

**Thèse de Doctorat de l'Université Paris VI  
Pierre et Marie Curie**

Spécialité

**SYSTÈMES INFORMATIQUES**

présentée par

**Laurent Bernaille**

pour obtenir le grade de

**Docteur de l'Université Pierre et Marie Curie**

**Classification temps réel d'applications  
sur l'Internet**

soutenue le 8 Juin 2007 devant le jury composé de

MM. :	Walid	DABBOUS	Rapporteurs
	Philippe	OWEZARSKI	
MM. :	Pierre	SENS	Examineurs
	Konstantina	PAPAGIANNAKI	
M. :	Eric	HORLAIT	Directeur
MM. :	Renata	TEIXEIRA	Encadrants
	Kave	SALAMATIAN	



**Thèse de Doctorat de l'Université Paris VI  
Pierre et Marie Curie**

Spécialité

**SYSTÈMES INFORMATIQUES**

présentée par

**Laurent Bernaille**

pour obtenir le grade de

**Docteur de l'Université Pierre et Marie Curie**

**Classification temps réel d'applications  
sur l'Internet**

soutenue le 8 Juin 2007 devant le jury composé de

MM. :	Walid	DABBOUS	Rapporteurs
	Philippe	OWEZARSKI	
MM. :	Pierre	SENS	Examineurs
	Konstantina	PAPAGIANNAKI	
M. :	Eric	HORLAIT	Directeur
MM. :	Renata	TEIXEIRA	Encadrants
	Kave	SALAMATIAN	



## Remerciements

Je remercie Walid DABBOUS, Directeur de Recherche à l'INRIA, et Philippe OWEZARSKI, Chargé de Recherche au CNRS, d'avoir bien voulu accepter la charge de rapporteurs.

Je remercie Pierre SENS, Professeur à l'Université Pierre et Marie Curie, et Konstantina PAPAGIANNAKI, Chercheur à INTEL Research, d'avoir bien voulu juger ce travail.

Je remercie enfin Eric HORLAIT, Professeur à l'Université Pierre et Marie Curie, Renata TEIXEIRA, Chargée de recherche au CNRS, et Kave SALAMATIAN, Maître de conférences à l'Université Pierre et Marie Curie, d'avoir bien voulu encadrer cette thèse. Qu'il me soit permis d'ajouter une mention toute particulière pour Renata Teixeira. Son niveau d'exigence m'a obligé à toujours plus de rigueur et de précision, et sa présence constante a été d'un grand soutien, notamment lors des moments difficiles que connaissent tous les thésards.

Bien sûr, merci à Georges, Chantal, Julien, Augustin et Guillaume qui ont toujours été là et qui partagent une valeur essentielle pour moi : le rire.

Merci à mes parents et à ma soeur pour m'avoir toujours soutenu et encouragé durant cette thèse.

Et enfin, surtout, merci à Marion, pour tout.



## Résumé

L'identification des applications utilisées sur un réseau est essentielle pour comprendre les évolutions du trafic et pour appliquer des politiques de qualité de service. Les méthodes de classification utilisant les numéros de port deviennent de moins en moins efficaces. Bien que plus précise, l'analyse systématique du contenu des paquets nécessite beaucoup de ressources et ne peut fonctionner lorsque le trafic est chiffré. Récemment, de nouvelles techniques utilisant des études statistiques sur les flux TCP ont fait leur apparition. Cependant, ces techniques nécessitent une analyse a posteriori des connexions, et ne peuvent donc être utilisées pour agir dynamiquement sur le trafic.

Dans cette thèse, nous proposons une méthode identifiant l'application associée à une connexion TCP à partir de ses premiers paquets uniquement. Tout d'abord, nous utilisons plusieurs techniques de clustering afin d'identifier les comportements de connexions les plus répandus. Ensuite, nous présentons des méthodes de classification qui associent les connexions à identifier aux clusters que nous avons trouvés et déterminent leur application. Nous évaluons ces méthodes sur des traces de trafic et montrons que la taille des quatre premiers paquets suffit pour identifier 90% des connexions. Nous décrivons ensuite une implémentation de notre méthode capable de supporter un débit de 6Gbit/s. Enfin, nous étendons notre méthode aux connexions chiffrées avec SSL et montrons que nous pouvons identifier l'application encapsulée dans plus de 80% des cas.

## Mots-Clés

Classification du trafic Internet, TCP, Applications, Intelligence Artificielle, SSL



## **Abstract**

The automatic detection of applications associated with network traffic is an essential step to apply quality-of-service policies and profile network usage. Unfortunately, simple port-based classification methods are not always efficient, systematic analysis of packet payloads is too slow and cannot work on encrypted traffic. Recent research proposes to address these issues by using flow statistics to classify traffic flows. However, these methods rely on statistics about whole connections and can only classify traffic after the connection is finished. Therefore, these approaches cannot be used to apply dynamic per-application policies on the traffic.

In this thesis, we present a method to identify the application at the beginning of a TCP connection. We identify common connection behaviors using three clustering techniques: K-Means, Gaussian Mixture Model and spectral clustering. We use resulting clusters 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 for 60% of the connections. Then, we present an implementation of our method that is able to classify traffic at up to 6 Gbit/s, which is more than enough for edge routers. Finally, we extend our method to the classification of connections encrypted using SSL and show that we can identify the encapsulated application with more than 80% accuracy.

## **Keywords**

Internet traffic classification, TCP, Applications, Machine learning, SSL



# Contents

<b>1</b>	<b>Introduction</b>	<b>15</b>
1.1	Challenges . . . . .	17
1.1.1	Fast evolution of applications . . . . .	17
1.1.2	Limited information on connections . . . . .	18
1.1.3	Online implementation . . . . .	18
1.2	Early Application Identification . . . . .	19
1.3	Contributions . . . . .	20
1.4	Outline . . . . .	22
<b>2</b>	<b>Background on traffic classification</b>	<b>23</b>
2.1	Measurement techniques . . . . .	24
2.1.1	Per-packet capture . . . . .	24
2.1.2	Summarizing traffic statistics in routers . . . . .	25
2.2	Methods to classify traffic . . . . .	26
2.2.1	Port-based classification . . . . .	26
2.2.2	Content-based classification . . . . .	28
2.2.3	Behavior-based classification . . . . .	29
2.3	Summary . . . . .	33
<b>3</b>	<b>Trace analysis for feature selection</b>	<b>35</b>
3.1	Description of data sets . . . . .	36
3.1.1	Packet traces . . . . .	36
3.1.2	Parsing packet traces . . . . .	37
3.1.3	Filtering Packet Traces . . . . .	38
3.1.4	Per-Application Traces . . . . .	38

3.2	Feature selection . . . . .	39
3.2.1	Comparison of connection metrics . . . . .	40
3.2.2	Feature stability across networks . . . . .	43
3.3	Summary . . . . .	43
<b>4</b>	<b>Offline training</b>	<b>45</b>
4.1	Training Traces . . . . .	46
4.2	Representation of a TCP connection . . . . .	48
4.2.1	Euclidean Representation . . . . .	48
4.2.2	Representation using Hidden Markov Models . . . . .	49
4.3	Clustering Algorithms . . . . .	50
4.3.1	Choice of the algorithm . . . . .	50
4.3.2	K-Means Clustering in the Euclidean Space . . . . .	52
4.3.3	GMM Clustering in the Euclidean Space . . . . .	55
4.3.4	Spectral Clustering in the HMM space . . . . .	56
4.4	Filtering . . . . .	58
4.5	Calibration . . . . .	59
4.5.1	Clustering quality metric . . . . .	59
4.5.2	Number of clusters . . . . .	61
4.5.3	Number of packets . . . . .	63
4.6	Summary . . . . .	64
<b>5</b>	<b>Online Classifier</b>	<b>65</b>
5.1	Assignment Heuristics . . . . .	66
5.1.1	K-Means . . . . .	66
5.1.2	Gaussian Mixture Model . . . . .	69
5.1.3	Hidden Markov Model . . . . .	70
5.2	Labeling Heuristics . . . . .	71
5.2.1	Predominant heuristic . . . . .	72
5.2.2	Cluster+Port heuristic . . . . .	72
5.3	Heuristic evaluation . . . . .	73
5.3.1	Assignment Accuracy . . . . .	73
5.3.2	Labeling Accuracy . . . . .	75
5.3.3	Detection of new applications . . . . .	79
5.3.4	Discussion . . . . .	79
5.4	Implementation . . . . .	80
5.4.1	Algorithm . . . . .	81

5.4.2	Optimizations . . . . .	81
5.4.3	Complexity of Online Classification . . . . .	83
5.4.4	Evaluation . . . . .	85
5.5	Challenges . . . . .	86
5.6	Summary . . . . .	87
<b>6</b>	<b>Identification of encrypted traffic</b>	<b>89</b>
6.1	Packet traces with encrypted traffic . . . . .	90
6.2	Analysis of SSL traffic . . . . .	91
6.2.1	Description of SSL . . . . .	91
6.2.2	Identifying SSL connections . . . . .	92
6.2.3	Description of SSL traffic . . . . .	93
6.3	Classification of SSL connections . . . . .	95
6.3.1	Detection of the first data packet . . . . .	95
6.3.2	Inferring the original packet sizes . . . . .	96
6.3.3	Method overview . . . . .	98
6.4	Evaluation . . . . .	99
6.4.1	Recognition of SSL traffic . . . . .	99
6.4.2	Recognition of encrypted Applications . . . . .	99
6.5	Summary . . . . .	100
<b>7</b>	<b>Conclusion</b>	<b>103</b>
7.1	Contributions . . . . .	104
7.2	Future directions . . . . .	105
<b>A</b>	<b>Thesis' French Version</b>	<b>107</b>
A.1	Introduction . . . . .	108
A.1.1	Problématique . . . . .	109
A.1.2	Plan . . . . .	111
A.2	Identification des applications en temps réel . . . . .	112
A.2.1	La distinction des applications . . . . .	114
A.2.2	L'apprentissage . . . . .	116
A.2.3	La classification temps réel . . . . .	122
A.2.4	Implémentation de notre méthode . . . . .	127
A.2.5	La classification du trafic chiffré . . . . .	128
A.3	Conclusion . . . . .	133
A.3.1	Contributions . . . . .	133

A.3.2 Perspectives . . . . .	135
<b>Bibliography</b>	<b>142</b>
<b>List of figures</b>	<b>144</b>
<b>Liste of tables</b>	<b>146</b>

Chapter **1**

# Introduction

## Contents

---

<b>1.1</b>	<b>Challenges . . . . .</b>	<b>17</b>
1.1.1	Fast evolution of applications . . . . .	17
1.1.2	Limited information on connections . . . . .	18
1.1.3	Online implementation . . . . .	18
<b>1.2</b>	<b>Early Application Identification . . . . .</b>	<b>19</b>
<b>1.3</b>	<b>Contributions . . . . .</b>	<b>20</b>
<b>1.4</b>	<b>Outline . . . . .</b>	<b>22</b>

---

Over the last twenty years, the Internet has evolved from a small academic network into an interconnection of tens of thousands of autonomous networks. This fast development of the infrastructure led to a drastic evolution of the usage of the Internet. At first, the Internet was limited to exchanges of information through e-mails and newsgroups. Then, the need to find and organize information led a few years later to the development of the first web pages using hypertext. Today, millions of people use the Internet, end-hosts have fast processors, large hard-drives and fast connections to the Internet. This evolution has allowed the development of a large variety of applications, which continue to appear on a monthly basis.

In parallel to the evolution of applications, the structure of the Internet has evolved. We can now divide the networks that compose the Internet in two main categories: those that provide connectivity to other networks (transit networks), and those that connect primarily end-hosts (edge networks). Transit networks belong to Internet Service Providers (such as Sprint or France Telecom). Edge networks (for example, campus or enterprise networks) depend on transit networks to access the Internet. Figure 1.1 illustrates this relationship. Transit and edge networks have different needs and therefore are engineered differently. Transit networks need very fast links (often 10Gbits/s) to transfer traffic from multiple edge networks, whereas edge networks only need to provide Internet connectivity to their end-hosts and do not require as much bandwidth (from a few Mbits/s for small enterprises to 1Gbits/s or 2.5Gbits/s for large enterprises or campuses).

Administrators of edge networks often want to apply different quality of service policies depending on the applications (to prioritize Voice over IP connections over data transfers, for instance). Besides, network administrators need to know what applications traverse their network to enforce institutional policies (such as no peer-to-peer traffic) and plan for traffic evolution. This requirement leaves network administrators with the daunting task of (1) identifying the application associated with a traffic flow as early as possible, and (2) applying network policies when needed. Therefore, accurate and early classification of traffic flows is an essential step for administrators to apply dynamic quality-of-service policies and have a comprehensive view of current trends in the traffic.

In this thesis, we present a method to identify the applications associated with a traffic flow early (i.e., after a few packets of the flow), on the link that connects an edge network to the Internet. Our method relies on the payload sizes of the first packets in a TCP connection that contain application data. In this chapter, we first describe how application data is transferred in the Internet. Then, after explaining

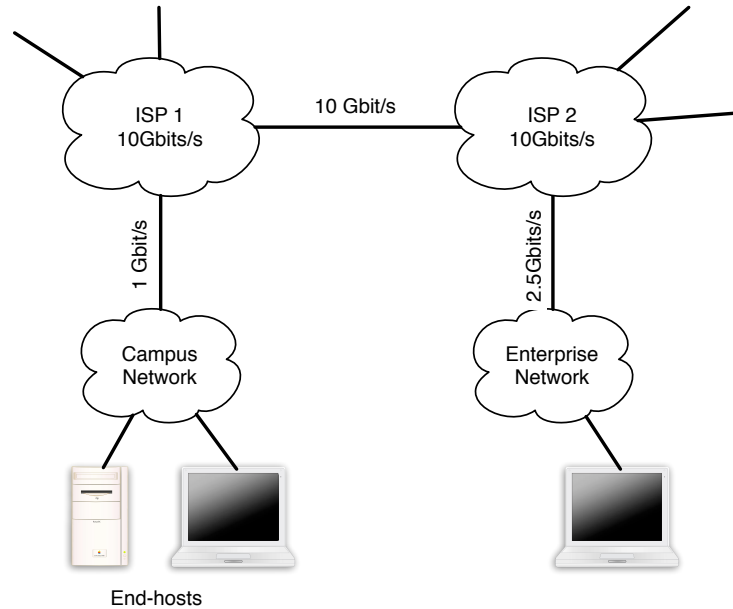


Figure 1.1: Sample internetwork. Transit: ISP 1 and ISP 2. Edge: Campus and Enterprise

the challenges associated with traffic classification, we present our method for early application identification and detail the main contribution of this thesis.

## 1.1 Challenges

To design a classification method that identifies the applications associated with TCP connections early, we face many challenges.

### 1.1.1 Fast evolution of applications

Until the end of the 1990s, Internet applications relied on the Client-Server architecture and their behavior was simple: some Internet hosts were dedicated to provide services, while the vast majority of hosts were simply clients of these services. The communication between clients and servers relied on connections using standardized ports and were easy to identify. However, many new applications have appeared in the last decade. Some of them are hard to identify because they do not use standard ports [65, 24].

First, **some applications use dynamically negotiated connections**. For instance, the File Transfer Protocol (FTP) uses several connections between the two hosts in communication: one connection for control traffic and the others for data. The parameters of the connections used to transfer data are negotiated in the control connection. Therefore, it is often easy to identify the control connection, which usually uses destination port 21, but much more difficult to recognize data connections. Voice-over-IP protocols such as H323 also use several connections for a single communication.

Second, **some applications are not standardized**. These applications use proprietary protocols and are hence more difficult to recognize. This is typical of recent peer-to-peer software, which now represent a large share of Internet traffic [42]. Furthermore, these **applications often try to hide** because they are used to illegally transfer movies or software and are therefore forbidden on many networks.

### 1.1.2 Limited information on connections

To classify connections early (i.e., after only a few packets), we can only have access to the **information that is available in the first packets**. Moreover, we can only **rely on information from packet headers**, because packet payloads can be encrypted. The use of SSL encryption has increased over the last few years, as supported by the SSL surveys from Netcraft [52]. We can expect SSL to continue to develop, especially for peer-to-peer traffic (widespread bittorrent software such as Azureus already support SSL encryption). In addition, even when application data is sent without encryption, it may not be possible to parse packet payloads for privacy reasons.

### 1.1.3 Online implementation

The connection of large networks to the Internet usually relies on **very fast links**. Many universities use 1Gbits/s Ethernet connections (for instance, the link connecting the Université Pierre et Marie Curie to Renater, the French academic network, is 1Gbits/s). Some larger networks already use OC-48 links (2.5Gbits/s), and we can expect the bandwidth of edge links to continue to increase. These high speeds impose strict performance requirements on tools analyzing the packets on the link.

**Memory is a major bottleneck** for tools that keep track of ongoing TCP connections [26], because they need to keep information about all active connections. On high-speed links, the number of active connections can exceed one million [38], which involves a large amount of memory. Besides, the traffic analyzers need to perform nu-

merous lookups in the list of connections to verify for each packet if it belongs to an existing connection, or to a new one. These lookups require fast memories.

## 1.2 Early Application Identification

Several techniques are currently available to identify the application associated with a TCP connection.

- **Port-based methods** label connections based on TCP server ports. They are simple but often inaccurate.
- **Content-based methods** analyze packet content to identify the application. They are accurate but computationally intensive.
- **Behavior-based methods** compute statistics on TCP connections to classify them. They are efficient but can only label connections when they are finished.

We present all these techniques in detail in Chapter 2.

In this thesis, we propose a method to classify connections using behavior-based methods that only rely on information from the first few packets in a connection, and addresses the challenges presented in 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 Chapters 4 and 5, respectively. Figure 1.2 presents an overview of our method. The left part represents the steps of the training phase and the right part the components of the classifier. These two phases run at separate locations and time frames. The training phase runs offline at a management site, whereas the classifier runs online in a distinct host that has online access to packet headers or in a network processor at the monitored router.

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 Hidden Markov Models (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 that lists the applications that are present at each cluster.

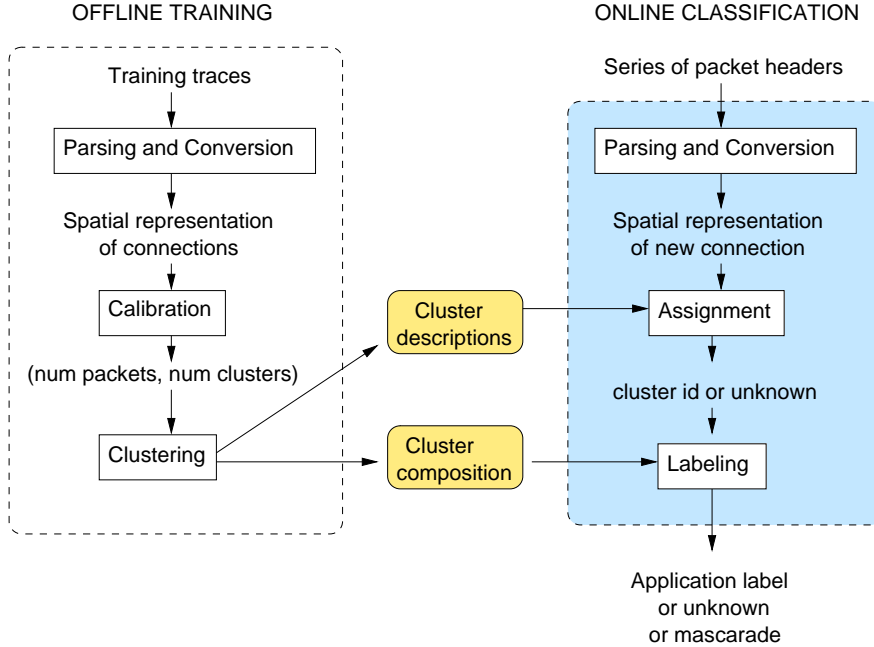


Figure 1.2: Approach overview

Our classifier takes as input the series of packet headers for both directions of an edge link (since our classification runs on the link connecting edge networks to the Internet, we have access to both directions for most networks, as shown in Figure 1.1 for instance). 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. By using the composition of this cluster, the labeling module selects which application is most likely associated with the connection.

### 1.3 Contributions

This thesis makes the following contributions.

**Study of the most appropriate features for early traffic classification:** we compute different features based on information from the first packets in TCP connec-

tions (mean packet size, or mean Inter-Arrival Time, for instance). Then, we compare these features over a set of applications to find the best metric to distinguish these applications. Our experimental analysis shows that the sizes of just the first few packets help to identify applications, whereas time-related metrics are not effective. In addition, we compare the sizes of the first packets for several applications using traces from different networks. Our study shows that these sizes do not vary across networks, and therefore that a classifier using this feature and designed for one network can be applied to others.

**Selection of a training method:** we analyze different techniques to design a model using the sizes of the first packets. These techniques rely on clustering algorithms to find connections with similar behaviors according to the sizes of the first packets. We present two spatial models to represent TCP connections: Euclidean and based on Hidden Markov Models (HMM). Based on these representations, we describe and evaluate three clustering methods: K-Means and Gaussian Mixture Model (GMM) in the Euclidean space, and spectral clustering in the HMM-based space. We propose a method to calibrate our clustering algorithms and choose the appropriate number of packets and the optimal number of clusters. 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. In addition, we use different training sets to verify that our results are robust to the choice of the training set.

**Design of an online classifier:** this classifier uses the model obtained during the training phase: the identified clusters and their composition (i.e. the applications associated to the training connections assigned to each cluster). We define and compare a number of classification heuristics. We describe different heuristics to associate incoming connections with clusters identified in the training phase. In particular, we propose heuristics that detect connections that should not be assigned to any of the clusters and label them as *unknown*. We also present two heuristics to label connections. The heuristics rely on the cluster compositions and on additional information such as port numbers. We evaluate these different heuristics in detail and show that our classifier achieves over 90% accuracy, and is able to label most connections of new applications as *unknown*.

**Classification of encrypted traffic:** since our method does not inspect content, it is possible to use it to classify encrypted traffic. We characterize the usage of a

common encryption protocol, SSL, on two campus networks and show that this usage is growing, and that the number of applications using SSL is increasing. We present a modified version of our method that identifies the applications encapsulated in SSL tunnels. First, we show that our method can effectively detect SSL connections. Then, we describe a method to find the first packet in an SSL connection with application data, and a heuristic to evaluate the size of the original packets based on their encrypted size. Finally, we evaluate this method and show that we can use this method to effectively identify the application associated with a connection encrypted with SSL.

**Implementation and evaluation of our classifier:** we describe an implementation of our method along with a set of optimizations to speed-up the classification and limit memory usage. We show that our implementation can classify traffic at rates up to 6Gbits/s, which is considerably higher than typical links for edge networks. In addition, we profile our program and show that our classification method can be added to any tool that gathers per-connection statistics at a very small cost.

## 1.4 Outline

Chapter 2 presents existing techniques to measure traffic on high-speed links and to classify TCP connections. Chapter 3 presents the traces we used for this thesis and how we selected the features to design our classifier. Chapter 4 describes the offline training phase. We present our classification heuristics and an implementation of our classifier in Chapter 5 and Chapter 6 present an extension of our method to connections encrypted with SSL. Finally, Chapter 7 summarizes the contributions of this thesis, analyzes our results and discusses future work and research directions.

Chapter **2**

# Background on traffic classification

## Contents

---

<b>2.1</b>	<b>Measurement techniques</b>	<b>24</b>
2.1.1	Per-packet capture	24
2.1.2	Summarizing traffic statistics in routers	25
<b>2.2</b>	<b>Methods to classify traffic</b>	<b>26</b>
2.2.1	Port-based classification	26
2.2.2	Content-based classification	28
2.2.3	Behavior-based classification	29
<b>2.3</b>	<b>Summary</b>	<b>33</b>

---

This chapter details current techniques to analyze traffic on edge links. We start by describing traffic measurement techniques, which are central to any classification method. Then, we focus on the current methods employed to identify the applications associated with TCP connections.

## 2.1 Measurement techniques

This section discusses different methods to measure traffic. After presenting techniques to capture packets traversing a link, we describe measurement tools embedded in IP routers.

### 2.1.1 Per-packet capture

A common method to analyze traffic consists in capturing all packets on a link for a certain duration and analyze them later. Offline analysis allows complex computations because it does not need to run at line speed. `Tcpdump` [77] is the most common tool to acquire packets entering and leaving a machine. `Tcpdump` captures all packets on a specified network interface, and provides filters to store only specific packets. Packet traces can get very large: a host transferring at a 1Gbit/s rate generates 3.6TB of traffic every hour. Therefore, an important parameter of `Tcpdump` is the number of bytes captured for each packet. It is usual to capture TCP and IP headers only. In this case, `Tcpdump` captures  $14 + 20 + 20 = 54$  bytes for each packet on Ethernet links (Size of Ethernet header: 14 bytes, size of IP header without options: 20 bytes, size of TCP header without options: 20 bytes). `Tcpdump` traces use a format called *pcap*. `Pcap` stores all packets with an additional header that includes a time-stamp, the actual size of the packet captured, and the number of bytes from the packet stored in the trace.

`Tcpdump` can capture traffic from individual hosts but it can also be used to capture traffic from a whole network. An administrator can run `tcpdump` on a monitoring machine that receives all packets entering and leaving the network. This operation is possible because most modern switches have port-mirroring capabilities and can therefore forward the traffic entering and leaving the network on a specific interface connected to the monitor. When `tcpdump` captures all traffic from a network, it raises concerns about privacy. Packets transport user data, which is often unencrypted. This privacy issue is another reason not to capture packet payloads.

`Tcpdump` is an efficient tool to capture traffic. However, it does not work on very fast networks because neither the network stack of most operating systems, nor existing

network interface cards can handle high throughput. To solve this problem, the Wand research group at the Waikato university developed specialized network monitoring boards able to capture 100% of the packets on very fast links, using a technology called DAG (Data Acquisition Generation) [14]. The success of the DAG project led to the foundation of Endace [23], a company that develops the most widespread monitoring cards (*DAG cards*). These cards behave exactly like tcpdump and copy the first bytes of all packets (the number of bytes is configurable) entering the card to a packet trace. DAG traces are similar to traces obtained with tcpdump but use a different format. The research team from the IPMON project [38] used DAG cards extensively to capture and analyze traffic on the Sprint backbone, because no other tool was capable of capturing all traffic on their high-speed links. To associate all captured packets with an application, the IPMON project relied on port numbers.

The DAG cards were the first to enable the capture of all packets on high-speed links. Today, other alternatives are available, like the Combo6 card developed by Cesnet [13] and used in the Scampi [67] and Liberouter [45] projects. In this thesis, we relied on a DAG card to capture the traffic from the Paris 6 university network, because the university uses a 1Gbit/s link to connect to the Internet, which is too fast for tcpdump. However, we could have used any other tool allowing the capture of all packets on 1Gbit/s links. To perform this capture, we configured a switch between the edge router and the university router to mirror all traffic (from and to the router) on a port where we plugged the DAG card.

### 2.1.2 Summarizing traffic statistics in routers

The main limitation of traffic measurement techniques is the volume of data. On very fast links, it is not possible to capture all the traffic for more than a few minutes. Therefore, another method consists in performing a part of the traffic analysis online, without storing the packets. Most routers have the capability to extract statistics of the traffic that traverses them (for instance, Cisco routers provide a tool called Netflow [54] that gathers statistics about TCP and UDP connections). On very fast links, routers cannot analyze every packet and hence rely on packet sampling (they analyze one packet out of one thousand, for instance).

When a router uses sampling, it cannot retrieve all packets from a given flow, which makes the computation of flow statistics difficult. To address this challenge, Duffield et al. [21] propose a method to recover the mean number of packets per flow from sampled traffic. However, Hohn and Veitch [35] prove that it is not possible to precisely infer more complex statistics, such as the distribution of the number of packets per flow,

when using packet sampling. They recommend the use of flow sampling (i.e., collect all packets from a subset of the connections), which makes the inference of such statistics possible and accurate. In addition, Netflow shows some limitations in terms of scalability (processing power and memory usage) as described in [26]. Therefore, Netflow only gathers very simple statistics and does not permit any additional computation on packets. Choosing the right metrics to precisely describe IP flows with as few bytes as possible and efficient packet sampling are still open problems. Two working groups at the Internet Engineering Task Force (IETF) are currently working on these topics. The *IP Flow Information Export* working group (ipfix [37]) is developing standards to describe IP flows and transfer IP flow data; and the *Packet Sampling* working group (psamp [61]) is defining a standard set of capabilities to sample subsets of packets.

In this thesis, we propose a classification method for edge networks. Therefore, we can analyze all packets and we do not need to use sampling. However, our method could work on faster networks where sampling is compulsory as long as we have access to both directions of the traffic, and use flow sampling (to make sure we have the first packets in sampled connections).

## 2.2 Methods to classify traffic

After capturing information about the traffic using one of the techniques from the previous section, the next step is to perform the classification. There are several approaches to classify connections according to their application. In this section, we first present the simplest method, which relies on TCP ports. Given the limitations of this approach, we present the latest alternatives proposed by the research community: techniques based on protocol signatures and behavior-based methods.

### 2.2.1 Port-based classification

The most common classification method relies on TCP and UDP port numbers. For standard services such as web or mail, the server port is standardized. Standardized port numbers allow any client to connect to the correct application on a host providing this service. A central registry, IANA (Internet Assigned Numbers Authority [36]), is responsible for defining the port number for each standard service. For instance, web servers using the *HTTP* protocol to transfer web pages should run on port 80.

The classification of TCP connections based on ports is pretty straightforward. When the classifier identifies a connection (using the 5-tuple), it assigns the connection to an application according to the TCP port on the server side of the connection, using

Service	Protocol Name	Port
Web (clear)	HTTP	80
Web (encrypted)	HTTPS	443
Mail (clear)	POP	110
Mail (clear)	IMAP	143
Mail (encrypted)	POPS	995
Mail (encrypted)	IMAPS	993
Mail (sending)	SMTP	25
Remote administration	SSH	22
File transfer (control)	FTP control connection	21

Table 2.1: Example of mapping between applications and ports

a simple translation table such as Table 2.1. The first application classifiers were based on ports. The main advantage of port-based classification compared to other techniques is its simplicity: extracting TCP ports from IP packets involves very little computation. Most firewalls, both software (like iptables [53]), and hardware (like Cisco PIX [59]) use port-based classification. When a company wants to limit the access of Internet users to only a few services provided by their servers (usually mail and web), they rely on a firewall that only allows connections from outside hosts to these servers on the associated ports (25 and 80). Similarly, many companies or universities filter outgoing traffic (connections initiated inside their network) to enforce policies on the use of the network (for instance, the UPMC university forbids the use of Skype, and therefore blocks connections on ports associated to Skype).

Nevertheless, many recent studies show that this method is not as reliable as it was a few years ago. Several reasons explain this phenomenon. First, applications such as FTP, peer-to-peer and games use dynamic port-negotiation. These applications use several connections: a control connection on a standard port to negotiate the ports that will be used in the following connections. Thus, even though port analysis can identify the control connections using well-known ports (port 21 for FTP, for instance), it will not find the other connections, which use dynamically negotiated ports<sup>1</sup>. FTP, H323 or SIP are very good examples of applications using dynamic ports. Moreover, some applications use non-standard ports to avoid detection. Peer-to-peer applications, which many operators try to forbid on their network, use ports not assigned by IANA.

---

<sup>1</sup>When FTP is in active mode, the server initiates the data connection using port 20 as source port. Port-based methods could identify this connection using this information. However, most FTP traffic now relies on passive mode due to NAT and firewall limitations. When FTP uses passive mode, data connections cannot be identified using ports.

A recent peer-to-peer study by CAIDA [42] confirms that peer-to-peer applications are increasingly difficult to identify using ports. It shows that peer-to-peer traffic seems to have decreased according to ports, but that a detailed analysis of packet contents proves the contrary. This problem shows up clearly when we look at the application breakdown on the Sprint backbone, as presented on the IPMON website [38]. Classification using well-known ports labeled 15% of the packets as unknown in 2000, whereas in February 2005, 35% of packets were not labeled with a standard application on the same link. These two studies represent a general trend: an increasing proportion of the traffic that uses non-standard ports.

Finally, some applications may even use a well-known port for a different usage to bypass firewall rules or to avoid detection. For example, MSN Messenger, a common instant messaging application, runs on port 80 and on port 443 to be usable on networks blocking all traffic except web (and therefore blocking traffic on all ports except 80 and 443). We call this behavior *Masquerade*.

Since port-based classification methods lead to an increasing amount of unclassifiable traffic and cannot flag instances of masquerade traffic, the research community and security experts developed new methods.

### 2.2.2 Content-based classification

An alternative approach is to inspect the payload of every packet searching for specific protocol signatures. For instance, a simple signature for HTTP would be “GET HTTP”. A content-based classifier would label any connection with a packet containing this string as HTTP. This method is very close to techniques from Intrusion Detection Systems (IDS) like Snort [70] or Bro [58]. IDSes analyze packet payloads and match them against known signatures. Pattern-matching methods [46, 44] are efficient to identify most applications, because protocols rely on standardized messages, which differ from one application to another. Table 2.2 shows a few simple signatures for common protocols.

L7-filter [44] is a Linux classifier that identifies applications using signatures. It consists of a patch for the Linux kernel and of list of signatures for a large set of applications. Several tools rely on the signatures from l7-filter to identify connections (for instance, the analyzer proposed in [20]). Finding protocol signatures is not an easy task and the signature list must be frequently updated to include new applications. To tackle this issue, Ma et al. [46] propose a method to automatically find application signatures by grouping connections based on their content.

Content-based approaches are much more reliable than port-based methods: chang-

Protocol Name	Signature
HTTP	GET * HTTP/1.?
POP	+OK Password required for *
IMAP	OK *IMAP *
SMTP	220 * SMTP *
SSH	SSH-*
FTP control connection	*LIST*

Table 2.2: Common protocol signatures

ing TCP ports is easy, but altering the protocols themselves is far more difficult. [42] presents a peer-to-peer recognition system which relies on pattern matching and is much more effective than port-based analysis. Some companies (like Qosmos [62]) now sell content-based classifiers that are more efficient than classic firewalls to detect and stop specific applications.

This method 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). Moreover, finding signatures for all protocols remains difficult and may sometimes not be good enough. For instance, many peer-to-peer applications rely on the HTTP protocol to transfer data between peers, and a pattern-based mechanism might thus mistakenly classify these connections as HTTP. Finally, payload information is not useful when applications use encryption.

### 2.2.3 Behavior-based classification

Since port-based classification is not always efficient and given the limitations of searching payloads for signatures, there has been a new trend to develop alternative classification techniques in the research community. These methods share a common basis: instead of looking at ports or packet content they analyze the behavior of TCP connections. The behavior corresponds to metrics that capture the communication pattern of a connection, such as the packet size, inter-arrival time or direction (client to server or server to client). Behavior-based classifiers associate communication patterns with applications. Table 2.3 presents the most common metrics used in behavior-based classification.

These methods create models of applications based on these behaviors and can rely on different machine learning techniques [31]:

Metric
Duration of the connection
Total volume transferred
Number of packets exchanged
Inter-Arrival Time between packets
Distribution of packet sizes

Table 2.3: Metrics for behavior based classification methods

- *Supervised learning* techniques use a training data set with application labels for all connections. These techniques define a function that associates an observation (behavior of the connection) described with attributes (such as number of packets, duration of the connection) with a class (application label). They estimate the parameters of this function using the labeled samples from the training set. A common supervised learning technique is Naive Bayes Classifiers. Naive Bayes Classifiers model each class independently using the observed attributes, with the assumption that these attributes are normally distributed. They estimate the parameters of these distributions (mean and variance) using all the observations associated to each class. Naive Bayes Classifiers then associate new observations to a class using a maximum-likelihood criterion.
- *Unsupervised learning* techniques involve two steps. First, these techniques re-group connections from the training set into clusters based on their behavior (they do not take application labels into account). Then, they design a classifier based on the obtained clusters and their composition (i.e., the labels of the connections assigned to each cluster).
- *Semi-supervised learning* techniques are an alternative to the previous techniques when labeling all connections in the training set is difficult and time consuming. These techniques propose to regroup connections into clusters and to label only a few connections in each cluster [25]. Then, they associate each cluster with an application based on these labels.

We can classify behavior-based methods according to their goal, their modeling approach, and their metrics to capture connection behaviors.

### Identification of classes of applications

Some behavior-based approaches [47, 65, 84, 50] do not identify the protocols behind connections. Instead, they associate connections to classes of traffic (such as bulk transfer, interactive, or games) and therefore cannot identify protocols precisely.

McGregor et al. [47] presents one of the first behavior-based classification method. This method uses supervised learning to model applications. It relies on a probabilistic clustering method using Bayesian classification. This technique models each application based on metrics such as packet size, duration, total volume and inter-arrival time, and find the parameters of the models with the Expectation Maximization algorithm. This work only describes a method without an evaluation on test traffic.

Moore et al. [84], [50] improve the classification technique from [47]. They also design a model that uses Bayesian techniques. However, they explore a richer set of metrics and describe a method to identify the relevant metrics for their classifier. They study 250 different metrics [83], which mostly derive from packet sizes, timing information and TCP control information (such as window size, number of ACKs and header size). They use Fast Correlation-Based Filter (FCBF) to identify the metrics that discriminates better between applications. They find that the best metrics derive from packet sizes (mean, average, median) and TCP information (Server Port, Initial Window, Minimum Segment Size). They design a classifier using these metrics and evaluate it on test traffic from a university network. Their classifier accurately labels 93% of the connections in their test trace.

Roughan et al. [65] propose an alternative method to find classes of applications in large and fast networks. Instead, of using supervised learning, they use unsupervised learning. They first regroup connections with similar behaviors without a priori knowledge of applications and then analyze the resulting clusters. To identify clusters, they test two algorithms: Linear Discriminant Analysis and k-Nearest Neighbor. They compare different features and conclude that average packet size and connection duration give the best clustering. Their evaluation shows that their classifier is very efficient to detect if a connection corresponds to a large transfer, interactive traffic or streaming (90% accuracy). However, this method cannot classify connections according to their protocol. Among methods that identify classes of applications, this method is the closest to our approach, because instead of building per-application models, it starts by clustering connections according to their behavior. In addition, they use simple metrics and show that they can design an efficient classifier with them. The main difference with our technique is that they use statistics on complete connections, whereas we only rely on the first packets.

### **Identification of protocols**

The earlier work on traffic classification, proposed by McGregor et al. [47], Moore et al. [50], and Roughan et al. [65] only recognize classes of traffic. We now discuss more recent proposals that identify application protocols.

Wright et al. [81] propose the first classifier that identifies protocols. They model each protocol using an HMM (Hidden Markov Model) that models the sequence of packets in the connection. The output of each state in their HMM rely on two metrics: packet sizes and time stamps. To label a connection, their classifier computes the probability of generating the packet sequence for the HMMs associated to all protocols. Then, they use Maximum Likelihood to associate the connection with a protocol. They can label connections with accuracies ranging from 60% to 90% depending on the protocol. This is the only approach that relies on information from each packet and not on aggregate features such as mean packet size. We also propose to use HMM in our technique. However, instead of using all connections from an application to compute an HMM that models this application, we choose to associate each connection with an HMM and use the resulting HMMs to compute a similarity between connections. In addition, we only use the first packets in a connection, whereas Wright et al. uses all packets.

Erman et al. [24] rely on clustering of connections in the training set. This method is similar to Roughan et al. [65], with the exception that [24] identifies protocols instead of classes. To regroup connections according to their behavior, this method represents these connection spatially based on their number of packets, mean packet size, mean inter-arrival time and total volume, and then uses K-Means and DBSCAN clustering. This method achieves 90% accuracy, and can identify protocols precisely. It is partially similar to our technique because we also evaluate K-Means clustering. However, this method requires metrics computed on the complete connection.

All approaches discussed so far [65], [47], [84], [50] and [24] rely on different sets of metrics and different methods to calibrate their model. However, they all use metrics computed about complete connections (duration or total volume, for instance). Therefore, these methods cannot classify connections early and cannot be applied online, which is the main goal of this thesis.

### **Identification of applications on a host**

BLINC [43] introduces a very different 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, their goal is not to classify a single TCP connection. Our method could be used together with BLINC to analyze all connections from a host with a connectivity pattern suggesting forbidden applications (such as peer-to-peer).

## 2.3 Summary

In this chapter, we described tools to collect traffic and to classify connections into applications.

We showed that port-based classification is no longer accurate and that content-based methods are computationally intensive, and not usable on fast networks. Behavior-based methods avoid these limitations and are promising. However, the current techniques need information about complete TCP connections, and, therefore, cannot classify connections early.

We propose to design a classifier that uses behavior-based techniques to identify applications early in TCP connections. This classifier will run on edge routers and therefore needs to involve little computation and to have small memory requirements. To design this classifier, we capture a training data set using a DAG card on the link that connects our university to the Internet, and use unsupervised learning to cluster the connections in this data set. To analyze the composition of the obtained clusters, we label all the connections in the training set using a content-based classifier.

The following chapters present the design of our classifier. Chapter 3 describes the traces we use for this study, and how to choose a feature that helps to distinguish between applications based on the first packets in a connection. Then, Chapter 4 explains how to design a model relying on this feature, and Chapter 5 presents a classification method designed to run on high-speed links.



# Trace analysis for feature selection

## Contents

---

<b>3.1</b>	<b>Description of data sets . . . . .</b>	<b>36</b>
3.1.1	Packet traces . . . . .	36
3.1.2	Parsing packet traces . . . . .	37
3.1.3	Filtering Packet Traces . . . . .	38
3.1.4	Per-Application Traces . . . . .	38
<b>3.2</b>	<b>Feature selection . . . . .</b>	<b>39</b>
3.2.1	Comparison of connection metrics . . . . .	40
3.2.2	Feature stability across networks . . . . .	43
<b>3.3</b>	<b>Summary . . . . .</b>	<b>43</b>

---

Although some of the classification metrics proposed in previous work presented in Chapter 2 need information of the complete flow (such as duration and number of packets), nothing prevents us from applying metrics like mean packet size or mean inter-arrival time to a sub-set 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 this chapter, we first present the packet traces we use in this thesis, and how we parse them to extract connection-level information. Then, we analyze the different metrics we can compute after the first few packets in a TCP connection.

## 3.1 Description of data sets

We use different packet traces to evaluate our assumptions and methodology. This section describes these traces and our procedure to extract the sizes of the first packets in TCP connections and the associated application labels.

### 3.1.1 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 [23]. We capture data traversing the Gigabit Ethernet Link that connects the university to the Internet. The fourth trace was captured at the edge of 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 points of Dartmouth university, and is referred to as ResBldg13 in the repository (this trace is described in detail in [32]). Finally, we use a trace captured at the edge of the network of the University of Massachusetts Amherst campus (described in [76]).

### 3.1.2 Parsing packet traces

To parse the payload traces and gather metrics about all TCP connections, we adapted the *crl\_flow* tool from the CoralReef suite developed by CAIDA [15]. *Crl\_flow* parses packets and gives statistics for each connection (number of packets, total volume, time-stamps for the first and last packet in the connection). In addition to these statistics, we store the payload sizes and the time-stamps of each of the first ten application packets. We remove all TCP control packet (SYN, Keep-Alive, or Ack with no data), because they do not contain application data.

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 content-based tools to identify application. To label connections in these traces we rely on a commercial classification tool called Traffic Designer [62]. Traffic Designer can identify more than 300 different applications using pattern matching. To find the accurate application label for all connections in payload traces, we plugged the classification engine from Traffic Designer into our parsing tool. Even though Traffic Designer works for our offline analysis of payload traces, it is not always 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. This method only works for traffic using standard ports and, therefore, cannot classify all traffic. Besides, this method may be misleading, because malicious applications may use these standard ports to avoid detection (masquerade traffic). Our study of payload traces (for which we can compare labels obtained using content analysis and using ports) shows that masquerade connections represent a very small proportion of the traffic. So, we believe that it is safe to use port classification for connections using standard ports.

After parsing all traces, we stored information related to each trace in a MySQL [51] table, where each entry corresponds to a connection. We describe each connection with the following fields:

- 5-tuple, to identify the connection (source and destination IP addresses, source and destination port numbers and transport protocol);
- Initial flags, to verify if we have the first packets in the connection (i.e., if the connection did not start before the capture);
- Payload sizes of the first 10 packets; and
- Time-stamps of the first 10 packets.

### 3.1.3 Filtering Packet Traces

We process our SQL tables to keep only the connections that are relevant to our analysis. We remove all non-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).

Table 3.1 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 presents 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, because most connections with less than four packets are bogus (most of them consist of port scans or connections to closed ports).

Grouping all payload traces we have “enough” sample TCP connections (as described in Chapter 4) 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. We hesitated to take HTTP into consideration, because many applications use port 80 to avoid detection. However, our study of payload traces showed that the proportion of masquerade traffic on port 80 is very small (on Paris 6 traces, legitimate HTTP traffic represents more than 99% of the traffic on port 80).

### 3.1.4 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

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%
Crowdad	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 3.1: Description of packet traces for connections that started during the capture, and proportion of connection and traffic with more than four packets

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 disabled most services on the host with the bittorrent client. Then, we configured a software firewall to filter out the traffic associated with the remaining services on this host. 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 manually generated traces in Table 3.1.

## 3.2 Feature selection

We use the data sets described in the previous section to compare connection-related features to find the best metric to distinguish applications.

### 3.2.1 Comparison of connection metrics

We can compute a large number of metrics describing a TCP connection. In [83], the authors present 250 different connection metrics (including all the metrics used in other studies). We can regroup these metrics in three main categories.

- Metrics related to packets, such as number of packets, average packet size, variance of packet size, total volume transferred, distribution of packet sizes;
- Metrics related to timing information, such as duration of the connection, mean inter-arrival time, variance of the inter-arrival time;
- Metrics related to TCP, such as initial sequence number, window size, header size.

In our study, we do not consider metrics related to TCP, because they depend mostly on the TCP implementation and little on the applications. Therefore, we focus on metrics related to packet sizes and timing information only. In addition, we can only consider metrics which we can compute using only the first packets in a connection. To decide on the best feature to distinguish applications, we evaluate the classification power of these metrics when restricted to the first four packets of a TCP connection<sup>1</sup>. 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 3.1. Figure 3.1 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 the range between the 5-percentile and 95-percentile (as a 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 addition, if we analyze in detail the distributions of mean IAT for our applications, as presented in Figure 3.2, we can observe that these distributions are really similar. Therefore, we cannot rely on mean IAT to distinguish applications. We performed the same study for mean jitter and reached a similar conclusion. In contrast, Figure 3.1 shows that features related to

<sup>1</sup>We select four packets because of the results from the following Chapters. However, the results presented in this section are similar for other numbers of packets.

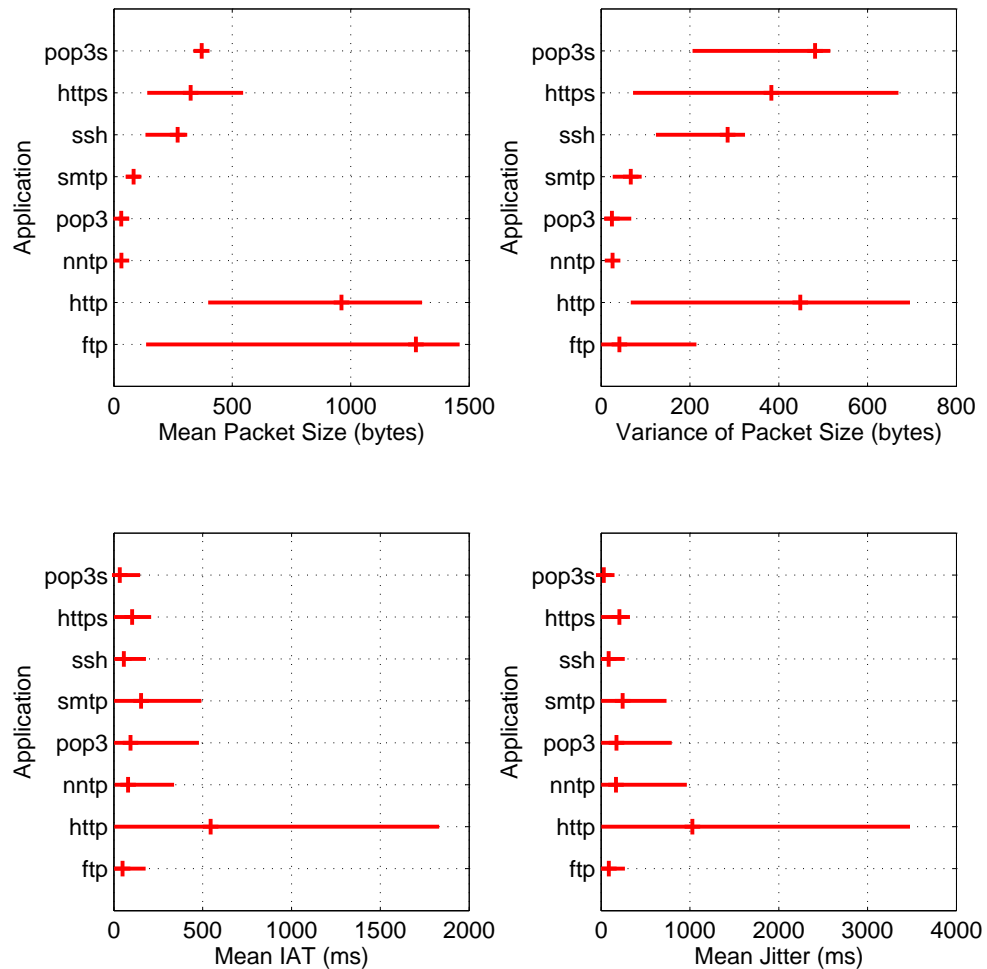


Figure 3.1: Comparison of classification metrics for the most common applications for Paris6-1 trace

packet sizes do distinguish among applications. For instance, the mean payload size of SMTP connections varies from 30 to 150 bytes, which is different from all other applications. This figure also shows that the mean size of the first four packets is similar for POP3 and NNTP connections, but very different between POP3 and other applications. We can also observe that the mean size does help distinguishing between POP3S and FTP, but that the variance is significantly different for both protocols.

Instead of computing the mean and the variance of packet sizes, we study the size and direction of each of the first four packets (the size is positive for packets sent by the client of the connection and negative for packets sent by the server). Figure 3.3

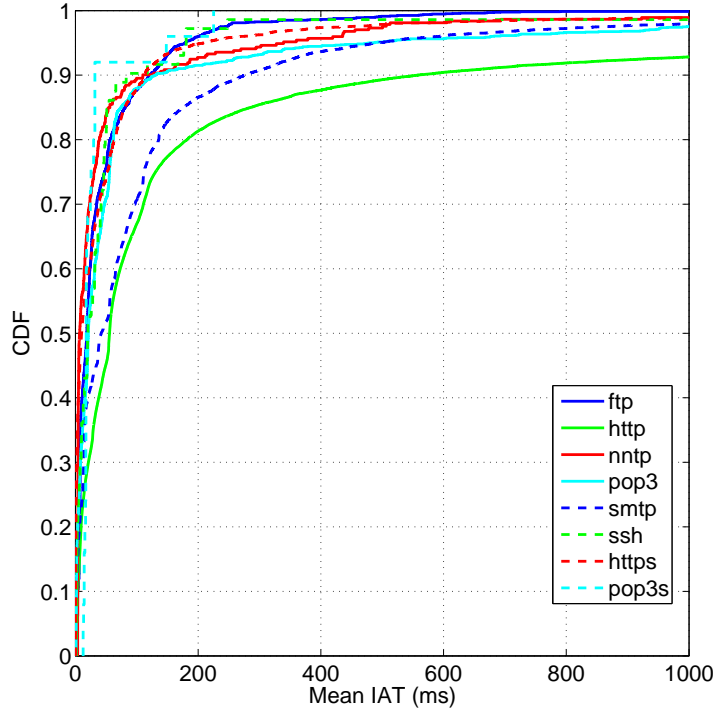


Figure 3.2: Cumulative distribution functions for mean IAT

presents the mean and standard deviation of each packet size for five traces<sup>2</sup>. 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.

Therefore, 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.

<sup>2</sup>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 3.1.2.

### 3.2.2 Feature stability across networks

To verify if we can apply a model obtained on a network to traces from different places, we compare the sizes of the first packets in TCP connections for eight applications on five different networks. Figure 3.3 presents the results of this comparison. It shows that the sizes of the first four packets for one application are similar across all data sets. 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.

## 3.3 Summary

This chapter presents the traces we use to develop our classification methodology. We describe the approach we use to extract connections from these traces and get the ground truth about the application associated to each connection. In addition, we find that the payloads size of the first packets is a good metric to distinguish applications. This result is promising because we can therefore design a classifier relying on packet sizes, which are easy to retrieve online. Besides, we can train such a classifier on one network and use it in another, because the sizes of the first packets are similar across networks for a given application. In the following chapters, we detail the methods we use to design this classifier.

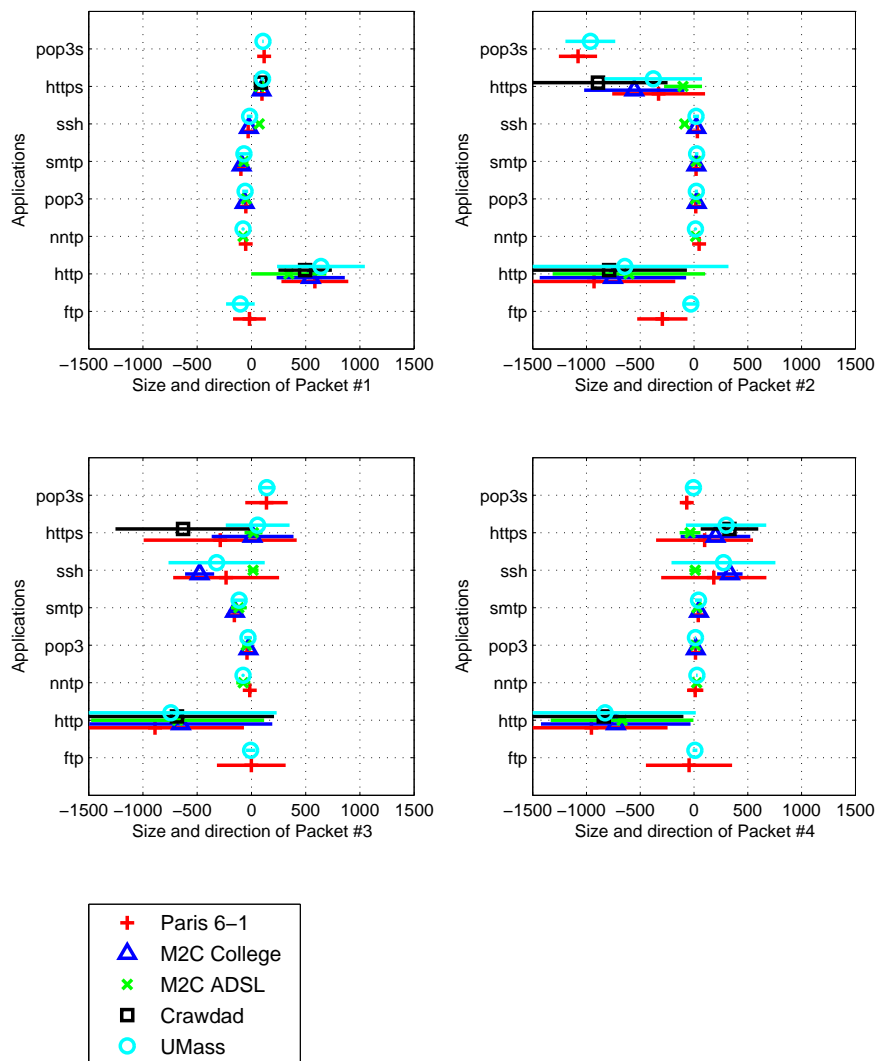


Figure 3.3: Comparison of the sizes of the first 4 packets for different applications on several networks

# Offline training

## Contents

---

<b>4.1</b>	<b>Training Traces</b>	<b>46</b>
<b>4.2</b>	<b>Representation of a TCP connection</b>	<b>48</b>
4.2.1	Euclidean Representation	48
4.2.2	Representation using Hidden Markov Models	49
<b>4.3</b>	<b>Clustering Algorithms</b>	<b>50</b>
4.3.1	Choice of the algorithm	50
4.3.2	K-Means Clustering in the Euclidean Space	52
4.3.3	GMM Clustering in the Euclidean Space	55
4.3.4	Spectral Clustering in the HMM space	56
<b>4.4</b>	<b>Filtering</b>	<b>58</b>
<b>4.5</b>	<b>Calibration</b>	<b>59</b>
4.5.1	Clustering quality metric	59
4.5.2	Number of clusters	61
4.5.3	Number of packets	63
<b>4.6</b>	<b>Summary</b>	<b>64</b>

---

This chapter discusses our methodology to obtain models of application protocols from sample TCP/IP packet traces. We describe how to construct “good” training traces, and how to convert TCP connections to a spatial representation. Then, we explain how to regroup connections according to the sizes of the first packets using clustering algorithms and how to calibrate these algorithms. Table 4.1 presents a summary of the notations used in this Chapter.

$\mathcal{U}$	Connections in the training set
$P$	Number of packets considered
$\epsilon(x)$	Euclidean representation of $x$
$\rho(x)$	Sequence of symbols associated to $x$ (HMM)
$\eta(\rho(x))$	HMM associated to $x$
$d_e(\epsilon(x), \epsilon(x'))$	Euclidean distance between $x$ and $x'$
$d_h(x, x')$	Distance between $x$ and $x'$ using the HMM model
$K_k, K_g, K_h$	Number of clusters for K-Means, GMM, HMM

Table 4.1: Notations for the training phase

## 4.1 Training Traces

The training traces are the input for the clustering algorithm. A poor choice of training data may lead to models that do not capture all applications or all modes of operation of an application. To avoid this problem, we follow a number of rules: select samples of all target applications; select a representative number of connections for each application; and select a similar number of connections for all applications. If the number of connections is not balanced, the most prevalent applications would bias the clustering. To follow these rules, we need the ground truth for the application associated to each connection in our training set.

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 one advantage: we know exactly the application in each connection. However, manually-generated traces can take long to create. In addition, it is challenging and time consuming to capture all modes of operation for all target applications. On the other hand, packet traces from the studied network contain a representative sample of the applications in use on the network, but we need access to TCP payload to accurately identify the applications.

Considering these requirements and possibilities, we use connections from the three packet traces from the Paris 6 network. Even though our traces have more than 50 appli-

cations total, three applications account for 90% of the connections: HTTP, SMTP and HTTPS. Some applications have very few connections, which would not be “enough” to create a representative model. Table 4.2 shows the number of connections for all applications in the Paris6-1 trace. “Other” represents the total number of connections associated with applications having less than 100 connections. Based on this table, we choose to select the ten applications with more than 300 connections, which is the best trade-off between the number of applications in our training set and the number of connections per application.

Application	Number of connections
HTTP	249 589
SMTP	23 094
HTTPS	14 718
FTP	13 326
POP3	8 803
NNTP	3 238
edonkey	630
SSH	455
POP3S	396
Kazaa	332
MSN	184
IMAP	180
LDAP	151
IRC	150
Other	466

Table 4.2: Number of connections with at least four packets for each application in Paris6-1 trace

To ensure that our results are not biased by the particular samples in the training set, we use 50 different training sets and present mean values with standard deviations in the rest of this thesis. For each training set, we randomly select the same number of connections of the ten target applications. We select the best number of connections in the training sets experimentally. We use the overall accuracy metric presented in Section 5.3 to compare models obtained with training sets consisting of 10 to 300 connections from each application. Figure 4.1 shows the mean accuracy obtained for each number of connections over 50 runs (based on different training sets), with the standard deviation represented by the vertical bars. The accuracy increases when the number of connections varies from 10 to 100 but remains stable afterwards. Therefore, 100 connections of each application are enough to model our training set. In the rest of this thesis, we use training traces with 1,000 connections (100 connections for each

of the ten applications), and use  $\mathcal{U}$  to refer to the set of training connections. Since we focus on applications with at least 300 connections, we have at least  $\binom{300}{100}$  possible training sets of each application. In addition, for each training set, we have at least 200 connections of each application to validate our model.

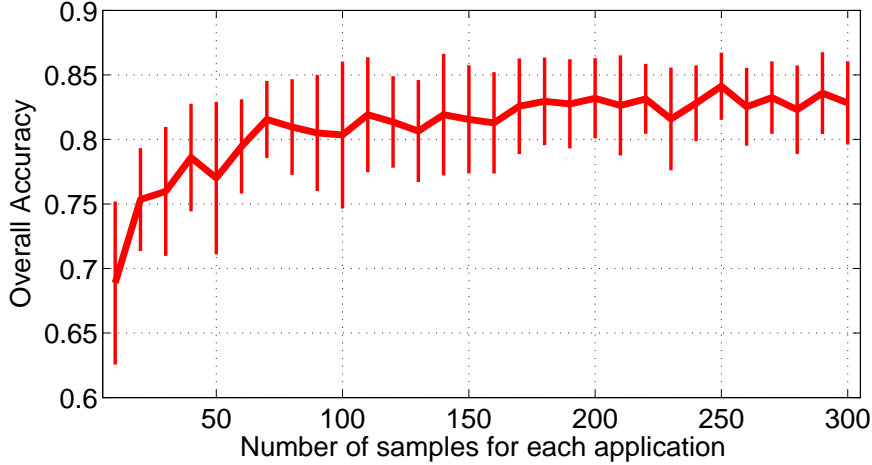


Figure 4.1: Overall Accuracy on the full training set depending on the number of samples (No threshold and using the Dominant heuristic)

## 4.2 Representation of a TCP connection

Chapter 3 shows that the size of the first packets of a connection is a good representation of application behaviors. To model applications, we regroup connections in the training data set based on the sizes of the first few packets. To perform this clustering, we need a representation of each connection and a measure of similarity between the representations of two connections. We only use a very small amount of information on connections: the sizes and directions of the first packets. Therefore, we have a limited choice to represent connections. This section describes two different connection representations based on the sizes of the first packets. First, we simply represent connections in a Euclidean space. Then, we take into account sequence information using Hidden Markov Models.

### 4.2.1 Euclidean Representation

Let  $x$  be a connection in the training set ( $x \in \mathcal{U}$ ). 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 coordinate associated to packet  $i$  in the vector representing  $x$ . The absolute value of  $s_i(x)$  is the payload size of packet  $i$  and, to take into account the direction of the packet,  $s_i$  is positive for packets sent by the TCP client and negative for packets sent by the TCP server. Let  $\epsilon$  be the function that transforms a connection into this representation:  $\epsilon : \mathcal{U} \rightarrow \mathbb{Z}^P$  and  $\epsilon(x) = (s_1(x), \dots, s_P(x))$ . To measure the distance between two connections  $x$  and  $x'$  we use the classical  $l^2$ -norm:  $d_e(\epsilon(x), \epsilon(x')) = \|\epsilon(x) - \epsilon(x')\|$ .

### 4.2.2 Representation using Hidden Markov Models

The Euclidean representation is simple but does not take into account the order of the packets: a permutation of the  $P$  dimensions would have no influence on the clustering results. This limitation led us to evaluate a more complex representation based on Hidden Markov Models (HMM) [69]. Hidden Markov models assume that the outputs of a system are generated by a Markov process with hidden parameters. Modeling a system with an HMM consists in finding these hidden parameters based on the observed outputs. In our case, the outputs are the sizes and direction of the first  $P$  packets in the connection.

To represent connections using HMM, we use the method proposed by Jeannin in [5]. Since we evaluate a clustering based on this HMM representation in Chapters 4 and 5, we summarize it in the following paragraphs for completeness (for more details, refer to [5]). To simplify the computation of the HMM parameters, this method does not use packet sizes directly<sup>1</sup>. Instead, it uses a set  $\mathcal{L}$  of 8 symbols representing four different ranges of packet size ( $[0, 150]$ ,  $[150, 700]$ ,  $[700, 1300]$ ,  $[1300, 1500]$ ) in each direction (symbol 1 represents packets sent by the TCP client with 0 to 150 bytes of payload, ..., symbol 8 packets sent by the TCP server with 1300 to 1500 bytes). We define  $\rho$  the function that transforms the size and direction of the first  $P$  packets in connection  $x$  into a sequence of  $P$  symbols  $\rho(x)$ .

This method uses the HMM structure proposed in [10, 60], where each state corresponds to an output symbol in the sequence. This method estimates the HMM parameters as follows:

- Initial probabilities ( $\Pi$ ).  $\Pi(i)$  is the probability to start at state  $i$ . With the HMM structure from [10], the Markov chain has  $P$  states and only goes from left to right. Therefore,  $\Pi(1) = 1$  and  $\Pi(j) = 0$  for  $j \in 2..P$ .

<sup>1</sup>With this simplified model, HMM training is already twenty times slower than GMM training.

- Transition matrix ( $A$ ):  $A(i, j)$  is the probability that state  $j$  will follow state  $i$ . Since this Markov chain only goes from left to right,  $A(i, i+1) = 1$  and  $A(i, j) = 0$  for  $j \neq i+1$ .
- Emission probabilities ( $B$ ),  $B(i, s_j)$  is the probability to output symbol  $s_j$  when in state  $i$ . First, the method empirically estimates the probability that symbol  $\sigma_1$  follows symbol  $\sigma_2$ ,  $\mathbb{P}(\sigma_1, \sigma_2)$ , for the eight symbols using the overall sequence set,  $\rho(\mathcal{U})$ . Then, it defines a similarity measure  $c(\sigma_1, \sigma_2)$  between  $\sigma_1$  and  $\sigma_2$  as the correlation between the probability laws  $\mathbb{P}(\sigma_1, \cdot)$  and  $\mathbb{P}(\sigma_2, \cdot)$ . Finally, it defines the emission probability law of state  $i$  as:

$$B(i, \sigma_j) = \frac{c(\sigma_{i-1}, \sigma_j)}{\sum_{\sigma_k \in \mathcal{L}} c(\sigma_{i-1}, \sigma_k)}, \sigma_j \in \mathcal{L}.$$

The generic method for evaluating the similarity between HMM-based sequences is given in [69]. Let  $\eta$  the transformation that associates a sequence  $\rho(x)$  with an HMM  $\eta(\rho(x))$  that represents connection  $x$ . To derive a distance matrix from these representations, this method first computes 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.  $\mathbb{L}(\rho(x), \rho(x')) = -\log \mathbb{P}(\rho(x') | \eta(\rho(x)))$ . To obtain these log-likelihoods, this method relies on the forward-backward algorithm [4]. Based on this matrix, it then defines the distance matrix  $D_h$ :

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

### 4.3 Clustering Algorithms

Depending on the representation, we need to use different clustering algorithms. After explaining how we choose our clustering algorithms, we present the algorithms we use for the Euclidean space and for the HMM-based space.

#### 4.3.1 Choice of the algorithm

A large number of clustering algorithms are available in the literature. We can divide them in four main categories:

**Partition-based algorithms.** These algorithms divide the studied space in distinct regions. The Nearest-Neighbor (NN) algorithm [16] is a simple algorithm in this

category. This method is used to design simple classifiers: it associates a new vector with the label from the closest vector in the training set. The k-Nearest-Neighbor (k-NN) algorithm [29] extends this method by taking into account the k closest vectors to decide on the label. The main limitation of both methods is that they do not result in simple models for the clusters: to label a new vector, we need to evaluate the distance between this vector and all vectors in the training set. This operation is time-consuming for large training sets. Another approach is Linear Discriminant Analysis [28] (LDA). This method finds linear boundaries between previously identified groups. The main drawback of this approach is the shape of the obtained regions, which results in complex computations to associate new vectors with these region. Finally, a well known partition-based algorithm is K-Means [48]. This method finds spherical clusters in the training set. These clusters can then be simply modeled using only their center.

**Hierarchical algorithms.** These algorithms [41] can use an agglomerative or divisive technique. Agglomerative techniques builds a hierarchy of clusters by regrouping clusters at each step. At the beginning, clusters consist of a single element and at the end a single cluster regroup all elements. These techniques regroup clusters based on the distance between all clusters from the previous step. The divisive alternative results in a similar hierarchy but instead of regrouping clusters, it divides them at each step. The main limitation of these techniques is the difficulty to associate a new vector with one of the clusters, because the agglomeration (or division) results in clusters with very complex shapes that cannot be modeled easily.

**Model-based algorithms.** The idea of this technique is that the vectors in the training set are independent samples from a series of heterogeneous groups. A common model for this method is a Gaussian mixture (GMM), where each Gaussian in the mixture is associated with a cluster. The main advantage of this method is that it results in a simple classifier, because we can compute the probability that a new vector belongs to each cluster and choose the cluster with highest probability.

**Similarity-based algorithms.** These methods perform the clustering when we cannot represent the training sample in a regular space and can only evaluate the similarity between samples. A very efficient algorithm in this situation is spectral clustering [55]. This algorithm relies on a similarity matrix, which contains the similarity between all samples in the training set, and regroup them using an eigen-decomposition. The main advantage of this algorithm is to be efficient when there is no real spatial

representation but only similarities. In addition, spectral clustering is very fast.

We want to classify connections very fast to handle high-speed links. This excludes the NN, k-NN, LDA and hierarchical algorithms because they result in complex models to associate new connections with the clusters. We choose to study the K-Means algorithm because it is the only partition-based algorithm that fits our needs. In addition, we also evaluate the GMM algorithm because it is a model-based method that fulfills our requirements in terms of complexity. Finally, we use spectral clustering to cluster connections represented using HMMs. To decide on the best algorithm among these three, we evaluate their efficiency on our data sets.

### 4.3.2 K-Means Clustering in the Euclidean Space

The Euclidean representation of connections leads to a  $P$ -dimensional space where  $P \leq 10$ . A common type of clustering algorithms is partition-based clustering techniques. A well-known example of this type of clustering is *K-Means*. K-Means finds the  $K$  clusters that minimize the total intra-cluster variance (i.e., minimize the sum of the distances between each point in the data set and the centroid of its closest cluster).

We illustrate the behavior of K-Means using the example in Figure 4.2. It presents three groups of points in a two-dimensional space. K-Means is an iterative algorithm that proceeds in two steps. First, it randomly chooses  $K$  cluster centroids and assigns each connection to the closest cluster. Then, it computes the centroid of each cluster based on the points assigned to it. It repeats these steps until it finds a minimum for the total intra-cluster distances. If we apply the K-Means algorithm to the data set in Figure 4.2 with  $K = 3$  we should find the decomposition shown in Figure 4.3.

The K-Means algorithm is simple, efficient, and fast. However, it has some limitations. First, depending on the initial choice of centroid, it can converge to a sub-optimal solution. A common solution to this problem is to run the algorithm several times with different initial centroids and to choose the best solution. The second drawback of the K-Means method is that it only uses spherical clusters, as shown on Figure 4.3. When the shape of clusters is not spherical it leads to a poor representation of the data set. It can lead to an unnecessarily large number of clusters (an ellipsoidal cluster may be represented as multiple spheres) or to clusters that are too “large” (an ellipsoidal cluster represented by a large sphere). Besides, clusters obtained with K-Means do not take into account the dispersion of points in the clusters. The only information given by K-Means is the centroids of the  $K$  clusters. This limitation is a major drawback when trying to assign new data points to existing clusters. For instance, consider the

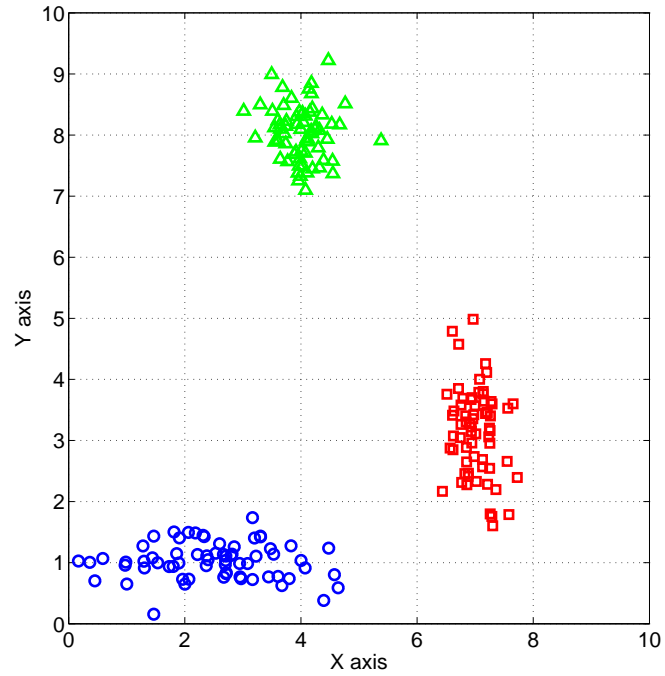


Figure 4.2: Three groups of points in a two-dimensional space

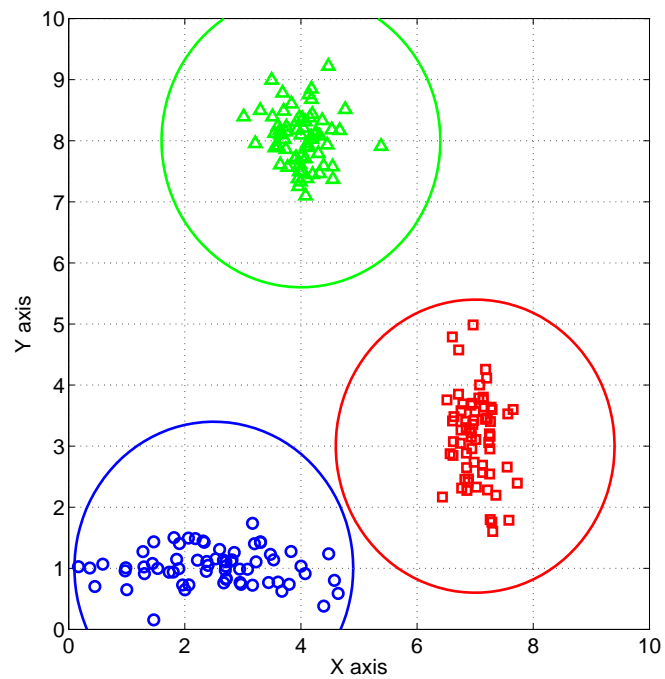


Figure 4.3: Clusters found with K-Means

example in Figure 4.4 and say that we want to map the diamond-shaped point with a cluster. This point is closer to the centroid of the cluster on the right (consisting of squares) and would therefore be mapped to this cluster, whereas it would be better to associate it to the cluster on the left (consisting of circles), if we consider the dispersion of points in both clusters. Finally, associating a new point to the closest cluster does

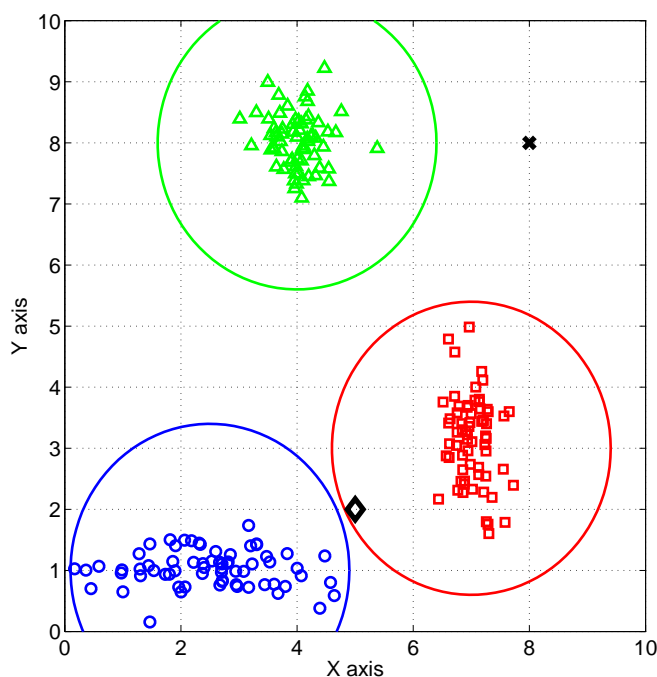


Figure 4.4: Clusters found with K-Means

not allow the identification of points that do not belong to any cluster. For instance, on Figure 4.4, the “x” point is closest to the cluster consisting of triangles and would be associated with this cluster, whereas it is clearly far from all points in this cluster.

There are simple extensions to K-Means to solve some of these issues. For instance, instead of simply modeling each cluster by its centroid, we can also use the variance of the distance of all points to the centroid. Under the hypothesis that the distances to the centroid are normally distributed, we can use the centroids and variances to compute the probability that a point belongs to each cluster. To associate a new point to a cluster we then choose the cluster with highest probability. This new model takes into account the dispersion of points in clusters and the diamond point would be associated to the cluster on the left. Besides, we can then use a threshold on the probability to belong to a cluster to limit the size of the clusters and detect that a point should

not be associated to any cluster. With such a threshold the “x” point would not be mapped to any cluster. These improvements of the K-Means algorithm give a clustering method that relies on more than simple space partitioning, and is close to probabilistic model-based methods.

### 4.3.3 GMM Clustering in the Euclidean Space

A second family of clustering algorithms uses probabilistic models. In this case, the aim is to represent the data set with a probabilistic model. Gaussian Mixture Model (*GMM*) is a well-known clustering method in this family of algorithms. GMM assumes that each point in the space has been generated according to a probability distribution function  $f$ . This distribution consists of a mixture of  $K$  multi-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., centroid),  $c_i$ , and covariance matrix (i.e., shape),  $\Sigma_i$ . The covariance matrix defines the shape of the cluster. If  $\Sigma_i$  is proportional to the identity matrix, then resulting clusters are spherical. If  $\Sigma_i$  is diagonal, then clusters are ellipsoidal with axis parallel to the coordinate axis. Finally, a generic  $\Sigma_i$  represents clusters with any ellipsoidal shapes. GMM finds the probability distribution function  $f$  (represented by  $c_i$  and  $\Sigma_i$ ) that maximizes the likelihood of generating the points in the data set. We use the Expectation Maximization algorithm [18] to perform this optimization.

Consider the data set shown in Figure 4.2. It consists of three different groups in a two-dimensional space. GMM clustering models this data set with a mixture of three Gaussian distributions represented in Figure 4.5. Each Gaussian distribution corresponds to a cluster. GMM directly evaluates the probability for a new point to belong to any cluster: given a point  $x$ , we simply need to compute  $\mathcal{N}(c_i, \Sigma_i)(x)$ . To associate a new point to a cluster we can use the maximum likelihood method (i.e. compute the probability to belong to all clusters and chose the cluster with highest probability). We can also use threshold on these probabilities to detect points that should not be associated to any cluster.

K-Means and GMM are well-known and efficient clustering algorithms. However, they have some limitations. First, they work fast on low-dimensional spaces, but take time to converge and often fail to produce meaningful clusters on high-dimensional spaces. Second, K-Means and GMM methods always lead to convex clusters, and therefore cannot detect clusters with more complex shapes.

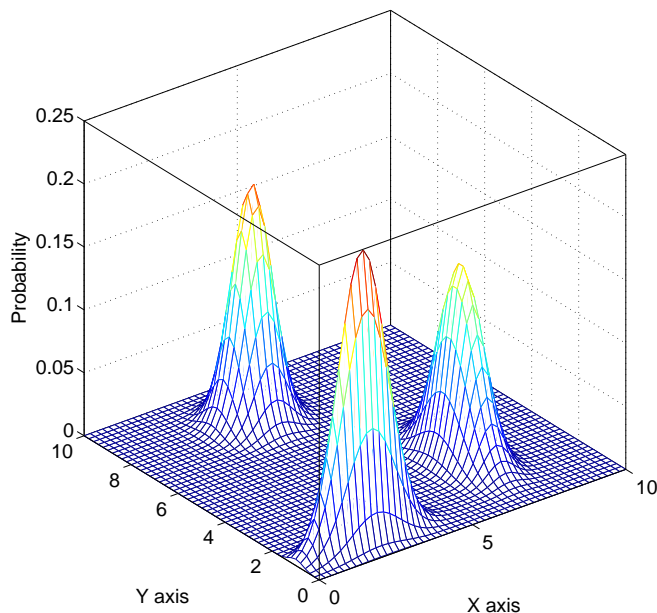


Figure 4.5: Probability density function obtained with GMM

#### 4.3.4 Spectral Clustering in the HMM space

Modeling the training set with HMMs leads to an  $N$ -dimensional space, with  $N$  the number of connections in our training set (1,000 in our case). The problem of clustering in high-dimensional spaces requires new algorithms such as *Spectral Clustering* ([55], [27]). Spectral clustering partitions the rows of a matrix in  $K$  clusters according to their components in the top  $K$  singular vectors of the matrix.

If we want to group the  $N$  points represented in Figure 4.6 in three classes, the first step is to compute a similarity matrix. A similarity matrix,  $S$ , is an  $N \times N$  matrix where  $S_{ij}$  measures the similarity between points  $i$  and  $j$ .  $\forall(i, j), 0 \leq S_{ij} \leq 1$ ; if  $i$  and  $j$  are not similar,  $S_{ij} = 0$ ; and if  $i$  and  $j$  are similar,  $S_{ij} = 1$ . There are different methods to compute  $S$  depending on the data set we want to classify. A simple method consists in evaluating the Euclidean distance between any pair of points  $(i, j)$ ,  $d_{ij}$  and comparing this distance to a threshold  $T$ : if  $d_{ij} \leq T$ ,  $S_{ij} = 1$ , else  $S_{ij} = 0$  (for the HMM representation, we cannot directly compute a Euclidean distance between HMMs, and we use the distance matrix,  $D_h$ , defined earlier). Table 4.3 presents such a similarity matrix for the points in figure 4.6. This measure of similarity is simple, but has some drawbacks because finding  $T$  is difficult, and  $T$  can vary from one cluster to another. A common solution is to measure an affinity  $a$  between the points in our

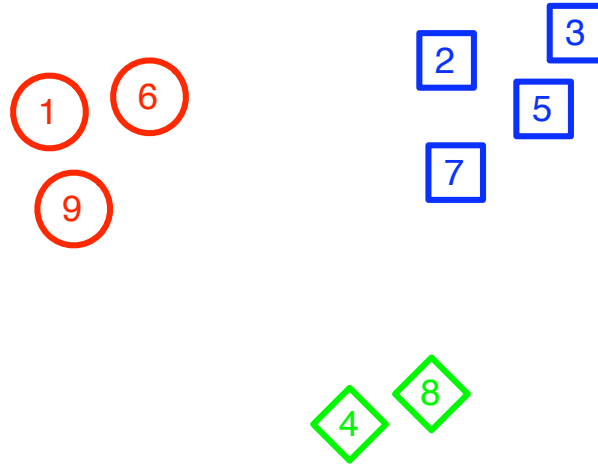


Figure 4.6: Points belonging to three distinct classes

data set using a Gaussian kernel as the mapping function between the affinity and distance, *i.e.*  $a_{ij} = \exp\left(-\frac{d_{ij}^2}{2\sigma^2}\right)$ , where  $\sigma$  is the radius of the Gaussian kernel. This non-linear transformation reinforces the information from the distance matrix: close points (according to sigma) will have an affinity close to 1, whereas distant points will have an affinity close to 0. To finalize the similarity matrix, we can apply additional transformations, such as the evaluation of the “conductivity” between points from our data set. The conductivity discovers if there is a “path” between two points  $i, j$  (i.e., if they are connected by paths of short “hops” over other points), and allows the detection of non-convex clusters.

	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	1	0	0	1
2	0	1	1	0	1	0	1	0	0
3	0	1	1	0	1	0	1	0	0
4	0	0	0	1	0	0	0	1	0
5	0	1	1	0	1	0	1	0	0
6	1	0	0	0	0	1	0	0	1
7	0	1	1	0	1	0	1	0	0
8	0	0	0	1	0	0	0	1	0
9	1	0	0	0	0	1	0	0	1

Table 4.3: Similarity matrix

When the similarity matrix is ready, it should be close to a block-diagonal matrix

(rows and columns may be mixed) as shown in Table 4.4, where each block corresponds to a cluster. Given this block-diagonal matrix, we can perform an eigen-decomposition to identify the clusters. If we want to regroup the data in  $K$  clusters, we perform the clustering the  $K$ -dimensional space spanned by the first  $K$  eigenvectors relative to the  $K$  largest eigenvalues. Then, we assign points belonging to the subspace spanned by the eigenvectors corresponding to block  $k$  to cluster  $k$ .

	1	6	9	2	3	5	7	4	8
1	<b>1</b>	<b>1</b>	<b>1</b>	0	0	0	0	0	0
6	<b>1</b>	<b>1</b>	<b>1</b>	0	0	0	0	0	0
9	<b>1</b>	<b>1</b>	<b>1</b>	0	0	0	0	0	0
2	0	0	0	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	0	0
3	0	0	0	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	0	0
5	0	0	0	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	0	0
7	0	0	0	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	0	0
4	0	0	0	0	0	0	0	<b>1</b>	<b>1</b>
8	0	0	0	0	0	0	0	<b>1</b>	<b>1</b>

Table 4.4: Clusters found in the similarity matrix

To regroup the HMMs representing the connection of our training set in  $K$  clusters, we compute a similarity matrix based on the distance matrix  $D_h$  using affinity and conductivity. Then, we perform an eigen-decomposition to identify the  $K$  clusters.

## 4.4 Filtering

We use clustering to detect common behaviors in our training set. To avoid “outliers” which may degrade the quality of our model, we use a simple heuristic: when there is only a single connection from a given application in a cluster, we consider this connection to be an outlier and remove it. The proportion of filtered connections increases with the number of clusters, because an increase in the number of clusters leads to smaller clusters. We have experienced with 10 to 100 clusters for each clustering method, and in none of these experiments the percentage of filtered connections was more than 4% of the total number of connections, as shown in Figure 4.7.

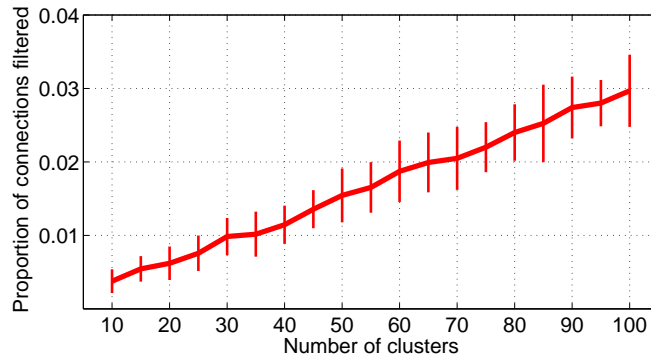


Figure 4.7: Proportion of connections filtered (four packets)

## 4.5 Calibration

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

### 4.5.1 Clustering quality metric

Clustering literature propose several metrics to evaluate the quality of a clustering. We can divide them in two main categories: internal criteria metrics and external criteria metrics.

**Internal criteria metrics** verify if the obtained clusters fit the data using information from the data set only. Well-known metrics using internal criteria are the Dunn index [22], the Davies-Bouldin index [17] and the silhouette index [66]. These three metrics measure if a given clustering lead to clusters that are compact and well-separated.

**External criteria metrics** measure if the results of a clustering is similar to another partition of the data set. To determine this similarity, the Rand index [64] and the Jaccard index [39] consider every pair of entries in the data set. For any pair, both entries can:

- belong to the same cluster and to the same group for the other partition;
- belong to the same cluster but to different groups;
- belong to different clusters but to the same group;

- belong to different clusters and to different groups.

The Rand index then measures the proportion of pairs that are in the same cluster and in the same partition or in different clusters and in different partitions. The Jaccard index differs a little: it measures the proportion of pairs that belong to the same cluster and to the same group, without considering pairs that are neither in the same cluster nor in the same group. The Rand index and the Jaccard index are bounded between 0 and 1 and values close to 1 correspond to a high similarity between the clustering and the partition.

Normalized Mutual Information (NMI) [75] is another metric using an external criteria. Instead of considering proportions, the metric relies on mutual information to evaluate the similarity between the clustering and the partition. NMI works as follows. 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:

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

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 as follows:

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

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 ids. Many applications have several behaviors (FTP, for instance, consists of a control connection and multiple data connections), and some applications may share a similar behavior. As a consequence, we cannot obtain  $NMI=1$  for our clustering, but the higher the  $NMI$  the better the clustering. Therefore, this metric is useful to compare the quality of clusterings obtained with different methods and parameter settings.

In our case, we do not perform a “blind” clustering. We want the clusters to separate applications: an optimal clustering is a clustering that has one cluster per application. Therefore, we choose to rely on an external criteria metric by comparing the results of our clustering with the partition defined by application labels obtained

as described in Section 3.1.2. We chose to use Rand index and NMI because Rand index and Jaccard index are very similar. We evaluated both metrics and found that they lead to comparable results, as illustrated in the following subsection in Figure 4.9. Therefore, we focus on NMI and give the results of the calibration for this metric.

#### 4.5.2 Number of clusters

We use NMI to choose the number of clusters. For the K-Means algorithm, we run K-Means on our training data using from 5 to 100 clusters and compute the NMI between the resulting clusters and the optimal clustering. Figure 4.8 presents the mean NMI over 50 different training sets for four packets. The vertical bars represent two standard deviations. This figure shows that the best number of clusters for four packets is 50. Increasing from 10 to 30 clusters leads to a sharp increase in NMI, whereas more than 50 clusters 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 has a direct impact on the speed of the classification engine (described in Chapter 5). 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 30 clusters for four packets:  $K_k(4) = 30$ .

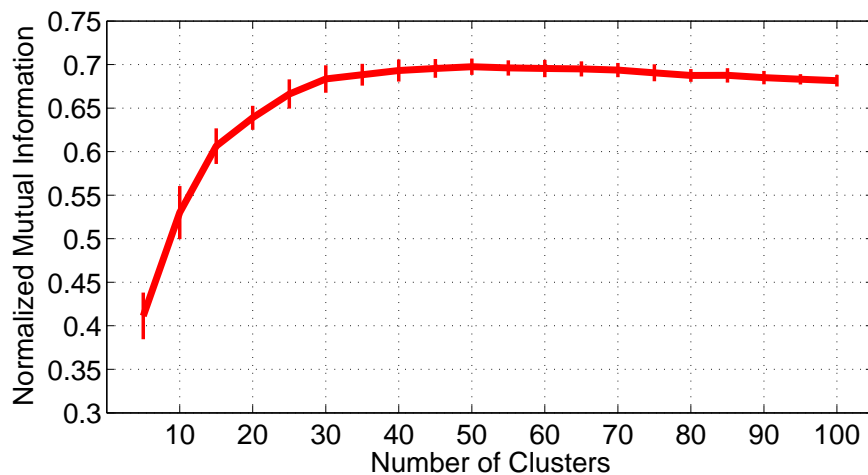


Figure 4.8: Influence of the number of clusters on clustering quality (4 packets, K-Means clustering)

Figure 4.9 illustrates the choice of the number of clusters for the GMM algorithm with two packets, considering both NMI and Rand index to measure the quality of the clustering. For both metrics, the optimal number of clusters is 25.

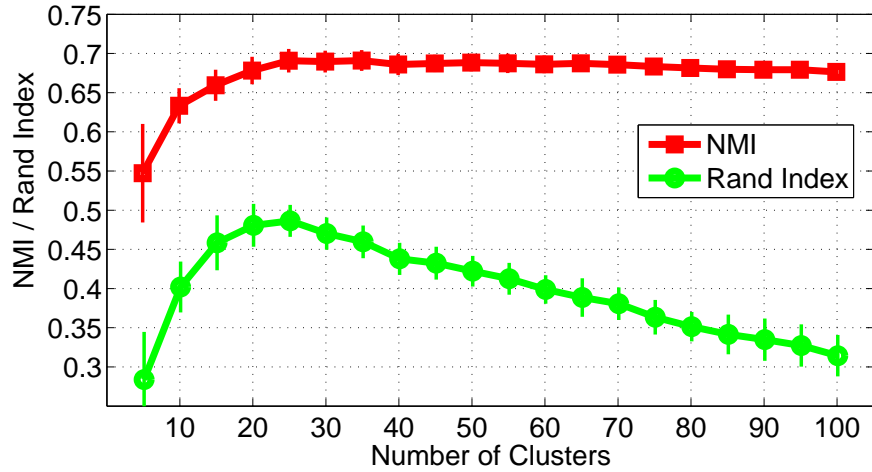


Figure 4.9: Comparison between NMI and Rand Index (2 packets, GMM clustering)

We apply the same methodology to determine the best number of clusters for GMM and HMM-based clusterings. Figure 4.10 shows the best number of clusters for different number of packets, with vertical bars representing two standard deviations (obtained with our 50 training sets). The optimal number of cluster increases with the number of packets because a larger number of packets leads to an increasing number of behaviors.

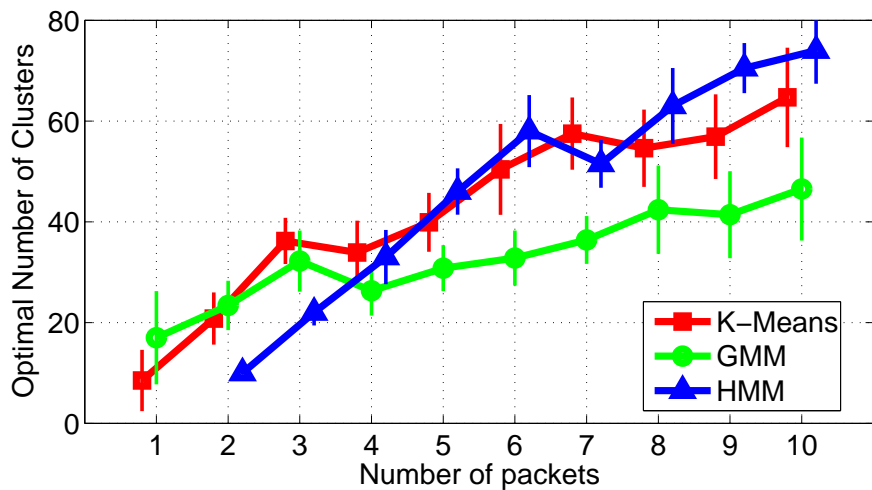


Figure 4.10: Optimal number of clusters according to the number of packets

### 4.5.3 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. Figure 4.11 shows the influence of the number of packets on the quality of the clustering evaluated by the NMI metric. As in the previous section, we present the mean NMI over all training sets with vertical bars representing two standard deviations. 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. We want to classify connections early, so a classifier that uses fewer packets is better as long as accuracy is not sacrificed. Based on this trade-off,  $P = 4$  packets is the best for the three clustering methods.

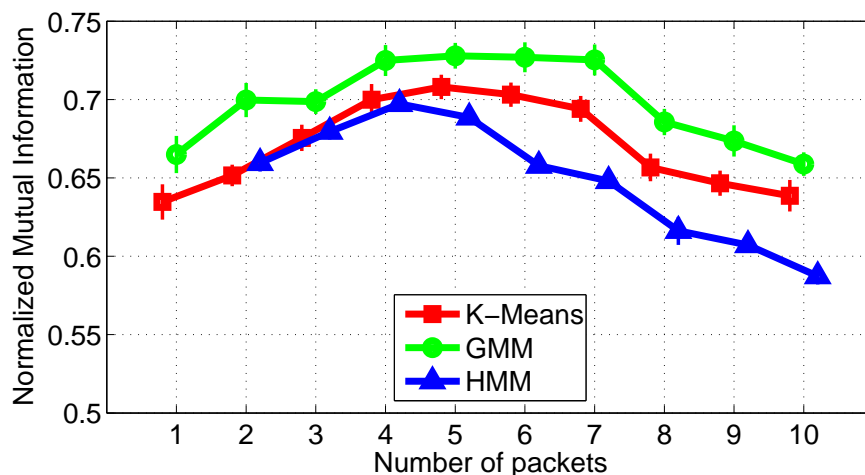


Figure 4.11: Influence of the number of packets on clustering quality

We evaluate the expressiveness of our clustering by studying the composition of resulting clusters. Figure 4.12 presents the proportion of clusters with one to four applications (no cluster contains more than four). The bars represent the mean over the 50 runs and the additional lines two standard deviations. 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.

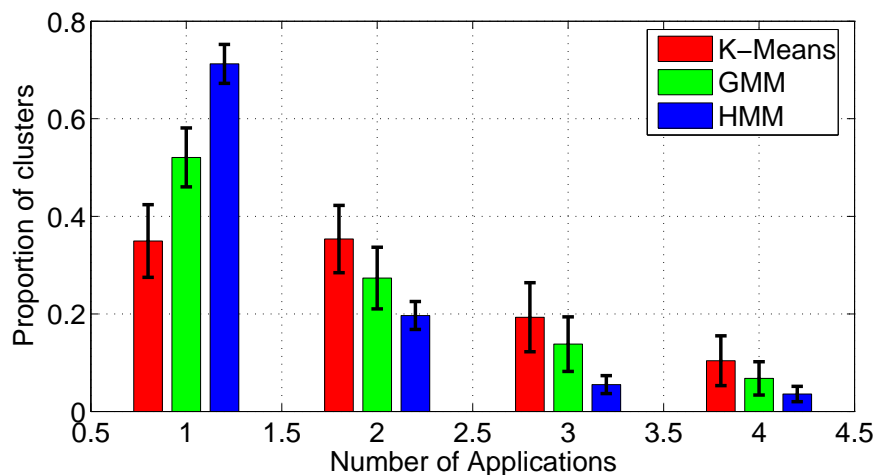


Figure 4.12: Proportion of clusters with N Applications (four packets)

## 4.6 Summary

This chapter presents our methods to obtain application models. Given a training data set with samples from 10 applications, we found that we can distinguish applications using only the first four packets in a connection. We described three different methods to obtain application models (namely, K-Means, GMM and HMM) and presented techniques to find the best number of clusters. The best method to obtain clusters with one application is HMM. Using the NMI metric, we observe that GMM outperforms K-Means. The models we obtained consist of the descriptions of all clusters (i.e., the connections assigned to each cluster) and their composition (i.e., the applications found in each cluster, obtained by labeling the connections assigned to it). The following chapter evaluates the adaptability of these models to design classifiers and implement an online classifier.

# Chapter 5

## Online Classifier

### Contents

---

<b>5.1</b>	<b>Assignment Heuristics</b>	<b>66</b>
5.1.1	K-Means	66
5.1.2	Gaussian Mixture Model	69
5.1.3	Hidden Markov Model	70
<b>5.2</b>	<b>Labeling Heuristics</b>	<b>71</b>
5.2.1	Predominant heuristic	72
5.2.2	Cluster+Port heuristic	72
<b>5.3</b>	<b>Heuristic evaluation</b>	<b>73</b>
5.3.1	Assignment Accuracy	73
5.3.2	Labeling Accuracy	75
5.3.3	Detection of new applications	79
5.3.4	Discussion	79
<b>5.4</b>	<b>Implementation</b>	<b>80</b>
5.4.1	Algorithm	81
5.4.2	Optimizations	81
5.4.3	Complexity of Online Classification	83
5.4.4	Evaluation	85
<b>5.5</b>	<b>Challenges</b>	<b>86</b>
<b>5.6</b>	<b>Summary</b>	<b>87</b>

---

Given the clusters obtained from each clustering method in the previous chapter, we now design and evaluate multiple classifiers. A classifier captures packets that traverse a link. Once it has the size of the first four packets in a connection, it assigns this connection to a cluster and labels it with one of the applications in the cluster. After presenting assignment and labeling heuristics in Section 5.1 and 5.2, we present a detailed evaluation of all possible classifiers in Section 5.3. Then, we describe and evaluate an implementation of the method that represents the best accuracy/complexity trade-off in Section 5.4 and discuss the main challenges faced by our classifier in Section 5.5.

Table 5.1 presents a summary of the notations used in this chapter.

$\alpha$	Assignment function
$\lambda$	Labeling function
$T_k, T_g, T_h$	Thresholds for K-Means, HMM, GMM
$\mathcal{A}(i)$	Set of applications present in cluster $i$
$\pi(i, a)$	% of connections associated to cluster $i$ from app. $a$
$\mathcal{S}$	Set of standard ports
$std(p)$	Application associated with standard port $p$
$\mathcal{T}$	Set of test connections
$\mathcal{T}_a$	Subset of test connections from app. $a$
$\gamma(x)$	Real application of connection $x$
$\beta(x)$	Indicates if $x$ has been assigned to a cluster containing its app.
$\psi(x)$	Indicates if $x$ has been labeled correctly
$\phi_a(x)$	Indicates if $x$ has been labeled with app. $a$
$\omega$	Overall Accuracy

Table 5.1: Notations

## 5.1 Assignment Heuristics

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

### 5.1.1 K-Means

A simple model to summarize clusters obtained with K-Means is 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,  $\alpha_{kc}$ , which associates a

connection  $x$  to the cluster with the closest centroid. Let  $c_i$  the centroid of cluster  $i$ :

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

Finding cluster centers and computing  $\alpha_{kc}$  involves little computation. However, this heuristic is too naive. It does not take into account cluster dispersion. Some clusters are very tight, whereas others are more disperse because the size of packets vary more (HTTP connections, for instance). We analyze the distribution of distances between connection representations and cluster centers in Figure 5.1. This figure shows that this distribution is close to Normal (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.

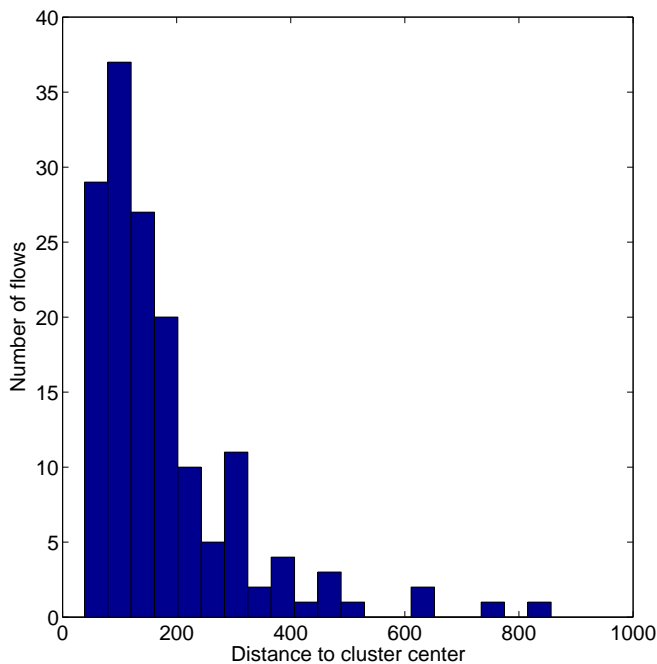


Figure 5.1: Distribution of distances to the center for the largest cluster (K-Means, 4 packets, 30 clusters)

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  $\sigma_i^2$ :

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

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 the assignment function  $\alpha_{kp}$ . We evaluate the logarithm of the probability to simplify the computation.

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

This heuristic will always associate new connections to a cluster even if the probability to belong to any cluster is very small. Therefore, to detect *unknown* connections (i.e., corresponding to behaviors not in the training set), we introduce a threshold on these probabilities. Since we are using a Normal distribution of the distances to cluster centers, 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  $i$ .

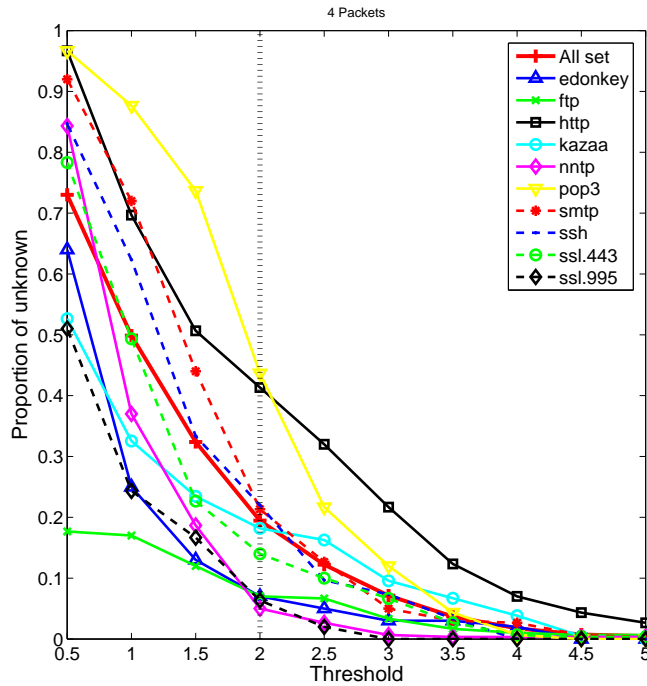


Figure 5.2: Choice of Threshold (K-Means)

To determine the best threshold, we apply our assignment heuristic to one of the training data set with different values of  $T_k$ . Figure 5.2 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 behaviors. We choose the smallest threshold that allows a good detection of known traffic. This corresponds to the knee of the “all set” curve, i.e.  $T_k = 2$ .

We define the assignment function  $\alpha_{kt}$  (called the **K-Means Thresh** heuristic) to assign a connection to a cluster with the probabilistic model using a threshold:

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

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.

### 5.1.2 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  $\Sigma_i$ ):

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

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 (we evaluate the logarithm of the probability to simplify the computation):

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

We use a probability threshold  $T_g$  to detect unknown traffic using this model, and define the **GMM Thresh** heuristic, which relies on the following assignment function:

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

We apply a similar method as the one presented in Section 5.1.1 to choose the

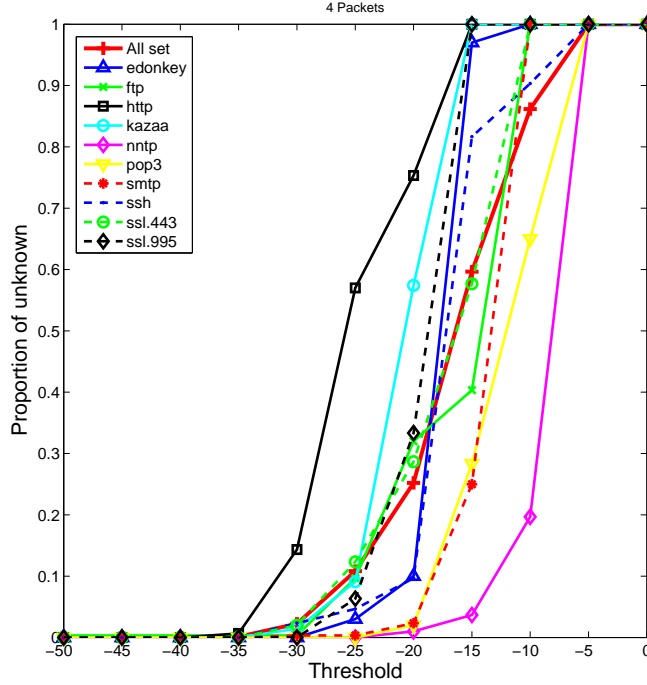


Figure 5.3: Choice of Threshold (GMM)

threshold  $T_g$ . We study the proportion of unknowns among training connections based on different values of  $T_g$  and choose the value that corresponds to a knee. Figure 5.3 shows these proportions, and based on this graph we choose  $T_g = -25$ .

### 5.1.3 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.  $h_i$  has a structure similar to the HMMs presented in Chapter 4 (four states, corresponding to the first four packets in the connection; and transitions between states  $i$  and  $i + 1$  only). We obtain the parameters of  $h_i$  with the Expectation-Maximization algorithm, using all connections in cluster  $i$ . We can now compute the log-likelihood that sequence  $\eta(x)$ , has been generated by  $h_i$ :  $\mathbb{L}_h(h_i, \eta(x))$ .

We define the **HMM likelihood** heuristic to associate a connection to a cluster.

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

We apply the same method used with GMM to detect unknown traffic. We define a

threshold  $T_h$  and the **HMM Thresh** heuristic, which relies on the following assignment function:

Function  $\alpha_{ht}(x)$

If  $\nexists i \in 1..K_h$  such that  $\mathbb{L}_h(h_i, \eta(x)) > T_h$

$\alpha_{ht}(x) = \text{unknown}$

else  $\alpha_{ht}(x) = \text{argmax}_{i \in 1..K_h} \mathbb{L}_h(h_i, \eta(x))$

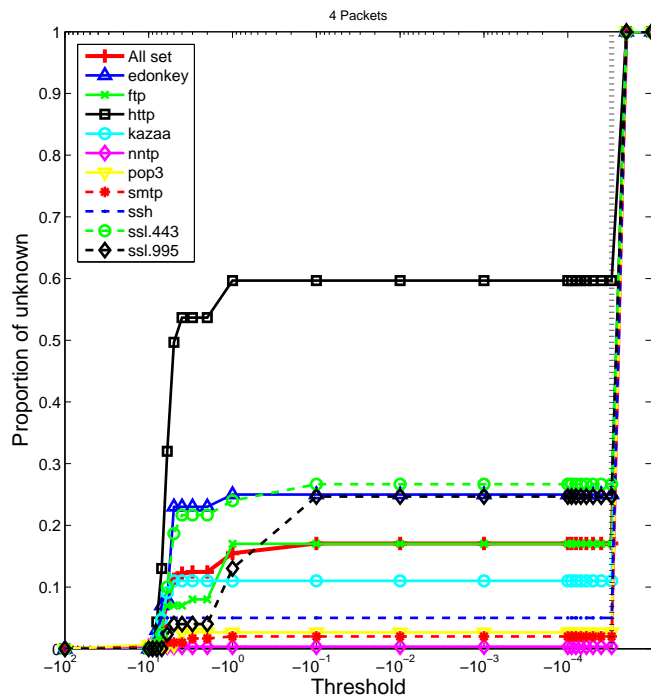


Figure 5.4: Choice of Threshold (HMM)

Figure 5.4 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  $T_h = -3e^{-5}$ .

## 5.2 Labeling Heuristics

When the cluster assignment heuristic finds the best cluster for a connection, the next step is to label it with an application. In this section, we describe two different labeling heuristics.

### 5.2.1 Predominant heuristic

The **Predominant** labeling heuristic is naive. This heuristic labels connections with the predominant application in their assigned cluster. Although Figure 4.12 shows that a cluster may consist of several applications, most clusters contain only one, and the predominant heuristic should accurately label connections assigned to these clusters. Let  $\mathcal{A}(i)$  be the set of applications present in cluster  $i$ ,  $\pi(i, a)$  the proportion of connections associated to cluster  $i$  from application  $a$ , and  $\alpha$  an assignment function. We define the predominant labeling function as:

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

This heuristic is simple but it will misclassify all connections from non-predominant applications when clusters have more than one application. We design another heuristic to handle this limitation.

### 5.2.2 Cluster+Port heuristic

We refine our labeling heuristic by exploring other information present in the first four packets. Although port numbers by themselves cannot classify all applications, many applications still rely on IANA port 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  $port(x)$  be 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  $std(\mathcal{S}) = \{\text{FTP, SSH, SMTP, HTTP, POP3, NNTP, HTTPS, POP3S}\}$ ). We define the Cluster+Port labeling function as:

```

Function  $\lambda_c(x)$ 
If  $\alpha(x) = \text{unknown}$ 
   $\lambda_c(x) = \text{unknown}$ 
Else If  $port(x) \in \mathcal{S}$ 
  If  $std(port(x)) \in \mathcal{A}(\alpha(x))$ 
     $\lambda_c(x) = std(port(x))$ 
  Else  $\lambda_c(x) = \text{masquerade}$ 
Else
  If  $|\mathcal{A}(\alpha(x)) \setminus \mathcal{A}(\mathcal{S})| = 0$ 

```

$$\lambda_c(x) = \textit{masquerade}$$

Else  $\lambda_c(x) = \operatorname{argmax}_a(\pi(\alpha(x), a), a \in \mathcal{A}(\alpha(x)) \setminus \mathcal{A}(\mathcal{S}))$

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 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 predominant 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.

### 5.3 Heuristic evaluation

In this section, we use the traces presented in Table 3.1 to evaluate all assignment and labeling heuristics. We evaluate models obtained with four packets and use the thresholds found in the previous section.

#### 5.3.1 Assignment Accuracy

Assignment accuracy measures the proportion of test connections that are assigned to clusters that contain the actual application of the connection. We define an accuracy metric as follows. 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  $\alpha$  (i.e., all connections not labeled unknown). Let  $\gamma(x)$  be the actual application of connection  $x$  (obtained as described in section 3.1.2). We define an indicator function  $\beta(x)$  that verifies if the application from connection  $x$  is in the set of applications of the cluster ( $\beta(x) = 1$ , if  $\gamma(x) \in \mathcal{A}(\alpha(x))$ ; and  $\beta(x) = 0$ , otherwise). The assignment accuracy of heuristic  $\alpha$  is:

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

Tables 5.2, 5.3 and 5.4 compare assignment accuracy for the different clustering methods and assignment heuristics for all our test traces. For clarity, all results presented in this section use models derived from one of our 50 training sets (results are similar for other training sets). Assignment accuracy is above 95% for all heuristics (for

Heuristic	Center	Proba	Thresh Tk=2
	Accuracy	Accuracy	Accur. (Unknown)
P6-1	95.6%	98.8%	99.7% (37.9%)
P6-2	95.8%	98.7%	99.7% (37.8%)
P6-3	95.8%	97.7%	98.8% (38.4%)
Enterprise	95.5%	97.9%	99.3% (31.8%)
M2C College	97.3%	99.7%	99.9% (41.3%)
M2C ADSL	68.9%	71.3%	65.5% (35.7%)
Crawdad	96.8%	99.7%	99.8% (41.8%)
UMass	94.6%	95.1%	98.6% (47.6%)

Table 5.2: Assignment accuracy and proportion of unknown traffic for K-Means

Heuristic	Proba	Thresh Tg=-25
	Accuracy	Accur. (Unknown)
P6-1	99.6%	99.7% (32.0%)
P6-2	99.6%	99.8% (28.5%)
P6-3	99.6%	99.7% (30.7%)
Enterprise	99.3%	98.8% (41.7%)
M2C College	99.5%	99.7% (29.6%)
M2C ADSL	92.1%	91.1% (38.0%)
Crawdad	99.9%	99.9% (22.2%)
UMass	98.3%	98.8% (29.1%)

Table 5.3: Assignment accuracy and proportion of unknown traffic for GMM

HMM and GMM, accuracy is above 99%) except for the M2C ADSL trace. All other traces are from university networks, and an ADSL commercial network may include different modes of operation for certain applications, which were not in our training set. In addition, the M2C ADSL trace is a packet-header trace and we rely on standard port numbers to label applications, which can involve some incorrect labels.

The accuracies with thresholds is comparable across all methods, but the proportion of unknown traffic varies. For instance, there is around 50% of unknown for HMM Thresh, 40% for K-Means Thresh, and 30% for GMM Thresh. A likely reason for the large proportion of unknown traffic with HMM is that HTTP traffic represents more than 50% of the total traffic in our traces, and HTTP is often labeled unknown with the threshold we chose, as shown in Figure 5.4. The sizes of HTTP packets significantly vary, which can lead to low likelihoods when the size of a packet in the connection is not the same size range as the packets from the model.

Heuristic	Likelihood	Thresh Th= $-3e^{-5}$
	Accuracy	Accur. (Unknown)
P6-1	99.6%	99.8% (53.9%)
P6-2	99.8%	99.9% (53.2%)
P6-3	99.3%	99.9% (50.8%)
Enterprise	99.2%	99.2% (42.3%)
M2C College	99.97%	99.98% (56.0%)
M2C ADSL	78.3%	67.1% (41.4%)
Crowdad	99.9%	99.7% (64.4%)
UMass	98.9%	98.6% (54.7%)

Table 5.4: Assignment accuracy and proportion of unknown traffic for HMM

### 5.3.2 Labeling Accuracy

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

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

We also define true-positive and false-positive metrics to compute per application results. Let  $\mathcal{T}_a$  be the set of connections from application  $a$  in  $\mathcal{T}$  ( $\mathcal{T}_a \subset \mathcal{T}$  such that  $\gamma(x) = a$ ),  $\overline{\mathcal{T}}_a$  the set connections from the other applications ( $\overline{\mathcal{T}}_a = \mathcal{T} \setminus \mathcal{T}_a$ ), and  $\phi_a(x)$  a function that indicates if  $x$  has been labeled with application  $a$  ( $\phi_a(x) = 1$ , 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 5.5 compares the overall accuracy for four test traces (three from Paris 6 and one from an enterprise network<sup>1</sup>), for different number of packets. We evaluate overall accuracy using the 50 models obtained from our training sets. The vertical bars represent two standard deviations. This plot allows us to compare methods and verify that our choice of number of packets from calibration works in practice. For the K-Means and the HMM models, we achieve the highest accuracy on this test set with four packets, which is coherent with our calibration. However, for GMM, we obtain the best

<sup>1</sup>We can only use payload traces because we need to be certain of the actual application

overall accuracy with two packets (92%), and 89% accuracy with four. We obtain very good results of GMM with two packets because the ellipsoidal shape of GMM clusters models well the sizes of the first two packets in HTTP connections, which represent more than 50% of the traffic in our test sets.

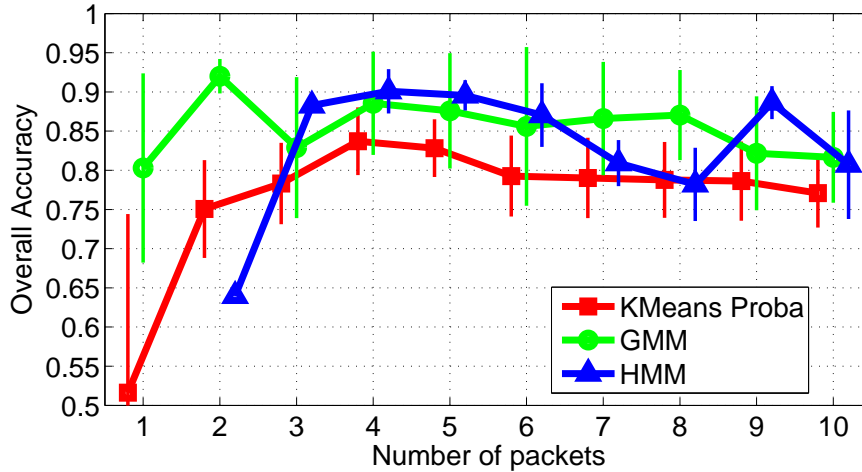


Figure 5.5: Overall Accuracy (Predominant)

Overall accuracy depends on the mix of applications in the test set: if 90% of the test set consists of a single application, overall accuracy will only measure how well we recognize this application. To overcome this limitation, we compute the mean accuracy across applications (i.e., the mean of true positives for each application). This new metric no longer depends on the composition of the test sets. Figure 5.6 presents mean accuracy for the same traces, and shows that models with four packets outperform models with two packets.

Table 5.5 compares overall accuracies for our two labeling heuristics<sup>2</sup>. For the Predominant heuristic, HMM performs poorly compared to GMM and K-Means. This result is counter-intuitive, because most HMM clusters contain only one application (as shown in Figure 4.12), so the Predominant heuristic should work best for HMM. To explain this result, we study 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. This misclassification of POP3 traffic greatly degrades the overall accuracy of HMM with the Predominant heuristic. When

<sup>2</sup>These results differ from the ones shown on Figure 5.5 for four packets, because the figure uses connections from our four test traces, whereas this table focuses on one trace from Paris 6 and one from an enterprise network. In addition, the plots in Figure 5.5 present an average over 50 models with standard deviations, whereas this table uses only one of these models.

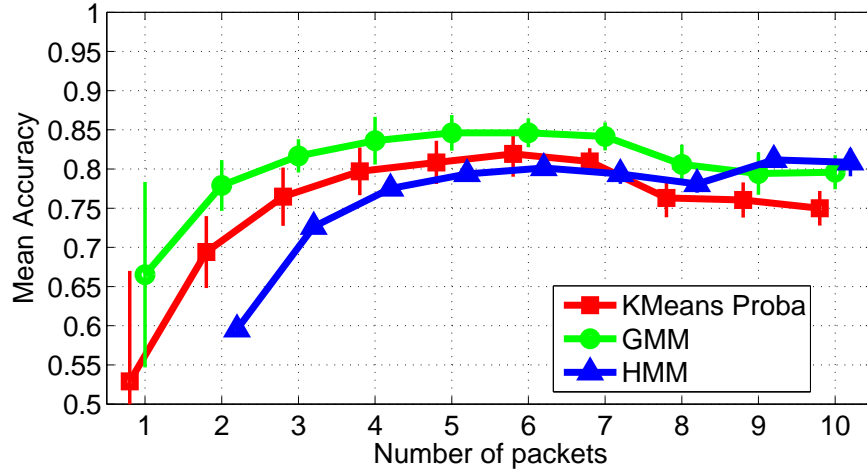


Figure 5.6: Mean Accuracy (Predominant)

using the Cluster+Port heuristic, all methods give higher accuracy. GMM and HMM achieve overall accuracy above 98% (POP3 is no longer mistaken for NNTP with the HMM model).

Heuristic	Trace	K-Means	GMM	HMM
		Proba	Proba	Likelihood
Predominant	P6-1	92.4%	93.5%	92.4%
Predominant	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 5.5: Accuracy of labeling heuristics (4 packets)

Table 5.3.2 presents per-application results. This table shows a detailed comparison of both labeling heuristics for the Paris 6-1 trace (the result for the other traces are similar). For each application  $a$  in our data sets, it shows  $TP(a)$  and  $FP(a)$ . For instance, with the Predominant heuristic and GMM Proba assignment, 94.1% of Edonkey connections are labeled Edonkey (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 labeled masquerade.

As expected, the Cluster+Port heuristic outperforms Predominant. 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  $\mathcal{S}$  and an application in  $\bar{\mathcal{S}}$ . For such clusters, the heuristic chooses

App	GMM without Threshold					HMM without Threshold				
	Predominant		Cluster+Port			Predominant		Cluster+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 5.6: Labeling accuracy for Paris 6-1 trace.

the application in  $\bar{\mathcal{S}}$  for connections not using standard ports. Furthermore, the False Positives are low for most applications.

Let us examine in detail all applications with low accuracies. The HMM model with the Predominant 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, HMM assigns all POP3 connections to this cluster and the Predominant heuristic labels them 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), because the assignment heuristic maps some modes of operation of these applications 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 these 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). We could decrease the proportion of SSH labeled masquerade by adding the SSH connections with new behaviors to our training set.

We also evaluated port-based classification for our target applications. On the Paris 6-1 test trace, port-based classification correctly identified more than 95% of standard applications, but resulted in about 5% false positives (in particular, connections on port 80 that were not HTTP). Port-based classification correctly labeled 65% of FTP connections (the control connections), and did not succeed in identifying Edonkey or

Kazaa connections. As expected, port-based classification is not suitable for applications not using standard ports and is subject to a high level of false positives due to masquerade traffic.

### 5.3.3 Detection of new applications

To test the efficiency of the thresholds presented in Section 5.1 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 5.7 shows the labeling accuracy with thresholds. We focus on GMM and HMM because these methods outperform K-Means. We report labeling accuracy 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 equal to 1 when the label of a connection and its actual application do not match, and to 0 otherwise.

In this table, TP+FN+Unknown+Masquerade=100% for each application. Therefore, low True Positives do not necessarily mean high misclassification. For instance, we accurately label only 60.8% of SMTP connections, but misclassify only 1.2%. Among SMTP connections not labeled as unknown, 98% are accurately identified. With the thresholds defined in Section 5.1, we label unknown 30% of the connections from target applications for GMM (50% for HMM), and detect 65% of the new traffic (i.e., label unknown) for both models. This proportion varies greatly between applications. Some of them are almost never misclassified, whereas we mistakenly label some others with application from our training set in a large proportion. For instance, the GMM model labels 100% of Gnutella traffic as Kazaa, and 40% of MSN connections as Kazaa or Edonkey.

### 5.3.4 Discussion

We can accurately label over 98% of the traffic, which is better than methods presented in previous work [50, 65, 24, 81]. This accuracy is sufficient for most network administrators if they want to profile the traffic on their network, or apply quality-of-service policies that depend on the applications. If a network administrator wants to

App	GMM Thresh $T_g=-25$					HMM Thresh $T_h=-3e^{-5}$				
	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 5.7: Detection of unknown traffic (Paris 6-1) using Cluster+Port

use this technique for security, it needs to couple it with other techniques (such as a content-based analyzer [20]), because we cannot guarantee 100% detection.

Our technique can also detect new traffic. In this case, an administrator could log the traffic labeled unknown for later forensic analysis. If the proportion of unknown traffic increases significantly, this likely indicates a change in the application mix. In this case, administrators should perform a new training phase including this traffic to create new models for the classifier.

Our evaluation shows that GMM represents the best trade-off between accuracy and complexity. The accuracy of GMM is comparable to that of HMM and GMM is much simpler. HMM requires complex computations to assign connections to clusters. Therefore, we choose to implement our classifier using the GMM method.

## 5.4 Implementation

In this section, we present an implementation of our classifier configured with cluster descriptions obtained with GMM. After presenting an algorithm to classify connections online, we describe some optimizations to this algorithm and evaluate its speed and memory consumption.

### 5.4.1 Algorithm

Our online classifier is described in Algorithm 1. It takes as input descriptors of the packets entering and leaving the network. Upon capturing the header of a new packet, the classifier first maps the packet to a connection  $x$  and verifies whether  $x$  is a known connection (lines 2 and 3 of Algorithm 1). All known connections are stored in a hash table,  $\mathcal{C}$ . We identify each connection using a four-tuple (IPsrc, IPdst, Port src, Port dst) of the first data packet in the connection. Follow-up packets identified as either (IPsrc, IPdst, Port src, Port dst) or (IPdst, IPsrc, Port dst, Port src) belong to the same connection. When a packet does not belong to any known connection and the SYN flag is set (which indicates a new connection), we create a new entry for this connection (lines 25 to 29). If the packet flags do not correspond to a new connection, then we cannot classify the connection because we have missed the first packets of the connection. We label such connections “unclassifiable” (line 23). Lines 4 and 5 ignore all connections that have already been labeled. If the packet contains application data, then we store the payload size in the connection descriptor (lines 7 to 15). Otherwise, we ignore the packet. Upon the reception of the  $P$ -th packet, we can label the connection using our labeling function (lines 16 to 18). This algorithm only labels connections and should be used as a function of a management system. This management system could check these labels to apply policies online, for instance.

This algorithm defines the data structures *packet* and *connection*. *Packet* is a structure describing an incoming packet with attributes `payloadSize` (size of TCP payload), `dport` (destination port, to decide whether the packet was sent by the client or the server of the TCP connection), `flags` (TCP flags) and `IPsrc`, `IPdst`, `Port src`, `Port dst` (4-tuple of the packet). Each *connection* is described by the attributes `label` (application associated with the connection), `pkts` (number of packets with application data of the connection that have already traversed the link), `sizes` (vector with the sizes of the first  $P$  payloads) and the 4-tuple of the connection.

### 5.4.2 Optimizations

Running our classifier online on fast links is challenging because the classifier has little time to process each packet, and it must use as little memory as possible because fast memories (such as SRAM) are expensive and limited in size. To improve the efficiency of our implementation we explored several techniques:

1. **Compute the logarithm of the probability** to belong to each cluster, to avoid the evaluation of exponentials, as explained in Section 5.1.

```

1: procedure CLASSIFIER(packet)
2:    $x \leftarrow \text{connection}(\text{packet})$ 
3:   if  $x \in \mathcal{C}$  then
4:     if  $x.\text{label} \neq \text{"None"}$  then
5:       Connection already labeled, ignore packet
6:     else
7:       if  $\text{packet.payloadSize} = 0$  then
8:         No application data, ignore packet
9:       else
10:         $x.\text{pkts} \leftarrow x.\text{pkts} + 1$ 
11:        if  $\text{packet.dport} = x.\text{serverPort}$  then
12:           $x.\text{sizes}[x.\text{pkts}] \leftarrow \text{packet.payloadSize}$ 
13:        else
14:           $x.\text{sizes}[x.\text{pkts}] \leftarrow -\text{packet.payloadSize}$ 
15:        end if
16:        if  $x.\text{pkts} = P$  then
17:           $x.\text{label} \leftarrow \lambda(x)$ 
18:        end if
19:      end if
20:    end if
21:  else
22:    if  $\text{packet.flags} \neq \text{SYN}$  then
23:       $x.\text{label} \leftarrow \text{"Unclassifiable"}$ 
24:    end if
25:     $x.\text{label} \leftarrow \text{"None"}$ 
26:     $x.\text{pkts} \leftarrow 0$ 
27:     $x.\text{serverPort} \leftarrow \text{packet.dport}$ 
28:     $x.\text{sizes}[1..P] \leftarrow 0$ 
29:     $\mathcal{C} \leftarrow \mathcal{C} \cup c$ 
30:  end if
31: end procedure

```

Alg. 1: Online Classifier

2. **Limit lookups to packets that contain application data.** Looking up the connection associated with a packet in the list of connections based on a 4-tuple takes time. We reduce the number of accesses to the list of connections by only performing the lookup for packets that contain application data. Besides, we only create entries for connections with a SYN packet. This represents a small reduction of the list of active connections, but greatly reduces the number of lookups.
3. **Storage optimization:** when we label a connection, we update statistics and remove the connection from the list. This optimization reduces the amount of memory used by our algorithm. However, we cannot use it if we want to apply policies on connections according to their application (we do not keep information about active connections, which is necessary to apply the appropriate policy to all packets).
4. **Garbage collection:** we regularly parse the list of known connections to remove connections that have expired (i.e., for which there has been no packet for one minute). We also remove connections from the list when we see a flag indicating the end of the connection (RST or FIN).

### 5.4.3 Complexity of Online Classification

This section compares the complexity of our classifier to port-based and content-based classifications. 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 only requires an additional look up of the port number in a list of pre-defined ports with the associated applications (one basic operation).

Payload-analysis tools involve an analysis of packet content (which may not always be possible, for instance, in the case of encrypted traffic). They rely on substring matching algorithms. 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 [11]<sup>3</sup>. A payload analyzer needs to search for the signature of each target application in all packets until the connection is classified. Let  $|\mathcal{A}|$  be the number of target applications, and  $\tilde{p}$  the average id of the packet containing the

---

<sup>3</sup>This analysis focuses on simple patterns and does not apply to regular expressions involving wildcard characters such as “\*”, which are necessary to identify some protocols, and involve higher complexities.

signature for these applications. Payload analysis involves  $\frac{n}{m} \times |\mathcal{A}| \times \tilde{p}$  basic operations to label a connection. For traffic classification, the searched text is the TCP payload. In our data set, the average payload size of data packets (i.e., packets containing actual application data) is 600 bytes, which is consistent with data from a large network [38]. Considering an average application signature of four characters (HTTP, for instance), payload analyzers need in average  $n/m = 150$  comparisons to find a signature in a TCP payload. To evaluate  $\tilde{p}$ , we slightly modified Traffic Designer to store the id of the packet which contains the signature that allows the labeling of the connection. We found that for most applications, this signature is in the third or fourth packet. In addition to the number of comparisons required to identify applications, payload analysis requires a considerable amount of memory to store packets while they are processed.

Our classifier needs to store information concerning the first  $P$  packets of the connection, namely, 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. The probability that a connection  $x$  with packet sizes  $x_1, \dots, x_P$  belongs to a cluster with center  $c = [c_1, \dots, c_P]$  and diagonal covariance matrix  $\Sigma = [\sigma_1, \dots, \sigma_P]$  is:

$$f_x = \frac{1}{(2\pi)^{P/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}(x-c)^T \Sigma^{-1} (x-c)} = \frac{1}{(2\pi)^{P/2} (\prod_{i=1}^P \sigma_i^2)^{1/2}} e^{-\frac{1}{2} \sum_{i=1}^P \left(\frac{x_i - c_i}{\sigma_i}\right)^2}.$$

To simplify the computation, we compute  $2 \times \ln f_x$  (the use of diagonal covariance matrices makes this computation very simple):

$$2 \times \ln(f_x) = cst - \sum_{i=1}^P (x_i - c_i)^2 \times \frac{1}{\sigma_i^2},$$

where  $cst$  is a constant relative to the cluster ( $cst = -P \times \ln(2\pi) - \sum_{i=1}^P \ln(\sigma_i^2)$ ). Therefore, our method involves  $3 \times P$  multiplications to evaluate the probability that a connection belong to a cluster, and  $3 \times P \times K$  multiplications to evaluate this probability for all clusters. Table 5.8 summarizes the complexity of the three methods, and shows that our method is at least an order of magnitude faster than signature-matching algorithms.

	Port-based	Content-based	Early Classification
Basic operations to label a connection	1	$\frac{n}{m} \times  \mathcal{A}  \times \tilde{p}$	$3 \times K \times P$
Example with $n/m = 150$ , $ \mathcal{A}  = 10$ $\tilde{p} = 3$ , $K = 30$ , $P = 4$	1	4500	360

Table 5.8: Compared complexities

Duration	3600 s
Size	35.5 GB
Packets	60 236 915
Non TCP/IP packets	3 123 504
Connections	659 178
Connections with SYN	634 314
New Connections/sec	180

Table 5.9: Trace description

#### 5.4.4 Evaluation

To evaluate our algorithm, we implemented it in C using the optimizations described in Section 5.4.2. We did our evaluations with a 2.4GHz opteron using a one-hour trace from our university network described in Table 5.9.

Table 5.10 presents different metrics comparing different versions of our algorithm. Without any optimization our algorithm classifies the trace in 53 CPU seconds (the total time is longer due to delays in disk accesses). The maximum number of active connections stored in memory reaches 659,000 (i.e., all connections are stored) for a total storage of 55MB (we obtained these values using valgrind [80], a well-known memory profiler). Our algorithm does 86 millions lookups. The minimization of lookups reduces the classification time to 50 seconds and reduces the number of lookups to 60 millions. Storage optimization reduces the maximum number of connections stored to 313,000 (for a total storage of 24MB), but increases a little the number of lookups (removing a connection from the list adds lookups). Finally, we evaluate garbage collection. We remove expired connections (i.e., without any packet for more than 60s) with two time intervals: 300s and 60s. Garbage collection greatly reduces the use of memory (less than 2MB) and even decreases the classification time (lookups are faster).

Table 5.10 shows that we are able to classify 35.5 GB of traffic in 46s, which corresponds to 6Gbits/s. This is very fast compared to implementations based on pattern-matching methods. The studies presented in [82] and [20] analyze two well-known

Optimization	Max Entries	Lookups	Time	Max Memory
None	659k	86M	53s	55MB
Lookups	634k	60M	50s	50MB
Storage	313k	72M	50s	24MB
Garbage (300s)	20k	74M	46s	2MB
Garbage (60s)	12k	74M	47s	1.2MB

Table 5.10: Evaluation

Handling connection list	55%
Parsing packets	35%
Assignment and labeling	10%

Table 5.11: Time spent

classifiers: bro [58] and l7-filter [44], and show that they classify traffic at rates lower than 500Mbits/s (on machines similar to the one we used). In addition, our implementation is fast enough for most edge networks because their connections are usually 1Gbit/s (Ethernet) or 2.5Gbits/s (OC-48).

To understand where our classifier spends most of the time, we profiled it using gprof [30]. Table 5.11 presents the proportion of time spent handling active connections (lookups, additions, deletions), parsing packets (finding TCP payload, creating new connections, storing packet sizes) and assigning and labeling connections. We see that managing active connections and parsing packets (which all classification methods must perform) account for 90% of the processing. Cluster assignment and labeling only account for 10% of the processing. Parsing packets and managing connections are simple tasks, which could be performed much faster on dedicated hardware such as network processors. Therefore, with such hardware our classifier could probably run on even faster links.

## 5.5 Challenges

The most important challenges faced by our approach is the necessity to know the size of the first four packets of the flows we want to analyze. This can be difficult for several reasons.

- Packet order: In IP networks, packets 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 [40]. Out of order packet sequences will impact our clustering and thus the quality of our classification. However,

we need only the first few packets to be in the correct order. On most network order will thus not be an issue (on the network we studied, less than 4% of TCP flows show an out of order packet before the fourth data packet). Moreover, it is possible to identify and fix out-of-order packets using TCP sequence numbers.

- **Sampling:** On high speed networks, monitors usually only sample packets, which is problematic for our method since it relies on the sizes of the first packets in connections. This issue is not restricted to our method but to all methods trying to get flow information through packet samples, as discussed in [35]. In this work, the authors point out that a flow sampling strategy must be used instead of packet sampling to retrieve meaningful flow information. With such a strategy, our method will work unaltered.
- **Multi-homed networks:** Large networks often are multi-homed. In this case, a monitor is not guaranteed to see both TCP semi-connections. Our approach needs both half of the TCP connection but this will not be an issue if it is applied to access networks which are most of the time single-homed. For larger networks, aggregating techniques could be used to centralize flow samples and reconstitute full TCP connections.

## 5.6 Summary

In this chapter, we described different labeling and assignment heuristics to design classifiers based on the three clustering methods presented in Chapter 4. We showed that HMM and GMM models outperform K-Means, and therefore that GMM is the best trade-off between accuracy and complexity.

The GMM model with the Cluster+Port labeling heuristic achieves 98% accuracy on our test traces and can identify more than 60% of the new traffic when using a threshold. These results indicates that we can use our classifier to profile the traffic on edge networks, or to apply dynamic per-application quality-of-service policies. We still misclassify a small proportion of the connections, and cannot always flag new applications. Therefore, security tools cannot rely on our method only. However, we could use our tool to speed-up content-based analyzers used for security, by indicating the most likely application of a given connection.

Finally, we presented and evaluated an implementation of our method using the GMM model. This implementation is able to classify traffic at rates up to 6Gibts/s on our server. In addition, our program only spends 10% of its time classifying connections,

and 90% parsing packets and maintaining the list of active connections. This result indicates that our method can be used in addition to any tool gathering statistics about connections at a small cost.

In the following chapter, we extend this classification method to identify the applications in connections encrypted with SSL.

# Chapter 6

## Identification of encrypted traffic

### Contents

---

<b>6.1</b>	<b>Packet traces with encrypted traffic . . . . .</b>	<b>90</b>
<b>6.2</b>	<b>Analysis of SSL traffic . . . . .</b>	<b>91</b>
6.2.1	Description of SSL . . . . .	91
6.2.2	Identifying SSL connections . . . . .	92
6.2.3	Description of SSL traffic . . . . .	93
<b>6.3</b>	<b>Classification of SSL connections . . . . .</b>	<b>95</b>
6.3.1	Detection of the first data packet . . . . .	95
6.3.2	Inferring the original packet sizes . . . . .	96
6.3.3	Method overview . . . . .	98
<b>6.4</b>	<b>Evaluation . . . . .</b>	<b>99</b>
6.4.1	Recognition of SSL traffic . . . . .	99
6.4.2	Recognition of encrypted Applications . . . . .	99
<b>6.5</b>	<b>Summary . . . . .</b>	<b>100</b>

---

In the previous chapters, we presented a new classifier to identify the application of a TCP connection. We showed that this classifier, which we will refer to as *early identification*, is much faster than current content-based techniques [58, 44]. However, content-based methods can outperform our classifier in terms of accuracy. These methods are very efficient because they rely on unambiguous protocol signatures, but they are easy to evade by using encryption. To make matters worse, Secure Sockets Layer (SSL), which can easily be used to encrypt any application communication, is widely available. In this chapter, we extend the classifier presented in Chapters 4 and 5 to detect the underlying application in encrypted SSL connections.

Although any flow-level classifier [65, 47, 84, 50, 24, 8, 81] can potentially identify encrypted applications, this dissertation is the first to design an application-recognition mechanism for encrypted traffic and test it on real SSL traffic. The method presented in [81] is the only one that shares our goal of classifying encrypted traffic. As other flow-level classifiers, however, this mechanism also requires *all* packets in the connection before classifying it. In addition, we evaluate our classifier against real SSL connections, whereas their classifier was only evaluated under simulated encrypted traffic.

After presenting the packet traces we used to train our classifier and to evaluate our mechanism in Section 6.1, we briefly introduce SSL in Section 6.2. Then, we describe a content-based approach to identify SSL connections, and characterize SSL usage in our traces. Section 6.3 presents our mechanism to identify application using SSL, and Section 6.4 evaluates it.

## 6.1 Packet traces with encrypted traffic

Our study of SSL traffic relies on two sets of data: packet traces collected at the edge of two campus networks and manually-generated traces. Packet traces allow us to characterize the usage of SSL in operational networks, when manually-generated traces help us validate our classification mechanism.

We use two one hour traces collected at the edge of the Paris 6 network and described in Table 3.1 (Chapter 3): Paris6-1 and Paris6-3, captured in 2004 and 2006, respectively. Both traces contain packet payload, which allows the identification of SSL versions and options. In addition, we also use the UMass trace, which only captures 58 bytes for each packet. Fortunately, for many connections, 58 bytes are enough to capture 4 bytes of the TCP payload. These first four bytes are enough to accurately identify SSL connections and the versions they use.

To validate our method, we need a ground truth (SSL connections for which the

underlying application is known), which is difficult for encrypted traffic. We use two methods to obtain this ground truth. First, we filter the Paris 6 traces to keep only connections directed to well known HTTPS and POP3S servers in the university. To extend our validation to other types of traffic, we also manually encrypt traces consisting of other applications. We design a methodology to replay packet traces over an encrypted tunnel and capture the resulting connections. We use three machines, say A and B and a controller. Machine A represents the server of the TCP connection and machine B the client. First we establish a tunnel between A and B using *stunnel*<sup>1</sup>. Then, the controller parses an existing packet trace. For each packet in a TCP connection, if the packet was sent from the TCP server, the controller asks A to send the packet to B over the encrypted tunnel, otherwise it asks B to send it. Using this method, we manually encrypt FTP, Bittorrent and Edonkey traffic.

## 6.2 Analysis of SSL traffic

Before constructing a classifier for encrypted traffic, we characterize the use of SSL on two campus networks by studying packet traces. This characterization sheds light on the prevalence of SSL in today's networks, the SSL versions in use, and the types of application that use encryption. Our traces show an increase in the use of SSL between 2004 and 2006. This increase coincides with the surge of new applications that use SSL. For instance, Bittorrent clients (Azureus and uTorrent) now offer SSL encryption as a way to hide from content-based blocking of peer-to-peer applications. These factors indicate that SSL usage will continue to increase.

This section presents a brief background on SSL and a content-based method to identify SSL connections in packet traces. We end with a characterization of SSL in our traces.

### 6.2.1 Description of SSL

Secure Sockets Layer (SSL) authenticates and encrypts TCP connections. SSL runs between the transport layer (usually TCP) and the application layer. Three different versions of SSL have been developed: SSLv2 [73], SSLv3.0 [74] and TLS [78]. As SSL version 2 (SSLv2) presents several security flaws, its use is now strongly discouraged. Its follow-up version is SSL version 3 (SSLv3.0). The latest version, Transport Layer Security (TLS or SSLv3.1), is the standard specified by the Internet Engineering Task Force (IETF) and is similar to SSLv3.0. The differences between SSLv3.0 and TLS are

---

<sup>1</sup><http://www.stunnel.org>

minor (for instance, types of ciphers supported, pseudo-random functions and padding policies) and do not affect the handshake or the packet sizes. Therefore, we use SSLv3 to refer to both protocols.

Figure 6.1 presents an SSL handshake for SSLv2. The exact messages exchanged differ for SSLv3, but the main steps remain the same. First, the client and the server negotiate the SSL version they are going to use and choose an encryption algorithm. Second, the server authenticates itself to the client (the client might do likewise if required by the server) and both peers negotiate an encryption key. Finally, they terminate the handshake, and can start exchanging application data over the encrypted channel.

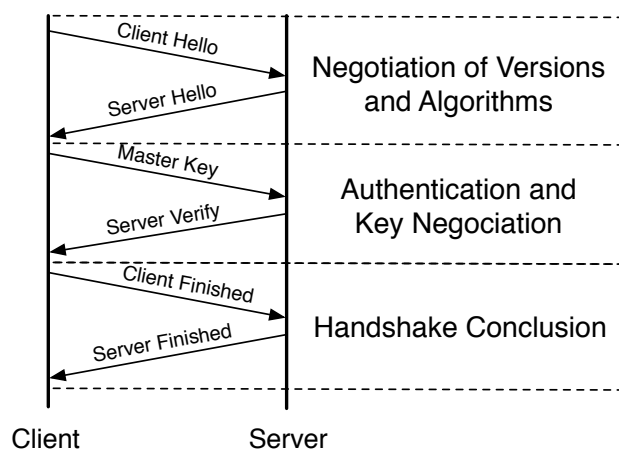


Figure 6.1: Example of SSL handshake (SSLv2)

### 6.2.2 Identifying SSL connections

Many applications can be detected based on a well-known signature (for instance “GET /index.html HTTP/1.1” for HTTP traffic). Unfortunately, there is no such pattern for SSL. Therefore, we rely on the value of specific bits in the TCP payload to precisely identify SSL connections.

The first data packet of an SSL connection is sent by the client of the TCP connection and gives information about the client capabilities (in particular, the SSL versions and encryption algorithms it supports). Based on this information and on its own capabilities, the server decides on the final configuration when it sends its first packet.

Therefore, to verify if a connection is using SSL and determine the version, we can simply analyze the second data packet in the connection (“Server Hello”).

The first two bits in SSLv2 headers are always 1 and 0, the following 14 bits contain the size of the SSL record and the third byte is the message type (1 for “Client Hello” and 4 for “Server Hello”). The first byte of SSLv3.x (i.e. SSLv3.0 or TLS) packets is the message type (22 for handshake packets). The second byte indicates the major version (3) and the third the minor version (0 for SSLv3.0 and 1 for TLS).

Let  $bit_i[j]$  be the content of bit  $j$  in the payload of packet  $i$  in a connection,  $bit_i[j : k]$  the integer represented by the sequence of bits from  $j$  to  $k$ ,  $Byte_i[l]$  the value of byte  $l$ , and  $s_i$  the payload size of packet  $i$ . We summarize the decision process to determine if a connection is using SSL and the associated version in the following algorithm:

```

If  $bit_2[0] = 1$  and  $bit_2[1] = 0$  and  $bit_2[2 : 15] = s_2$  and  $Byte_2[3] = 4$ 
    Connection is an SSLv2 connection
Else If  $Byte_2[1] = 22$  and  $Byte_2[2] = 3$ 
    If  $Byte_2[3] = 0$ 
        Connection is an SSLv3.0 connection
    Else If  $Byte_2[3] = 1$ 
        Connection is a TLS connection
    Else Connection is not using SSL
Else Connection is not using SSL

```

### 6.2.3 Description of SSL traffic

We applied our identification mechanism to three traces: Paris6-1, Paris6-3 and UMass. Table 6.1 shows the proportion of each SSL version found in our traces. We see that most SSL traffic consists of SSLv3.0 and TLS, although there are still a few instances of SSLv2 in the Paris6-3 trace. By comparing the proportion of SSL connections in the Paris6 traces from 2004 and 2006 (4.6% and 8.6%, respectively), we see a sharp increase in the usage of SSL. This trend is supported by the SSL surveys achieved by netcraft [52] (in 2005 only, the use of SSL in the web servers they surveyed increased by 30%). On the UMass trace, this proportion is lower. This difference is because the Paris6 traces consist only of academic traffic, whereas the UMass campus also has a dorm and, therefore, contain many other types of applications (such as online games).

Table 6.2 presents connections associated to non-standard SSL traffic (either SSL connections on non-SSL ports, or non-SSL connections on SSL ports). An interesting

Trace	SSL Conn.	SSLv2	SSLv3	TLS
Paris6-1 (2004)	4.6%	0.6%	81.0%	18.4%
Paris6-3 (2006)	8.6%	0.2%	53.2%	46.6%
UMass	1.2%	0 %	48%	52%

Table 6.1: SSL versions

observation is the proportion of non-SSL traffic in connections using standard SSL ports (labeled as “SSL Port but not SSL”). We studied this traffic in detail. In Paris6-1, all non-SSL connections on SSL ports were non-encrypted traffic using port 443 (probably misconfigured web servers). In Paris6-3, we still find this non-encrypted HTTP traffic, but also observe other types of traffic. For instance, the trace contains unencrypted SIP traffic (VoIP connections from Instant Message software using port 443 to avoid firewalls), and HTTP connections using the CONNECT method. This method is used when a web client connects to an SSL web page using a proxy. However, the contacted servers were not proxies but web servers. It turned out these clients were trying to connect to SMTP servers using the web servers as TCP proxies, probably to send spam (this method works for misconfigured Apache servers with proxy capability). In the UMass trace, we also found Bittorrent connections using port 443 to avoid firewalls.

Trace	SSL Port but not SSL	SSL on non-SSL Ports
Paris6-1 (2004)	1.9%	1.1%
Paris6-1 (2006)	1.1%	4.2%
UMass	5.0%	1.5%

Table 6.2: Non-standard SSL Traffic

Finally, using the detection method presented in Section 6.2.2, we evaluated the proportion of SSL on ports not usually associated with SSL (“SSL on non-SSL Ports”). This proportion is not negligible and is even increasing on the Paris6 network. This indicates that SSL is spreading to applications for which it was not formerly used.

Our characterization shows that SSL traffic has increased in the Paris6 traces. We also find an increase of SSL on non-standard ports. These observations highlight the importance of identifying applications in SSL connections. Section 6.3 addresses this problem.

### 6.3 Classification of SSL connections

Our classifier for encrypted traffic builds on the observation that SSL does not modify significantly the number of packets, their size, and their inter-arrival time [71, 34]. Applying early identification to connections encrypted with SSL raises two challenges: we need to identify the packets that carry encrypted application, and we need to infer the original sizes of these data packets. In this section, we present methods to address these challenges and then describe our mechanism to identify encrypted traffic.

#### 6.3.1 Detection of the first data packet

Before sending encrypted application data, SSL connections start with a handshake, as described in 6.2. For SSLv2, the handshake can take four or six packets. If in the negotiation, the client and the server discover they share an encryption key that is still valid, the handshake takes 4 packets, otherwise it takes six packets. To decide which handshake is used in a given connection, we check if the second packet sent by the client starts a key negotiation (as in Figure 6.1).

For SSLv3, this detection is more difficult because the last packet in the SSL handshake may contain an SSL negotiation record as well as encrypted data. Therefore, to detect the first application packet in SSLv3 connections we inspect SSL records until we find the first record with content type equal to 23, which indicates an application payload. This inspection is not computationally intensive because the header of each record contains the size of the record (bytes four and five of the record header), not including the length of the header (five bytes). Let  $s$  be the payload size of the packet and  $B[l]$  be the content of byte  $l$  in this payload. To check if an SSLv3 contains a record with application payload we use the following algorithm:

```

cumulativeSize=0
While cumulativeSize < s
  If Byte[cumulativeSize] = 23
    Packet contains application payload
    Break While
  Else
    cumulativeSize = cumulativeSize + Byte[cumulativeSize + 3] × 256
                      + Byte[cumulativeSize + 4] + 5
End While

```

We applied this algorithm on the Paris6-3 trace. Figure 6.2 shows the distribution of the position of the first SSL packet that contains application data across all SSL connections. This result shows that there is never application data in the first two packets (which is expected from the RFC). This implies that it is safe to start inspecting TCP payloads after the third packet to identify the first application packet.

Figure 6.2 also shows that application data may start at any packet between 3 and 12, there is no clear pattern. Therefore, our classifier will need to rely on online packet inspection to detect packets with application data.

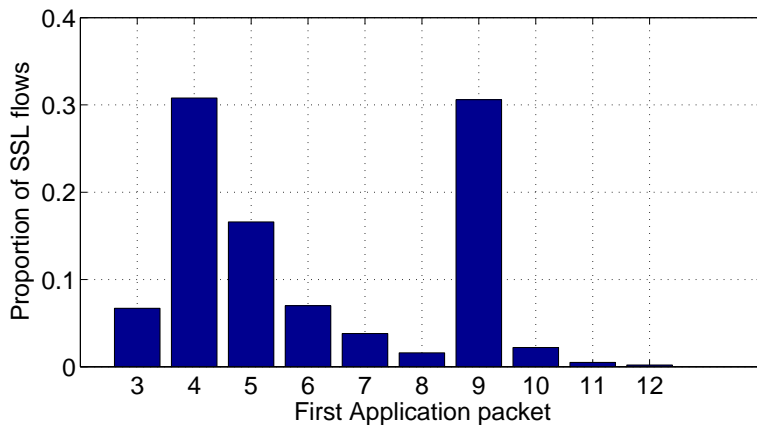


Figure 6.2: Position of the first packet with application data

### 6.3.2 Inferring the original packet sizes

Once we have identified the packets corresponding to application data in a connection, we need to infer the original sizes of these packets to use the classifier described in Chapter 5. SSL can use different encryption algorithms, which will modify packet sizes differently. Table 6.3 shows the most common encryption algorithm used in the Paris6-1 and Paris6-3 traces. The specification of SSL allows for more than 50 encryption methods. However, we can see that the five most common algorithms account for more than 98% of SSL connections. Although the use of the RC4 based algorithms has decreased between 2004 and 2006, RC4\_128\_MD5 still account for 66% of the traffic, whereas less than 25% of the connections use AES-based ciphers.

To evaluate the influence of encryption algorithms on the sizes of exchanged packets, we design a small application that sends packets with different sizes over an SSL connection. Figure 6.3 shows the relationship between the size of application payload

Cipher	P6-2004	P6-2006
RC4_128_MD5 (0x04)	79.7%	66.0%
DHE-RSA-AES256-SHA (0x39)	5.9%	13.6%
AES_256_SHA (0x35)	1.0%	10.4%
RC4_128_SHA (0x05)	9.7%	7.0%
RC4_40_MD5 (0x03)	2.0%	2.1%
Other	< 2%	< 1%

Table 6.3: Proportion of each cipher

and the final size of the ciphered packet depending on the encryption algorithm (we focus on packets with less than 100 bytes, but the evolution remains the same for larger values). This figure shows that encryption mechanisms increase the size of the packets by an amount that depends on the cipher. For RC4 based ciphers, this increase is fixed: 25 bytes for RC4\_128\_SHA and 21 bytes for the other two. For AES-based ciphers, the increase varies by steps, because they use blocks.

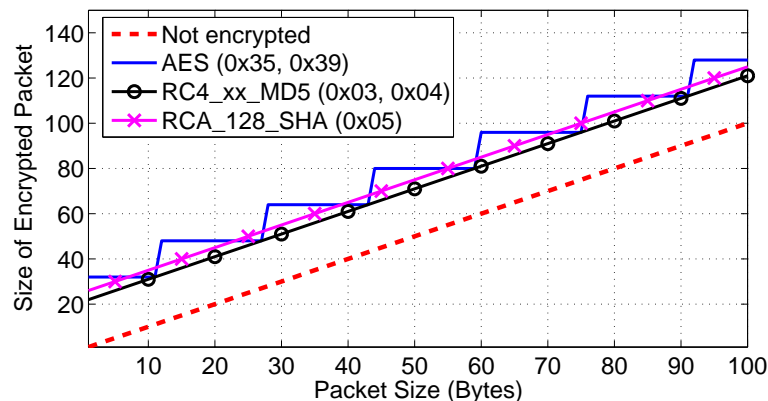


Figure 6.3: Size of encrypted packets according to cipher

These results are encouraging because even if encryption alters packet sizes, this alteration is limited and predictable. The most accurate method to decide on the size of the original packet is to look up the encryption method in the handshake packets and transform the size of application packets accordingly. However, for the five most common ciphers this method is overkill because the increase varies from 21 to 33 bytes. Therefore, instead of keeping track of the cipher, we use a simple heuristic to decide on the size of the original packet: subtract 21 from the size of the encrypted packet regardless of the cipher. Even though this inferred size is not perfect, this simple heuristic

will bring the points representing the connections closer to the clusters containing the original applications.

### 6.3.3 Method overview

Figure 6.4 describes our classification mechanism. This classifier takes as input a stream of packets from a TCP connection and outputs the application associated to the connection. It runs in three steps: recognition of SSL connections, detection of the first packet containing application data, and recognition of the encrypted applications.

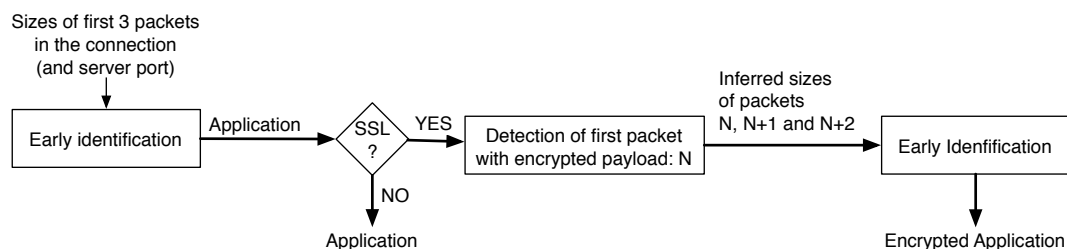


Figure 6.4: Classifier Overview

To recognize SSL traffic, we use a slightly modified version of the classifier described in Chapter 5. First, we run the training phase with different target applications: HTTP, FTP, NNTP, POP3, SMTP, SSH, MSN, Bittorent, Edonkey, SSLv2 and SSLv3. In the previous chapters, we had HTTPS and POP3S in our training sets. However, since we focus on the first packets in TCP connections, our classifier detected SSL and was able to tell HTTPS from POP3S because they often use different SSL implementations, with different behaviors (they regroup SSL records differently). To select 100 SSLv2 and SSLv3 connections, we use the method presented in Section 6.2

To design this new classifier, we applied GMM clustering to this training set and calibrated it as described in Chapter 4. We found that the best parameters for this training set are four packets and 40 clusters. However, this calibration gave results almost as good for three packets and 35 clusters. Since we need to start inspecting SSL packets to look for the first application packet in the third packet, we decide to use three packets for our application classifier. A classifier using four packets would involve the storage of the payload of the third packet, which would require additional memory. We evaluate the efficiency of this classifier in Section 6.4 using the Predominant and Cluster+Port heuristics.

After the classifier establishes that the connection is SSL, it analyzes the packets in

the connection to find the first application packets. Then, we apply the same classifier to the transformed sizes of the first packets with encrypted data to decide on the encrypted application. We extend the *Cluster+Port* labeling heuristic to take into account SSL-specific ports: we use 443 for HTTPS, 993 for IMAPS and 995 for POP3S.

## 6.4 Evaluation

In this section, we first evaluate our method to recognize SSL on the Paris6-3 trace. Then, we validate our method to recognize encrypted applications on HTTPS and POP3S traffic extracted from the Paris6-3 trace and on manually encrypted connections.

### 6.4.1 Recognition of SSL traffic

To evaluate the accuracy of our classifier to recognize SSL connections, we use two metrics: the proportion of connections accurately classified for all applications in our test data set (True Positives) and the proportion of connections of each application that are incorrectly labeled (False Positives). Table 6.4 presents both metrics for our two labeling heuristics: Predominant and Cluster+Port. This table shows that our classifier, based on the sizes of the first three data packets achieves a very high accuracy and that it recognizes SSLv2 and SSLv3 for more than 80% of the connections. With the Predominant heuristic, 82% of SSLv2 traffic is accurately classified and 68% of SSLv3. Some SSL connections are assigned to clusters that contain SSL, and another application that predominates. Therefore, the use of the Cluster+Port heuristic increases the accuracy because the port information helps to decide on the final application. The reason for the 2.3% false positives for SSLv2 is that, some SSLv3 connections are classified as SSLv2 (hence only 81% true positives for SSLv3). This is not unexpected because behaviors of SSLv2 and SSLv3 are similar in some cases. However, we can easily limit the impact of this misclassification by inspecting packets from connections classified as SSLv2 to decide on the real SSL version that is used.

### 6.4.2 Recognition of encrypted Applications

We evaluate the recognition of encrypted applications on two different test sets described in Section 6.1. Table 6.5 presents the results for traffic from Paris6-3 trace. Since we test each application separately we limit ourselves to true positives. These results show that we correctly label more than 98% of connections of applications that

Heuristic	Predominant		Cluster+Port	
Application	True Positives	False Positives	True Positives	False Positives
bittorent	74.65%	0.01%	97.30%	0.23%
edonkey	94.76%	2.89%	95.08%	0.18%
FTP	91.00%	0.04%	97.95%	0.04%
HTTP	96.50%	2.96%	98.95%	0.00%
MSN	95.36%	0.90%	100.00%	0.00%
NNTP	94.40%	0.34%	99.15%	0.00%
POP3	96.65%	2.67%	99.25%	0.00%
SMTP	86.35%	0.42%	98.85%	0.00%
SSH	97.73%	0.00%	96.10%	0.00%
SSLv2	82.07%	2.20%	94.71%	2.30%
SSLv3	67.75%	0.33%	81.20%	0.27%

Table 6.4: Application detection, including SSL(Paris6-3 Trace)

often use SSL (HTTP and POP3). Table 6.6 evaluates our mechanism for applications that cannot be detected with port-based methods and are usually recognized based on signatures. Our classifier accurately classifies these applications when they are encrypted, with more than 85% accuracy for the Cluster+port heuristic.

Applications	Predominant	Cluster+Port
http	99.95%	99.95%
pop3	98.45%	98.45%

Table 6.5: Detection of Encrypted Applications in Paris6-3 trace

Applications	Predominant	Cluster+Port
ftp	90.58%	92.67%
bittorent	77.87%	86.48%
edonkey	94.56%	96.57%

Table 6.6: Detection of Encrypted Applications for manually encrypted traffic

## 6.5 Summary

This chapter characterized the usage of SSL on two campus networks. Results from Paris 6 showed that the use of SSL is growing and that the number of applications using SSL is increasing. Applications that want to hide are using SSL encryption, because it

is simple and efficient against content-based analyzers. For instance, some Bittorrent clients (such as Azureus and uTorrent) now offer SSL encryption as a way to hide from content-based blocking of peer-to-peer applications.

We proposed a mechanism to recognize the application inside SSL encrypted connections based on the classification technique presented in the previous chapters. To adapt early identification to SSL traffic, we described a method to identify the packets carrying encrypted application data, and a heuristic to infer the original sizes of these data packets. Our technique classifies correctly more than 85% of connections of the five applications encrypted with SSL in our traces.

The classifier we described in this chapter identifies applications encrypted using “basic” SSL. The latest SSL implementations include options for compressing data and sending empty segments, which would affect our detection mechanism. To handle these new options, we would need to develop new heuristics to infer the original sizes of compressed packets (which might be more or less difficult depending on the compression algorithm), and to detect “fake” segments.

In this chapter, we focused on traffic encrypted with SSL. However, we believe that our algorithm could work with other encryption mechanisms such as SSH and IPSEC. These protocols use the same encryption algorithms as SSL, and we could, therefore, easily infer the original packet sizes. However, for SSH and IPSEC, identifying application packets would be much more difficult, and we would need to rely on heuristics to demultiplex connections, such as the methods presented in [81].



# Conclusion

Identifying the application associated with a traffic flow is essential for network administrators. Today, in operational networks, this identification usually relies on TCP (or UDP) port numbers. However, the development of applications performing dynamic port-negotiation or using non-standard ports (sometimes even trying to hide) has seriously reduced the efficiency of this technique. The current state-of-the-art methodologies consist of two types of techniques: content-based approaches, which inspect packet payloads to find specific application signatures; and behavior-based approaches, which model connections using metrics such as packet sizes and inter-arrival time to identify applications. Content-based methods are very efficient, but cannot work on very fast links because they are computationally intensive, and cannot identify encrypted traffic. Behavior-based techniques are promising, but current behavior-based methods rely on metrics computed on whole connections. This requirement prevents the use of these methods online.

In this thesis, we propose a behavior-based classification method that only uses metrics available in the first packets of a connection, which enables online control of traffic. Section 7.1 presents the main contributions of this thesis and we present future research perspectives in Section 7.2.

## 7.1 Contributions

This thesis makes the following contributions.

**The payload size of the first four packets is enough to distinguish applications.** We compared different metrics on TCP connections that are available early in the connection (i.e., after a few packets). Metrics related to inter-arrival time do not help to distinguish applications, whereas payload size is a good metric. Our analysis shows that four packets is the best for distinguishing among applications in our traces. Besides, we showed that this metric does not change across networks, and therefore that a classifier using this feature and designed for one network can be applied on others.

**GMM represents the best trade-off between accuracy and complexity to identify application behaviors.** We evaluated three different clustering methods to regroup connections according to their behavior: K-Means and GMM on Euclidean space, and spectral clustering on HMM sequences. Even though the HMM representation is richer, the quality of the clustering is comparable to the simpler Euclidean representation when using GMM clustering. These results are robust to the choice of the training set.

**Early application identification labels correctly 98% of the connections after only four packets.** Using the results from the training phase: the set of clusters and their composition (i.e. the applications associated to the training connections assigned to each cluster), we defined different assignment and labeling heuristics. The GMM clustering combined with TCP port numbers correctly classifies over 98% of known applications for all the payload traces studied. The GMM or HMM classifiers can also label more than 60% of the connections from applications not in the training set as “unknown” or “masquerade”. Our approach results in an accuracy that is comparable (and sometimes superior) to current state-of-the-art behavior-based methods, using only information from the first packets whereas these methods use statistics computed over complete connections.

**Our implementation of early application identification is an order of magnitude faster than content-based methods.** We implemented our classification method and proposed optimizations to speed-up the classification and limit memory usage. We showed that our implementation can classify traffic at rates up to 6Gbits/s,

which is enough for current edge networks, and more than ten times faster compared to implementations of content-based methods in similar conditions. In addition, we profiled our program and showed that our classification method can be added to any tool that gathers per-connection statistics at a very small cost. The Matlab library used to create models and the code of our classifier are available for download at: <http://rp.lip6.fr/~bernaill/earlyclassif.html>.

**Our method is capable of identifying applications inside SSL connections with a high accuracy.** We were the first to: (i) analyze SSL traffic on different networks and show that the usage of SSL is growing and that the number of applications using SSL is increasing; and (ii) to describe a methodology to transform encrypted packets and show experimentally that it is possible to identify encrypted applications using only the first packets in the connections.

## 7.2 Future directions

There are numerous possible directions to extend the work presented in the thesis:

**Classification of UDP traffic.** Classification of UDP traffic based on the sizes of the first packets is challenging, because it is difficult to identify the first packets in a UDP flow. Our method can be extended to UDP traffic by using heuristics to identify the beginning of a UDP flow, such as presented in [76].

**Adaptive number of packets:** our classifier uses a fixed number of packets to classify connections. The advantage is that it is deterministic. However, some applications may be easier to identify with more or less packets. Therefore, one could evaluate an approach that labels connections after  $1, 2, \dots, N$  packets until it can identify the application with enough confidence.

**Analysis of the stability of our application models:** the model of applications we compute during the training phase remains valid as long as the application mix do not change on the monitored network. When new applications appear, we need to perform a new training phase to take them into account. During this thesis, we studied the Paris 6 network over three years. In this period, we did not see any new application, but noticed the disappearance of Kazaa and edonkey traffic (due to network policies). However, this is specific to this academic network, and we would like to analyze the

evolution of applications on commercial networks and to measure how often we need to re-train our classifier to take these evolutions into account.

**Router implementation:** our current implementation is a C program running on pcap traces. We would like to implement the classifier on a Network Processor to test it on live traffic and evaluate the maximum throughput.

**Extension to other encryption mechanisms:** currently our classifier can identify connections in SSL encrypted connections. We would like to extend our method to other encryption mechanisms such as SSH and IPsec. For both these protocols, the isolation of connections and the determination of the first application packets will be more challenging. Besides, the latest SSL implementations include options for compressing data and sending empty segments. These options would affect our detection mechanism and we plan to extend it to take them into account.

Although the methodology and the classifier presented here represent an advancement for online classification, there is still an important issue to be addressed before they can be used for security: evasion. 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. An “attacker” could easily evade our method by padding packet payloads in order to modify sizes. To design a classifier that needs to be used in a hostile environment, it is important to rely on several approaches. To improve our classifier, we could couple our method with other techniques to verify applications. For instance, we could use it as a first step to the analyzer proposed in [20] to identify the potential protocols quickly. Another promising possibility would be to use our method in addition to tools performing deep packet inspection, when they cannot identify application because of encryption. Building classifiers that can resist complex evasion techniques remains an important challenge for the traffic classification community.

# Thesis' French Version

## Contents

---

<b>A.1 Introduction</b>	<b>108</b>
A.1.1 Problématique	109
A.1.2 Plan	111
<b>A.2 Identification des applications en temps réel</b>	<b>112</b>
A.2.1 La distinction des applications	114
A.2.2 L'apprentissage	116
A.2.3 La classification temps réel	122
A.2.4 Implémentation de notre méthode	127
A.2.5 La classification du trafic chiffré	128
<b>A.3 Conclusion</b>	<b>133</b>
A.3.1 Contributions	133
A.3.2 Perspectives	135

---

The next pages correspond to a french resumé of this thesis, which is a requirement of the University Pierre et Marie Curie.

## A.1 Introduction

Au cours des vingt dernières années, l'Internet s'est métamorphosé. Le petit réseau universitaire des débuts est devenu un immense système qui interconnecte des dizaines de milliers de réseaux autonomes. Ce développement rapide a été accompagné par de profonds changements en ce qui concerne les utilisations de l'Internet. A l'origine, les échanges étaient limités au courrier électronique et aux groupes de discussions. Puis, quelques années plus tard, la nécessité de trouver et d'organiser l'information a entraîné la création des premières pages Web utilisant les liens hypertextes. Aujourd'hui l'Internet est utilisé par des millions de personnes. Les ordinateurs ont maintenant des processeurs très puissants, des disques durs de grandes capacités et des connexions hauts débits. Cette évolution a permis le développement d'une grande variété d'applications. De plus, de nouvelles applications voient le jour chaque mois.

Parallèlement à l'évolution des applications, la structure de l'Internet a beaucoup changé. Nous pouvons maintenant distinguer deux types de réseaux : ceux qui permettent à d'autres réseaux d'être connectés à l'Internet (les réseaux de transit), et ceux qui permettent à des utilisateurs finaux d'accéder à l'Internet (les réseaux de bordure). Les réseaux de transit appartiennent aux grands fournisseurs d'accès (comme Sprint ou France Télécom). Les réseaux de bordure (par exemple, celui d'une université ou d'une entreprise) ont besoin des réseaux de transit pour accéder à la totalité de l'Internet. Cette relation est illustrée par la figure A.1. Les réseaux de transit et les réseaux de bordure ont des besoins différents et ne sont donc pas conçus de la même manière. Les réseaux de transit ont besoin de liens à très haut débit (souvent 10Gbit/s) pour transférer le trafic de plusieurs réseaux de bordure. De leur côté, ces réseaux de bordure permettent uniquement à leurs utilisateurs de se connecter, et peuvent donc se contenter de liens moins rapides (quelques MBit/s pour une petite entreprise, et 1Gbit/s ou 2.5Gbit/s pour une grande entreprise ou pour une université).

Les administrateurs des réseaux de bordure désirent souvent appliquer des politiques de qualité de service qui varient selon les applications (pour que la voix sur IP ait priorité sur les transferts de fichiers, par exemple). De plus, ces administrateurs ont besoin de connaître les applications qui sont présentes sur leur réseau pour garantir des règles d'utilisation (pas de trafic pair à pair, par exemple) et s'adapter aux évolutions du trafic. Du fait de ces exigences, les administrateurs ont besoin (1)d'identifier les applications associées aux flux qui transitent sur leur réseau le plus tôt possible, et (2) d'appliquer les règles en usage, si nécessaire. C'est pourquoi il est essentiel de classer

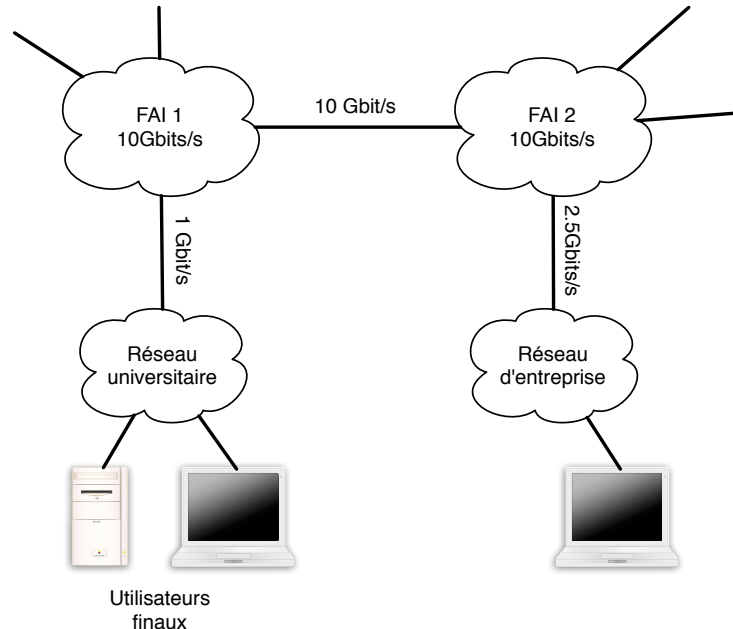


FIG. A.1 – Exemple de réseaux interconnectés. Transit : FAI 1 et FAI 2. Bordure : Université et Entreprise

chaque flux précisément et rapidement pour appliquer des politiques de qualité de service dynamiques et pour comprendre les évolutions du trafic.

Dans cette thèse, nous décrivons une méthode pour identifier rapidement (c'est-à-dire après quelques paquets) les applications associées aux flux qui transitent sur le lien connectant un réseau de bordure à l'Internet. Pour ce faire, notre méthode utilise la taille des données des premiers paquets d'une connexion TCP. Dans cette section, nous commençons par décrire comment les données de l'application sont transmises sur l'Internet. Ensuite, nous décrivons les problèmes soulevés par la classification du trafic.

### A.1.1 Problématique

L'identification rapide des applications associées avec des connexions TCP est difficile pour plusieurs raisons.

#### Des applications qui évoluent rapidement

Jusqu'à la fin des années 1990, les applications utilisaient toutes l'architecture client-serveur et avaient des comportements simples : certaines machines spécialisées fournis-

saient des services, tandis que la vaste majorité des hôtes de l'Internet étaient de simples clients de ces services. Les communications entre les clients et les serveurs se faisaient avec des connexions sur des ports standards et étaient faciles à identifier. Cependant, beaucoup de nouvelles applications ont vu le jour au cours de la dernière décennie. Une partie de ces nouvelles applications est plus difficile à identifier car elles n'utilisent plus les ports standards [65, 24].

Tout d'abord, certaines applications utilisent des connexions dynamiques. C'est le cas de FTP (File Transfer Protocol) par exemple. FTP utilise plusieurs connexions entre les hôtes en communication : une pour le trafic de contrôle et les autres pour le transfert des données. Les paramètres des connexions utilisées pour le transfert des données (et en particulier les ports TCP utilisés) sont négociés dans la connexion de contrôle. C'est pourquoi il est souvent facile d'identifier la connexion de contrôle, qui utilise en général le port 21, mais beaucoup plus difficile de reconnaître les connexions de données. D'autres protocoles, tels que ceux de voix sur IP (H323 par exemple) utilisent aussi plusieurs connexions de manière similaire.

D'autre part, certaines applications ne sont pas standardisées. Ces applications utilisent des protocoles propriétaires, ce qui les rend beaucoup plus difficiles à reconnaître. C'est le cas de nombreuses applications pair à pair, qui représentent aujourd'hui une part importante du trafic sur l'Internet [42]. De plus, ces applications font souvent tout leur possible pour ne pas être détectées car elles sont utilisées pour transférer illégalement des films ou des logiciels et sont donc interdites sur de nombreux réseaux.

### **Une information disponible limitée**

Pour identifier les connexions rapidement (c'est-à-dire après quelques paquets), nous pouvons seulement avoir accès aux informations qui sont présentes dans les premiers paquets. De plus, nous pouvons seulement utiliser les informations contenues dans les entêtes de ces paquets puisque leur contenu peut être chiffré. L'utilisation de SSL s'est beaucoup développé ces dernières années, comme l'indiquent les évaluations faites par Netcraft [52]. Nous pouvons penser que SSL va continuer à se développer, en particulier pour le trafic pair à pair (les client Bittorent les plus répandus, tels qu'Azureus, proposent déjà de chiffrer leurs communications avec SSL). De plus, même pour le trafic qui est transmis sans chiffrement, l'étude du contenu des paquets peut être impossible pour des raisons de protection de la vie privée.

### **Des débits de réseaux de bordure élevés**

Les connexions de grands réseaux à l'Internet utilisent souvent des liens à très haut débit. Beaucoup d'universités utilisent ainsi des liens 1Gbit/s Ethernet (c'est le cas par exemple du lien qui connecte l'université Pierre et Marie à Renater, le réseau universitaire français). Certains réseaux plus importants utilisent même déjà des liens OC-48 (2.5Gbit/s), et il est probable que la bande passante utilisée par les réseaux de bordure continue à augmenter. Ces haut débits requièrent des performances très élevées pour les outils qui analysent le trafic.

La mémoire est un goulot d'étranglement pour les outils qui analysent les connexions TCP [26] parce que ceux-ci doivent conserver des informations sur toutes les connexions actives. Sur des liens à haut débit, le nombre de connexions actives peut dépasser le million [38], ce qui implique une utilisation importante de mémoire. De plus, les analyseurs de trafic doivent faire de nombreuses recherches dans la liste des connexions, ce qui nécessite des mémoires rapides.

#### **A.1.2 Plan**

Ce résumé est composé de trois parties. Nous introduisons tout d'abord le problème dans la Section A.1. Ensuite, dans la Section A.2, nous décrivons le fonctionnement de notre méthode de classification temps réel. Enfin, dans la Section A.3, nous concluons ce résumé en présentant les principaux résultats obtenus au cours de cette thèse et en présentant les perspectives d'extensions de ces travaux.

## A.2 Identification des applications en temps réel

La méthode la plus simple pour identifier les applications consiste à observer les numéros de ports dans les entêtes TCP puis à utiliser les correspondances port-application définies par l'IANA. Les méthodes s'appuyant sur les ports peuvent être efficaces parce que la majorité des applications les plus classiques utilise les numéros de ports standards (ainsi, la majorité des connexions HTTP et POP3 utilise les ports 80 et 110, respectivement). Cependant, de nombreuses applications effectuent de la négociation de ports dynamique (donc des ports non standards) ou utilisent vraiment des numéros de port différents pour ne pas être détectées. C'est pourquoi la classification par port est de moins en moins efficace [42, 65, 50, 43]. Une solution alternative consiste à chercher des signatures applicatives spécifiques dans le contenu des paquets [46]. Cette méthode est très précise mais ne peut pas être employée dans toutes les situations. En effet, l'analyse du contenu des paquets soulève le problème du respect de la vie privée. De plus, l'analyse du contenu a un coût élevé en termes de traitement et d'utilisation de la mémoire. Enfin, on ne peut appliquer ce type de méthode lorsque le contenu des paquets est chiffré.

La communauté scientifique a proposé plusieurs techniques de classification pour contourner les limites des techniques reposant sur les ports ou le contenu des paquets [65, 47, 84, 50, 43]. La plupart de ces méthodes utilisent des statistiques sur les connexions : durée, nombre de paquets, taille moyenne des paquets ou temps inter-arrivées par exemple. Malheureusement, on ne peut pas utiliser ces techniques pour faire de la classification en temps réel car elles ne peuvent identifier l'application que lorsque la connexion est terminée. Par conséquent, on ne peut les utiliser pour effectuer des traitements dynamiques sur les connexions (blocage ou qualité de service par exemple).

Dans cette thèse, nous proposons une méthode de classification qui utilise des statistiques sur les connexions qui sont disponibles après quelques paquets. Notre technique de classification comporte deux étapes : une étape d'apprentissage et une étape de classification. Nous décrivons ces deux étapes dans les sous-sections qui suivent. La Figure A.2 décrit le fonctionnement général de notre méthode. La partie gauche représente les différents éléments de l'étape d'apprentissage et la partie droite décrit le fonctionnement de notre outil de classification. Ces deux étapes ne se déroulent pas au même endroit, ni au même moment. L'étape d'apprentissage peu prendre du temps et est effectuée hors ligne sur des serveurs de calcul dédiés, alors que l'étape de classification

se déroule en temps réel au coeur du réseau sur une machine qui a accès aux entêtes des paquets qui traversent le lien étudié.

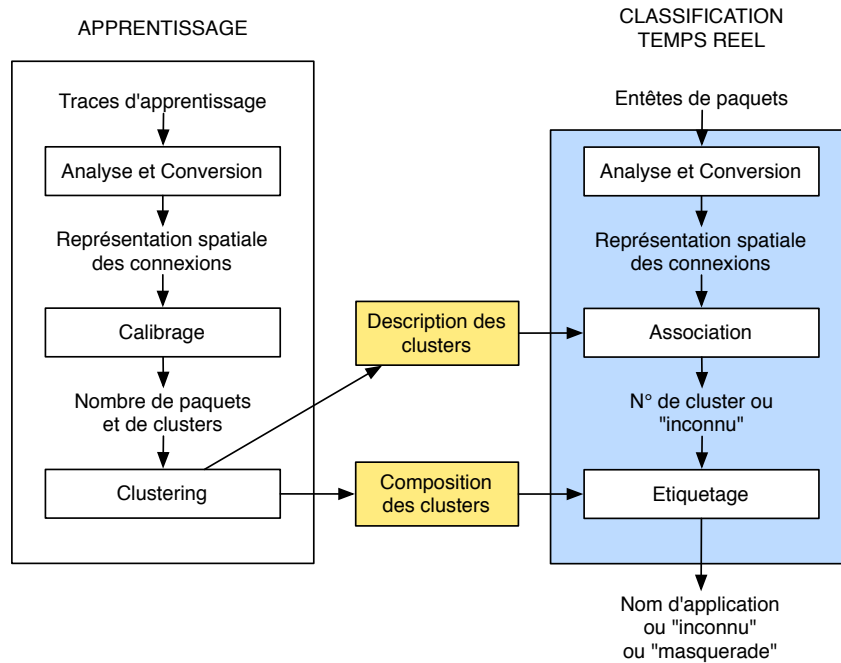


FIG. A.2 – Méthode de classification

L'objectif de l'étape d'apprentissage est d'obtenir des modèles de comportements d'applications en utilisant des algorithmes de clustering sur des traces qui contiennent un échantillon représentatif des applications que l'on souhaite détecter. Nous commençons par analyser cette trace pour extraire les différentes connexions. Puis, nous donnons à chacune des connexions une représentation spatiale, obtenue en utilisant seulement la taille des premiers paquets. Ensuite, nous effectuons un calibrage de notre algorithme de clustering pour trouver le nombre de paquets et de clusters qui donne le meilleur résultat. Enfin, nous utilisons notre algorithme calibré pour trouver des groupes de connexions qui ont des comportements similaires. Nous avons étudié trois algorithmes différents : K-Means, Gaussian Mixture Model et clustering spectral sur une représentation des connexions utilisant les Hidden Markov Model (HMM). A la fin de l'étape d'apprentissage, nous obtenons deux éléments qui serviront à notre outil de classification : la description de chacun des clusters et la liste des applications qui se trouvent dans chacun de ces groupes.

Notre outil de classification reçoit en entrée les entêtes de tous les paquets qui

traversent le lien connectant un réseau de bordure à l'Internet, pour les deux directions du trafic (nous avons accès aux deux directions dans la majorité des réseaux de bordure, comme illustré dans la Figure A.1). Le module d'analyse et de conversion extrait le quintuplet des paquets (protocole de transport utilisé, adresses IP source et destination, numéros de ports source et destination) et la taille du contenu application (c'est-à-dire la taille totale du paquet IP à laquelle nous soustrayons la taille des entêtes IP et TCP). Notre analyseur filtre le trafic de contrôle (les trois paquets de négociation à l'ouverture de la connexion TCP ainsi que les ACK qui ne contiennent pas de données) et stocke la taille de tous les autres paquets pour les deux directions. Quand nous avons reçu les  $P$  premiers paquets d'une connexion, ces informations sont transférées au module qui associe cette connexion à l'un des clusters en utilisant leur description. A l'aide du contenu de ce cluster, le module d'étiquetage sélectionne l'application la plus probable pour cette connexion.

Dans cette section, nous décrivons le fonctionnement de notre méthode. Après avoir montré que la taille de premiers paquets est un bon indicateur pour distinguer les applications, nous étudions différentes représentations des connexions et différents algorithmes de clustering. Ensuite, nous décrivons et comparons les heuristiques que nous avons étudiées pour associer de nouvelles connexions à un cluster et pour les étiqueter. Nous présentons ensuite une implémentation de notre outil de classification qui utilise le meilleur algorithme de clustering et les heuristiques les plus performantes. Enfin, nous décrivons une extension de notre mécanisme qui permet d'identifier les applications lorsque le trafic est chiffré avec SSL.

### A.2.1 La distinction des applications

Pour créer notre outil de classification, la première étape est de trouver une métrique qui permet de distinguer les connexions selon les applications qui les ont générées. Les méthodes de classification comportementale ont étudié de nombreuses métriques qui peuvent se regrouper en trois catégories :

- Les métriques analysant les paquets, comme par exemple le nombre de paquets dans une connexion, la taille moyenne des paquets, la variance de ces tailles, le volume de données échangées ou la distribution de la taille de paquets.
- Les métriques analysant les informations temporelles, comme par exemple la durée d'une connexion, le temps moyen d'inter-arrivées ou sa variance.
- Les métriques liées à TCP, comme le numéro de séquence initial, la taille de la fenêtre ou la taille des entêtes.

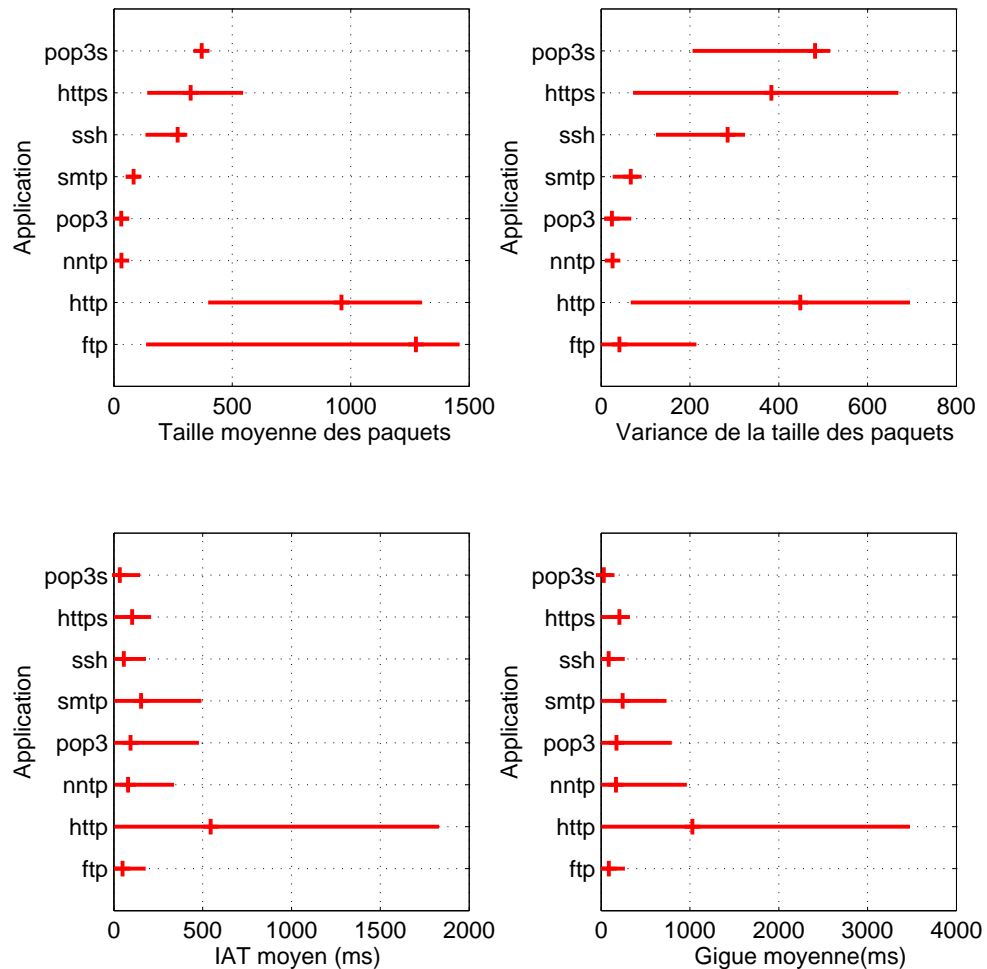


FIG. A.3 – Comparaison des métriques pour les applications les plus représentées dans les traces de Paris 6

Dans cette étude, nous avons choisi de ne pas considérer les métriques liées à TCP, car celles-ci dépendent essentiellement de l'implémentation de TCP et très peu des applications. De plus, comme nous souhaitons identifier l'application rapidement, nous devons utiliser des métriques qui peuvent être évaluées après quelques paquets. Pour choisir la meilleure métrique, nous comparons des métriques évaluées avec quatre paquets<sup>1</sup>. Nous avons considéré les métriques suivantes : taille moyenne des paquets, variance de la taille des paquets, temps d'inter-arrivées moyen, variance du temps d'inter-

<sup>1</sup>Nous présentons ces résultats avec quatre paquets puisque les chapitres suivants montrent que ce nombre est le plus efficace. Cependant, nous obtenons des résultats similaires pour d'autres nombres de paquets

arrivées. Pour comparer ces métriques, nous avons étudié les connexions de plusieurs applications dans différentes traces. La Figure A.3 présente le résultat de cette analyse pour les applications les plus représentées dans les traces de Paris 6. Les quatre graphiques correspondent aux différentes métriques étudiées. Pour chaque application, nous représentons la moyenne de la métrique considérée (centre des traits) et l'intervalle correspondant à 90% des connexions. Une métrique est efficace pour distinguer les applications si ces traits ne se superposent pas, c'est-à-dire si l'intervalle des valeurs est différent entre deux applications.

Ces graphes montrent que les métriques liées au temps sont très peu efficaces pour distinguer les applications par rapport aux métriques liées à la taille des paquets. De plus, les métriques "taille moyenne" et "variance de la taille" apportent des informations complémentaires. C'est pourquoi nous avons décidé d'utiliser directement la taille des premiers paquets et non une métrique agrégée telle que la taille moyenne. De plus, nous considérons aussi la direction du paquet (a-t-il été envoyé par le client ou le serveur de la connexion TCP ?) en utilisant une taille négative pour les paquets envoyés par le serveur. Pour vérifier que ces résultats ne dépendent pas du réseau étudié, nous avons comparé la taille des premiers paquets de différentes applications avec des traces provenant de plusieurs réseaux. Cette analyse a montré que ces tailles ne varient pas en fonction du réseau, et donc qu'un modèle développé pour un réseau donné fonctionnera sur d'autres.

## A.2.2 L'apprentissage

Dans cette sous-section, nous décrivons la méthode que nous avons utilisée pour obtenir des modèles d'applications à partir de traces TCP utilisées pour l'apprentissage. Nous commençons par expliquer comment nous avons créé de "bonnes" traces d'apprentissage. Puis, nous présentons comment nous avons représenté spatialement les connexions de ces traces. Ensuite, nous décrivons les différents algorithmes de clustering que nous avons comparés et comment calibrer ces algorithmes.

### Données d'apprentissage

Nous utilisons les traces d'apprentissage pour trouver des groupes d'applications aux comportements similaires. Si nous choisissons mal ces traces, nous risquons d'obtenir des modèles qui ne capturent pas toutes les applications ou tous les modes de fonctionnement d'une application. Pour éviter ce problème, nous choisissons un nombre similaire de connexions de chaque application. Si le nombre de connexions n'est pas

le même pour chaque application, les applications qui prédominent risquent de biaiser le clustering. Nous avons donc besoin de déterminer de manière certaine l'application qui est associée à chaque connexion dans nos traces. Nous avons utilisé deux méthodes pour créer nos jeux de données d'apprentissage. La première méthode consiste à extraire des connexions des applications que nous souhaitons étudier à partir de traces de trafic réel. Pour déterminer l'application associée à chacune des connexions, nous avons utilisé un outil de classification analysant le contenu des paquets : Traffic Designer de Qosmos [62]. L'avantage de l'extraction est de contenir un bon échantillon des applications utilisées sur un réseau, mais cette méthode nécessite d'avoir accès au contenu des paquets. La seconde méthode que nous avons utilisée consiste à générer manuellement du trafic de certaines applications. L'avantage de cette méthode est que nous sommes certains de l'application qui a généré les connexions. En revanche, cette méthode prend beaucoup de temps et nous n'avons pas la garantie de capturer tous les modes de fonctionnement d'une application.

Dans les traces de Paris 6 que nous avons étudiées se trouvent un grand nombre d'applications. En revanche, pour la plupart, nous n'avons qu'un très faible nombre de connexions. Après analyse, le meilleur compromis entre le nombre d'applications modélisées et le nombre de connexions disponibles pour créer les modèles est d'utiliser les applications pour lesquelles nous avons 300 connexions ou plus. Pour vérifier que nos résultats ne dépendent pas du jeu de données d'apprentissage, nous avons, de plus, créé 50 jeux d'apprentissage différents. Par la suite, pour tous les résultats, nous présenterons la moyenne sur ces 50 jeux de données ainsi que la variance de ces résultats.

### Représentation des connexions

Nous avons montré précédemment que la taille des premiers paquets d'une connexion permet de distinguer les applications. Pour modéliser les applications, nous allons les regrouper en fonction de ces tailles. Afin d'effectuer ce clustering nous devons donner une représentation aux connexions qui permette de mesurer leurs similarités. Nous avons utilisé deux méthodes pour représenter les connexions.

La méthode la plus naturelle pour représenter les  $P$  premiers paquets d'une connexion consiste à lui associer un point dans un espace euclidien à  $P$  dimensions. Nous avons donc associé les connexions de notre jeu de données d'apprentissage à des points. Chacune des coordonnées représente la taille du contenu applicatif d'un paquet d'une connexion. De plus pour prendre en considération le sens, nous utilisons des coordonnées positives pour les paquets envoyés par le client de la connexion TCP et négatives pour

les paquets du serveur. Pour mesurer la distance entre deux connexions, nous utilisons alors simplement la distance euclidienne entre les deux points qui les représentent.

Cette méthode de représentation est simple, mais elle ne prend pas en compte l'information de séquence : si l'on permute les dimensions associées aux paquets 1 et 2, cela n'aura pas d'influence sur le clustering. Pour tenir compte de la séquence nous nous sommes intéressés à une représentation plus complexe des connexions utilisant les chaînes de Markov cachées (HMM), en utilisant la méthode proposée par [5]. Nous avons modélisé chacune des connexions par une HMM de  $P$  états, où chaque état correspond à l'un des paquets. Une fois que nous avons associé une HMM à chacune des connexions, nous pouvons évaluer la similarité entre deux connexions en utilisant les probabilités que chacune des séquences de paquets ait été générée par la HMM associée à l'autre connexion.

### Algorithmes de clustering

En fonction de la représentation choisie, nous devons utiliser des algorithmes de clustering différents.

Un algorithme de clustering souvent utilisé dans les espaces euclidiens est K-Means. Cet algorithme trouve les  $K$  clusters qui minimisent la somme des distances entre chaque point et le centroïde le plus proche. Cet algorithme fonctionne de manière itérative. On commence par placer les centroïdes de façon aléatoire, et on associe chaque point au centroïde le plus proche. On calcule les nouveaux centroïdes des clusters ainsi formés et on recommence jusqu'à ce que l'on trouve un minimum pour la somme des distances. Cet algorithme est simple et efficace. En revanche il possède plusieurs défauts. Il se peut tout d'abord que l'algorithme converge vers un optimum local. Pour résoudre ce problème, on utilise généralement plusieurs fois l'algorithme et on garde le meilleur résultat. Un second défaut est que les clusters obtenus avec K-Means sont toujours de forme sphérique. Si les points de chaque groupe ne sont pas répartis de cette manière les résultats d'un clustering avec K-Means peuvent ne pas être bons. De plus, K-Means ne prend pas en compte la dispersion des points à l'intérieur d'un cluster. Ce défaut peut être un véritable problème lorsque l'on souhaite associer de nouveaux points à des clusters que l'on a identifiés avec K-Means (le cluster le plus proche n'est pas nécessairement le meilleur).

Pour éviter les défauts de K-Means, nous avons utilisé une autre méthode qui consiste à modéliser les points de l'espace euclidien avec un mélange de gaussiennes (GMM). Chacune des gaussiennes du mélange correspond alors à un cluster. Un algorithme de clustering utilisant les GMM trouve les paramètres (c'est-à-dire les centres

et les matrices de covariance) des  $K$  gaussiennes qui décrivent le mieux les points considérés. L'avantage de cette méthode est que les clusters que l'on identifie n'ont plus de simples formes sphériques, mais peuvent avoir des formes ellipsoïdales. De plus, si l'on veut associer un nouveau point à l'un des clusters existants, on peut évaluer la probabilité qu'il appartienne à chacune des gaussiennes et choisir le cluster pour lequel cette probabilité est maximale.

Pour la représentation utilisant les HMM, nous obtenons une matrice qui donne la similarité entre toutes les connexions du jeu de données d'apprentissage. Une méthode de clustering couramment utilisée lorsque l'on a ce type de matrice est le clustering spectral. Le principe de fonctionnement de cet algorithme est de considérer que la matrice de similarités est une matrice par bloc dans laquelle chaque bloc correspond à un cluster. Pour identifier les blocs, il suffit alors de diagonaliser la matrice. Nous pouvons ensuite associer chacun des points à l'un des  $K$  premiers sous-espaces propres (qui correspondent aux  $K$  valeurs propres les plus importantes).

## Calibration

Nous avons présenté plusieurs méthodes de clustering. Le résultat de ces algorithmes dépend de deux paramètres importants : le nombre de paquets de la connexion que l'on considère et le nombre de clusters que l'on choisit. Nous allons maintenant montrer comment choisir ces deux paramètres.

Nous définissons un clustering optimal comme un clustering pour lequel chaque cluster correspond exactement à une application (nous avons identifié de façon certaine l'application associée à chacune des connexions d'apprentissage à l'aide d'un outil de classification par contenu). Nous avons choisi de mesurer la qualité d'un clustering en le comparant à ce clustering optimal. Pour effectuer cette comparaison, nous utilisons un indicateur souvent utilisé dans les travaux de clustering : l'information mutuelle normalisée (NMI) [75]. Nous mesurons l'information mutuelle entre la distribution des clusters et la distribution des noms d'application pour l'ensemble des connexions, puis nous normalisons le résultat qui est donc compris entre 0 et 1. Lorsque le NMI vaut 1, cela signifie que l'association entre les clusters et les applications est parfaite et que nous avons le clustering optimal. Cependant, certaines applications peuvent avoir plusieurs comportements (par exemple FTP contient des connexions de contrôle et de transfert de données) et certaines applications peuvent avoir le même comportement. C'est pourquoi le NMI ne peut atteindre 1. En revanche, plus le NMI est élevé, meilleur est le clustering. Nous avons utilisé cet indicateur pour calibrer nos algorithmes de clustering.

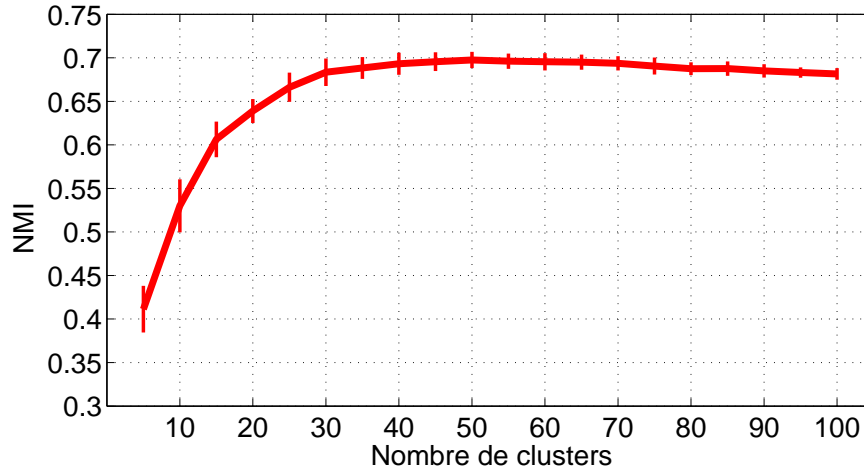


FIG. A.4 – Influence du nombre de clusters sur la qualité du clustering (4 paquets, K-Means)

Nous utilisons le NMI pour déterminer le nombre de clusters optimal pour un nombre donné de paquets en faisant varier le nombre de clusters de 5 à 100. La Figure A.4 présente le résultat de cette étude pour l’algorithme K-Means lorsque l’on considère les quatre premiers paquets des connexions. Nous évaluons le NMI pour nos 50 jeux de données différents et la figure donne la moyenne avec deux écarts types (les barres verticales). Cette figure montre que le meilleur nombre de clusters est 50. Nous observons une augmentation rapide du NMI lorsque le nombre de clusters passe de 10 à 30, tandis qu’à partir de 50 clusters, le NMI n’augmente plus. Ceci indique qu’avoir plus de 50 clusters n’améliore pas le clustering. De plus, nous souhaitons avoir un nombre de cluster minimal pour simplifier la procédure de classification. C’est pourquoi nous cherchons le “coude” dans cette courbe, puisque qu’il représente le meilleur compromis entre la qualité du clustering et la complexité. En utilisant cette méthode, le meilleur nombre de clusters pour quatre paquets avec l’algorithme K-Means est 30. Nous effectuons un calibrage similaire pour d’autres nombres paquets (de 1 à 10), et nous faisons aussi cela pour les autres algorithmes.

Il existe plusieurs méthodes pour évaluer la qualité d’un clustering. Ces méthodes peuvent être regroupées en deux catégories. Tout d’abord, les méthodes qui utilisent des critères intrinsèques au clustering. Ces méthodes mesurent si un clustering aboutit à des clusters compacts et bien séparés. Ensuite, les méthodes qui utilisent des critères externes en mesurant la similarité entre un clustering et une partition de référence. Dans notre cas, nous n’effectuons pas un clustering à l’aveugle : nous voulons que nos clusters séparent les applications. C’est pourquoi nous avons choisi une méthode uti-

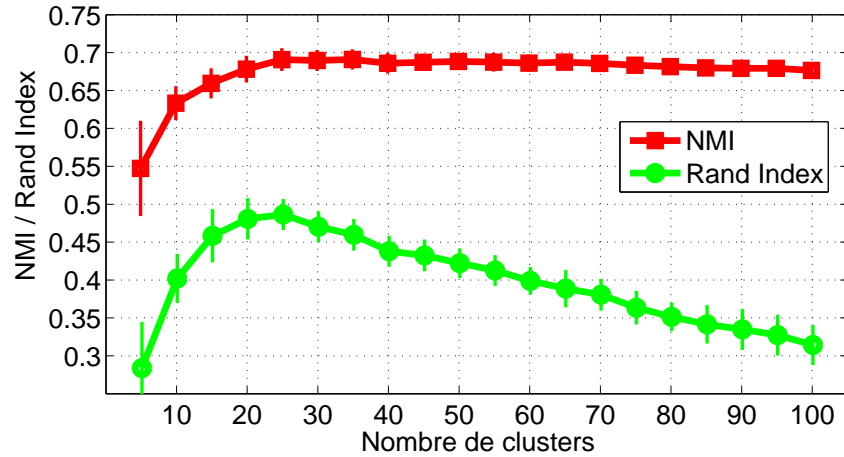


FIG. A.5 – Comparaison entre NMI et Rand Index (2 paquets, GMM)

lisant un critère externe et mesurant la similarité entre le clustering et la partition définie par les étiquettes d'application. Un certain nombre de méthodes entrent dans cette catégorie. Nous en avons évalué plusieurs, en particulier NMI et Rand Index. Nous avons observé que ces méthodes aboutissent à des résultats similaires, comme illustré par la Figure A.5. C'est pourquoi nous avons choisi d'utiliser NMI pour notre étape de calibration. La Figure A.5 présente le choix du nombre de clusters pour la méthode GMM avec deux paquets. Nous voyons que le nombre optimal de cluster avec les deux méthodes est 25.

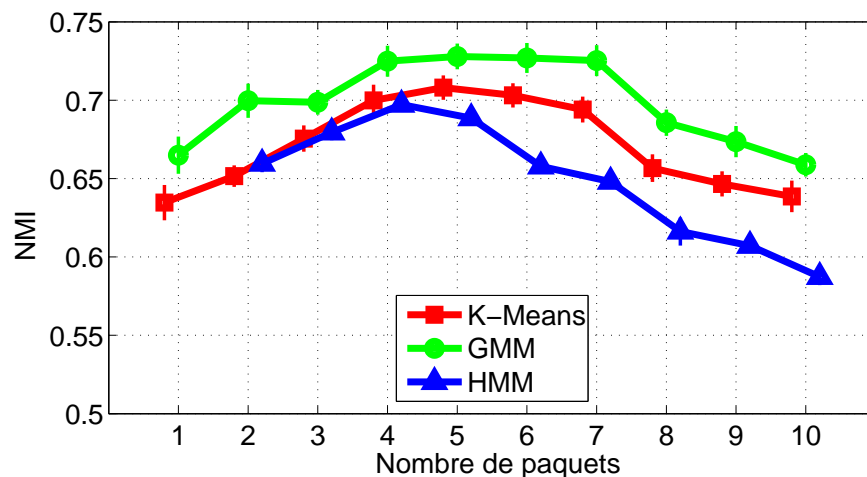


FIG. A.6 – Influence du nombre de paquets sur la qualité du clustering

Pour déterminer le meilleur nombre de paquets, nous comparons les NMI obtenus

pour différents nombres de paquets (en utilisant chaque fois le nombre de clusters optimal pour un nombre de paquets donné). La Figure A.6 montre cette comparaison pour nos trois algorithmes de clustering. Là encore nous représentons la moyenne pour nos 50 jeux de données d'apprentissage, ainsi que deux écarts types. Nous constatons que le NMI décroît pour un grand nombre de paquets. Ceci peut s'expliquer par le fait qu'ajouter de nouveaux paquets ajoute du bruit : les paquets échangés après la négociation ne sont plus standards et leur taille n'aide plus à séparer les applications. Nous souhaitons considérer le moins de paquets possible pour identifier les applications le plus tôt possible. Le meilleur compromis entre la qualité du clustering et le nombre de paquets d'après ces courbes est donc  $P = 4$  paquets.

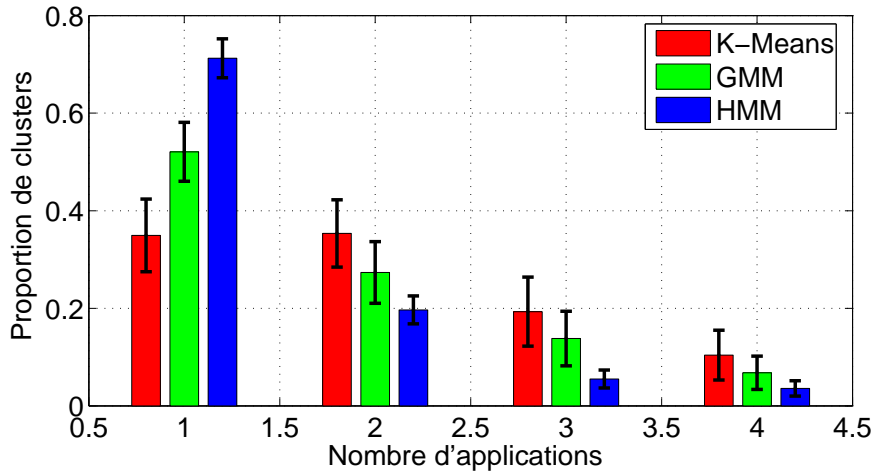


FIG. A.7 – Proportion de clusters avec N Applications (4 paquets)

Enfin, pour évaluer l'efficacité de notre méthode, nous étudions la composition des clusters obtenus. La Figure A.7 montre la proportion de chaque cluster qui contient une, deux, trois ou quatre applications (aucun cluster n'en contient plus). Les barres représentent la moyenne de ces proportions sur nos 50 jeux de données, et les traits supplémentaires, deux écarts types. Dans le cas d'un clustering parfait 100% des clusters contiendraient une seule application. HMM est la méthode la plus proche de cet objectif grâce à sa représentation plus riche des connexions. De plus, GMM donne de meilleurs résultats que K-Means grâce aux formes plus complexes de ses clusters.

### A.2.3 La classification temps réel

Nous utilisons les résultats du clustering pour créer notre outil de classification temps réel. Celui-ci capture les  $P$  premiers paquets de chaque connexion sur un lien et

identifie l'application en utilisant la taille de ces paquets. Cette identification repose sur deux heuristiques. La première associe une nouvelle connexion avec l'un des clusters que l'on a identifiés, tandis que la seconde étiquette la connexion en fonction des applications qui se trouvent dans ces clusters.

### Heuristiques d'association

La méthode utilisée pour associer une nouvelle connexion avec un cluster dépend de l'algorithme utilisé.

Dans le cas de K-Means, la méthode la plus simple, que nous nommons **K-Means Center**, associe une nouvelle connexion au cluster dont le centroïde est le plus proche. Cette méthode est simple, mais elle ne prend pas en compte la dispersion des clusters : certains sont très concentrés alors que d'autres sont plus étalés. Nous avons analysé la distribution de la distance au centroïde pour chaque connexion de nos clusters, et nous avons observé que cette distribution est proche de la distribution normale. Pour prendre en compte cette information, nous modélisons alors chaque cluster par son centre et la variance de ces distances au centre. Ceci nous permet d'évaluer la probabilité qu'une nouvelle connexion appartienne à chacun des clusters. Nous introduisons donc une nouvelle heuristique **K-Means Proba** qui associe une nouvelle connexion au cluster pour lequel cette probabilité est la plus élevée. Ces deux heuristiques associent toute nouvelle connexion avec un des clusters. Or, lorsque la probabilité d'appartenir à tous les clusters est faible, cela peut signifier que la connexion considérée correspond à un type de trafic que l'on connaît pas. C'est pourquoi nous introduisons une nouvelle heuristique **K-Means Thresh** qui utilise un seuil : lorsque la probabilité d'appartenir à chacun des clusters est plus faible que ce seuil, on étiquette la connexion "inconnu". Pour déterminer la valeur de ce seuil, nous appliquons notre heuristique d'association aux données d'apprentissage avec différents seuils et nous étudions la proportion de trafic inconnu. Nous choisissons alors comme seuil la valeur qui correspond au coude dans la courbe obtenue : le meilleur compromis entre la proportion de trafic étiqueté "inconnu" et la précision.

Avec le modèle GMM, nous pouvons directement évaluer la probabilité qu'une connexion appartienne à chacune des gaussiennes du mélange. Nous définissons donc l'heuristique **GMM Proba** qui associe une connexion au cluster le plus probable. De la même manière que pour K-Means, nous introduisons aussi l'heuristique **GMM Thresh** qui vérifie que cette probabilité dépasse un seuil (que nous déterminons de façon similaire).

Pour les modèles à base de HMM, nous commençons par modéliser les connexions de

chaque cluster avec une HMM unique en utilisant l’algorithme “Expectation-Maximization”. Ensuite, nous pouvons évaluer la probabilité qu’une nouvelle connexion ait été générée par chacune des HMM qui représente un cluster. Nous définissons alors l’heuristique **HMM Likelihood** qui associe une connexion au cluster pour lequel cette probabilité est maximale. Comme pour K-Means et GMM, nous introduisons aussi l’heuristique **HMM Thresh**.

### Heuristiques d’étiquetage

Lorsque nous avons associé une connexion avec l’un des clusters, l’étape suivante consiste à étiqueter la connexion. Pour cela nous proposons deux heuristiques.

L’heuristique **Predominant** est très simple : elle étiquette une connexion avec l’application qui prédomine dans le cluster auquel elle a été associée. Nous avons vu grâce à la Figure A.7 que la majorité des clusters contient une seule application, donc cette heuristique pourra étiqueter précisément la plupart des connexions. Cette méthode est simple mais n’étiquettera pas bien les connexions appartenant aux applications qui sont minoritaires dans leur cluster.

Pour traiter ce problème nous proposons d’améliorer notre étiquetage en prenant en compte d’autres informations disponibles dans les premiers paquets. Les numéros de port TCP ne sont pas suffisants pour identifier les applications, mais apportent quand même une information, en particulier lorsqu’ils correspondent à des services standards. Nous introduisons donc une heuristique hybride **Cluster+Port** qui utilise les ports quand ceux-ci peuvent aider. Si une connexion utilise un port standard et que l’application associée se trouve dans le cluster auquel elle a été associée, on l’étiquette avec cette application. Dans le cas contraire, la connexion utilise un port standard mais ne se comporte pas en conséquence. Nous choisissons d’étiqueter ces connexions “masquerade” parce-qu’elles correspondent potentiellement à des connexions dangereuses qui utilisent un port standard pour éviter la détection ou les règles de firewall. Dans le cas où le port utilisé n’est pas standard, nous l’étiquetons avec l’application qui prédomine parmi celles du cluster qui n’utilisent pas les ports standards. Si aucune application de ce type n’est présente dans le cluster, nous l’étiquetons “masquerade” parce-qu’elle correspond probablement à une application standard utilisant un port non standard.

### Évaluation des heuristiques

Nous allons maintenant évaluer les différentes heuristiques que nous avons proposées en utilisant plusieurs traces de test. Nous commençons par évaluer la précision

des heuristiques d'association. Nous définissons cette précision comme la proportion de connexions des traces de test qui sont associées à des clusters contenant leur véritable application. La Table A.1 présente ces résultats pour l'algorithme K-Means. Nous pouvons remarquer que nous obtenons les meilleurs résultats quand nous utilisons un seuil pour associer les connexions. Les résultats obtenus avec l'heuristique K-Means Proba sont aussi très bons mais légèrement plus faibles. En revanche l'heuristique K-Means Center donne des résultats significativement moins bons. Nous avons fait la même étude pour GMM et HMM et nous avons obtenu des résultats similaires.

Heuristic	Center	Proba	Thresh
	Précision	Précision	Précision (Inconnu)
P6-1	95.6%	98.8%	99.7% (37.9%)
P6-2	95.8%	98.7%	99.7% (37.8%)
P6-3	95.8%	97.7%	98.8% (38.4%)
Entreprise	95.5%	97.9%	99.3% (31.8%)
M2C College	97.3%	99.7%	99.9% (41.3%)
M2C ADSL	68.9%	71.3%	65.5% (35.7%)
Crowdad	96.8%	99.7%	99.8% (41.8%)
UMass	94.6%	95.1%	98.6% (47.6%)

TAB. A.1 – Précision de l'association et proportion de trafic inconnu pour K-Means

Pour mesurer la qualité de nos heuristiques d'étiquetage, nous mesurons la précision globale de nos heuristiques. Celle-ci correspond à la proportion de connexions qui obtiennent la bonne étiquette d'application. La Figure A.8 représente cette précision globale pour les trois traces de Paris 6 et la trace d'entreprise pour différents nombres de paquets. Les points représentent la moyenne de la précision globale pour les 50 modèles obtenus avec nos 50 jeux de données d'apprentissage, et les barres verticales deux écarts types. Ces courbes nous permettent de vérifier que le nombre de paquets que nous avons choisi lors du calibrage fonctionne en pratique. Pour les modèles K-Means et HMM le meilleur résultat est obtenu avec quatre paquets, ce qui est cohérent avec les résultats du calibrage. En revanche, pour GMM, nous obtenons le meilleur résultat, 92%, avec deux paquets alors que nous la précision est de 89% pour quatre paquets. Les résultats obtenus avec GMM sont très bons avec deux paquets parce que la forme ellipsoïdale des clusters de GMM capture parfaitement les deux premiers paquets des connexions HTTP qui représentent plus de 50% des connexions de nos traces de test.

La précision globale dépend des proportions de chaque application dans les données de test : si 90% des connexions correspondent à une application, la précision globale mesurera la précision avec laquelle nous reconnaissons cette application. Pour contour-

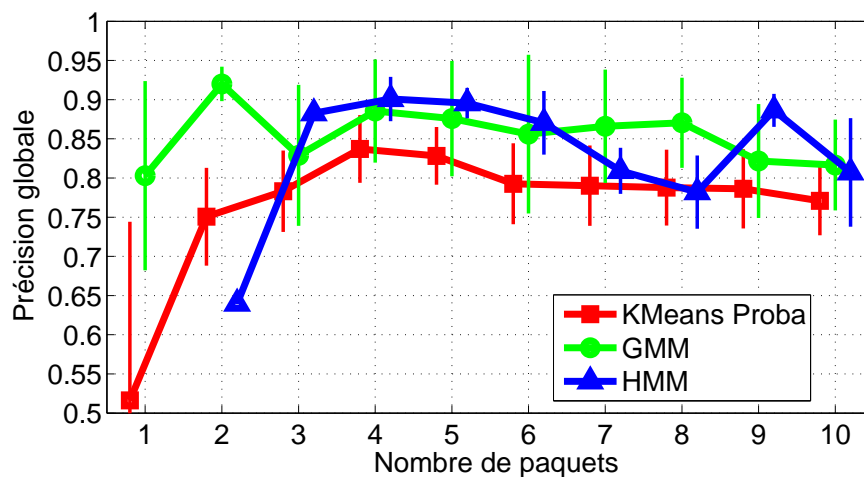


FIG. A.8 – Précision globale (Predominant)

ner ce problème, nous mesurons la précision moyenne sur l'ensemble des applications (c'est-à-dire la moyenne des précisions avec laquelle nous reconnaissons chaque application dans les traces de test). La Figure A.9 représente cette précision moyenne pour les même traces, et nous voyons que les modèles utilisant quatre paquets donnent de meilleurs résultats que les modèles avec deux paquets.

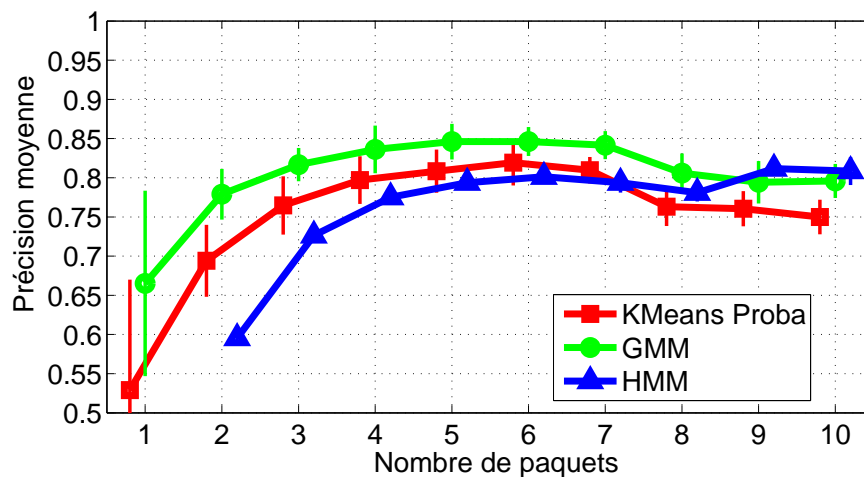


FIG. A.9 – Précision moyenne (Predominant)

La Table A.2 compare la précision globale pour nos deux heuristiques d'étiquetage pour deux traces. Nous pouvons voir que les résultats obtenus avec l'heuristique Cluster+Port sont meilleurs que les résultats obtenus avec la méthode Predominant. De plus HMM et GMM donnent de meilleurs résultats que K-Means.

Heuristic	Trace	K-Means	GMM	HMM
		Proba	Proba	Likelihood
Predominant	P6-1	92.4%	93.5%	92.4%
Predominant	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%

TAB. A.2 – Précision des heuristiques d'étiquetage (4 paquets)

Pour évaluer l'efficacité des seuils que nous avons introduits précédemment, nous avons généré des traces contenant des applications qui n'étaient pas présentes dans la phase d'apprentissage. Nous avons seulement étudié cela pour GMM et HMM puisque ces deux modèles sont plus performants que K-Means et nous avons obtenu les résultats suivants. GMM et HMM sont capables de détecter les deux tiers des connexions des nouvelles applications (mais cette proportion varie en fonction des applications). Cette identification à un coût : 30% des connexions appartenant à des applications présentes lors de l'apprentissage sont étiquetées "inconnu" pour GMM (et 50% pour HMM).

Les résultats de cette évaluation montrent que GMM et HMM donnent des résultats similaires. Étant donné que la méthode GMM est plus simple, celle-ci représente le meilleur compromis entre précision et complexité. C'est pourquoi l'implémentation que nous présenterons dans la sous-section suivante utilise la méthode GMM.

#### A.2.4 Implémentation de notre méthode

Afin d'évaluer les performances de notre outil de classification sur des traces de trafic réel, nous avons implémenté notre méthode en C. Nous avons effectué nos tests sur un opteron à 2.4GHz avec une trace de trafic d'une heure, collectée sur le lien qui connecte notre université à Renater. Notre programme est capable de classer l'ensemble de cette trace, qui correspond à 35.5GB de trafic, en 46s CPU (le temps réel est plus long car la vitesse de lecture sur le disque est un facteur limitant). Ceci signifie que notre outil est capable d'analyser le trafic à un débit de 6Gbit/s, ce qui est beaucoup plus rapide que les méthodes utilisant le contenu des paquets. En effet, des études présentées dans [82] et [20] analysent deux outils répandus de classification par le contenu, bro [58] et 17-filter [44], et montrent que ces outils peuvent analyser le trafic à des débits inférieurs à 500Mbit/s (sur des machines semblables à celles que nous avons utilisées). De plus, notre implémentation est assez rapide pour les réseaux de bordure puisque la plupart de ces réseaux utilisent des liens à 1Gbit/s (Ethernet) ou 2.5Gbit/s (OC-48).

Nous avons également étudié le profil de notre implémentation, afin de déterminer

Gestion de la liste des connexions	55%
Analyse des paquets	35%
Association et étiquetage	10%

TAB. A.3 – Temps passé

les tâches qui prenaient le plus de temps. La Table A.3 présente la proportion du temps que notre programme passe à gérer la liste des connexions actives (recherche, ajout, suppression), à analyser les paquets (analyse des entêtes, création des nouvelles connexions, stockage de la taille des paquets) et à associer et étiqueter les connexions avec notre méthode. Nous voyons que la gestion de la liste des connexions et l'analyse des paquets représentent 90% du temps de traitement. Or ces tâches doivent être accomplies par tous les outils de classification. La part du traitement que représente notre méthode est limitée à 10%. Ce résultat indique que notre outil de classification pourrait être ajouté à tout outil analysant les connexions TCP à un coût minime. De plus notre programme effectue des tâches très simples, donc une implémentation matérielle ou utilisant des processeurs réseaux permettrait d'atteindre relativement facilement des débits encore plus élevés.

### A.2.5 La classification du trafic chiffré

Dans la partie précédente, nous avons montré comment identifier l'application associée à une connexion TCP. Nous avons montré que cet outil de classification temps réel est beaucoup plus rapide que les techniques analysant le contenu des paquets. En revanche celles-ci peuvent être plus précises que notre outil puisqu'elles recherchent des signatures qui identifient les applications de façon unique. La principale limitation de ces méthodes (outre la complexité des traitements qu'elles induisent) est qu'il est aisé de les empêcher de fonctionner en chiffrant le contenu des paquets. De plus, le mécanisme Secure Sockets Layer (SSL), qui peut être mis en œuvre pour chiffrer les connexions, est très répandu et facile à utiliser. Dans cette partie, nous décrivons une extension de notre outil de classification qui permet d'identifier les applications associées à des connexions chiffrées.

#### Identification du trafic chiffré avec SSL

Notre outil de classification du trafic chiffré s'appuie sur l'observation suivante : SSL ne modifie pas de manière significative le nombre de paquets, leur taille et le temps d'inter-arrivées [71, 34]. Pour appliquer notre méthode de classification, nous

devons traiter deux problèmes : nous devons identifier les paquets qui transportent les données chiffrées (avant de transmettre les données, SSL commence par une phase de négociation pour identifier les machines en communication et négocier une clé de chiffrement), et nous devons estimer la taille d'origine de ces paquets.

Il existe trois versions de SSL différentes : SSLv2, SSLv3.0 et SSLv3.1 (ou TLS). SSLv3.0 et SSLv3.1 fonctionnent de manière similaire, contrairement à SSLv2 qui est très différent. Selon la version utilisée, selon le déroulement de la négociation et selon les implémentations de SSL la durée de la négociation de SSL peut être très différente. La figure A.10 illustre cette variabilité pour SSLv3.x pour des connexions du réseau de Paris 6. Pour déterminer la position du premier paquet contenant des données, nous avons étudié les types des "records" SSL présents dans les entêtes SSL. Ce graphique montre que le nombre de paquets utilisés pour la négociation varie entre 3 et 12. Notre outil de classification devra donc analyser les entêtes SSL pour déterminer quel est le premier paquet qui contient des données chiffrées.

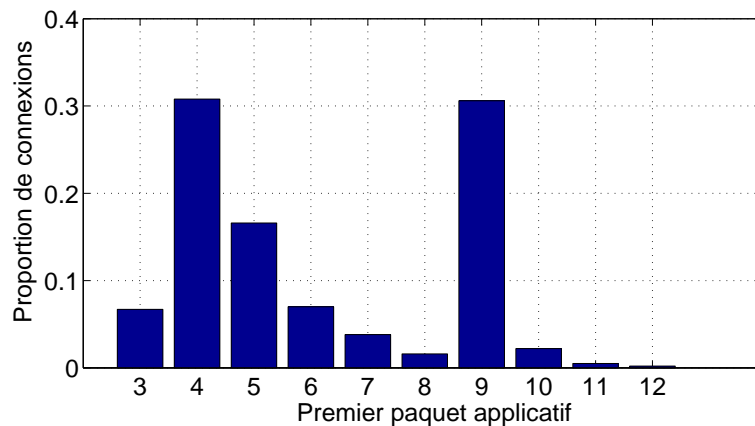


FIG. A.10 – Position du premier paquet avec des données chiffrées

La seconde étape importante pour notre classification consiste à estimer la taille d'origine des paquets chiffrés. SSL peut utiliser un grand nombre d'algorithmes de chiffrement qui vont altérer la taille des paquets de façons différentes. En étudiant le trafic de Paris 6, nous avons cependant remarqué que les cinq algorithmes de chiffrement les plus utilisés représentent 99% du trafic SSL. Nous avons donc étudié l'influence du chiffrement sur la taille des paquets pour ces algorithmes. Pour cela, nous avons développé une application simple qui envoie des paquets contenant 1, 2, ..., 100 octets en utilisant une connexion SSL, et nous avons observé la taille des paquets chiffrés. La Figure A.11 présente les résultats de cette analyse. Cette figure montre que les algorithmes de chif-

frement considérés augmentent tous légèrement la taille des paquets, et ce de façons différentes selon l'algorithme (en particulier selon qu'il s'agit d'un chiffrement par bloc ou pas). Cependant, cette figure montre aussi que la variation est faible et prévisible. De plus, pour ces algorithmes les variations sont très similaires. C'est pourquoi nous avons décidé d'utiliser une heuristique simple pour estimer la taille d'origine des paquets chiffrés : nous soustrayons 21, quelque soit l'algorithme (d'après notre étude les variations sont comprises entre 21 et 37 octets).

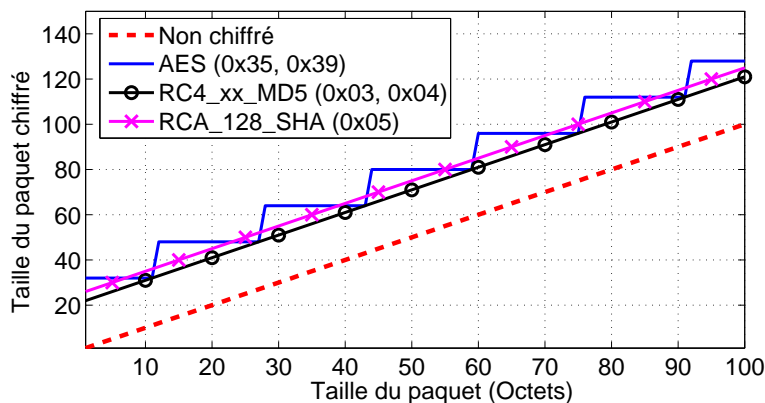


FIG. A.11 – Taille des paquets chiffrés en fonction de l'algorithme de chiffrement

La Figure A.12 décrit le fonctionnement de notre outil de classification. Cet outil reçoit les paquets d'une connexion TCP et identifie l'application qui lui est associée, y compris lorsque celle-ci est chiffrée avec SSL. Cette classification se déroule en trois étapes. Tout d'abord nous utilisons notre outil de classification temps réel pour reconnaître les connexions utilisant SSL. Pour ces connexions, nous détectons ensuite le premier paquet contenant des données chiffrées. Enfin, nous estimons la taille d'origine de ces paquets pour identifier l'application encapsulée à l'aide de notre outil de classification temps réel.

Pour détecter les connexions SSL, nous utilisons une version légèrement modifiée de notre outil de classification temps réel. Tout d'abord, nous incluons des connexions SSLv2 et SSLv3 dans les données d'apprentissage. Ensuite, nous effectuons l'apprentissage en utilisant la méthode GMM. En calibrant notre algorithme nous avons trouvé que les nombres optimaux de paquets et de clusters étaient respectivement de 4 et 40. Cependant, nous obtenons des résultats similaires en utilisant seulement 3 paquets et 35 clusters. Étant donné que nous devons inspecter le contenu des paquets à partir du troisième pour identifier le premier paquet transportant des données chiffrées, nous

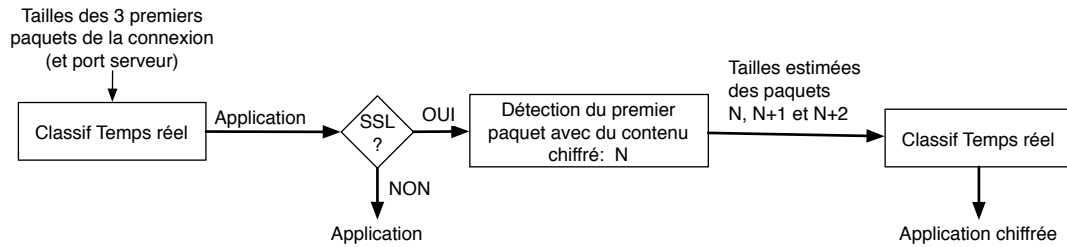


FIG. A.12 – Vue d'ensemble

décidons d'utiliser trois paquets pour la première phase de classification. Ceci nous permet de ne pas stocker le contenu du troisième paquet de chaque connexion et donc de limiter l'utilisation de la mémoire. Lorsque nous détectons une connexion SSL, nous cherchons ensuite le premier paquet avec données chiffrées en analysant les entêtes SSL. Enfin, nous utilisons les tailles estimées des trois premiers paquets de données chiffrées pour identifier l'application transportée en appliquant de nouveau notre outil de classification temps réel. Pour cela, nous utilisons une version légèrement modifiée de l'heuristique Cluster+Port pour prendre en compte les ports standards associés au trafic SSL : nous utilisons 443 pour HTTPS, 993 pour IMAPS et 995 pour POP3S.

## Évaluation

Nous évaluons notre nouvel outil de classification en deux étapes. Tout d'abord, nous vérifions que nous pouvons détecter les connexions SSL. Ensuite, nous testons notre méthode pour détecter les applications chiffrées avec SSL.

Nous commençons par mesurer l'efficacité de notre outil de classification temps réel utilisant trois paquets. Pour cela nous prenons l'ensemble des connexions présentes dans une trace d'une heure capturée sur le réseau de Paris 6, puis nous mesurons la précision globale obtenue avec les heuristique Predominant et Cluster+Port. Nous obtenons alors une précision globale supérieure à 90% pour les deux heuristiques. Nous identifions correctement 68% des connexions SSLv3, 82% des connexions SSLv2 et plus de 85% des connexions des autres applications avec l'heuristique Predominant. En utilisant Cluster+Port, nous obtenons de bien meilleurs résultats : 81% pour le trafic SSLv3, 95% pour SSLv2 et plus de 95% pour les autres applications. Ces bons résultats montrent que notre outil de classification temps réel peut détecter les applications SSL en utilisant seulement la taille des trois premiers paquets contenant des données.

Pour évaluer l'identification des applications encapsulées dans SSL, nous avons

utilisé deux jeux de données différents. La Table A.4 présente les résultats pour des connexions SSL observées sur le réseau de Paris 6. Pour déterminer de façon certaine l'application chiffrée dans ces connexions, nous avons filtré le trafic en nous limitant aux connexions destinées à des serveurs de l'université fournissant des services que nous connaissions (HTTPS ou POP3S). Cette table montre que notre outil peut étiqueter correctement plus de 98% du trafic des applications qui utilisent fréquemment SSL (HTTP et POP3).

Applications	Predominant	Cluster+Port
http	99.95%	99.95%
pop3	98.45%	98.45%

TAB. A.4 – Détection des applications chiffrées dans la trace Paris6-3

Pour vérifier l'efficacité de notre méthode pour d'autres types de trafic, nous avons manuellement chiffré des connexions FTP, Bittorent et Edonkey, et nous avons testé notre méthode sur ces connexions. La Table A.5 présente ces résultats et montre que pour ce type de trafic, nous pouvons identifier correctement l'application qui est chiffrée dans plus de 85% des cas.

Applications	Predominant	Cluster+Port
ftp	90.58%	92.67%
bittorent	77.87%	86.48%
edonkey	94.56%	96.57%

TAB. A.5 – Détection des applications pour le trafic chiffré manuellement

## A.3 Conclusion

L'identification de l'application qui a généré un flux est une tâche essentielle pour les administrateurs réseau. Aujourd'hui, dans la plupart des réseaux opérationnels, cette identification se fonde sur les numéros de port TCP (ou UDP). Cependant, le développement rapide de nouvelles applications qui font de la négociation dynamique de ports ou utilisent des numéros de port non standards (et qui tentent même parfois de se cacher) a beaucoup réduit l'efficacité de cette technique. Les deux méthodes les plus efficaces aujourd'hui utilisent des techniques qui analysent le contenu des paquets à la recherche de signatures applicatives spécifiques et les techniques qui se fondent sur le comportement "extérieur" des flux en utilisant des métriques telles que la taille des paquets ou le temps inter-paquet pour identifier les applications. Les méthodes qui analysent le contenu des paquets sont très performantes mais ne peuvent pas être utilisées sur des liens à haut débit puisqu'elles nécessitent beaucoup de traitements. De plus, elles ne peuvent fonctionner lorsque le trafic est chiffré. Les méthodes comportementales n'ont pas ces défauts et sont pleines de promesses. Cependant, jusqu'ici, ces techniques utilisent des statistiques qui ne peuvent être évaluées que lorsque les flux sont terminés (volume total transféré, nombre total de paquet ou durée, par exemple). De ce fait, ces techniques ne peuvent classer le trafic en temps réel.

Dans cette thèse, nous proposons une méthode comportementale qui s'appuie seulement sur des statistiques que l'on peut évaluer au début de la connexion, ce qui permet une classification en temps réel. Après avoir décrit les contributions principales de cette thèse dans la Sous-section A.3.1, nous présentons les perspectives de recherche qui pourraient étendre ces travaux dans la Sous-section A.3.2.

### A.3.1 Contributions

Cette thèse apporte les contributions suivantes.

**Nous avons montré que la taille des quatre premiers paquets est suffisante pour distinguer les applications.** Nous avons comparé plusieurs métriques évaluées sur les premiers paquets d'une connexion TCP. Les métriques relatives au temps ne permettent pas de différencier les applications. En revanche, la taille des premiers paquets permet cette distinction. Pour les données que nous avons étudiées, le meilleur nombre de paquets à considérer est quatre. De plus, nous avons aussi montré que la taille des premiers paquets d'une application ne varie pas entre les réseaux, et donc qu'un outil de classification développé pour un réseau donné fonctionnera sur les

autres.

**Nous avons montré que nous obtenons le meilleur compromis entre précision et complexité avec les GMM.** Nous avons évalué trois méthodes de clustering différentes pour regrouper les connexions selon leur comportement : K-Means et GMM dans un espace euclidien et le clustering spectral dans un espace obtenu en modélisant les connexions avec des HMM. La représentation utilisant les HMM est la plus riche, mais nous obtenons des résultats comparables en utilisant des modèles plus simples à base de GMM. Ces résultats ne varient pas lorsque l'on change les connexions utilisées pour l'apprentissage.

**Nous avons développé un outil de classification temps réel qui identifie correctement 98% des connexions.** Nous avons utilisé les résultats du clustering : les clusters et leur composition (c'est-à-dire les applications associées aux connexions du jeu de données d'apprentissage qui se trouvent dans chaque cluster) pour définir des heuristiques d'association et d'étiquetage. Le clustering utilisant la méthode GMM associé aux numéros de ports TCP permet de classer correctement 98% des connexions générées par les applications que nous avons étudiées, et cela pour toutes les traces que nous avons testées. De plus, les outils de classification reposant sur les HMM ou les GMM permettent d'étiqueter "inconnu" 60% des connexions générées par des applications qui n'étaient pas dans le jeu de données d'apprentissage.

**Nous avons implémenté notre outil de classification et nous avons montré qu'il peut classer le trafic à des débits de 6Gbit/s.** Nous avons implémenté notre méthode de classification et proposé des optimisations pour accélérer la classification et diminuer la quantité de mémoire utilisée. Nous avons montré que cette implémentation peut classer le trafic à des débits de 6Gbit/s, ce qui est suffisant pour les réseaux de bordure. De plus, nous avons étudié le profil de notre application et montré que notre méthode de classification peut être ajoutée aux outils qui font des analyses de connexions TCP à un coût minime. La librairie Matlab que nous avons développée pour générer les modèles, et le code de notre outil de classification sont tous deux disponibles à l'adresse suivante : <http://rp.lip6.fr/~bernaill/earlyclassif.html>.

**Nous avons présenté une technique qui permet d'identifier l'application associée à une connexion chiffrée.** Nous avons étudié le trafic SSL de deux réseaux universitaires et montré que l'utilisation de SSL se répand et que le nombre d'applica-

tions qui emploient ce type de chiffrement augmente. Nous avons présenté une technique qui permet de reconnaître l'application qui génère un trafic chiffré avec SSL en utilisant la taille des premiers paquets. Nous avons montré que cette méthode identifie correctement l'application dans plus de 85% des cas.

### A.3.2 Perspectives

Le travail présenté dans cette thèse peut être approfondi de plusieurs manières.

**Classification des connexions UDP.** La classification du trafic UDP à partir de la taille des premiers paquets d'une connexion est difficile parce-qu'il est difficile de trouver les premiers paquets d'une connexion UDP. Notre méthode pourrait être étendue à ce type de trafic en utilisant des heuristiques permettant de trouver ces paquets, comme proposé dans [76] par exemple.

**Nombre de paquets variable.** Notre outil de classification utilise un nombre de paquets fixe pour classer les connexions. L'avantage de cette méthode est qu'elle est déterministe. En revanche, il peut être plus facile d'identifier certaines applications avec un nombre de paquets plus faible ou plus élevé. C'est pourquoi nous pourrions modifier notre méthode pour qu'elle tente d'identifier l'application après 1, 2, ..., N paquets, jusqu'à ce que l'on reconnaisse l'application de façon certaine.

**Analyse de la stabilité de notre modèle.** Le modèle que nous créons durant la phase d'apprentissage est valide tant que les applications n'évoluent pas sur le réseau étudié. Lorsqu'une nouvelle application apparaît, nous devons effectuer un nouvel apprentissage pour la prendre en considération. Durant cette thèse, nous avons étudié le trafic de l'université Paris 6 pendant trois ans. Pendant cette période, nous n'avons pas vu apparaître de nouvelle application, mais nous avons observé la disparition du trafic Kazaa et edonkey (ces applications sont maintenant interdites sur le réseau de l'université). Cependant, cette stabilité est spécifique au réseau académique que nous avons étudié et nous souhaiterions pouvoir analyser l'évolution des applications sur des réseaux commerciaux et mesurer la fréquence à laquelle nous devons refaire l'apprentissage pour prendre en compte cette évolution.

**Implémentation dans un routeur.** Nous avons implémenté notre méthode de classification en C, et celle-ci s'applique à des traces pcap. Nous souhaiterions l'implémenter sur un processeur réseau pour la tester sur du trafic réel et évaluer le débit maximum

qu'elle peut traiter.

**Extension à d'autres algorithmes de chiffrement.** Notre outil de classification peut identifier les applications transportées dans des connexions chiffrées avec SSL. Nous aimerions étendre notre méthode de reconnaissance à d'autres mécanismes de chiffrement tels que SSH ou IPSEC. En revanche, pour ces deux protocoles, isoler les paquets appartenant à une connexion donnée sera très difficile. D'autre part, les implémentations de SSL les plus récentes proposent des options permettant la compression des données ou l'envoi de segments vides. L'utilisation de ces options affectera notre méthode de classification. Nous avons donc prévu de la modifier pour prendre en compte ces nouvelles options.

La méthode que nous avons proposée représente une avancée dans le domaine de la classification temps réel. Cependant, un problème important devra être réglé si l'on souhaite utiliser cette méthode pour la sécurité des réseaux : l'évasion. Toutes les méthodes de classification sont sensibles aux techniques d'évasion : les techniques fondées sur l'analyse du contenu des paquets ne peuvent pas étudier le trafic chiffré, les techniques utilisant les numéros de ports peuvent être contournées à l'aide d'un simple changement de port, et les approches utilisant des statistiques sur les connexions sont sensibles aux modifications de tailles de paquets ou des temps inter-arrivées. Un "attaquant" pourrait facilement contourner notre technique en ajoutant des octets de padding à la fin des paquets pour modifier leur taille. Pour développer un outil de classification devant fonctionner dans un milieu hostile, il faut utiliser plusieurs techniques simultanément. Afin d'améliorer notre méthode, nous pourrions l'utiliser en parallèle avec des outils qui vérifient la syntaxe des applications. Nous pourrions l'utiliser avec l'analyseur décrit dans [20]. Dans ce cas notre outil ferait une pré-classification qui permettrait d'identifier les applications potentielles très rapidement et ainsi limiter le volume de traitement de l'analyseur syntaxique. Le développement d'outils de classifications pouvant résister à des attaques complexes est aujourd'hui encore un défi pour les équipes de recherche qui travaillent sur la reconnaissance des applications.

# Bibliography

- [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] A. Barron, J. Rissanen, and B. Yu. The minimum description length principle in coding and modeling. . *IEEE Trans. Information Theory*, 44, 1998.
- [4] L. 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.
- [5] L. Bernaille, A. Soule, M. 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/~bernaille/techreport.pdf>, 2005.
- [6] L. Bernaille and R. Teixeira. Early recognition of encrypted applications. In *Passive and Active Measurement*, 2007.
- [7] 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.
- [8] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *To appear in Conference on Future Networking Technologies*, 2006.
- [9] J. Bilmes. A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models, 1997.

- 
- [10] H. Binsztok, T. Artières, and P. Gallinari. A model-based approach to sequence clustering. In *ECAI*, 2004.
- [11] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Communications of the ACM*, 1977.
- [12] <http://bro-ids.org/>.
- [13] Cesnet. <http://www.ces.net>.
- [14] J. Cleary, S. Donnelly, I. Graham, T. McGregor, and M. Pearson. Design principles for accurate passive measurement. In *Passive and Active Measurement*, 2000.
- [15] <http://www.caida.org/tools/measurement/coralreef/>.
- [16] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 1967.
- [17] D. Davies and D. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1979.
- [18] 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.
- [19] C. Dewes, A. Wichma, and A. Feldmann. An analysis of internet chat systems. In *IMC*, 2003.
- [20] H. Dreger, M. Mai, A. Feldmann, V. Paxson, and R. Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *Usenix Security Symposium*, 2006.
- [21] N. Duffield, C. Lund, and M. Thorup. Properties and prediction of flow statistics from sampled packet streams. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, 2002.
- [22] J. Dunn. Well separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 1974.
- [23] Endace. [www.endace.com](http://www.endace.com).
- [24] J. Erman, M. Arlitt, and A. Mahanti. Traffic classification using clustering algorithms. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 281–286, 2006.

- 
- [25] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. A semi-supervised approach to network traffic classification. In *SIGMETRICS*, 2007.
- [26] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 245–256, 2004.
- [27] I. Fischer and J. Poland. New methods for spectral clustering. Technical report, IDISA, 2004.
- [28] R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 1936.
- [29] E. Fix and J. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review*, 1989.
- [30] <http://www.gnu.org/software/binutils/manual/gprof-2.9.1/>.
- [31] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning: Data mining, inference, and prediction*. Springer, 2001.
- [32] 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, 2004.
- [33] F. Hernandez-Campos, A.B. Nobel, F.D. 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.
- [34] A. Hintz. Fingerprinting websites using traffic analysis, 2002.
- [35] N. Hohn and D. Veitch. Inverting sampled traffic. In *IMC*, 2003.
- [36] <http://www.iana.org/>.
- [37] ipfix: <http://www.ietf.org/html.charters/ipfix-charter.html>.
- [38] IPMON. [ipmon.sprintlabs.com](http://ipmon.sprintlabs.com).
- [39] P. Jaccard. Bulletin de la societe vaudoise des sciences naturelles, 1901.
- [40] 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*, 2003.

- 
- [41] N. Jardine and R. Sibson. The construction of hierarchic and non-hierarchic classifications. *The Computer Journal*, 1968.
- [42] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is p2p dying or just hiding? In *Globecom*, 2004.
- [43] T. Karagiannis, D. Papagiannaki, and M. Faloutsos. Blinc: Multilevel traffic classification in the dark. In *SIGCOMM*, 2005.
- [44] <http://l7-filter.sourceforge.net/>.
- [45] Liberouter. <http://www.liberouter.org>.
- [46] Ma, Levchenko, Kreibich, Savage, and Voelker. Unexpected means of protocol inference. In *Internet Measurement Conference*, 2006.
- [47] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. In *Passive and Active Measurement*, 2004.
- [48] J. McQueen. Some methods for classification and analysis of multivariate tions. In *Symposium on Mathematical Statistics and Probability*, 1967.
- [49] A. Moore and K. Papagiannaki. Toward the accurate identification of network applications. In *Passive and Active Measurement*, 2005.
- [50] A. Moore and D. Zuev. Internet traffic classification using bayesian analysis. In *SIGMETRICS*, 2005.
- [51] <http://www.mysql.org/>.
- [52] Netcraft: <http://www.netcraft.com>.
- [53] <http://www.netfilter.org/>.
- [54] [www.cisco.com/en/US/products/ps6601/products\\_ios\\_protocol\\_group\\_home.html](http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html).
- [55] A.Y. Ng, M. Jordan, and Y. Weiss. On spectral clustering : analysis and an algorithm. In *NIPS*, 2001.
- [56] A. Oveissian, K. Salamatian, A. Soule, and N. Taft. Fast flow classification over internet. In *CNSR*, 2004.
- [57] K. Papagiannaki, N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian, and C. Diot. A pragmatic definition of elephants in internet backbone traffic. In *IMC*, 2002.

- 
- [58] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.
- [59] [www.cisco.com/en/US/products/hw/vpndevc/ps2030/index.html](http://www.cisco.com/en/US/products/hw/vpndevc/ps2030/index.html).
- [60] F. Porikli. Trajectory distance metric using hidden markov model based representation. In *IEEE European Conference on Computer Vision, PETS Workshop*, 2004.
- [61] psamp: <http://www.ietf.org/html.charters/psamp-charter.html>.
- [62] Qosmos. <http://www.qosmos.com>.
- [63] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 1989.
- [64] W. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 1971.
- [65] 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.
- [66] P. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Computational and Applied Mathematics*, 1987.
- [67] Scampi. <http://www.ist-scampi.org>.
- [68] N. Shah. Understanding network processors. Master’s thesis, University of California, Berkeley, 2001.
- [69] P. Smyth. Clustering sequences with hidden markov models. In *Advances in Neural Information Processing*, 1997.
- [70] Snort: <http://www.snort.org>.
- [71] D. Xiaodong Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on ssh. In *Proc. 10th USENIX Security Symposium*, 2001.
- [72] A. Soule, K. Salamatian, R. Emilion, N. Taft, and K. Papagiannaki. Flow classification by histograms or how to go on safari in the internet. In *ACM Sigmetrics*, 2004.

- 
- [73] SSLv2: [http://wp.netscape.com/eng/security/SSL\\_2.html](http://wp.netscape.com/eng/security/SSL_2.html).
- [74] SSLv3.0: <http://wp.netscape.com/eng/ssl3/draft302.txt>.
- [75] A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal on Machine Learning Research (JMLR)*, 2002.
- [76] K. Suh, D. Figueiredo, J. Kurose, and D. Towsley. Characterizing and detecting relayed traffic: A case study using skype. In *IEEE Infocom*, 2006.
- [77] TCPDUMP: <http://www.tcpdump.org/>.
- [78] TLS: <http://www.ietf.org/rfc/rfc2246.txt>.
- [79] C. Trivedi, H.J. 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.
- [80] <http://valgrind.org/>.
- [81] Wright, Monrose, and Masson. On inferring application protocol behaviors in encrypted network traffic. *the Journal of Machine Learning Research Special Topic on Machine Learning for Computer Security*, 2006.
- [82] F. Yu, Z. Chen, Y. Diao, T. Lakshman, and R. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 93–102, 2006.
- [83] D. Zuev and A. Moore. Discriminators for use in flow-based classification. Technical report, Intel Research, Cambridge, 2005.
- [84] D. Zuev and A. Moore. Traffic classification using a statistical approach. In *Passive and Active Measurement*, 2005.

# List of Figures

1.1	Sample internetwork. Transit: ISP 1 and ISP 2. Edge: Campus and Enterprise . . . . .	17
1.2	Approach overview . . . . .	20
3.1	Comparison of classification metrics for the most common applications for Paris6-1 trace . . . . .	41
3.2	Cumulative distribution functions for mean IAT . . . . .	42
3.3	Comparison of the sizes of the first 4 packets for different applications on several networks . . . . .	44
4.1	Overall Accuracy on the full training set depending on the number of samples (No threshold and using the Dominant heuristic) . . . . .	48
4.2	Three groups of points in a two-dimensional space . . . . .	53
4.3	Clusters found with K-Means . . . . .	53
4.4	Clusters found with K-Means . . . . .	54
4.5	Probability density function obtained with GMM . . . . .	56
4.6	Points belonging to three distinct classes . . . . .	57
4.7	Proportion of connections filtered (four packets) . . . . .	59
4.8	Influence of the number of clusters on clustering quality (4 packets, K-Means clustering) . . . . .	61
4.9	Comparison between NMI and Rand Index (2 packets, GMM clustering) . . . . .	62
4.10	Optimal number of clusters according to the number of packets . . . . .	62
4.11	Influence of the number of packets on clustering quality . . . . .	63
4.12	Proportion of clusters with N Applications (four packets) . . . . .	64

5.1	Distribution of distances to the center for the largest cluster (K-Means, 4 packets, 30 clusters) . . . . .	67
5.2	Choice of Threshold (K-Means) . . . . .	68
5.3	Choice of Threshold (GMM) . . . . .	70
5.4	Choice of Threshold (HMM) . . . . .	71
5.5	Overall Accuracy (Predominant) . . . . .	76
5.6	Mean Accuracy (Predominant) . . . . .	77
6.1	Example of SSL handshake (SSLv2) . . . . .	92
6.2	Position of the first packet with application data . . . . .	96
6.3	Size of encrypted packets according to cipher . . . . .	97
6.4	Classifier Overview . . . . .	98
A.1	Exemple de réseaux interconnectés. Transit : FAI 1 et FAI 2. Bordure : Université et Entreprise . . . . .	109
A.2	Méthode de classification . . . . .	113
A.3	Comparaison des métriques pour les applications les plus représentées dans les traces de Paris 6 . . . . .	115
A.4	Influence du nombre de clusters sur la qualité du clustering (4 paquets, K-Means) . . . . .	120
A.5	Comparaison entre NMI et Rand Index (2 paquets, GMM) . . . . .	121
A.6	Influence du nombre de paquets sur la qualité du clustering . . . . .	121
A.7	Proportion de clusters avec N Applications (4 paquets) . . . . .	122
A.8	Précision globale (Predominant) . . . . .	126
A.9	Précision moyenne (Predominant) . . . . .	126
A.10	Position du premier paquet avec des données chiffrées . . . . .	129
A.11	Taille des paquets chiffrés en fonction de l'algorithme de chiffrement . . . . .	130
A.12	Vue d'ensemble . . . . .	131

# List of Tables

2.1	Example of mapping between applications and ports . . . . .	27
2.2	Common protocol signatures . . . . .	29
2.3	Metrics for behavior based classification methods . . . . .	30
3.1	Description of packet traces for connections that started during the capture, and proportion of connection and traffic with more than four packets	39
4.1	Notations for the training phase . . . . .	46
4.2	Number of connections with at least four packets for each application in Paris6-1 trace . . . . .	47
4.3	Similarity matrix . . . . .	57
4.4	Clusters found in the similarity matrix . . . . .	58
5.1	Notations . . . . .	66
5.2	Assignment accuracy and proportion of unknown traffic for K-Means . .	74
5.3	Assignment accuracy and proportion of unknown traffic for GMM . . .	74
5.4	Assignment accuracy and proportion of unknown traffic for HMM . . .	75
5.5	Accuracy of labeling heuristics (4 packets) . . . . .	77
5.6	Labeling accuracy for Paris 6-1 trace. . . . .	78
5.7	Detection of unknown traffic (Paris 6-1) using Cluster+Port . . . . .	80
5.8	Compared complexities . . . . .	85
5.9	Trace description . . . . .	85
5.10	Evaluation . . . . .	86
5.11	Time spent . . . . .	86
6.1	SSL versions . . . . .	94

6.2	Non-standard SSL Traffic . . . . .	94
6.3	Proportion of each cipher . . . . .	97
6.4	Application detection, including SSL(Paris6-3 Trace) . . . . .	100
6.5	Detection of Encrypted Applications in Paris6-3 trace . . . . .	100
6.6	Detection of Encrypted Applications for manually encrypted traffic . . . . .	100
A.1	Précision de l'association et proportion de trafic inconnu pour K-Means . . . . .	125
A.2	Précision des heuristiques d'étiquetage (4 paquets) . . . . .	127
A.3	Temps passé . . . . .	128
A.4	Détection des applications chiffrées dans la trace Paris6-3 . . . . .	132
A.5	Détection des applications pour le trafic chiffré manuellement . . . . .	132



## Résumé

L'identification des applications utilisées sur un réseau est essentielle pour comprendre les évolutions du trafic et pour appliquer des politiques de qualité de service. Les méthodes de classification utilisant les numéros de port deviennent de moins en moins efficaces. Bien que plus précise, l'analyse systématique du contenu des paquets nécessite beaucoup de ressources et ne peut fonctionner lorsque le trafic est chiffré. Récemment, de nouvelles techniques utilisant des études statistiques sur les flux TCP ont fait leur apparition. Cependant, ces techniques nécessitent une analyse a posteriori des connexions, et ne peuvent donc être utilisées pour agir dynamiquement sur le trafic.

Dans cette thèse, nous proposons une méthode identifiant l'application associée à une connexion TCP à partir de ses premiers paquets uniquement. Tout d'abord, nous utilisons plusieurs techniques de clustering afin d'identifier les comportements de connexions les plus répandus. Ensuite, nous présentons des méthodes de classification qui associent les connexions à identifier aux clusters que nous avons trouvés et déterminent leur application. Nous évaluons ces méthodes sur des traces de trafic et montrons que la taille des quatre premiers paquets suffit pour identifier 90% des connexions. Nous décrivons ensuite une implémentation de notre méthode capable de supporter un débit de 6Gbit/s. Enfin, nous étendons notre méthode aux connexions chiffrées avec SSL et montrons que nous pouvons identifier l'application encapsulée dans plus de 80% des cas.

## Mots-Clés

Classification du trafic Internet, TCP, Applications, Intelligence Artificielle, SSL