

Blind application recognition through behavioral classification

Laurent Bernaille, Augustin Soule, Marc-Ismael Jeannin, Kavé Salamatian

ABSTRACT

Application recognition appears to be an important task for a large number of applications in security and traffic engineering. Well-known port numbers can no longer be used to reliably identify network applications. There is a variety of new Internet applications that either do not use well-known port numbers or use other protocols, such as HTTP, as wrappers in order to go through firewalls without being blocked. One consequence of this is that a simple inspection of the port numbers used by flows may lead to the inaccurate classification of network traffic. Moreover because of some privacy concern or more simply because of used encryption mechanism it is frequently impossible to get access to the full payload of packets. This means that classification should be based only to the behaviour of the packet flow in term of size, inter-arrival time and interaction. We develop in this paper a Blind applicative flow recognition through behavioral classification. The approach is based on very simple sequences of quantified packet size and packet direction. These sequences are clustered through a powerful spectral clustering algorithm. We developed thereafter a recognition algorithm based on a mixture of HMM representative of the obtained clusters. The presented method appear to be very powerful as it reaches recognition performance of 90% with only observing seven packets of a flow!. This work is a first step toward an operational flow recognition system that will be robust toward flow morphing (tunnelling flow in other protocol) and payload encryption.

1. INTRODUCTION

Application recognition is a fundamental task for a large number of applications. Firewalls rely on it to decide if a particular flow is allowed to get inside (or outside) the company network. Charging mechanisms use it to decide which rate applies to a flow. Quality of Service management systems base their access control decision on the application, *etc.* The straightforward approach is to use the well-known port number assignments. In recent work [7] the idea that this classification scheme is no longer appropriate seems to grow. This is mainly because applications such as FTP, P2P, game *etc.* are using dynamic port negotiation. Thus, even if ports can be used to identify the control flows, resulting flows, using ports negotiated in this first connection, are not going to be recognized. FTP, H323 or SIP are very good examples of this kind of behavior. Some applications may even use a well-known port for a different usage (in order to bypass firewall rules for instance). Besides, the recent Peer to Peer study by CAIDA [7] shows that if P2P traffic seemed

to have decrease (according to port usage studies), packets payload analysis proves that it is not the case.

This problem shows up clearly by looking at the usual breakdown by application. As an example in August 2000 the application breakdown in the Sprint backbone as seen on the IPMON website (ipmon.sprintlabs.com) show a large amount of HTTP traffic. Using well-known port there was roughly 15% of packets representing 9% of the bytes classified as unknown. But in February 2005, on the same link 35% of packets representing 35% of the bytes are classified as unknown. Even if this is a particular example this increase in the unknown traffic is not only observed in Sprint backbone but appears to be a general trend.

In order to address this problem it seems that a recognition approach based on full packet payload is needed. However this approach raises several fundamental concerns. The first one may be the computing power needed to process full payload packets. Even if solutions can be found for low rate links, handling very fast ones remains a huge challenge. Besides, computation power is not the only issue to be faced when dealing with packets payloads. What can be achieved with encrypted packets for instance? Or, is it legal to read the data inside packets without written approval by the final user?

This paper fit into the gap between the unreliable port number classification and the costly, barely legal, full packet analysis. In this paper we propose a framework to do application classification based on behavior recognition. The idea is not to replace the monitoring box but to classify the usual traffic without the need of going inside the data part of the packet. Then the remaining unknown traffic could be sent through a classic packet analyzer. Our objective is then to avoid any false classification with a small number of unknown flows.

Throughout this paper we will use the term application with the meaning of the protocol assigned by the IANA in the well-known port numbers. An example includes FTP, HTTP, POP, *etc.*

We do believe that different applications exhibits fundamentally different characteristics in term of packet sizes, inter-packet time, as well as interaction with other flows. As an example, we can cite a simple HTTP transfer that consists of a small request from the client followed by a sequence of packets coming in the reverse direction containing the answer to the request. A chat application would behave in a very different way, generating a series of small packets in both directions. These two examples exhibit very different behaviors. In the first case the server tries to send its

answer as fast as possible while in the second one the rate is relatively slow and the inter packet time is human driven. Our main assumption is that the first packet of a connexion exhibits the behaviour of the application.

The method we propose is to create a model to describe the flows using the application labels and the flow sequence. The model is then transformed in a small dictionary. Then the new incoming flows are classified using this dictionary. Each flow is transformed in a discrete Hidden Markov Model (HMM) sequence. Then we find some clusters in the set of all the HMM. At the end each application is described as a mixture of HMM. This description is very lightweight and enables us to apply this classification in real-time.

HMM have been successfully used in different areas such as voice, handwriting or even image recognition. An important issue is the number of clusters needed to achieve a good classification. We propose a decision process based on the *MDL* (Minimum Description Length) method. This method chooses a number of clusters such that the trade-off between the computational complexity and the gain of adding new cluster is optimal. This trade off is evaluated through measuring the size of the description of the observation knowing a model fitted to the data.

To validate the method we uses full packet trace collected at the edge of our university. Whereas almost all the previous papers uses manually labelled flows or well-known application port we rely on a full packet analyzer developed by Qosmos [11]. This packet analyzer is heavily used by some ISP to understand the underlying traffic. This analyzer is able to detect an application with no false positive and with only a few unknown traffic.

The paper is organized as follows. The first section presents some previous works related to this subject. Then we describe the packet traces we collected and how to translate an application flow to a sequence of symbols on which the classification process can be applied. Classical port number classification will also be evaluated in this section. Section 4 will describe the proposed clustering mechanism for finding flows with similar behavior. The MDL criteria for choosing the number of clusters will be described in this section. An interpretation of the clustering in term of applications behaviors we be shown to conclude. The recognition algorithm will be presented with the results in section 5. We will conclude in section 6.

2. TRACE DESCRIPTION AND INITIAL RESULTS

2.1 Traces description

Our main data set consists in several packet traces collected at the exit of the UPMC university network. We used an optical splitter and a DAG card to capture data transiting through the edge router connecting the university to the French academic network RENATER. The link we monitored was a Gigabit Ethernet Link. We captured three traces of one hour each. Since most of the traffic on this link consisted in a few applications (HTTP, FTP, NNTP) we needed long traces to be also able to classify less common applications. The DAG card is a very powerful tool which allowed us to capture every packets coming in and out the edge router. Besides DAG cards are able to capture more than just packet headers which was very

helpful to find out what application was behind each flow. We tested thoroughly what payload size was needed to an accurate classification and it turned out that only 300 bytes per packets were enough. This is far better than full packet capture since a one hour traces shrunk down from a typical size of 50 GB to a more convenient 15GB. As an indication the last one our trace we collected was 16GB, for a total traffic of 58GB and contains 220,000+ flows (a flow being identified by the classical 5-tuple : source IP address, destination IP address, source port, destination port and IP protocol).

Once we had captured these traces we needed to analyze them and remove irrelevant flows for our study. In order to do so we used a tool based on the Coralreef suite developed by Caida (www.caida.org). First of all, we needed to remove non-TCP flows, which are easily filtered out using the IP protocol field. Besides, we had to keep only TCP-flows which started within the trace since our classification method is based on the first packets. We achieved this by checking each flow. We decided to keep a flow only if it began by a full successful TCP handshake. On the trace mentioned before, after filtering out non TCP traffic, 76% of the flows remains (representing more than 98% of the total volume). After the full handshake filtering 62% remains (accounting for 76% of the total volume).

In order to achieve a proper classification we also filtered out TCP control packets and kept only those with an application payload. As we want to focus on the behavior of applications we tried to avoid the pollution of the trace by TCP control packet (TCP Keep-Alive or TCP Ack with no data). Besides, in order to find out relevant patterns we needed flows which exchanged at least a certain number of packets with application data. Thus our final sequences were extracted from long enough flows and consisted in the sizes of the first data packets exchanged. We decided to keep only flows with more than seven packets containing actual application data. This decision was based on some initial testing and results in a trade-off between the quality of the clustering (the more packets kept, the more efficient) and the complexity. Besides, using more packets decrease the numbers of flows that can be dealt with in our method (since in most of the flows only a few packets are exchanged).

On the trace previously mentioned, this last filtering left us with 22% of the original but they accounted for 74% of total volume. In a situation where we would be doing real time analysis, flows without handshake would no longer be an issue and our filtering would leave with 35% of the TCP flows, accounting for more than 98% of the traffic (for the example trace).

Finally, another TCP related behavior has to be taken into account. Indeed, if you look at packets belonging to a flow on a link, they may not appear in order or they may appear more than once. There can be several reasons for packets not appearing in the sequence they were sent as shown in [6]. Out of order packets will decrease the quality of the clustering by introducing new non-legitimate HMM states. We need to consider this behavior by removing any re-transmission and deliver the packets in order to our analysis tool.

2.2 Application labeling

To describe applications with a mixture of clusters in a semi-supervised way we needed to label each flow with the

corresponding application. This labeling could have relied on TCP ports. However using IANA well-known ports is no longer reliable when studying traces as mentioned in the introduction. Since our mixtures was based on the application labels we needed it to be as reliable as possible.

That is why we decided to use an applicative classification tool : Traffic Designer (www.qosmos.fr). Traffic designer is able to label a flow with an application based on the TCP payload. A flow is said to belong to a specific application only if the syntax of the data exchanged between the two TCP peers matches to the application syntax. For instance if a flow runs on port 21 and contains something like "GET /index.html HTTP/1.1", Traffic Designer is going to classify it HTTP whereas a standard port classification would describe it as FTP. The classification engine of Traffic Designer is quite complex. The main idea behind it is to find out the protocol stack in each packet. In order to do so each layer starting at the data link level is thoroughly examined. Once Traffic designer has found out what protocol is running in a specific layer it tries to match its payload with a protocol that can be found on top of it. This allows Traffic Designer to successfully analyze complex protocol stacks such as those hidden inside in tunnels (which 5-tuple analysis can not find).

An example of the proportion of unknown traffic seen using well-know port is given in fig. 1. To compare these numbers with the application breakdown obtained using the Traffic Designer see fig. 2.

As we can see on these figures the use of an applicative classification tool improves a lot the quality of the analysis. The main difference here is due to FTP data flows which can not be found based on TCP ports only since server ports are dynamically negotiated. We can also observe a certain amount of misclassified flows (applications not running on their official port). However less than a percent falls in this category for our specific trace, which would not make it an issue. But in the case of commercial traffic (ADSL user for instance) this proportion should increase a lot.

2.3 Flow representation

In this work, we want to do application recognition using a minimal amount of information and especially without going into the packet payload. However, inter-packet time that might be easily extracted over a packet flow appears to be unusable for our purpose. This is related to the fact that inter-packet interval time is affected by the network load, TCP acknowledgments and window based mechanisms, as well as application related properties. In practice it is very difficult to separate TCP and network induced delay characteristics from applicative properties. This motivates our decision of not using the inter-packet delay and to keep for each packet only two piece of information : the direction of the packet, and its size. The side that opens the connection will be later referred as the client. The host waiting for the connection will be named server. This denomination is often the intuitive one but in some case when the port are dynamically assigned this can be misleading. However this does not affect our classification scheme.

The precise size of packets is not very important for our method. To facilitate later interpretation, we quantize packet size in four level $\{1, 2, 3, 4\}$ for small, small-mid, large-mid and large size packets. We choose the quantification steps by observing the multi-modal size distribution of packets

(see Fig. 3). This distribution shows clearly three modality for packet size and we choose the discretization levels based on these modality as $\{1 = [0, 150], 2 = [150, 700], 3 = [700, 1300], 4 = [1300, 1500]\}$.

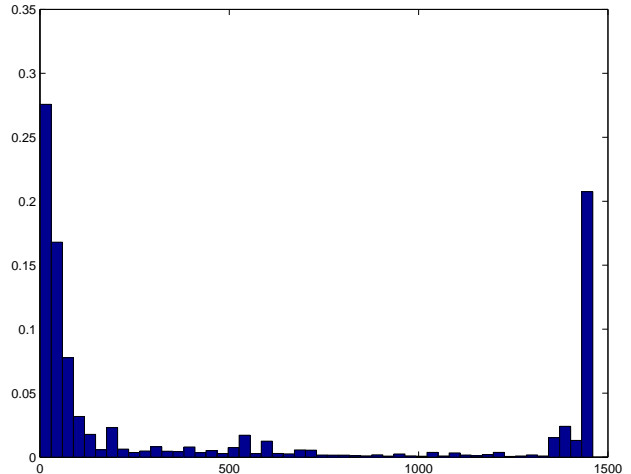


Figure 3: Empirical packet size distribution

We represent the direction of the packet as a sign (client: + and server: -). This lead to the representation of a flow as a sequence of positive or negative number in $\Sigma = \{\pm 1, \pm 2, \pm 3, \pm 4\}$. Each flow is therefore represented as $f_i = (\sigma_1, \dots, \sigma_{l_i})$ where l_i is the length of the sequence and σ_t represents the t^{th} packet of the sequence and $\sigma_t \in \Sigma$. Throughout this paper the recognition horizon, i.e the maximal value for l_i will be noted N_{max} . And finally let $\mathcal{F} = \{f_i\}$ be the set of all the sequences.

3. APPLICATION DETECTION

3.1 Previous works

Several approaches have been used to classify flows according to the application they belong to. The most common one is based on ports. This approach is pretty straightforward : each time a new flow is found, it is assign to an application according to the TCP port on the server side of the connection. This method has been used for quite a time and its main advantage compared to other techniques is its simplicity : finding out ports IP packets is an easy task that is besides computationally inexpensive. However, if this method used to be quite reliable a few years ago, many studies start to show it is not the case anymore. This phenomenon can be explained by several reasons. First of all applications such as FTP, P2P, game *etc.* are using dynamic port negotiation. Thus, even if ports can be used to identify the control flows, resulting flows, using ports negotiated in this first connection, are not going to be recognized. FTP, H323 or SIP are very good examples of this kind of behavior. Besides, some applications may even use a well-known port for a different usage (in order to bypass firewall rules for instance). Besides, the recent Peer to Peer study by CAIDA [7] shows that if P2P traffic seemed to have decrease (according to port usage studies), packets payload analysis proves that it is not the case. This problem shows

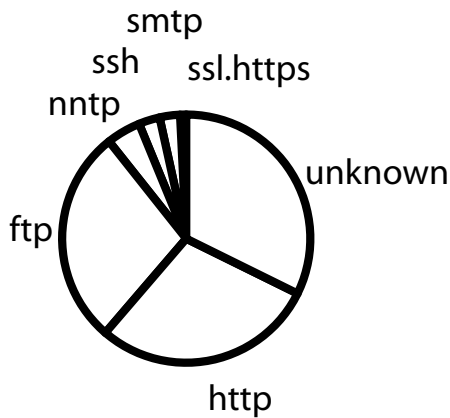


Figure 1: Breakdown according to TCP ports

up clearly by looking at the usual breakdown by application. As an example in August 2000 the application breakdown in the Sprint backbone as seen on the IPMON website (ipmon.sprintlabs.com) shows a large amount of HTTP traffic. Using well-known port there was roughly 15% of packets representing 9% of the bytes classified as unknown. But in February 2005, on the same link 35% of packets representing 35% of the bytes are classified as unknown. Even if this is a particular example this increase in the unknown traffic is not only observed in Sprint backbone but appears to be a general trend.

An other common approach relies on protocol signatures. This method is very close to what is used in Intrusion Detection Systems : packets payload are analyzed and matched against given signatures. For HTTP for instance a simple signature might be "GET HTTP". Any flow with a packet containing this string would be labeled HTTP. This approach is much more reliable than ports based ones since if application ports are easily modified, altering protocols themselves is far more difficult. In [7], the P2P recognition system is based on pattern matching and proves much more efficient than ports based analysis. However this method cannot answer all needs. First, it is computationally quite expensive and thus cannot be used on high speed links. Besides, finding a protocol signature can sometimes not be good enough. Many P2P applications for instance rely on the HTTP protocol to transfer data between peers. A pattern based mechanism might thus wrongly classify these flows as HTTP.

A third approach that has been used in recent studies relies on summarized flows information such as duration, number of packets and mean Inter-arrival Time for instance. In [8] or [14] the proposed classification methods rely on classes extracted from such metrics. In [2] the authors propose to represent flows as wavelet coefficients extracted from packet size distributions and arrival times, and then compare the results of clustering techniques. The authors of [5] propose a feature-based representation of the flows, and apply a hierarchical clustering technique to find groups of similar flows that can be recycled to form a classification process. Such approaches avoid full payload analysis. However they need the flow to be finished to compute the necessary metrics and determine the application it belongs to. This would make such methods very difficult to use in real time analy-

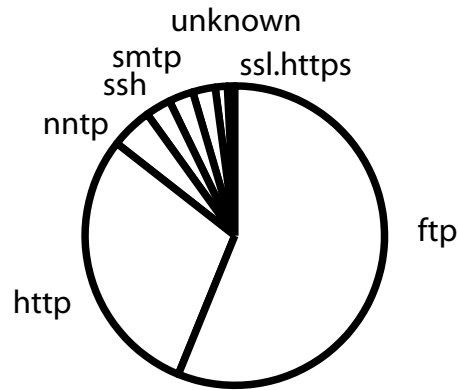


Figure 2: Breakdown according to Traffic Designer

sis, since flow metrics would have to be kept and updated for the whole duration of the connection.

3.2 Classification method

3.2.1 Unsupervised / Supervised learning

Two types of classification methods are commonly used in machine learning : unsupervised and supervised. In supervised learning, the learning algorithm is provided with a set of samples along with the corresponding category label. A model is then chosen to describe the classes and the classification process consists in estimating the model parameters based on the labeled samples. A good example of supervised learning methods is Naive Bayes Classifiers. Unsupervised learning on the other hand relies on unlabeled samples. The aim of such methods is to find "natural" groups (or *clusters in the data*). Such groups may then be modeled and the resulting model can be used as classifier. The main advantage of unsupervised learning is that it does not rely on classes defined a priori. In our case, we do not believe that applications have a single behavior that can be modeled. FTP flows for instance might be either control flows or data flows which have very different behavior at the packet level. Besides, it is very likely that different applications may in some case have the behavior at the packet level. That is why our classification method rely first on unsupervised learning : the goal is to identify behaviors independently of applications.

3.2.2 Method overview

The proposed methodology in this paper is described in figure 4. This is a two step method. In the first step we calibrates the model and build the dictionary. Then in the second step we uses the dictionary to classify the new flows in real time. The learning phase is composed of different transformation applied to the training data. The training data is the set of packet for every flow crossing our measurement point. Every flow is transformed in a HMM sequence. We then computes the euclidean distance between each sequence to obtain the distance matrix. We then use the classical spectral clustering [9, 3] to find the clusters. At the very end of the method we describe the application as a mixture of the clusters.

3.3 K-Means

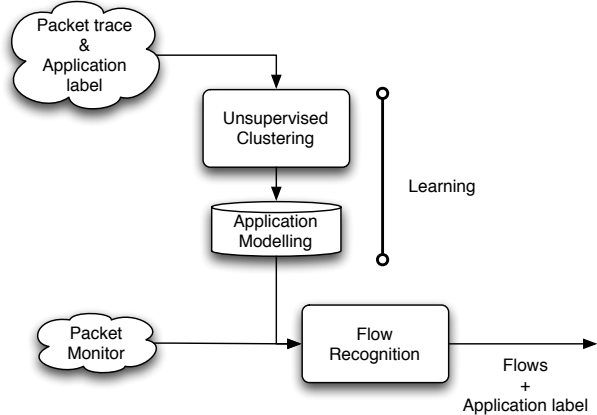


Figure 4: Method overview. The learning phase uses some recorded labeled packet traces. Then in the operational phase the algorithm can run in real-time using a simple packet monitor.

In this section we explain the K-Means method. This method is the simpler clustering method. It consist of grouping the flows into K groups such that the distance to the centroids of the clusters is minimized. ie:

$$\min_C \left(\sum_{k=1}^N \|f_k - C(k)\| \right)$$

. Where $C(k)$ is the closest centroid from the flow f_k and $\|\cdot\|$ is a norm. Our implementation of this algorithm using the Norm-2 as the distance metric. As the number of groups K is fixed it should be inferred using some other tools. For example we can use the Maximum Description Length algorithm described in [?].

The implementation of a K-Means algorithm is quite trivial. First assign randomly each flow to a cluster. Then computes the centroid of each cluster in order to minimize the distance between the centroid and the cluster member. Then using the updated centroids assigns the flows to the closest centroid. Repeat the operations until $\sum_{k=1}^N \|f_k - C(k)\|$ does not change.

Even if the algorithm seems very simple and computationally efficient the main disadvantage is that the initial partitions are very important. If the initial clusters are not well chosen then the K-Means can converge to a local minimum instead of the global minimum solution. To avoid that, a solution is to run the algorithm several times and keep the best solution. This however has the drawback of being very time consuming and computationally expensive.

3.4 Spectral Clustering

This section explains the algorithm to cluster flows with similar behavior. Hidden Markov Models (HMM) have been widely used as probabilistic model to describe the behavior of complex systems [12]. The idea of this class of model is that the observed system output at step i , $\sigma_i \in \Sigma$ can be explained as a random function of an unobserved internal state $q_i \in \mathcal{Q}$. The unobserved internal state q_i follows a Markov chain. In this paper we assume that the observed output σ_i comes from a finite set $\Sigma = \{\pm 1, \pm 2, \pm 3, \pm 4\}$. And the

state space \mathcal{Q} is also discrete. The random function relating the output symbol to the state is characterized through an emission probability distribution $\text{Prob}\{\sigma_t = i | q_t = j\}$. As explained before, the sequence of states is supposed to be hidden for HMMs. We therefore need to use the observed outputs of the system to infer the state sequence $Q_i = \{q_1, \dots, q_{l_i}\}$ for the sequence f_i of length l_i .

A k -state HMM with states $\{s_1; \dots, s_k\}$ is formally defined as a 3-tuple (Π_0, A, B) such that:

- $\Pi_0 = \{\pi_1, \dots, \pi_k\}$, $\pi_i = \text{Prob}\{q_0 = i\}$ are the initial state probabilities
- $A = \{a_{ij}\}$, $1 \leq i, j \leq |\mathcal{Q}|$, $a_{ij} = \text{Prob}\{q_{t+1} = i | q_t = j\}$ is the transition matrix between states.
- $B = \{b_{ij}\}$, $1 \leq i \leq |\mathcal{Q}|$, $1 \leq j \leq |\Sigma|$, $b_{ij} = \text{Prob}\{\sigma_t = i | q_t = j\}$ are the emission probability in each of the states.

The simple structure of HMM enables an easy and efficient computation of the likelihood of the observation of a sequence f_i given the parameters of a particular HMM. This derivation is based on the well-known Baum-Welch “Forward-Backward” algorithm.

The idea of using a HMM for our behavioural analysis is that any packet sent by an application depends on the overall applications context. Unfortunately this context is hidden and we only see it through the packet flow. The behavior of flows (as sequences) might be captured in a HMM through several means, with varying number of states and different structures. Here we focus on a specific HMM structure proposed in [4, 10] where each state corresponds to an element of the sequence. This structure has been intensively applied in the context of sequence analysis and handwriting recognition. As explained before, input sequences have a length of at most N_{\max} , meaning that the Markov chain has at most N_{\max} states. The state Markov chain is assumed to go only from left to right, i.e. $a_{ij} = 0$ if $j \neq i + 1$ and $a_{i, i+1} = 1$, which is more suitable for analyzing a sequential behavior.

The remaining parameters of the HMM (i.e. the emission probabilities) are estimated like in [4]. We first extract from the overall sequence set \mathcal{F} and for each symbol $s \in \Sigma$ a probability measure $P_s(r) = \text{Prob}\{\sigma_{t-1} = r | \sigma_t = s\}$. This represents the probability that the symbol r follows the symbol s . This probability measure is estimated by $P_s(r) = \frac{\#(\sigma_{t-1}=r, \sigma_t=s)}{\#(\sigma_t=s)}$, where $\#(\cdot)$ means the number of occurrences of the given sequence. The similarity measure between two symbols s_1 and s_2 $s_1, s_2 \in \Sigma$ noted $c(s_1, s_2)$, is defined as the correlation between each probability law P_{s_1} and P_{s_2} .

We define $b_{t,j}$ as the emission probability of the symbol j at the state t of the HMM derived from the sequence $f_i = \{\sigma_1, \dots, \sigma_{l_i}\}$. Then $b_{i,j} = \frac{c(\sigma_i, j)}{\sum_{k \in \Sigma} c(\sigma_i, k)}$, $j \in \Sigma$. This definition of emission probability relates the hidden context to affinities between the observed symbol σ_i to all the other symbols. This enables a highly generic relationship between context and observation. The emission probabilities are assigned to states of the HMM representing each existing flow in \mathcal{F} , i.e. each flow is now represented by a HMM with the structure depicted in Fig. 5.

Each sequence f_i has its properties summarized by its HMM. This new representation will allow us to quantify the

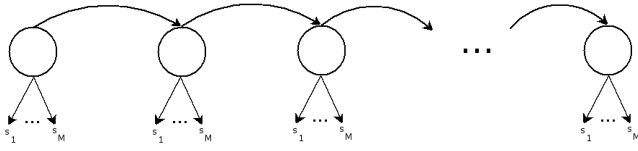


Figure 5: HMM Structure used throughout the paper.

similarity between the sequences. This similarity will be used in the clustering step.

3.5 Hidden Markov Model based Clustering

The method described in this paper consists of projecting the sequences representing the flows in a high dimensional space using Hidden Markov Models. The generic method for HMM-based clustering of sequences is given in [13]. Lets suppose that each sequence $f_i \in \mathcal{F}$ is associated with a HMM H_i as described in previous section. One might obtains for each HMM H_i , the log-likelihood of all other sequences f_j . This log-likelihood is easily obtained using the Baum-Welch forward-backward filter which is known to have a linear complexity in the number of symbols in the sequence.

We define a matrix of log-likelihood $\Lambda = (L_{ij})$ where L_{ij} is the log-likelihood that sequence f_j have been generated by HMM H_i related to sequence f_i *i.e.* $L_{ij} = -\log \text{Prob}\{f_j|H_i\}$. Now based on this log-likelihood matrix we can define a distance matrix $D = (d_{ij})$ through the euclidean distance as,

$$d_{ij} = \sqrt{\sum_{k=1}^N \|L_{ik} - L_{jk}\|^2}, i, j = 1, \dots, N.$$

It is noteworthy that each distance vector (L_{i*}) (each line of the matrix Λ) defines a point in \mathbb{R}^N , representing the sequence f_i . Similar flows with similar behaviors should be intuitively close together. We therefore have to regroup close enough points in the N -dimensional (N being the number of initial applicative flows) into homogeneous clusters. The distance matrix D extracted from Λ expresses distances between these points.

The application of the presented derivation on our learning set lead to a distance matrix of dimension 1000, and the clustering step have to cluster 1000 points in a 1000-dimensional space. We show in Fig. 6 the obtained distance matrix over the evaluation learning set.

The complexity involved in the calculation of the distance matrix is $\mathcal{O}(N^3)$ in time.

The term "clustering" refers to the task of identifying disjoint concentrations of points (clusters) in a multidimensional space. It is an instance of unsupervised learning, meaning that these groups have to be inferred only using a distance measure between the points and not from a class membership as for supervised learning. Intuitively the clusters found must have a high intra-cluster and a low inter-cluster similarities, as the desirable result is to find well-separated and compact clusters. Several methods of clustering might be applied here (e.g. K-means, Hierarchical Clustering, ...). However, despite a very rich literature clustering is known to be a hard task. Classical clustering approaches as k -means or approaches based on Gaussian mixtures work

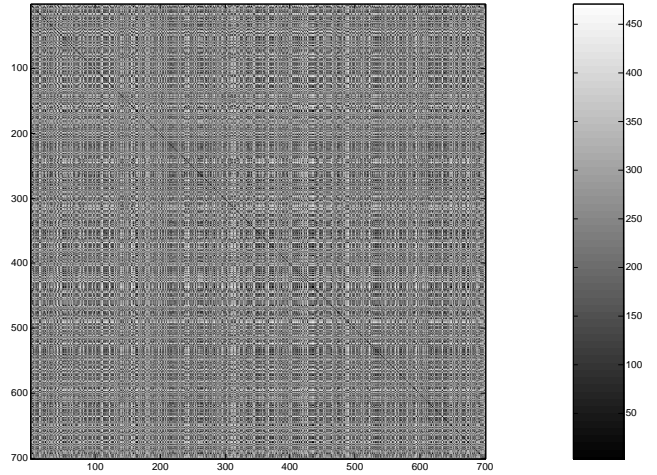


Figure 6: Distance Matrix obtained over the evaluation learning set

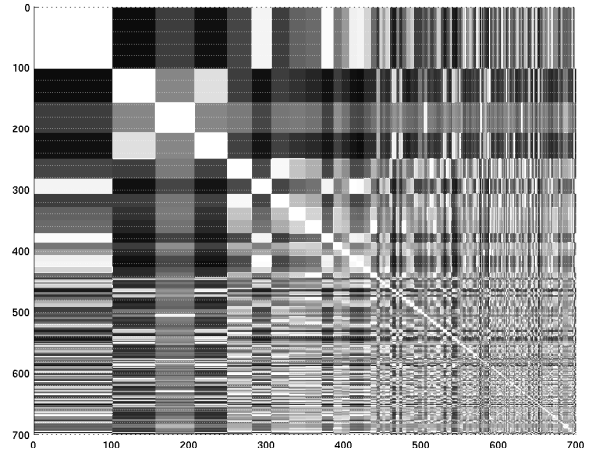


Figure 7: Distance Matrix after reordering following the clustering

well when clusters have a spherical or elliptic form. However in a lot of applications, clusters might have complex non-spherical structure, and even not being convex. Moreover, classical methods frequently fall into the pitfall of local minimum of likelihood function. We have chosen to apply here the so-called Spectral Clustering method.

3.6 Spectral clustering

In this section we explain how to find clusters in the likelihood space using the spectral clustering method. This technique has become popular recently [9] due to its efficiency on a wide range of problems when the clusters are not spherical. As the space is $N \times N$ we can not represent using a 2D-plot. To illustrate the shape of the clusters we can only show some projection as in figure 8. Each point is the projection of a sequence in the likelihood space and his coordinate is the likelihood of the sequence using the HMM 132 and HMM

12. As you can see in this particular projection there is not any circular structure. And there is no reason that there should be one. The k -means method is very likely to fail in this situation. This is not a proof that a simple method will fail in finding clusters in the likelihood space.

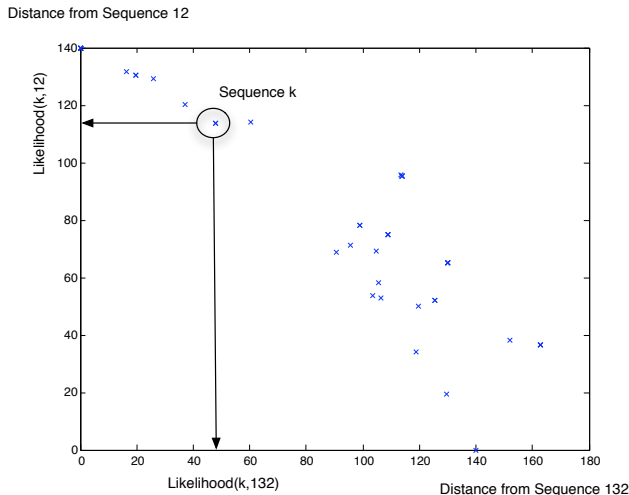


Figure 8: Projection on the HMM_{132} and HMM_{12} . As an example, the sequence k has the coordinates $(L_{132}(k), L_{12}(k))$.

The spectral clustering is divided in three steps. The first step is to transform the distance matrix into an affinity matrix. This can be viewed as increasing the contrast in an image darken the dark and light the white. Or in other words let the close sequence to be closer and the far sequence be further. As a second step we do the spectral analysis. This step can be viewed as a Principal Component Analysis. This rotates the likelihood space such that most of the energy is contained in the first axis. Then the last step is a k -line clustering in the transformed space.

3.6.1 From distance to affinity

To explain how the spectral clustering method works, we will assume that the clustering is *a priori* known and that points have been ordered so that points in the same cluster are consecutive. Moreover the clusters have been ordered in size-increasing order. This hypothesis will be applied through the analysis but as we will see at the end it is not necessary for the analysis. The distance matrix after reordering is shown in Fig. 7.

The new distance matrix has now a block structure that was not visible in the initial matrix before clustering (see Fig. 6). However even doing the clustering using this new distance matrix is not straightforward as the block structure is spoiled by elements far from the diagonal. To deal with this problem a non-linear operation is frequently applied to the distance to derive the so-called *Affinity Matrix*, $A = (a_{ij})$. The Affinity matrix contains values between 0 and 1, and evaluates the similarity of points. The non-linearity of the mapping function between distance and affinity is meant to reinforce the block-diagonal structure of the distance matrix. It is usual to take a Gaussian kernel as the mapping function between the affinity and distance, *i.e.*

$a_{ij} = \exp\left(-\frac{d_{ij}^2}{2\sigma^2}\right)$. This choice of the mapping function leads to a positive definite affinity matrix. This property will show to be very useful when we will have to calculate eigenvalues and eigenvectors of a large matrix, as it enables the application of Cholesky decomposition and simplify the calculations.

The choice of σ , the radius of the Gaussian kernel, is important for the efficiency of the clustering. We have followed in this paper the method proposed in [3]; we have chosen for each point i , the radius σ_i as the value where $\sum_{j=1}^N \exp\left(-\frac{d_{ij}^2}{2\sigma^2}\right) = \tau$. This choice will be motivated through the interpretation of the affinity matrix in terms of graph. The affinity matrix might be interpreted as a random adjacency matrix, *i.e.* an affinity a_{ij} can be interpreted as the existence of a link between i and j with a probability a_{ij} in a graph. Each particular clustering can be seen as a particular realization of this random graph where each cluster is a connected subgraph.

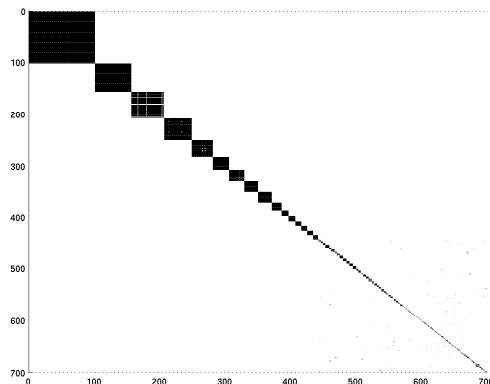


Figure 9: Affinity Matrix after reordering following the clustering

Moreover, the value $\sum_{j=1}^N a_{ij}$ represents the mean number of neighbors of the point i in this random graph. This motivates the choice of σ_i to fix the mean size of the neighborhood τ of each point. However this choice of σ_i lead to an asymmetric affinity matrix as a_{ij} is controlled by σ_i , where a_{ji} depend on σ_j . We make the affinity matrix symmetric by taking $\alpha_{ij} = \min(a_{ij}, a_{ji})$.

The application of the described derivation of the affinity matrix to the reordered distance matrix leads to the affinity matrix shown in Fig. 9. In this derivation the number of neighbors has been fixed to $\tau = 5$. This figure shows clearly that the affinity matrix has a reinforced block diagonal structure, meaning that clustering is easier using the affinity matrix in place of the distance matrix.

However, affinity matrix might be insufficient to tackle with clusters that are not convex. Fig. 8 gives the intuition that this might be case for the learning set under study. Moreover it can be seen that the block structure of the affinity matrix is almost perfect up to line and column 750, but we observe some dispersion after these values. This motivates another processing step to deal with such clusters. This step makes use of the interpretation of the affinity matrix as an adjacency matrix.

Conductivity matrix

In order to handle cases where data do not form compact and convex clusters, we have to extend our definition of a cluster : two points belong to a cluster if they are close enough (or equivalently they have enough affinity), or if they are well connected by paths of short "hops" over other points. The more such paths exists, the higher the chances are that the points belong to the same cluster. We are therefore defining a new affinity measure, based on the random graph view of the affinity matrix. Instead of considering two points similar if they are connected with a high probability by a link, we assign them high affinity if the overall graph conductivity between them is high. This new affinity matrix is called the *Conductivity Matrix*, $C = (c_{ij})$. This definition enables affinity to depend on the distance between points as well as to the neighborhood and the "weight" of a cluster helping to shape non-convex clusters. This definition is very similar to electrical circuit, where the conductivity between two nodes depends on all paths between them. The idea is to suppose that the graph is an electrical network with a resistance a_{ij} between each two nodes i and j .

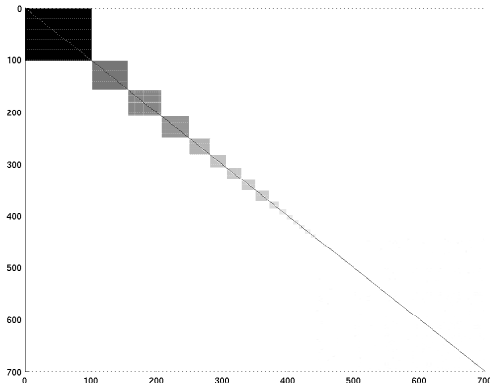


Figure 10: Conductivity matrix after reordering following the clustering

c_{ij} is computed by submitting the network to an entering current I at node i and an output current I at node j and deriving the voltage between these two points. The conductivity is found through Ohm law as the ratio between the current and the voltage. A look to an electrical circuit theory book will give the following formula for the conductivity matrix :

$$c_{ij} = \frac{1}{G^{-1}(i, i) + G^{-1}(j, j) - G^{-1}(i, j) - G^{-1}(j, i)} \quad i \neq j$$

$$c_{ii} = \max\{c_{ij}\}$$

where the matrix $G = (g_{ij})$, represents the matrix of voltage between point i and j . This matrix is defined through

$$G(1, 1) = 1,$$

$$G(1, j) = 0, \text{ for } j > 1$$

$$G(i, j) = \sum_{k \neq i} \alpha(i, k) \text{ for } i \neq j$$

$$G(i, i) = -\alpha(i, i),$$

The complexity involved in the derivation of the conductivity matrix is $\mathcal{O}(N^3)$ in time leading to an overall complexity of $\mathcal{O}(N^3)$ for the preprocessing step.

The application of this ultimate preprocessing step to the reordered distance matrix leads to the conductivity matrix shown in Fig. 10. This figure shows that the conductivity matrix has now a perfect block diagonal structure and that the clustering might be done easily on it. In next section we will explain the next step which is the spectral clustering by itself. This clustering will be applied on the conductivity matrix.

Spectral decomposition and k -lines

At the end of the preprocessing step we have a set of points (each line of the conductivity matrix) that has to be clustered and the conductivity matrix is supposed to be highly block diagonal. The block diagonal structure means that the eigenvalues and eigenvectors of the matrix C can be obtained as a union of eigenvalues and eigenvectors of its blocks (the latter padded with zeros at appropriate position). Now stacking eigenvectors of C column-wise in a proper order we obtain a matrix X with the following form:

$$X = \begin{bmatrix} \vec{x}^1 & \vec{0} & \dots & \vec{0} \\ \vec{0} & \vec{x}^2 & \dots & \vec{0} \\ \vdots & \vdots & \ddots & \vdots \\ \vec{0} & \vec{0} & \dots & \vec{x}^L \end{bmatrix}$$

where each sub-matrix \vec{x}^i denotes the eigenvectors of block i , and $\vec{0}$ is a null sub-matrix. The clustering is now done in the space spanned by the eigenvectors. Let's suppose that we want to regroup the data in K clusters, the clustering occurs in the K -dimensional space spanned by the first k eigenvectors relative to the K largest eigenvalues. Points belonging to each particular cluster will be along the subspace spanned by the eigenvectors. We are assuming here that the number of clusters K is known in advance. We will present in section 3.7 how to choose this number.

Spectral clustering is very flexible as it enables to choose the number of clusters, through choosing the number of eigenvectors and eigenvalues. Choosing a smaller number of eigenvectors is equivalent to doing the clustering in a space of lower dimensionality. Moreover, the spectral decomposition has an interpretation in terms of Principal Component Analysis (PCA) : the best clustering using K clusters is the one that is done using the eigenvectors relative to the K largest eigenvalues.

The spectral decomposition leads to the following algorithm for clustering: **calculate the eigenvectors of the conductivity matrix and regroup points in the subspace spanned by the eigenvectors along the plane defined by the eigenvectors subspaces.**

However the previous derivation makes the hypothesis of a "perfect" block diagonal structure. In practice, even after the mapping to the affinity and the conductivity matrices, the obtained matrix is not perfectly block diagonal. This means that the clusters might not perfectly be aligned with the eigenvectors. This means that we might have to search linear (or planar) alignment in the eigenvector spanned space. This search can be achieved by using the k -lines algorithm. This algorithm is similar with the well known k -means algorithm but it find linear alignments

in place of spherical clusters. It works through choosing a linear alignment and finding the set of closest points to this alignment. The alignment is readjusted at the end of each assignment round. This k -lines algorithm is described below.

- 1: Initialize vectors m_1, \dots, m_k as the k eigenvectors related to the largest eigenvalues.
- 2: **for** $j = 1 \dots k$: **do**
- 3: Define P_j as the set of indices of all points y_i that are closest to the line define by m_j and create the matrix $M_j = [y_i]_{i \in P_j}$ whose columns are the vectors y_i .
- 4: **end for**
- 5: Compute the new value of every m_j as the first eigenvector of $M_j M_j^T$.
- 6: Repeat until m_j are stable.

Algorithm 1: k -lines algorithm

This algorithm can be proved to converge to a stable solution (similarly to the k -means algorithm).

In the previous descriptions we assumed that the clustering is *a priori* known and that the distance, affinity and conductivity matrix were reordered to follow the block diagonal structure. However, the eigenvectors are insensitive to the order of column of a matrix, meaning that even if the conductivity matrix was not reordered, the obtained eigenvectors are the same. This means that distance, affinity and conductivity matrix need not to be block diagonal for the eigenvectors being helpful for clustering. We illustrate this point in Fig. 11, by showing the full methodology on the non-ordered distance matrix.

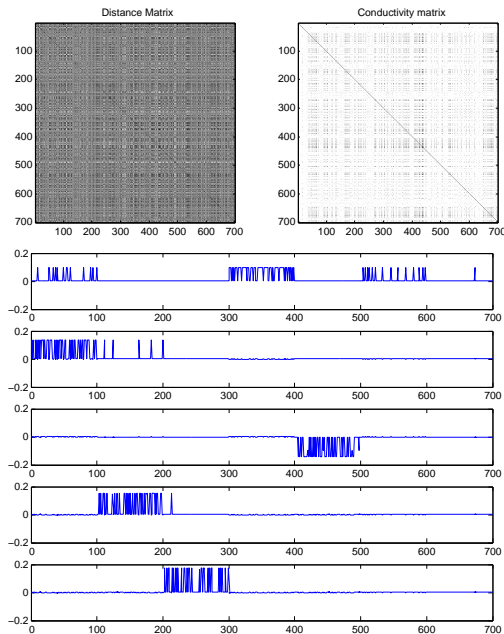


Figure 11: Real distance and conductivity matrix coming from the evaluation learning set followed by the first five eigenvectors

As can be seen in this figure the eigenvector have a clear block structure even if the initial distance matrix have not

been reordered to have a block-diagonal structure. The block structure of the eigenvector should ease the clustering using the k -lines algorithm.

Selecting Representative HMM for cluster. At the end of the clustering in eigenvectors subspace, each point related to a particular flow through a sequence $f_i = \{o_1, \dots, o_N\}$ and an HMM H_i will be assigned to one of the K clusters, C_k . However we have now to choose for each cluster the "best" representing HMM, H_k^* , $k \in \{1, \dots, K\}$. This choice might be done as usual through a Maximum likelihood criterion :

$$H_k^* = \arg \max_H \left(\sum_{i: f_i \in C_k} \log \text{Prob}\{f_i|H\} \right)$$

This choice finishes the first step of the methodology consisting of unsupervised learning. We will discuss in section 4 the results of this clustering on the learning set. However before doing this we need to decide how many clusters are needed to describe the learning set. This will be the subject of the next section.

3.7 Model selection and MDL criterion

Up to now we have assumed that the number of clusters K is a known value. However in real world the choice of this value is not straightforward and evaluating this number under general conditions is an open problem. Choosing the number of clusters can be seen as a *model selection* problem. We will describe here a model selection method based on the so-called Minimum Description Length (MDL) approach [1]. This method applies the Occam razor ¹ to choose the number of needed clusters through a trade-off between model accuracy as evaluated by the likelihood of the model and model complexity measured by the number of needed parameters.

Our aim here is to find a good clustering with not too many clusters and a sufficient goodness-of-fit to the data (a correct likelihood). The idea of MDL is to translate this problem into a *description length* problem. Let's suppose that one wants to describe the obtained clustering in any specific language he wants. The description of the data will consist of two parts : description of the model, and description of the data assuming the model. The description length \mathcal{L} is therefore,

$$\mathcal{L} = \text{Length}(\text{Data}|\text{Model}) + \text{Length}(\text{Model})$$

In our context the model is the cluster structure itself, meaning that we have to first describe the clusters and after that to describe the points inside the clusters.

MDL assumes that these two descriptions are done in the most compressed way through an Information Theoretic compression mechanism. Now, if the likelihood of observing data x through the model \mathcal{M} is given by $\text{Prob}\{x|\mathcal{M}\}$, the well-known Shannon compression theorem states that the length of the description of x will be larger than $-\log \text{Prob}\{x|\mathcal{M}\}$. We will assume that we are using an optimal compression code and therefore length of the representation converges asymptotically with large N (number of observations) to the log-likelihood, $\text{Length}(\text{Data}|\text{Model}) = -\log \text{Prob}\{x|\mathcal{M}\}$.

¹"*Pluralitas non est ponenda sine necessitate*" or in other terms, *entities should not be multiplied needlessly*

Moreover the description size of the model will depend on the number of parameters required by the model.

In our context the model consists of the obtained clustering containing K clusters, represented each by an HMM H_k^* with N_{\max} states. First of all we have to describe the value of K the number of clusters. As the number of bits needed for this description is not known *a priori* we have first to give this number through a sequence of $\log K$ zeros followed by a one used as a delimiter. After that the precise value of K , is provided using $\log K$ bits. All in all, $2 \log K + 1$ are needed to describe K . After that, we have to describe each one of the K representative HMMs. Each HMM has N_{\max} states and each state has $|\Sigma|$ emission probabilities associated. If we represent each emission probability using α (α is fixed and does not depend on K), each cluster would be represented by $\alpha \times N_{\max} \times |\Sigma|$ bits. This leads to a description size for the overall model equal to $Length(Model) = 2 \log(K) + \alpha K N_{\max} |\Sigma| + 1$.

The likelihood of the learning set knowing the model and the clustering might be derived as :

$$\mathbb{P}\text{rob}\{x|\mathcal{M}\} = \prod_{i=1}^N \mathbb{P}\text{rob}\{f_i|H_k^*, i \in \mathcal{C}_k\}$$

where $\mathbb{P}\text{rob}\{f_i|H_k^*, i \in \mathcal{C}_k\}$ is the probability that the sequence f_i is generated by the representative HMM H_k^* of the cluster that f_i belongs to. Mixing the model description length with the likelihood calculation we reach to the following description length $\mathcal{L}(K)$ as a function of cluster number :

$$\mathcal{L}(K) = -\sum_{i=1}^N \log(\mathbb{P}\text{rob}\{f_i|H_k^*, i \in \mathcal{C}_k\}) + 2 \log(K) + \alpha K N_{\max} |\Sigma| + 1$$

The MDL criterion chooses the value of K that minimizes the description size, *i.e.* $K^* = \arg \min_K \mathcal{L}(K)$. We are showing in Fig. 12 the description length curve obtained over the learning set as function of cluster number K .

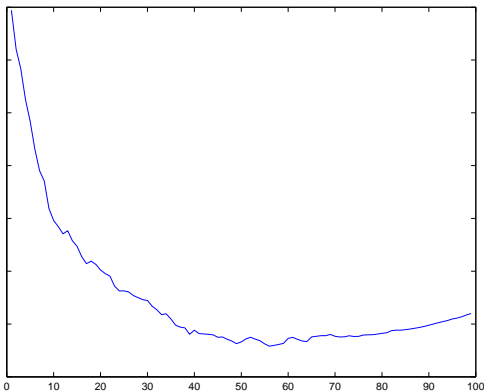


Figure 12: Description length as a function of number of clusters K

The curve shows a minimum of the description length at $K = 55$ meaning that 55 clusters generate the smallest description of the training set and is the MDL best estimate of cluster numbers. Doing clustering with a larger number of clusters yields some empty clusters, or too small clusters.

The fact that we have 55 clusters means that we have find over the trace 55 different type of behaviors. This observation have to be compared with the fact that we had in the training set 10 type of applications. This means that flow from the same application might exhibit different behaviors.

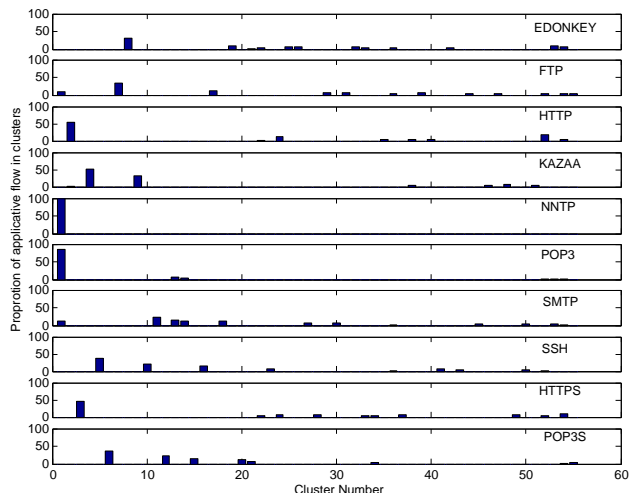


Figure 13: Distribution of different behavior among the 10 type of application used in the training set

Fig. 13 shows the Distribution of different behavior among different applications. Some application exhibits different behavior (as for example edonkey or FTP). The different behaviors are captured in different clusters. Some other applications appear remarkably homogeneous in term of behavior. NNTP is a very good example where 100% are in the same cluster. The dispersion of application between different behaviours lead to an applicative model that will be as a mixture of HMM. This point will be developed in section ???. In the next section we will analyze in more details the results of the application of the clustering method to the training set.

4. RESULTS

The methods presented in this paper are unsupervised. The idea is to find some similar behaviour without any knowledge about the application. Only after the clustering step we map the clusters to the applications. We will present the first step of clustering in the section ??. Then in the section ?? we will present the overall results.

4.1 Clustering results

Analyzing the clusters in detail can be very interesting for several reasons. First of all, each cluster describe one kind of behavior, and the relative weight of each cluster shows how common this behavior is in the studied training set. Moreover, looking at the applications appearing in each cluster (each flow is assigned a cluster and each flow is labeled with an application) shows that applications might exhibit different behaviors and that some applications have similar behaviors with others.

Using the trace described in 2 we ran the spectral clustering with 55 clusters. Figure 14 represents the quantized

packets size and the direction of clusters 1,2, 3 and 8 (in clockwise order). The first cluster contains 20.6% of the flows and the sequences for this cluster is shown in figure 14. Table 1 show the proportion of applications that fall in this cluster.

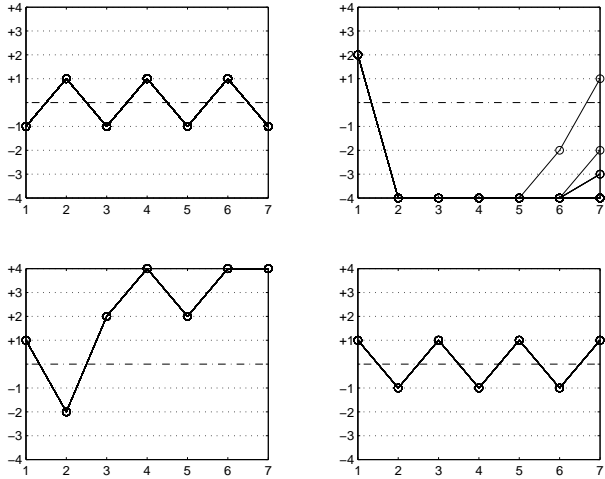


Figure 14: Packets Sequence for cluster 1,2,3,8 using the Spectral Clustering with 55 clusters.

Application	% in this cluster
NNTP	100%
POP3	86%
SMTP	11%
FTP	9%

Table 1: Applications in cluster 1

As we can see from 14 the first cluster gathers flows containing small packets sent alternatively by the client (i.e. the machine that started the TCP connection) and by the server (i.e. the machine that accepted the connection) starting with the server. If we give a closer look at the application seen in this cluster we can see that all of them, except SMTP, have a similar identification process (see Tab. 4.1 which make them having similar behavior.

Packet	NNTP	POP3	SMTP	FTP
1 (s → c)	Banner	Banner	Banner	Banner
2 (c → s)	User	User	EHLO	User
3 (s → c)	Pass?	User Ok	250 Ok	Pass?
4 (c → s)	Pass	Pass	MAIL FROM	Pass
5 (s → c)	Limit	Ok	Ok	Login OK
6 (c → s)	Quit	Stat	RCPT TO	Type
7 (s → c)	Quit	Ok	Ok	Ok

Table 2: Packets Exchanged for cluster 1.(s→c) describes a packet sent from the server to the client and (c→s) a packet sent by the client to the server.

Thus, the first packet is a welcome packet sent by the

server, while second, third, fourth and fifth are login exchanges and the sixth is a command sent from the client to the server. It is interesting to note that all the NNTP flows found in our data probably end with a "Daily Limit Exceeded" message. Some SMTP flows ends up in this cluster simply because the first packets exchanged to send an email can fit the identification behavior of the other applications (according to the way we build our sequences).

The second cluster is much simpler. It gathers 5.5% of the flows, all of which are HTTP ones. Besides 54% of the HTTP flows fall in this cluster. The packets sequence is represented in figure 14. Basically, the first packet is a medium sized one sent by the client, and the following packets are big ones sent back by the server. The first thing to say about this sequence is straightforward: the first packet is an "HTTP GET" and the following ones are the answer from the HTTP server. However there are a few interesting things to note. First of all the size of the answer vary and this is the reason why the sizes of packets 6 and 7 are not all quantized to the same value. Besides, some flows even end up with a packet being sent by the client. This can be interpreted as follow : some HTTP 1.1 flows can be found in this cluster and the next packet sent by the client is the following "HTTP GET".

We need to quantify how many behaviour can be identified within an Application.

We also need to quantify what are the overlapping of application within each cluster.

4.2 Application recognition

So far we described the unsupervised learning step of the methodology presented in Fig. 4. We will now analyze the second phase that is the recognition phase. In this phase, we use the results of clustering to derive a set of matched filter able to detect the applicative label of a flow using only the discretized packet size and the packet direction. We assume that the recognition have to be done based on only a small number of initial packets of a flow. Previous analysis and in particular Fig. 13 showed that a single application might exhibit different behaviors, and that a particular behavior is not specific to an application. This lead us to develop a recognition based on a mixture of HMM. This recognition mechanism is described in next section and it will be evaluated in section ??.

In order to evaluate the quality of our classification method we use a test set consisting in 5000 flows and check whether the application label found using our mixtures is right or not. Table 3 show the quality of our classification method. Real applications appear on the lines while applications found using our method appear in columns. Each cell represents the proportion of actual applications flows labeled with each possible application. This first line thus reads : among edonkey flows, 84.2% were classified edonkey, 4.2% ftp, 2.2% SMTP, 5.6% https and 2.8% pop3s.

A few conclusions can be derived from these results. First of all the overall classification performs quite well. If we take into account the relative weight of each application in the full initial trace, more than 89% of the flows would have been classified correctly. Besides, Peer To Peer applications which are known to be difficult to classify are labeled correctly 84.2% of the time for edonkey and 95% of the time for kazaa.

Another phenomenon needs to be explain : POP3 flows are labeled 100% of the time. This issue can be looked at

	edonkey	ftp	HTTP	kazaa	nntp	pop3	SMTP	ssh	https	pop3s
edonkey	84.2	4.2	1	0	0	0	2.2	0	5.6	2.8
ftp	0	87	2.4	0.2	6.2	0	4.2	0	0	0
HTTP	0	0	99	0	0	0	0	0	1	0
kazaa	0	0	4.76	95.24	0	0	0	0	0	0
nntp	0	0	0	0	99.6	0	0.4	0	0	0
pop3	0	0.6	0	0	86.8	0	12.6	0	0	0
SMTP	0	1.4	0	0	14.2	0	84.4	0	0	0
ssh	0	1.54	0	0	0	0	1.54	96.92	0	0
https	3.2	0	15	0	0	0	0	0	81.8	0
pop3s	2	0	0.4	0	0	0	0	0	7.8	89.8

Table 3: Classification results using *Spectral Clustering* with 55 clusters

from two different perspectives. From a networking point of view, the first seven packets of POP3 flows are identification packets, which once the quantization step done look a lot like SMTP and nntp first packets (in term of size and direction). This accounts for the confusion between the three applications. From a classification point of view, the reason why all POP3 flows fall either in the NNTP mixture or in the SMTP one (and never the other way around) can be explained by looking at figure 13. If a POP3 flows behaves exactly like the HMM describing cluster1, the computed probability of this flow being a POP3 one is going to be the log-likelihood multiplied by 86% (since 86% of the POP3 traffic belongs to cluster 1, which is going to affect our mixture composition accordingly). At the same time, the probability of this flow belonging to the NNTP mixture is going to be the same log-likelihood multiplied by 100%, since the mixture for NNTP consists only in the HMM describing cluster 1. Thus the NNTP probability is always going to be higher. The same could be said for SMTP if we consider clusters 13 and 14.

In the end our classification shows some limitation when quantized information about the first seven packets can not differentiate between two behaviors. However with a classification right more than 90% of the time except for POP3, it performs fairly well.

5. CONCLUSION

We presented in this paper a Blind applicative flow recognition through behavioral classification. The approach was based on very simple sequences of quantified packet size and packet direction. These sequences were clustered through a powerful spectral clustering algorithm. We developed thereafter a recognition algorithm based on a mixture of HMM representative of the obtained clusters. The presented method appear to be very powerful as it reach recognition performance of 90% with only observing seven packets of a flow!

This work is a first step toward an operational flow recognition system that will be robust toward flow morphing (tunnelling flow in other protocol) and payload encryption. It indeed remains some future to be done. We list here some of the extension we work on.

- We have used in this work only seven packet to recognize an application and we got good recognition rate. However, this number of packet might be too less or too much for specific applications. We are working on extension of this method to variable horizon recognition mechanism that will adapt the number of needed

packet to the suspected type of applications.

- We aimed toward a real-time running of the recognition algorithm on a Gigabit/sec link. For this purpose we are investigating how to implement the mechanism on a DSP or better on a Network processor architecture. The filter based structure of the algorithm should be very helpful in this context.
- In meanwhile we are developing hybrid recognition mechanism that will mix the blind recognition mechanism with the full packet payload system. The idea is to make use of the 99 % recognition rate of HTTP traffic of the proposed blind algorithm to reduce the load of a full packet payload analyzer but not forwarding these flow to the analyzer. As 90% of flows are HTTP, this will lead to a very important leverage of the full packet analyzer enabling it to cross the Gigabit/sec barrier.

6. REFERENCES

- [1] A.Barron, J.Rissanen, and B.Yu. The minimum description length principle in coding and modeling. *IEEE Trans. Information Theory*, 44, 1998.
- [2] C.Trivedi, H. Trussell, A. Nilsson, and M.-Y. Chow. Implicit traffic classification for service differentiation. In *15th ITC Specialist Seminar, Internet Traffic Engineering and Traffic Management*, 2002.
- [3] I. Fischer and J. Poland. New methods for spectral clustering. Technical report, IDISA, June 2004.
- [4] H.Binsztok, T. Artires, and P. Gallinari. A model-based approach to sequence clustering. In *ECAI*, Madrid, 2004.
- [5] F. Hernandez-Campos, A. Nobel, F. Smith, and K. Jeffay. Statistical clustering of internet communication patterns. In *Proceedings of the 35th Symposium on the Interface of Computing Science and Statistics*, 2003.
- [6] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone. In *IEEE Infocom*, San Francisco, march 2003.
- [7] T. Karagiannis, A. Broido, N. Brownlee, kc claffy, and M. Faloutsos. Is p2p dying or just hiding? In *Globecom*, 2004.
- [8] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. In *PAM 2004*, 2004.

- [9] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering : analysis and an algorithm. In *NIPS*, 2001.
- [10] F. Porikli. Trajectory distance metric using hidden markov model based representation. In *IEEE European Conference on Computer Vision, PETS Workshop*, 2004.
- [11] Qosmos. Traffic designer.
- [12] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 1989.
- [13] P. Smyth. Clustering sequences with hidden markov models. In *Advances in Neural Information Processing*, 1997.
- [14] D. Zuev and A. Moore. Traffic classification using a statistical approach. In *PAM 2005*, 2005.