Early Application Identification

Laurent Bernaille, Renata Teixeira, Kavé Salamatian
Université Pierre et Marie Curie - LIP6, CNRS
Paris, France
{Laurent.Bernaille,Renata.Teixeira,Kave.Salamatian}@lip6.fr

ABSTRACT

The automatic detection of applications associated with network traffic is an essential step for network security and traffic engineering. Unfortunately, simple port-based classification methods are not always efficient and systematic analysis of packet payloads is too slow. Most recent research proposals use flow statistics to classify traffic flows once they are finished, which limit their applicability for online classification. In this paper, we evaluate the feasibility of application identification at the beginning of a TCP connection. Based on an analysis of packet traces collected on eight different networks, we find that it is possible to distinguish the behavior of an application from the observation of the size and the direction of the first few packets of the TCP connection. We apply three techniques to cluster TCP connections: K-Means, Gaussian Mixture Model and spectral clustering. Resulting clusters are used together with assignment and labeling heuristics to design classifiers. We evaluate these classifiers on different packet traces. Our results show that the first four packets of a TCP connection are sufficient to classify known applications with an accuracy over 90% and to identify new applications as unknown with a probability of 60%.

1. INTRODUCTION

Enterprise or campus networks usually impose a set of rules for users to access the network in order to protect network resources and enforce institutional policies (for instance, no sharing of music files or no gaming). This leaves network administrators with the daunting task of (1) identifying the application associated with a traffic flow as early as possible, and (2) controlling user's traffic when needed. Therefore, accurate and early classification of traffic flows is an essential step for administrators to detect intrusion, malicious attacks, or forbidden applications.

The simplest approach to traffic classification consists in examining the port numbers in TCP headers and mapping them to applications as defined by IANA. Port-based methods can be effective because many well-known applications use registered port numbers (for instance, HTTP traffic uses port 80 and POP3 port 110). However, many applications use dynamic port negotiation instead of standard port numbers. As a result, the applicability of port-based analysis is increasingly limited [14, 21, 17, 15]. An alternative approach is to inspect the payload of every packet searching for specific signatures. This alternative is extremely accurate, but has some limitations. First, there are privacy concerns with examining user data. Second, there is a high storage and computational cost to study every packet that traverses a link (in particular on very high-speed links). Finally, payload information is not useful when applications use encryption.

There have been several proposals [21, 16, 26, 17, 15] to address the limitations of port-based and payload-based classification. Most of the proposed mechanisms perform traffic classification using flow statistics such as duration, number of packets, mean packet size, or inter-arrival time. Unfortunately, these techniques are not appropriate for early application identification as they only classify a flow after it is finished. In [5], we argue that it is possible to perform traffic classification based on the inspection of just the *first few packets of a TCP connection* (typically, the first four or five packets). This paper further explores this argument. First, we analyze packet-header traces collected at eight different edge networks to determine which TCP-connection features and classification techniques work better to early application identification. Then, we build on this analysis to propose a new mechanism to identify accurately and early the application associated with a TCP connection.

Our analysis of packet traces, presented in Section 3, finds that the *size and direction* of the first data packets of a TCP connection are expressive enough to distinguish among different applications. Intuitively, the size of the payload of the first few packets captures the application's negotiation phase, which is usually a pre-defined sequence of messages different between applications. We define the *behavior* of an application as the size and direction of the first P packets it exchanges over a TCP connection, where P is an integer value that we determine experimentally.

We design a classifier to run at the *edge* of a network (i.e., where the network connects to the Internet). Positioning the classifier at the edge allows it to have access to all packets associated with a TCP connection in both directions (from sender to receiver and vice-versa). The classifier contains a set of rules to map each TCP connection (represented by its first P packets) into an application. Our methodology works in two distinct phases: an offline training phase and an online classification phase.

The *training phase*, discussed in Section 5, applies clustering techniques to a set of training data to group TCP connections with similar behavior. The training traces contain examples of TCP connections pre-labeled with the application name. We evaluate three well-known clustering methods (in order of increasing sophistication: K-means and Gaussian Mixture Models on an Euclidean space, and Spectral clustering on Hidden Markov Models).

The *classification phase* uses the clusters defined in the training phase and two heuristics to associate a new connection with an application. An *assignement heuristic* determines whether a new TCP connection belongs to a predefined cluster. Connections that do not belong to any cluster are labeled as unkonwn. This approach allows the classifier to detect new applications or new modes of operation of known applications. A *labeling heuristic* selects the application label for the connection among all the applications in a cluster. This heuristic can use any information available in the first packets of the connection to improve classification accuracy. In particular, we use port numbers when meaningful.

Section 6 describes different assignment and labeling heuristics and Section 7 evaluates these heuristics. We describe the eight traces used in the evaluation in Section 2. Our results show that a classifier using only the size and direction of the first four packets of a TCP connection correctly identifies over 92% of all TCP connections for all traces for known applications. Furthermore, the use of assignment heuristics with thresholds accurately label as *unknown* more than 60% of connections from applications that were not in the training set. We conclude this paper in Section 8 with a summary of our contributions and a discussion of possible extensions and limitations of our approach.

2. DESCRIPTION OF DATA SETS

We use a number of packet traces to evaluate our assumptions and methodology. This section describes each of these traces and our procedure to extract application behavior and labels.

2.1 Packet traces

Our study uses two sets of traces: payload and packet-header traces. None of the monitors use sampling, so all traces contain all packets traversing the monitor during the measurement period. *Payload traces* capture entire packets. We have four payload traces collected on two different networks. The first three traces were collected during 2004 and 2005 at the University of Paris 6 network using an optical splitter and a DAG card [9]. We capture data traversing the Gigabit Ethernet Link that connects the university to the Internet. The fourth trace was captured at the edge on an enterprise network.

Packet-header traces only capture the first 64 bytes of every packet, which contain IP and layer-4 headers. We use four different packet-header traces collected in different networks. Two of these traces are available as part of the M2C project [2]. One was captured in 2003 on a 1Gbit/s link between a large college and the Dutch academic and research network (Location #3 in the M2C repository). The other was collected in 2004 on a 1Gbit/s ADSL access network (Location #4). We also study a trace from a wireless network from the Crawdad repository [1]. It was collected at the fall of 2003 on one of the access point of Darthmouth university, and is referred to as ResBldg13 in the repository (this trace is described in detail in [11]). Finally, we use a trace captured at the edge of the network of the University of Massachusetts Amherst campus (described in [25]).

2.2 Filtering Packet Traces

We process the raw traces to extract the packets that are relevant to our analysis. We remove all packets that do not belong to TCP connections. For the traces we study, this step typically removes 30% of flows and 3% of the traffic. We also remove TCP connections that started before the beginning of the traces, because we cannot identify the first packets of these connections (this discards 25% of the flows for one hour traces). For each of the remaining TCP connections, we remove all TCP control packets (SYN, Keep-Alive, or Ack with no data), because these packets do not contain application data.

Table 1 presents a summary of all the traces we used. Connections refer to total number of TCP connections that start during the capture (similarly for the volume of traffic). Our analysis and evaluation focus on TCP connections that have at least four data packets. The last column present the proportion of connections and traffic with at least four packets. Although most traces have only half connections with more than four packets, these connections represent the vast majority of traffic.

Trace	Type	Con.	Vol.	Time	%Con./%Vol.
Paris6-1	Payload	650k	27GB	1h	49%/98%
Paris6-2	Payload	720k	35GB	1h	49%/ 99%
Paris6-3	Payload	510k	28GB	1h	52%/99%
Enter.	Payload	5k	300MB	1h20	46%/85%
College	Header	35k	900MB	0h15	47%/97%
ADSL	Header	70k	2.3GB	0h15	75%/99%
Crawdad	Header	5k	330MB	5h30	62%/99%
Umass	Header	17M	47GB	1h	27%/98%
Bittorrent	Manual	59k	4GB	10h	41%/99.7%
IMAP	Manual	458	8MB	1h	100%/100%
Gnutella	Manual	26	12MB	1h	31%/99.6%
IRC	Manual	473	0.3MB	0h20	35%/46%
LDAP	Manual	151	0.4MB	1h	100%/100%
MSN	Manual	417	4.5MB	1h	93%/99%
Mysql	Manual	59	0.3MB	1h	100%/100%

Table 1: Description of packet traces for connections that started during the capture, and proportion of connection and traffic with more than four packets

2.3 Reference Applications

To calibrate and evaluate our classification method, we need to know the *real* application associated with each TCP connection. The approach to label TCP connections with applications depends on the trace we study. Payload traces contain full application data, which allows the use of payload analysis tools. We use a commercial classification tool called Traffic Designer [20]. This tool searches the application data for a number of pre-defined patterns (similar to SNORT signature search for intrusion detection [23]) to recognize the application. For instance, if a TCP connection on port 21 contains a string like `GET /index.html HTTP/1.1`, then it is classified as HTTP. Standard port classification would mistakenly label it as FTP. Traffic Designer identifies more than 300 applications. Even though this tool works for our offline analysis of payload traces, it is not appropriate for online traffic classification because the signature-matching engine is too complex for high-speed links. Packet-header traces do not allow payload analysis. Therefore, for these traces we limit our study to standard client-server applications, which use standard port numbers, and label connections according to IANA assignments.

Grouping all payload traces we have "enough" TCP connections (as described in 5) of ten applications: NNTP, POP3, SMTP, SSH, HTTPS, POP3S, HTTP, FTP, Edonkey, and Kazaa. In packet-header traces, we focus on the subset of these applications that use standard ports: NNTP, POP3, SMTP, SSH, HTTPS, POP3S, HTTP, and FTP. For FTP we only use control connections, which use port 21.

2.4 Per-Application Traces

We use manually-generated traces to study the behavior of specific applications. There are two approaches to generate these traces: extract packets exchanged by the target application from payload or packet-header traces; or generate traces for the target application in a controlled environment. We use our payload traces to extract traces with packets exchanged by several applications, namely, IMAP, Gnutella, IRC, LDAP, MSN, and Mysql. We select these applications because the number of connections for each of them is too small to include them in the training traces.

We also generate traces for Bittorrent (which was not available in our payload or packet-header traces) in a controlled environment. Before generating the trace, we configured a software firewall to allow only Bittorrent communications. This filtering during the collection time ensures that the trace does not contain any other application. We used Bittorrent to download a large Linux distribution and left it available for other users. During this period we run tcpdump to capture all packets sent by the machine. We also summarize all manual traces in table 1.

3. CONNECTION-LEVEL FEATURES

Previous studies propose to use different features of packets, flows, or connections to classify traffic. We start by a discussion of the methodology adopted by these studies, then we evaluate whether they are appropriate for identifying the application associated with a TCP connection early.

3.1 Prior Work

Since port-based classification is not always efficient and given the limitations of searching payloads for signatures, there has been a new trend to classify traffic based on summarized flow information such as duration, number of packets and mean inter-arrival time [21], [16], [26], [17]. BLINC [15] introduces a new approach for traffic classification. It associates Internet hosts with applications. Instead of studying TCP (or UDP) flows individually, it looks at all the flows generated by specific hosts. BLINC is able to accurately associate hosts with the services they provide or use (application server, web client, etc.). However, it cannot classify a single TCP connection.

All classification techniques that use flow statistics can only identify the nature of a flow when the flow is finished. BLINC has to gather information from several flows for each host before it can decide on the role of a host. These requirements prevent the use of these methods online. In contrast, our method relies only on the first few packets of a TCP connection. This early classification is essential to allow automatic blocking, filtering, or recording of specific applications. It also limits the amount of memory required to store information associated with each connection.

3.2 Adaptability to Early Classification

All previous works only classify a flow when it is finished. Although some of the classification metrics they propose need information of the complete flow (such as duration and number of packets), nothing prevent us to apply metrics like mean packet size or mean inter-arrival time to just a subset of the packets in a connection. In particular, applying these metrics to just the first few packets of a connection can enable early classification.

In [5], we argue that the first four packets of a TCP connection are enough to identify the application. We now evaluate the classification power of the metrics used in previous studies when restricted to the first four packets of a TCP connection. We characterize mean packet size, variance of packet size, mean inter-arrival time (IAT), and mean jitter of the first four packets of TCP connections for all traces presented in Section 2. Figure 1 presents the results of our analysis for the most common applications for the Paris6-1 trace. Each plot corresponds to a metric and presents its mean (as a point) and standard deviation (as an horizontal bar) per application. A metric is effective to distinguish a set of applications if its value is distinct for all applications in the set. In these plots, we identify that a metric is good when the horizontal bars do not overlap, i.e., the value range of the metric is different between applications.

We see that the range of values for both features based on arrival time (i.e., mean IAT and mean jitter) do not help to distinguish among applications. The range values of IAT and jitter overlap for all applications. In contrast, features related to packet sizes do distinguish among groups of applications. The mean packet size separate applications into four groups: (i) POP3s, HTTPS, and SSH; (ii) SMTP; (iii) POP3 and NNTP; and (iv) HTTP and FTP. The mean payload size of SMTP connections varies from 30 to 150 bytes, which is different from all other applications. Adding the information of the variance of packet size helps distinguish HTTP from FTP, but not the other applications.

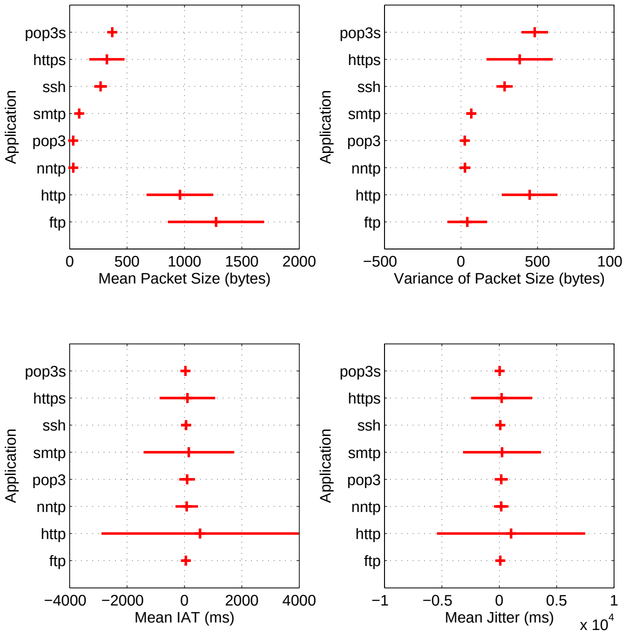


Figure 1: Comparison of classification metrics for the most common applications for Paris6-1 trace

Instead of computing the mean and the variance of packet sizes, we study the size and direction of each of the first four packets. Figure 2 presents the mean and standard deviation of each packet size for five traces[1]. This figure shows that some packets have a very precise size and direction (for instance, the first packet of POP3s is always between 100 and 120 bytes, whereas others present more variation (all HTTP packets). The mean and the variance of packet sizes combine precise and imprecise values, which leads to an imprecise value. By using each packet size and direction separately we gain precision to distinguish applications. Even the information that the size of a particular packet has a large standard deviation is useful to distinguish applications.

These plots lead to two important observations. First, the size and direction of each packet adds more information to distinguish applications than arrival time related metrics. Intuitively, the size of the payload of the first four packets captures the application's negotiation phase, which is usually a pre-defined sequence of messages and distinct among applications. Second, the range of packet sizes for each application is similar across traces. This observation implies that if we extract models of applications from packet traces collected at one network, these models can be used to classify the same set of applications at another network.

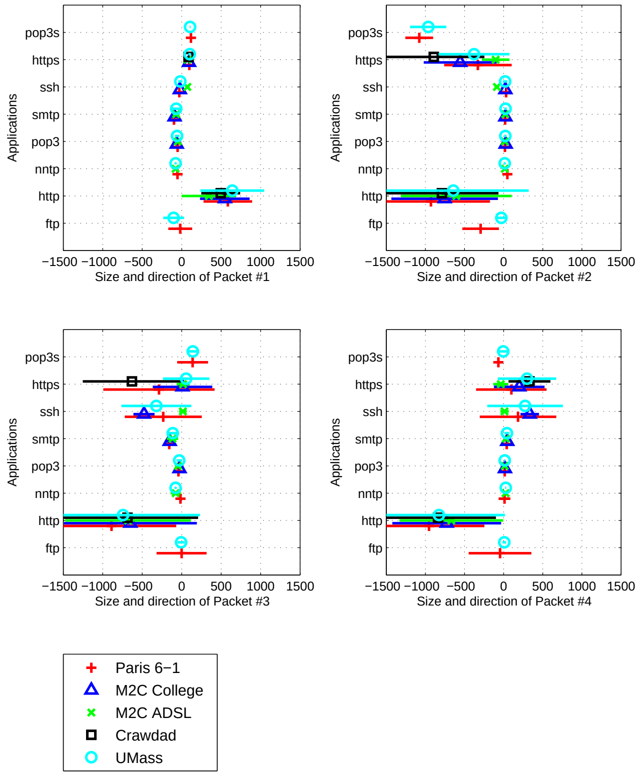


Figure 2: Comparison of the sizes of the first 4 packets for different applications on several networks

4. CLASSIFICATION APPROACH

Based on the observations from the previous section, we propose a traffic classification mechanism that works in two phases: a training phase and a traffic classification phase. We describe the training and classification phases in detail in Sections 5 and 6, respectively. Figure 3 presents an overview of our method. The left side represents the steps of the training phase and right side the components of the classifier. These two phases run at separate locations and timeframes. The training phase runs offline at a management site, whereas the classifier runs online at at a management host that has online access to packet headers or in a network processor at the monitored router.

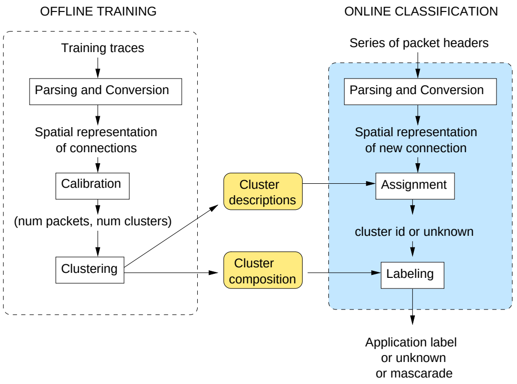


Figure 3: Approach overview

The training phase obtains models of application behaviors by applying clustering techniques to a trace that contains a representative sample of TCP connections from all target applications. First, we parse this trace and convert each connection into a spatial representation based on the sizes of its first P packets. Then, we calibrate our clustering algorithm. This step searches for the number of clusters and packets that give the best clustering. Finally, we run a clustering algorithm that finds groups of TCP connections that have similar behavior. We evaluate three well-known algorithms: k-means, Gaussian mixture model, and spectral on HMMs. The training phase outputs two sets that are later used in the classification phase: one set that contains the description of each cluster and another the applications that are present at each cluster.

Our classifier takes as input the series of packet headers for both directions of an edge link. The parsing and conversion module extracts the 5-tuple (protocol, source IP, destination IP, source port, destination port) and the packet size. The analyzer filters out control traffic (the three packets of the TCP handshake) and stores the size of every packet in both directions of the connection. When it has the size for the first P packets of the connection, it sends this information to the assignment module which associates the connection with a cluster based on cluster descriptions. Based on the composition of this cluster the labeling module selects which application is most likely associated with the connection.

5. OFFLINE TRAINING PHASE

In this section, we explain how to extract cluster descriptions and compositions from training traces. We describe how to construct the data to train our classifier. Then, we discuss the conversion of TCP connections to a spatial representation. Finally, we explain the clustering algorithms and how to calibrate them.

5.1 Training Traces

The training traces are the input for the clustering algorithm. There are several requirements regarding these traces

[1] Since each trace does not contain all applications, some values are missing. For packet-header traces, we determine the application using port numbers as described in section 2.3.

to obtain clusters that model well the target applications. First, they need to contain all target applications. In addition they need to have a minimum number of connections for each applications. Finally, the number of connections for each application needs to be similar, otherwise the most prevalent applications would bias the clustering.

There are two methods to create a training data set: extraction of target applications from packet traces or manual generation of traces for each application. Manual generation has several advantages: we do not need access to packet traces and we know exactly what application generated each connection. However, manually generated traces can take long to create and may not capture all possible modes of operation of the applications. On the other hand, packet traces from the studied network contain a representative sample of the applications used on the network.

Considering this requirements and possibilities, we decided to use connections from the three packet traces from the Paris 6 network. Even though our traces have more than 50 applications total, three applications account for 75% of the connections: HTTP, FTP and NNTP (Paris-6 house the largest GNU mirror in France, and a very large News server). Some applications have very few connections, which would not be enough to create a representative model. We analyzed the number of connections per application for our payload traces and found that the best trade-off is to select the ten applications that have more than 500 connections. For applications that have more connections, we randomly select 500 of them.

5.2 Representation of a TCP connection

To detect applications behaviors, we need a representation of each connection and a measure of similarity between the representations of two connections. Section 3 shows that the size of the first packets of a connection is a good representation of application behaviors. This section describes two different connection representations based on the sizes of the first packets.

Let x be a connection and $\mathcal{U}$ the set of connections in the training set. The most natural way of representing a connection x according to the sizes of its first P packets is to associate it with a vector in a Euclidean P-dimensional space. Let $|s_i(x)|$ be the payload size of packet i in connection x. We take into account the direction of the packet, so s_i is positive for packets sent by the TCP client and negative for packets sent by the TCP server. Let ϵ be the function that transforms a connection into this representation: $\epsilon : \mathcal{U} \to \mathbb{Z}^P$ and $\epsilon(x) = (s_1(x), \ldots, s_P(x))$. To measure the distance between two connections we use the classical l^2-norm: $d_e(\epsilon(x), \epsilon(x')) = \|\epsilon(x) - \epsilon(x')\|$.

This representation is simple but does not take into account the order of the packets: a permutation of the P dimensions of such a space would have no influence on the clustering results. This limitation led us to evaluate a more complex representation based on Hidden Markov Models (HMM) [22]. HMMs have been widely used to describe sequence of symbols. The idea behind HMMs is that the observed symbol at a given time can be explained as a random function of an unobserved internal state, which follows a Markov chain. In our case, we can only observe the sequence of packet sizes, which depends on a hidden application context. There are several ways to capture connection behaviors with HMMs. We focused on a specific structure proposed in [6], [19] where each state corresponds to a packet in the connection. Each connection x is then summarized by a single HMM with P states.

The generic method for evaluating the similarity between HMM-based sequences is given in [22]. Let ρ be the function that transforms the size of the first packets of a connection x into a sequence of symbols $\rho(x)$, and η the transformation that associates this sequence with an HMM $\eta(\rho(x))$ that represents connection x. To derive a distance matrix from these representations, we first compute a log-likelihood matrix $\mathbb{L}$ where $\mathbb{L}(\rho(x), \rho(x'))$ is the log-likelihood that sequence $\rho(x')$ has been generated by HMM $\eta(\rho(x))$, i.e. $\mathbb{L}(\rho(x), \rho(x')) = -\log \mathbb{P}(\rho(x')|\eta(\rho(x)))$. We obtain these log-likelihoods using the Baum-Welch algorithm [3]. Based on this matrix we define the distance:

$$d_h(x, x') = \sqrt{\sum_{x'' \in \mathcal{U}} \|\mathbb{L}(\rho(x''), \rho(x)) - \mathbb{L}(\rho(x''), \rho(x'))\|^2}.$$

Due to space limitation we explain the details of this method in [4].

5.3 Clustering Algorithms

Depending on the representation, we need to use different clustering algorithms. In this section, we first describe two algorithms for the Euclidean Space and another algorithm for the HMM based space.

5.3.1 Clustering in the Euclidean Space

The Euclidean representation of connections leads to a P dimensional space where $P \leq 10$. The K-Means algorithm is a method that is often used to find clusters in such spaces. This algorithm finds the set $\mathcal{C}_k$ of K_k clusters that minimizes the sum of the distances between each connection representation and the centroid of its closest cluster.

The K-Means algorithm is an iterative one. It initiates by randomly choosing K_k cluster centroids and involves two steps. First, it assigns each connection to the closest cluster. Then, it computes the centroid of each cluster based on the connections assigned to it. It repeats these steps until it finds a minimum.

This algorithm is simple and computationally efficient. However, it has two limitations. First, the selection of initial centroids is poor, K-Means can converge to a sub-optimal solution. Second, the resulting clusters are always spherical. In our population, there is no reason for clusters to have such shapes: the variation of the sizes of packet #1 can be very different from the variation of the sizes of other packets as shown in figure 2.

A more sophisticated clustering method relies on Gaussian Mixture Models (GMM). In this case, we assume that each P-dimensional vector $\epsilon(x)$ in the Euclidean space has been generated according to a probability distribution function f. This distribution consists of a mixture of K_g P-dimensional Gaussian distributions: $f = \sum_{i=1}^{K_g} \mathcal{N}(c_i, \Sigma_i)$. Cluster i is characterized by a P-dimensional Gaussian with mean (i.e. center) c_i and covariance matrix (i.e. shape) Σ_i. We limit our analysis to diagonal covariance matrices because the correlations between the sizes of the first packets are small and this choice simplifies the clustering algorithm and makes our classifier faster, as explained in section7.4. Achieving a clustering using GMM consists in finding the values of c_i and Σ_i that maximize the likelihood of generating the vectors in $\epsilon(\mathcal{U})$. We achieve this optimization with the Expectation Maximization algorithm [8].

5.3.2 *Clustering HMM Sequences*

Modelling the training set with HMMs leads to N-dimensional space, with N the number of connections in our training set (around 5000 in our case). K-Means and GMM are not efficient on high-dimensional spaces. Clustering in such spaces is commonly achieved through spectral clustering ([18], [10]).

Spectral clustering transforms the distance matrix into a block diagonal matrix, where each block can be interpreted as a cluster. Given this block diagonal matrix, extracting clusters is done with an eigen-decomposition. Each of the K_h largest eigenvalues correspond to the K_h clusters. Points belonging to the subspace spanned by the eigenvectors corresponding to block i will be assigned to cluster i. For a detailed explanation of this method please refer to [4].

5.4 Calibration

The clustering algorithm relies on two important parameters: the number of clusters and the number of packets. We select these parameters by spanning the parameter space and selecting the solution that maximizes clustering quality.

5.4.1 *Clustering quality metric*

We choose to measure the quality of a clustering (C) by comparing it to the "optimal“ clustering (i.e. a clustering that has one cluster per application, computed using application labels obtained as described in 2.3). We used a metric commonly used in clustering literature: Normalized Mutual Information (NMI) [24].

Let X be a random variable representing the distribution of application labels and Y be a random variable representing the distribution of cluster labels. In order to compute $NMI(X, Y)$ we first compute the mutual information between X and Y as:

$$MI(X,Y) = \sum_{i,j} p_{i,j} log(\frac{p_{ij}}{p_i p_j}),$$

where p_{ij} is the probability that a connection in cluster j belongs to application i, p_i is the probability of application i and p_j is the probability of cluster j. $MI(X, Y)$ measures the shared information between X and Y. Since this value is not bounded by the same constant for all data sets, we normalize it between 0 and 1 :

$$NMI = \frac{MI}{\sqrt{(H(X)H(Y))}},$$

with $H(X)$ and $H(Y)$ the entropy of X and Y. NMI is bounded between 0 and 1. When $NMI(X, Y) = 1$ we have a one to one mapping between applications labels and cluster labels. Many applications have more than a single behavior (FTP for instance consists of control connections and of data connections), and some applications may share a similar behavior. As a consequence, we cannot obtain NMI=1 for our clustering, but a higher NMI corresponds to a better clustering.

5.4.2 *Number of clusters*

We use NMI to choose the number of clusters for each clustering algorithm. For the K-Means algorithm, we run K-Means on our training data using 5, 10, .., 100 clusters and compute the NMI between the resulting clusters and the optimal clustering. Figure 4 presents NMI for two, three and four packets. This figure shows that the best number of clusters for three packets is 50. Increasing from five to 40 clusters leads to a sharp increase in NMI, while using more than 50 does not improve it. This result indicates that having more than 50 clusters does not help to separate applications. However, we also want to have a minimum number of clusters because it determines the speed of the classification engine (describe in section 6). Therefore, we look for the "knee" of the NMI curve, which represents the best trade-off between clustering quality and number of clusters. Using this methodology we choose 40 clusters for three packets: $K_k(3) = 40$.

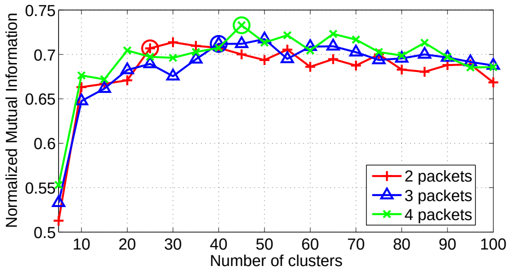


Figure 4: Influence of the number of clusters on clustering quality (K-Means)

We apply the same methodology to determine the best number of clusters for GMM and HMM-based clusterings and plot the best number of clusters for different number of packets in figure 5. This number increases with the number of packets because a larger number of packets leads to an increasing number of behaviors. HMM requires a lower number of clusters because its connection model is richer. It takes into account sequence information and do not need as many clusters to model our training data set.

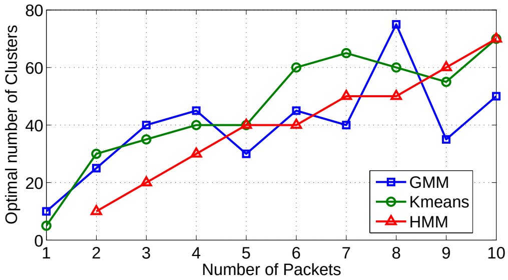


Figure 5: Optimal number of clusters according to the number of packets

5.4.3 *Number of packets*

To find the best number of packets (P), we apply the same methodology we used to determine the optimal number of clusters, varying the number of packets instead of the number of clusters. The influence of the number of packets on the quality of the clustering evaluated by the NMI metric is presented in 6. Using a 4-dimensional space gives best NMI for the three clustering algorithms. NMI decreases for higher numbers of packets because adding more packets brings in noise: packets exchanged after the application negotiation phase are not standardized and their sizes no longer help to recognize the application. Based on these results, we use $P = 4$ packets for the three clustering methods and $K_h = 30$ for HMM, $K_k = 40$ for K-Means and $K_g = 45$ for GMM.

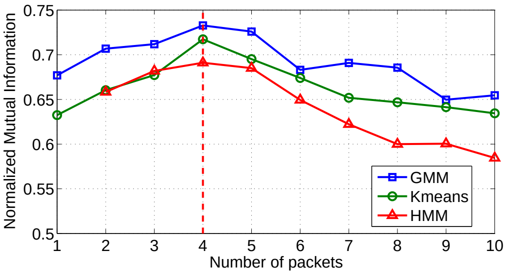


Figure 6: Influence of the number of packets on clustering quality

We evaluate the expressiveness of each clustering by studying the composition of resulting clusters. Figure 7 presents the proportion of clusters with one to four applications (no cluster contain more than four) for the three clustering methods. The ideal clustering for our purposes would only have one application per cluster. HMM is the closest to the ideal because of its richer representation. The more precise cluster shapes of GMMs outperform the simple spherical shapes induced by K-Means.

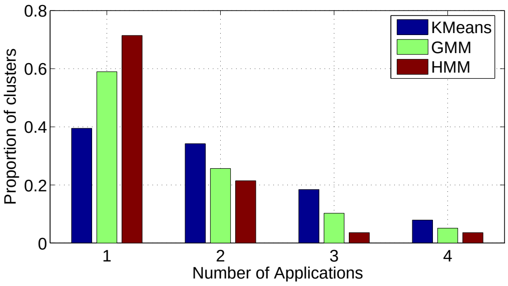


Figure 7: Proportion of clusters with N Applications (four packets)

6. CLASSIFICATION PHASE

We use the resulting clusters to classify connections observed online in two steps: assignment and labeling.

6.1 Assignment Heuristics

The assignment heuristics depend on whether we use the Euclidean or HMM representation of connections and on the clustering algorithm.

6.1.1 K-Means

A simple model to summarize clusters obtained with K-Means it to use the center of the clusters. To assign a connection to a cluster i with this model we use a heuristic called **K-Means Center**. We define an assignment function α_{kc} which associates a connection x to the cluster with the closest centroid. Let c_i the centroid of cluster i:

$$\alpha_{kc}(x) = \underset{i \in 1..K_k}{\operatorname{argmin}} \, d_e(\epsilon(x), c_i).$$

Finding cluster centers and evaluating α_{kc} involves little computation. However, this heuristic is too naive: it does not take into account cluster dispersion. Some clusters are very tight (for instance the clusters resulting from the representation of POP3 connections in figure 2), whereas others are more disperse because the size of packets vary more (for HTTP connections for instance). Our analysis (not presented here), shows that the distribution of distances between connection representations and cluster centers follows a distribution close to a Normal one (small distances to the center are more likely than larger ones). Therefore, we represent a cluster by its center and by the variance of the distances to the center. This extension of the K-Means Center heuristic consists in a simple GMM approximation with spherical clusters.

Using this normal distribution model we can now evaluate the probability that a connection x belongs to cluster i with center c_i and variance of distances to cluster center σ_i^2:

$$\mathbb{P}_k(i, \epsilon(x)) = \mathcal{N}(0, \sigma_i^2)(d(c_i, \epsilon(x))).$$

To determine to which cluster $\alpha_{kp}(x)$ a connection x should be assigned with this model we use a heuristic called **K-Means Proba**, and define this assignment function:

$$\alpha_{kp}(x) = \underset{i \in 1..K_k}{\operatorname{argmax}} \, \mathbb{P}_k(i, \epsilon(x)).$$

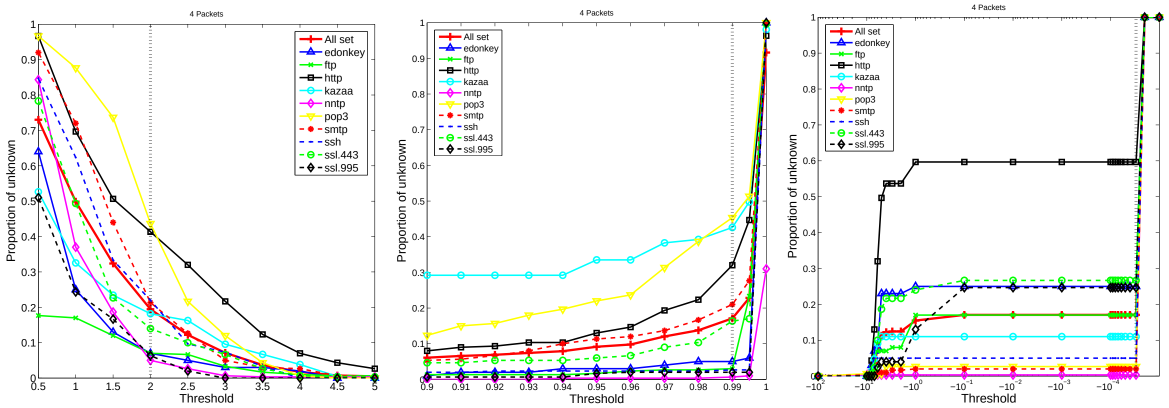
This heuristic will always associate a new connection to a cluster even if the probability to belong to any cluster is very small (which indicates that the connection belongs to an unknown behavior). Therefore, we introduce a threshold on the probability to belong to clusters, which will enable us to label as *unknown* connections that have a small probability to belong any cluster. Since we are using a normal distribution of the distances to the center of the cluster, we can translate the threshold on the probability to belong to a cluster into a threshold T_k on the distance to the center: if $d_e(c_i, \epsilon(x)) > T_k * \sigma_i^2$ the connection does not belong to cluster i.

To determine the best threshold, we apply our assignment heuristic to our training data set with different values for T_k. Figure 8(a) shows the proportion of training connections that are labeled as unknown for different T_k. When T_k increases, clusters become larger and the proportion of unknown traffic decreases (but the probability of finding a new application decreases). The choice of the threshold depends on the intended use of the classification method, which can be to correctly identify all connections from known applications or to pinpoint unknown applications. We choose the smallest threshold that allows a good detection of known traffic. This corresponds to the knee of the "all set“ curve in the graph, i.e. $T_k = 2$.

To assign a connection to a cluster with the probabilistic model using a threshold we define the assignment function α_{kt} (called the **K-Means Thresh** heuristic):

```
If $\nexists i \in 1..K_k$ such that $d_e(c_i, \epsilon(x)) < T_e * \sigma_i^2$
    Label x as unknown
else $\alpha_{kt}(x) = \operatorname{argmax}_{i \in 1..K_k} \mathbb{P}_k(i, \epsilon(x))$
```

If the proportion of unknown connections increases over a period of time, it may indicate an evolution of the traffic and the network administrator can rerun the learning phase to adjust the cluster descriptions to the new traffic patterns.



(a) K-Means Probabilistic (b) GMM (c) HMM

Figure 8: Choice of Threshold

6.1.2 *Gaussian Mixture Model*

Using our Gaussian Mixture Model, we can compute the a posteriori probability (i.e. the probability that a connection x belongs to a Gaussian mixture element i with center c_i and covariance matrix Σ_i):

$$\mathbb{P}_g(k, \epsilon(x)) = \frac{\mathcal{N}_P(c_i, \Sigma_i)(\epsilon(x))}{\sum_{i=1}^{K_g} \mathcal{N}_P(c_i, \Sigma_i)(\epsilon(x))}.$$

Based on these probabilities, the **GMM Proba** heuristic associates x with a Gaussian element using a maximum likelihood criterion. We define the following assignment function:

$$\alpha_{gp}(x) = \operatorname*{argmax}_{i \in 1..K_g} \mathcal{N}_P(c_i, \Sigma_i)(\epsilon(x)).$$

To detect unknown traffic using this model, we use a probability threshold T_g and define the **GMM Thresh** heuristic:

```
If $\nexists i \in 1..K_g$ such that $\mathbb{P}_g(i, \epsilon(x)) > T_g$
    Label x as unknown
else $\alpha_{gt}(x) = \text{argmax}_{i \in 1..K_g} \mathcal{N}_P(c_i, \Sigma_i)(\epsilon(x))$
```

To choose the threshold T_g we apply a similar method as the one presented in 6.1.1: we study the proportion of unknown among training connections based on different values of T_g and choose the value that corresponds to a knee. The second plot in figure 8(b) shows these proportions, and based on this graph we choose $T_g = 99\%$.

6.1.3 *Hidden Markov Model*

We model each cluster i in the HMM space by a representing HMM h_i, which is the HMM that is the most likely to have generated the sequences assigned to the cluster. We obtain h_i by fitting an HMM on all connections in cluster i with the Expectation-Maximization algorithm. We can now compute the log-likelihood that sequence $\eta(x)$, has been generated by h_i: $\mathbb{L}_h(h_i, \eta(x))$.

To associate a connection to a cluster k, we define the **HMM likelihood** heuristic, which uses a maximum likelihood criterion:

$$\alpha_{hl} = \operatorname*{argmax}_{i \in 1..K_h} \mathbb{L}_h(h_i, \eta(x)).$$

To detect unknown traffic, we apply the same method used with GMM. We define a threshold T_h and the **HMM Thresh** heuristic:

```
If $\nexists i \in 1..K_h$ such that $\mathbb{L}_h(h_i, \eta(x)) > T_h$
    Label x as unknown
else $\alpha_{ht}(x) = \text{argmax}_{i \in 1..K_h} \mathbb{L}_h(h_i, \eta(x))$
```

Figure 8(c) shows the proportion of unknown training connections for different T_h. This plot shows two knees for $T_h = -10$ and $T_h = -3e^{-5}$. However, the first value of T_h corresponds to a very low likelihood and almost all connections will be assigned to a cluster with this threshold. Therefore we choose $T_h = -3e^{-5}$.

6.2 Labeling Heuristics

When the cluster assignment heuristic finds a cluster for a connection, the next step is to label it with an application. Although figure 7 shows that a cluster may consist of several applications, most clusters contain only one. We propose a naive labeling heuristic called **Dominant**. This heuristic labels connections with the dominant application in their assigned cluster. Let $\lambda_d(x)$ be the function that associates a label to connection x according to this heuristic, $\mathcal{A}(i)$ the set of applications present in cluster i, $\pi(i, a)$ the proportion of connections associated to cluster i from application a, and α an assignment functions:

$$\lambda_d(x) = \operatorname*{argmax}_{a \in \mathcal{A}(\alpha(x))} \pi(\alpha(x), a).$$

This heuristic is simple but it will misclassify all connections from non-dominant applications when clusters have more than one. Therefore, we explore other information present in the first four packets that might help labeling connections. Although port numbers by themselves cannot classify all applications, many applications still rely on IANA assignments. Typically, client-server applications use a standard port to guarantee that anyone can access the service. We introduce a hybrid heuristic called **Cluster & Port** that uses ports when they are meaningful. Let λ_c be the labeling function for this heuristic, *port*(x) the server port used by connection x, $\mathcal{S}$ the set of ports corresponding to standard client-server applications (for the applications we study $\mathcal{S}$ ={21, 22, 25, 80, 110, 119, 443, 995}, and *std* the function that associates a standard port with an application label (for this study *std*$(\mathcal{S})$ ={FTP, SSH, SMTP, HTTP, POP3, NNTP, HTTPS, POP3S}). The Cluster & Port heuristic is the following:

```
If $port(x) \in \mathcal{S}$
```

```
    If $std(port(x)) \in \mathcal{A}(\alpha(x))$
        $\lambda_c(x) = std(port(x))$
    Else  $\lambda_c(x) =$ masquerade
Else
    If $|\mathcal{A}(\alpha(x)) \setminus \mathcal{A}(\mathcal{S})| = 0$
        $\lambda_c(x) =$ masquerade
    Else  $\lambda_c(x) = \text{argmax}_a(\pi(\alpha(x), a), a \in \mathcal{A}(\alpha(x)) \setminus \mathcal{A}(\mathcal{S})$
```

If x is using a standard port, and if the application associated to this port is part of the cluster, the connection is labeled with this application. Otherwise, the connection uses a standard port but does not behave accordingly. We choose to flag such connections as *masquerade*, because they typically can be dangerous connections using ports associated to "safe" applications to bypass firewall rules.

If the port is not standard, the connection is labeled with the dominant application among those that do not use standard ports. If the cluster consists of standard services only, we flag the connection as *masquerade* because it is probably a standard service using a non-standard port.

7. EVALUATION

In this section, we use the traces presented in table 1 to evaluate all assignment and labeling heuristics.

7.1 Assignment Accuracy

Assignment accuracy measures the proportion of test connections that are assigned to clusters which contain the actual application of the connection. Let $\mathcal{T}$ be a set of test connections and $\mathcal{T}(\alpha) \subset \mathcal{T}$ the subset of connections assigned to a cluster using heuristic α (i.e., all connections not labeled unknown). Let $\gamma(x)$ be the actual application of connection x (obtained as described in section 2.3). We define an indicator function $\beta(x)$ that verifies the assignment of connection x ($\beta(x) = 1$, if $\gamma(x) \in \mathcal{A}(\alpha(x))$; and $\beta(x) = 0$, otherwise). We define the assignment accuracy of heuristic α as follows:

$$\omega(\alpha) = \frac{\sum_{x \in \mathcal{T}(\alpha)} \beta(x)}{|\mathcal{T}(\alpha)|}.$$

Table 2 compares assignment accuracy for the different clustering methods and the different assignment heuristics for our test traces. Assignment accuracies are above 95% for all heuristics (for HMM and GMM these accuracies are even above 99%) except for the M2C ADSL trace. All other traces are from university networks. An ADSL commercial network may include different modes of operation for certain applications. Depending on the method, the use of thresholds lead to comparable accuracies, but different proportions of unknown traffic. For instance, there is around 50% of unknown for HMM Thresh, 40% for K-Means Thresh, and 30% for GMM Thresh. Intuitively, adding a threshold should increase accuracy. However, the results in Table 2 do not show a significant increase in accuracy. This result indicates that the cluster assignment in our test traces (which are different and larger than the training trace) are similar to that of the training trace. The test traces have no new modes of operation of the applications that could be misclassified without thresholds.

7.2 Labeling Accuracy

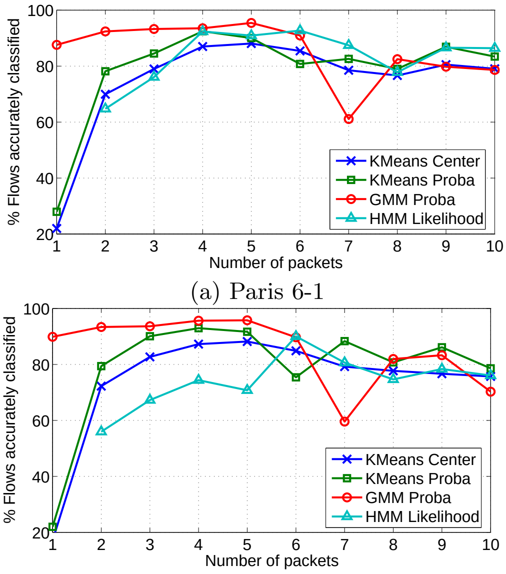
Labeling accuracy is the proportion of test connections correctly classified. A connection x is accurately labeled using heuristic λ if $\lambda(x) = \gamma(x)$. Let $\psi(x)$ be the indicator function that verifies the label of connection x ($\psi(x) = 1$, if $\lambda(x) = \gamma(x)$; and $\psi(x) = 0$, otherwise). We define the overall accuracy of a labeling λ as

$$\omega(\lambda) = \frac{\sum_{x \in \mathcal{T}(\alpha)} \psi(x)}{|\mathcal{T}(\alpha)|}.$$

Let $\mathcal{T}_a \subset \mathcal{T}$ such that $\gamma(x) = a$, $\overline{\mathcal{T}_a} = \mathcal{T} \setminus \mathcal{T}_a$, and $\phi_a(x)$ an indicator function defined as $\phi_a(x) = 1$, if $\lambda(x) = a$; and $\phi_a(x) = 0$, otherwise. We define the proportion of true positives TP and false positives FP for application a:

$$\text{TP}(a) = \frac{\sum_{x \in \mathcal{T}_a} \phi_a(x)}{|\mathcal{T}_a|}, \text{FP}(a) = \frac{\sum_{x \in \overline{\mathcal{T}_a}} \phi_a(x)}{|\overline{\mathcal{T}_a}|}.$$

Figure 9 compares the overall accuracy for the different assignment heuristics without thresholds for two different traces using the Dominant labeling heuristic. This figure shows that models based on four packets perform the best (which confirms the results obtained with the NMI metric in Section 5.4) and that GMM outperforms the other models. For all assignment heuristics with four packets, we achieve an overall accuracy above 85% except for the HMM model on the enterprise trace. This lower result can be explained by looking at how the different applications were classified: with the HMM model, all POP3 connections are labeled NNTP because both applications are present in a cluster where NNTP predominates.



(a) Paris 6-1

(b) Enterprise

Figure 9: Overall Accuracy (Dominant)

Table 3 compares overall accuracies for our two labeling heuristics. It shows that using the Cluster & Port heuristic, GMM and HMM achieve an overall accuracy above 98% (POP3 is no longer mistaken for NNTP with the HMM model).

Overall accuracy can be misleading. If a test set is composed mainly of one application, this metric will only evaluate the accuracy of classifying this applications. To avoid this limitation, table 4 presents per-application metrics. This table shows a detailed comparison of both labeling heuristics for the Paris 6-1 trace (the result for the other traces

Model	K-Means			GMM		HMM	
Heuristic	Center	Proba	Thresh Tk=2	Proba	Thresh Tg=99%	Likelihood	Thresh Th=$-3e^{-5}$
	Accuracy	Accuracy	Accur. (Unknown)	Accuracy	Accur. (Unknown)	Accuracy	Accur. (Unknown)
P6-1	95.6%	98.8%	99.7% (37.9%)	99.6%	99.7% (32.0%)	99.6%	99.8% (53.9%)
P6-2	95.8%	98.7%	99.7% (37.8%)	99.6%	99.8% (28.5%)	99.8%	99.9% (53.2%)
P6-3	95.8%	97.7%	98.8% (38.4%)	99.6%	99.7% (30.7%)	99.3%	99.9% (50.8%)
Enterprise	95.5%	97.9%	99.3% (31.8%)	99.3%	98.8% (41.7%)	99.2%	99.2% (42.3%)
M2C College	97.3%	99.7%	99.9% (41.3%)	99.5%	99.7% (29.6%)	99.97%	99.98% (56.0%)
M2C ADSL	68.9%	71.3%	65.5% (35.7%)	92.1%	91.1% (38.0%)	78.3%	67.1% (41.4%)
Crawdad	96.8%	99.7%	99.8% (41.8%)	99.9%	99.9% (22.2%)	99.9%	99.7% (64.4%)
UMass	94.6%	95.1%	98.6% (47.6%)	98.3%	98.8% (29.1%)	98.9%	98.6% (54.7%)

Table 2: Assignment Accuracy and proportion of unknown traffic (four packets)

		K-Means	GMM	HMM
Heuristic	Trace	Proba	Proba	Likelihood
Dominant	P6-1	92.4%	93.5%	92.4%
Dominant	Enter.	93.0%	95.6%	74.4%
Cluster&Port	P6-1	97.7%	98.5%	98.4%
Cluster&Port	Enter.	97.7%	99.1%	98.8%

Table 3: Accuracy of labeling heuristics (4 packets)

are similar). For each application a in our data sets it shows TP(a) and FP(a). For instance, the Dominant heuristic and GMM Proba assignment labels 94.1% of Edonkey connections correctly (True Positives), 5.9% as other applications, and 0.3% of non-Edonkey connections are labeled Edonkey (False Positives). For the Cluster & Port heuristic the table also shows the proportion of connections that were labeled masquerade connections.

As expected, the Clusters&Port heuristic outperforms Dominant. Moreover, adding the port number increases the accuracy even for applications that do not use standard ports. This improvement is due to connections that are assigned to a cluster shared by an application in S and an application in $\overline{S}$. For such clusters, the heuristic chooses the application in $\overline{S}$ for connections not using standard ports. The False Positives are low for most applications. We examine all applications with low accuracies. The HMM model with the Dominant labeling heuristic is not able to recognize POP3 connections. As explained before, POP3 and NNTP share similar handshakes in terms of packet sizes and are assigned to a same cluster where NNTP connections dominate in our training set. Therefore all POP3 connections are assigned to this cluster and labeled NNTP, as shown by the high level of False Positives associated with NNTP (3.6%). Similarly, the True Positives are low for Kazaa because 30% of Kazaa connections are labeled HTTP, and for HTTPS (25% of HTTPS connections are labeled HTTP and 20% POP3s). Some mode of operations of these applications are assigned to clusters where they do not dominate. However, these misclassifications disappear when we use the Cluster&Port heuristic. Finally, the high level of masquerade connections for SSH has two explanations: (i) half of masquerade SSH connections correspond to behaviors of SSH that are not present in our training traces; and (ii) the other half to SSH connections not using the standard SSH port (22).

7.3 Detection of new applications

To test the efficiency of the thresholds presented in section 6.1 to detect new applications, we use manually generated traces with applications not present in the training set: Bittorrent, IMAP, Gnutella, IRC, LDAP, MSN and Mysql. Table 5 shows the labeling accuracy with thresholds. We report both for new applications and for applications from the training set. This table presents two additional metrics: Unknown and False Negatives:

$$\text{Unknown}(a) = 1 - \frac{|\mathcal{T}_a(\alpha)|}{|\mathcal{T}_a|}, \text{FN}(a) = \frac{\sum_{x \in \mathcal{T}_a} 1 - \psi(x)}{|\mathcal{T}_a|},$$

where $1-\psi$ is an indicator function for misclassified connections. In this table, FP+FN+Unknown+Masquerade=100% for each application. Therefore, low True Positives do not necessarily involve a high misclassification. For instance, only 60.8% of SMTP connections are accurately labeled, but only 1.2% are misclassified. Among SMTP connections not labeled unknown, 98% are accurately identified. With the thresholds defined in section 6.1, the overall proportion of connections labeled unknown among known applications is around 30% for GMM (50% for HMM) while about two third of unknown traffic is labeled unknown for both models. This proportion varies greatly between applications. Some of them are almost never misclassified, while some others are in a large proportion. For instance, for the GMM model 100% of Gnutella traffic is classified Kazaa, and 40% of MSN connections are classified Kazaa or Edonkey. In future works, we plan to improve the detection of new behaviors to decrease the proportion of unknown among applications our classifier is designed to recognize and to increase this proportion for new traffic.

7.4 Complexity of Online Classification

This section compares the complexity of online classification using different assignment and labeling heuristics. As a reference, we also analyze the running time of port-based and payload-based classification. At running time, any of these classifiers needs to analyze the header of all incoming packets to assign the packet to a connection. Subsequent treatments depends on the classifier.

Port-based classification is very simple because it does not involve any other computation than the 5-tuple lookup. It only requires to look up the port number in a list of predefined ports with the associated applications.

Payload analysis tools rely on substring matching algorithms [7]. In the best case, these algorithms are known to have a running time in the order of n/m comparisons where n is the length of the searched text and m the length of the searched string. For traffic classification, the searched text is the TCP payload. In our data set the average size TCP payloads of data packets (i.e. packets containing actual application data) is 600 bytes, which is consistent with data from a large network [13]. Considering an average application signature of four characters (HTTP for instance),

App	GMM without Threshold					HMM without Threshold				
	Dominant		Clusters & Ports			Dominant		Clusters&Port		
	TP	FP	TP	Masq	FP	TP	FP	TP	Masq	FP
NNTP	81.2%	0.0%	99.3%	0.2%	0.0%	99.8%	3.6%	99.3%	0.2	0.0%
POP3	96.5%	0.7%	99.7%	0.3%	0.0%	0.0%	0.0%	98.2%	1.8	0.0%
SMTP	90.1%	0.1%	98.5%	1.3%	0.0%	83.6%	0.2%	96.4%	3.3	0.0%
SSH	89.4%	0.0%	91.7%	8.3%	0.0%	89.6%	0.0%	93.8%	6.3	0.0%
HTTPS	60.1%	0.0%	97.8%	2.2%	0.0%	53.1%	0.0%	99.1%	0.9	0.0%
POP3S	93.4%	1.1%	100.0%	0.0%	0.0%	96.4%	1.1%	100.0%	0.0	0.0%
HTTP	96.2%	1.3%	98.5%	0.3%	0.0%	99.0%	1.7%	98.6%	0.2	0.0%
FTP	92.4%	0.4%	98.7%	0.2%	0.4%	92.9%	0.3%	98.7%	0.2	0.1%
Edonkey	94.1%	0.3%	96.4%	1.8%	0.0%	71.4%	0.0%	96.4%	1.8	0.8%
Kazaa	88.9%	2.6%	90.3%	3.2%	0.7%	67.7%	0.8%	83.9%	3.2	0.1%
Overall	93.7%	X	98.7%	0.4%	X	92.3%	X	98.4%	0.5%	X

Table 4: Labeling accuracy for Paris 6-1 trace.

App	GMM Thresh Tg=99%					HMM Thresh Th=$-3e^{-5}$				
	TP	FN	Masq	Unknown	FP	TP	FN	Masq	Unknown	FP
NNTP	98.9%	0.5%	0.2%	0.4%	0.0%	99.3%	0.5%	0.0%	0.2%	0.0%
POP3	52.3%	0.0%	0.4%	47.3%	0.0%	97.9%	0.0%	0.0%	2.1%	0.0%
SMTP	60.8%	0.3%	0.9%	38.0%	0.0%	95.4%	0.0%	0.1%	4.4%	0.0%
SSH	85.4%	0.0%	4.2%	10.4%	0.0%	85.4%	0.0%	6.3%	8.3%	0.0%
HTTPS	78.4%	0.0%	1.1%	20.5%	0.0%	62.9%	0.0%	0.9%	36.1%	0.0%
POP3S	100.0%	0.0%	0.0%	0.0%	0.0%	76.8%	0.0%	0.0%	23.2%	0.0%
HTTP	65.2%	1.0%	0.0%	33.8%	0.0%	36.0%	0.1%	0.2%	63.7%	0.0%
FTP	97.8%	1.0%	0.1%	1.1 %	0.6%	70.7%	0.0%	0.1%	29.1%	0.0%
Edonkey	87.5%	0.9%	1.8%	9.8%	0.0%	69.6%	0.0%	1.8%	28.6%	0.0%
Kazaa	45.2%	0.0%	3.2%	51.6%	0.7%	83.9%	3.2%	3.2%	9.7%	0.2%
Overall	67.0%	1.0%	0.2%	31.8%	X	45.8%	0.1%	0.2	53.9%	X
Bittorrent	X	32.9%	0.3%	66.8%	X	X	16.1%	6.0%	77.9%	X
IMAP	X	8.6%	57.8%	33.6%	X	X	5.7%	85.7%	8.6	X
Gnutella	X	100.0%	0.0%	0.0%	X	X	0.0%	0.0%	100.0%	X
IRC	X	10.0%	0.0%	90%	X	X	0.0%	0.0%	100.0%	X
LDAP	X	8.8%	0.0%	91.2%	X	X	55.8%	0.0%	44.2%	X
MSN	X	38.5%	0.0%	61.5%	X	X	68.0%	0.0%	32.0%	X
Mysql	X	0.0%	0.0%	100.0%	X	X	0.0%	0.0%	100.0%	X

Table 5: Detection of unknown traffic (Paris 6-1) using Cluster&Port labeling heuristic

payload analysis performs approximately 150 comparisons. A payload analysis classification would thus perform a number of comparison in the order of 150 for each packet and for each application until the connection is classified. Besides, such payload analysis require an important amount of memory to store packets while they are processed.

Our classifier needs to store information concerning the first P packets of the connection. This information consists in the sizes of the first packets and the TCP port. This requires very little storage (a few bytes for each connection). After P packets, our classifier assigns the connection to a cluster and labels it. The labeling heuristic is very simple: for the Cluster&Port heuristic, it only involves a lookup of the connection port in a table of standard ports. The assignment heuristics involve an evaluation of the probability to belong to all clusters. For GMM, such computations involve a few multiplications for each luster, because we only need to evaluate $log(\mathcal{N}_P)$ (the use of diagonal covariance matrices makes this evaluation very simple). Considering the simple, four-states, Markov Chain from our HMM model, the HMM likelihood heuristic also involves a few multiplications (between the emission probabilities for the four symbols associated with the connection).

Even though the number of clusters is greater than the number of applications, our classification method is thus at least an order of magnitude faster than signature matching algorithms. Therefore, we can consider using our classifier on fast links where payload analysis can not be achieved. In future work, we plan to implement and evaluate an online classifier based on this method.

8. CONCLUSIONS

We propose new techniques for early identification of the application associated with TCP connections traversing the link that connects an edge network to the Internet. This paper makes three major contributions. First, a *study of TCP-connection features for early identification*. Our experimental analysis shows that the sizes of just the first few packets are a good metric to distinguish applications. Second, an *analysis of training techniques*. We evaluate three different clustering methods: k-means and GMM on Euclidean space, and spectral clustering on HMM sequences. Our results show that even though the HMM representation is richer, the quality of the clustering is comparable to the simpler Euclidean representation when using GMM clustering. Finally, an *evaluation of multiple classification heuristics*. We define a number of assignment and labeling heuristics. The GMM clustering combined with TCP port numbers correctly classifies over 98% of known applications for the payload traces we studied. The GMM or HMM classifiers can also label most connections of new applications as “unknown” or “masquerade”.

We can extend our classification methodology to work under different conditions or to classify other types of traffic. In multi-homed networks, we can monitor all access links and aggregate information on a machine where the classification will take place. We can also classify UDP traffic by using heuristics (such as presented in [25]) to identify the beginning of an UDP flow. One limitation of our method is that it needs each of the first four packets of a TCP connection in the correct order. In networks that use packet sampling or when packets arrive out of order, the classifier may have poor performance. Fortunately, the chance that there is reordering in the first four packets is small ($< 5\%$ of connections for our traces). Moreover, it is easy to identify and fix out-of-order packets using the IP ID field. The accuracy of our method will degrade fairly quickly under packet sampling. If instead the network adopts flow sampling [12], then our method will work unaltered.

The main challenge to traffic classification techniques in general is evasion. For instance, an "attacker" could easily evade our method by padding packet payloads in order to modify sizes. However, all classification methods can be evaded: payload analysis tools cannot classify encrypted packets, port-based methods are deceived by a simple change of port, and approaches relying on summarized flow information are sensitive to simple alterations of packet sizes and inter-arrival times. Building classifiers that are robust to evasion is an important topic for future research.

9. REFERENCES

[1] Community Resource for Archiving Wireless Data At Dartmouth: http://crawdad.cs.dartmouth.edu/.

[2] M2C Measurement Data Repository: http://m2c-a.ewi.utwente.nl/repository/.

[3] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[4] L. Bernaille, A. Soule, M.-I. Jeannin, and K. Salamatian. Blind application flow recognition through behavioral classification. Technical report, Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie, http://www-rp.lip6.fr/ bernaill/techreport.pdf, 2005.

[5] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic classification on the fly. *SIGCOMM Comput. Commun. Rev.*, 36(2):23–26, 2006.

[6] H. Binsztok, T. Artires, and P. Gallinari. A model-based approach to sequence clustering. In *ECAI*, Madrid, 2004.

[7] R. Boyer and J. Moore. A fast string searching algorithm. *Communications of the ACM*, 1977.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[9] Endace. www.endace.com.

[10] I. Fischer and J. Poland. New methods for spectral clustering. Technical report, IDISA, June 2004.

[11] T. Henderson, D. Kotz, and I. Abyzov. The changing usage of a mature campus-wide wireless network. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 187–201, New York, NY, USA, 2004. ACM Press.

[12] N. Hohn and D. Veitch. Inverting sampled traffic. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003.

[13] IPMON. ipmon.sprintlabs.com.

[14] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is p2p dying or just hiding? In *Globecom*, 2004.

[15] T. Karagiannis, D. Papagiannaki, and M. Faloutsos. Blinc: Multilevel traffic classification in the dark. In *SIGCOMM*, 2005.

[16] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. In *Passive and Active Measurement*, 2004.

[17] A. Moore and D. Zuev. Internet traffic classification using bayesian analysis. In *Sigmetrics*, 2005.

[18] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering : analysis and an algorithm. In *NIPS*, 2001.

[19] F. Porikli. Trajectory distance metric using hidden markov model based representation. In *IEEE European Conference on Computer Vision, PETS Workshop*, 2004.

[20] Qosmos. http://www.qosmos.com.

[21] 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*, 2004.

[22] P. Smyth. Clustering sequences with hidden markov models. In *Advances in Neural Information Processing*, 1997.

[23] Snort. http://www.snort.org.

[24] A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal on Machine Learning Research (JMLR)*, 2002.

[25] K. Suh, D. R. Figueiredo, J. Kurose, and D. Towsley. Characterizing and detecting relayed traffic: A case study using skype. In *IEEE Infocom*, 2006.

[26] D. Zuev and A. Moore. Traffic classification using a statistical approach. In *Passive and Active Measurement*, 2005.