or better accuracies when classifying traffic, including traffic originating from P2P applications.

8. CONCLUSION

This paper considered the problem of classifying network traffic when only one direction of network flows are observed, as may be the case when the point-of-observation is the network core. To address this problem, we developed a clustering-based machine learning framework for classifying network traffic using only unidirectional flow statistics.

We evaluated the aforementioned classification framework using a set of full-payload packet traces. The results show that, in general, rich traffic classification using only unidirectional statistics is feasible, with our experiments showing accuracies of 95% in terms of flows and 80% in terms of bytes. We also found better classification performance is achieved when statistics for the server-to-client direction are used than when statistics for the client-to-server direction are used. Because collection of the server-to-client statistics

may not always be feasible, we developed and validated an algorithm that can estimate the missing statistics from a unidirectional packet trace.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada and Informatics Circle of Research Excellence (iCORE) of the province of Alberta. We thank the anonymous reviewers for their comments on the original version of this paper.

9. REFERENCES

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

[2] Cache Logic. http://www.cachelogic.com/.

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

[4] T. Choi, C. Kim, S. Yoon, J. Park, H. Kim, H. Chung, and T. Jesong. Content-Aware Internet Application Traffic Measurement and Analysis. In *IEEE/IFIP NOMS'04*, Seoul, Korea, April 2004.

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

[6] C. Dews, A. Wichmann, and A. Feldmann. An Analysis of Internet Chat Systems. In *IMC'03*, Miami Beach, USA, October 2003.

[7] N. Duffield, C. Lund, and M. Thorup. Properties and Prediction of Flow Statistics from Sampled Packet Streams. In *IMW '02*, Marseille, France, August 2002.

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

[9] M. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, New Jersey, 1st edition, 2003.

[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, A. Mahanti, and M. Arlitt. Internet Traffic Identification using Machine Learning. In *GLOBECOM'06*, San Francisco, USA, November 2006.

[12] J. Erman, A. Mahanti, C. Williamson, M. Arlitt, and I. Cohen. A Semi-Supervised Approach to Network Traffic Classification (Extended Abstract). In *SIGMETRICS '07*, 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] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):254–323, September 1999.

[17] D. Jiang, C. Tang, and A. Zhang. Cluster Analysis for Gene Expression Data: A Survey. *IEEE Transactions On Knowledge and Data Engineering*, 15(11), November 2004.

[18] T. Karagiannis, A. Broido, and N. Brownlee. Is P2P Dying or Just Hiding? In *GLOBECOM '04*, Dallas, USA, November 2004.

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

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

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

[22] A. Madhukar and C. Williamson. A Longitudinal Study of P2P Traffic Classification. In *MASCOTS'06*, Monterey, USA, August 2006.

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

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

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

[26] Packeteer. http://www.packeteer.com/.

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

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

[29] 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.

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

[31] F. D. Smith, F. H. Campos, K. Jeffay, and D. Ott. What TCP/IP Protocol Headers Can Tell us about the Web. In *SIGMETRICS '01*, Cambridge, USA, June 2001.

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

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