**Université de Nice - Sophia Antipolis**

**École Doctorale STIC**
**SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION**

**THÈSE**

pour obtenir le titre de :

**Docteur en Sciences**

*de l'Université de Nice - Sophia Antipolis*
Mention : INFORMATIQUE

présentée et soutenue par

mohamad JABER

# INTERNET TRAFFIC PROFILING AND IDENTIFICATION

Thèse dirigée par chadi BARAKAT et philippe NAIN

Soutenue le (6 Octobre, 2011)

## Jury

| | | | |
|---|---|---|---|
| Mme claudine PEYRAT | Prof. | Université de Nice-Sophia Antipolis, France | Présidente |
| M. chadi BARAKAT | HDR | INRIA, France | Directeur |
| M. philippe NAIN | HDR | INRIA, France | Directeur |
| M. philippe OWEZARSKI | HDR | LAAS-CNRS, France | Rapporteur |
| M. kave SALAMATIAN | Prof. | Université de Savoie, France | Rapporteur |
| M. paulo GONÇALVES | HDR | ENS Lyon, France | Examinateur |
| M. manuel CROTTI | Dr. | Université de Brescia, Italy | Examinateur |

Thèse

# Caractèrisation et identification du trafic Internet

# Internet traffic profiling and identification

Mohamad Jaber

September 2011

## Dedications

To my father Kamal Mohamad Jaber

To my mother Nada Jaber Jaber

# Résumé

## Caractérisation et identification du trafic Internet

*Mohamad Jaber*
Directeur de thèse: *Chadi Barakat and Philippe Nain*
*Equipe Planete*, Inria Sophia Antipolis, France

L'évolution de l'Internet dans les dernières années a été caractérisée par des changements dramatiques dans la manière dont les utilisateurs se comportent, interagissent et utilisent le réseau. Ceci a été particulièrement accompagné par l'introduction de nouvelles classes d'applications telles que les jeux en ligne et et les réseaux pair-à-pair. L'un des défis les plus importants pour les administrateurs réseau et les ISPs est alors devenu l'identification du trafic Internet afin de pouvoir protéger leurs ressources contre le trafic indésirable et de prioriser certaines applications majeures. Les méthodes statistiques sont préférées à celles basées sur le numéro de port et l'inspection approfondie des paquets, car elles sont robustes au changement malveillant du numéro de port et fonctionnent avecle trafic crypté. Ces méthodes combinent l'analyse des paramètres statistiques des flux de paquets, tels que la taille des paquets et le temps les séparant, avec des techniques issues de la théorie d'apprentissage (machine learning). La majorité des méthodes statistiques ne peuvent pas identifier les flux applicatifs en temps réel et elles ont besoin d'atteindre la fin des flux avant de prendre une décision sur leur nature. Ceci est considéré comme trop long pour la plupart des administrateurs réseau, puisqu'il ne permet pas de bloquer un flux Internet indésirable à son début ni de lui donner en amant une qualité particulière de service. Un autre défi important pour les administrateurs réseau est de détecter et diagnostiquer tout changement dans le réseau comme une congestion à long terme, un changement dans le routage, une défaillance d'une liaison ou tout autre événement entraînant un changement dans les délais réseau. Dans la littérature il y a un grand nombre de méthodes pour détecter des anomalies dans le réseau, mais la plupart de ces méthodes ont besoin de générer un volume considérable de trafic destiné à la métrologie du réseau. La réduction de la charge des mesures est un besoin vital pour les administrateurs réseaux. Dans cette thèse nous décrivons les travaux que nous avons menés sur l'identification du trafic Internet et sur la détection des anomalies dans les réseaux. Dans la première partie, nous présentons nos trois méthodes que nous avons développées au cours de cette thèse, et qui permettent d'identifier avec précision et à la volée le trafic Internet. La première méthode, par sa nature itérative et probabiliste, identifie les applications rapidement et avec une grande précision en utilisant uniquement la taille des N premiers paquets. La deuxième méthode enrichit la première avec le temps entre paquets, pour cela nous avons eu besoin d'introduire un modèle pour filtrer le bruit dû aux conditions du réseau et d'extraire des mesures le temps d'attente due aux applications. Notre troisième méthode pour la classification du trafic en ligne combine les approches statistiques à des informations sur le comportement des machines hôtes afin de rendre l'identification du trafic Internet encore plus précis tout en profilant les activités réseaux des hôtes. Pour notre troisième méthode, nous utilisons la taille des paquets comme paramètre principale et nous exploitons les informations sur l'interaction des machines pour mieux affecter un flux à une application. Dans la deuxième partie de cette thèse, nous abordons le problème de détection des anomalies dans les réseaux. Nous commençons par une étude sur la stabilité des systèmes de coordonnées Internet (en particulier Vivaldi). Dans une première étape, nous confirmons le fait que les coordonnées de Vivaldi oscillent au fil du temps en raison de la nature adaptative du système. Toutefois, les variations de ces coordonnées sont dans la plupart du temps en corrélation les unes avec les autres, pointant par conséquent vers un cluster de noeuds stables vu de l'intérieur du réseau. Dans un deuxième temps, nous présentons un nouvel algorithme de clustering basé sur des méthodes de groupement hiérarchique afin d'identifier ce cluster de noeuds stables. Enfin, nous soulignons l'utilité d'une telle constatation avec une application qui permet de détecter les changements dans le réseau. En changeant artificiellement les délais du réseau dans différents scénarios, nous montrons que ces changements sont reflétées par ce corps de noeuds stables, permettant ainsi d'obtenir une image globale de la stabilité du réseau sans avoir besoin de mesures exhaustives des délais.

# INTERNET TRAFFIC PROFILING AND IDENTIFICATION

by
*Mohamad Jaber*
Directeur de thèse: *Chadi Barakat and Philippe Nain*
*Equipe Planete*, Inria Sophia Antipolis, France

## ABSTRACT

The evolution of the Internet in the last few years has been characterized by dramatic changes in the way users behave, interact and utilize the network. This was accompanied by the introduction of new categories of applications such as network games and peer-to-peer services. One of the most important challenges for network administrators and ISPs is then becoming the identification of Internet traffic applications in order to protect their resources from unwanted traffic and to prioritize some major applications. Statistical methods are preferred to port-based ones and deep packet inspection since they donât rely on the port number and they also work for encrypted traffic. These methods combine the statistical analysis of the application packet flow parameters, such as packet size and inter-packet time, with machine learning techniques. However, the majority of these statistical methods cannot identify flows early and require reaching the end of flows before taking any decision which is considered as too late for network administrators; indeed they do not provide means to stop an Internet flow or to give it a special quality of service early in its lifetime.

Another important challenge for network administrators is to detect and diagnose key network changes as a long-term congestion, a rerouting, a link failure or any other event causing a shift in network delays. In the literature there is a huge amount of anomaly detection methods but most of them require exhaustive measurements to function properly. Reducing the load of network-wide monitoring is always a vital need for network administrators.

In this thesis we present several contributions around Internet traffic identification and network-wide anomaly detection. In the first part we present three methods we have developed in order to identify accurately and on the fly the Internet traffic. The first method is a new online iterative probabilistic method that identifies applications quickly and accurately by only using the size of the first N packets. The second method enhances the first one with the inter-packet time in order to identify Internet traffic, this has required the introduction of a model to isolate the noise due to network conditions and to extract the time generated by the applications themselves. Our third method is a new online method for traffic classification that combines the statistical and host-based approaches in order to construct a robust and precise method for early Internet traffic identification. We use the packet size as the main feature for the classification and we benefit from the traffic profile of the host (i.e. which application and how much) to decide in favor of this or that application.

In the second part of this thesis, we aboard the problem of network-wide anomaly detection. We start by making a study about the stability of Internet coordinate systems (especially Vivaldi). In a first stage we confirm the fact that Vivaldi coordinates oscillate over time because of the adaptive nature of the system. However, the variations of these coordinates are most of the time correlated with each other pointing to a stable cluster of nodes seen from inside the network. In a second stage, we present a new clustering algorithm based on the data mining Hierarchical Grouping Method to identify this cluster of stable nodes. Finally, we highlight the utility of such finding with an application that tracks changes in network delays. By changing artificially the network delays in different scenarios, we show that these changes are easily reflected by this body of stable nodes, hence allowing to obtain a global picture about the stability of the underlying network without the need for exhaustive delay measurements.

# ACKNOWLEDGMENTS

This thesis would not have been completed without the help and contributions from a number of people who provided me their support both at technical and moral levels. Thus, I would like to take the opportunity to express my gratitude in order to thank them all here. I cannot possibly thank my thesis advisors Dr. Chadi Barakat and Dr. Philippe Nain for their continuous guidance, support and invaluable feedback on my work. They really helped me a lot in making the progress in my research, and I have learnt a lot while working under their supervision. (A special thanks to Chadi, he was my friend and my supervisor, i learned a lot from him).

I would also like to thank my colleagues in the Planete team at INRIA, Sophia Antipolis, especially Dr. Walid Dabbous, Dr. Thierry Turletti, Dr. Thierry Permentlat and Dr. Arnaud Legout who repeatedly provided their feedback on my work and helped me in improving it technically. Working in the Planete team at INRIA, Sophia Antipolis has been an unforgettable and very pleasant experience of my life, and I am going to miss the working environment here. Besides, I thank all my friends (Laurie, Anais, Mahmoud, Nicaise, Larissa, Abd, Ashwin, Shafqat, Anna, Blerina, Chafic, Salim, Maher, Sami, Rawad, Omar, Roberto, Giovanni, saed, Vincenzo, Marina, Dominique, Stevens) and colleagues who helped me in improving the thesis manuscript by providing their valuable feedback especially Ashwin and Roberto.

I would like also to thank the members of my PhD committee who patiently spent their effort in reading this dissertation: Prof. Kave Salamatian, Dr. Philippe Owezarski, Dr. Paulo Goncalves and Dr. Manuel Crotti . Many thanks go to Prof. Claudine Peyrat for serving as the chairman of the jury committee.

Nobody in this world can pay back for the love, affection and support of parents, and I am no exception. I would really like to thank my parents (Kamal and Nada) for extending their financial and moral support to me throughout my life. Their prayers and guidance have enabled me to achieve whatever I have attained in my life. Moreover, I would also want to mention the support I have been receiving from my sisters and brothers, Mariam, Hadi, Sarah, Nizar, Hiba and Houssam (he will stay alive in my heart for ever). A special thanks for my uncles, aunts and Grand-Mothers, Hayat, Mahmoud, Samia, Wissam, Hussein, Nadia, Nawal, Ali, Fayza, Omar and Omaira. A special dedicates for my brother Houssam, my uncle Hassan and my grand-fathers Mohamad and Jaber (they will stay alive in my heart).

*Mohamad JABER*
*Mohamad.jaber@inria.fr*
Sophia Antipolis, France

# CONTENTS

# FIGURES

# TABLES

# 1

# INTRODUCTION

The Internet was created as a simple, open and flexible network, where the only service offered to users is the best effort. This choice was deliberate for the simple reason to facilitate the interconnection of networks, the creation of applications and the addition of services to the Internet. Over the last thirty years, the Internet has evolved to a giant network interconnecting tens of thousands of autonomous networks. At the beginning, the usage of the Internet was limited to information exchanging by ”E-mails” and ”newsgroups”. Then, the need to find and organize information a few years later led to the development of the first web pages using hypertext. Today, hundreds of millions of people use the Internet, stimulating the development of a wide variety of applications, which continue to appear on a monthly basis. A fundamental principle of the Internet is that the network should remain as simple as possible and so the intelligence should be in users’ machines that perform the standard operations like errors recovery, rate adaptation, localization of resources, and so on. It’s the famous principle of end to end. This principle allows any Internet application to send packets for any IP address and any port number. The network does then its best to route the packets to this correct destination. Unfortunately, nothing is free in life. The opening of the network to traffic makes the task of the administration of the Internet and its engineering complicated, and consequently leads to a harmful behavior for both users and network operators. The present thesis fits within this context and tries to provide solutions to some of the problems encountered in the Internet of today.

## 1.1 Motivation

The identification of applications at the origin of Internet traffic is of major importance for network operators and Internet Service Providers (ISPs). On one side, this allows to treat flows in a different way based on their quality of service requirements. On the other side, they can use this information for security reasons in order to protect their networks from unwanted traffic by blocking or looking closely at those users who run unwanted or non legacy applications. Finally application identification can help operators for the good dimensioning of their networks. The recognition of applications in IP traces becomes increasingly complex. Historically, this recognition was based on static and standard port numbers in the transport header [16]. But the evolution of applications that do not communicate on standard port numbers, and the fact that there are many applications that use well-known port numbers of other applications causes this solution to be ineffective. Current techniques of Deep Packet Inspection (DPI)[26, 14, 40, 9, 55, 35, 11] make it possible to go further in the identification of the applications but they require a complete and costly exploration of the payload of the packets. This induces an important load and is not practical when packets are encrypted.

Statistical techniques [38, 41, 67, 63, 7, 54, 45, 43, 13, 12, 5, 2, 46, 3, 59, 25] seem to be today a promising alternative. They allow to recognize and to classify the applications according to their statistical signatures. These signatures can be among many others data volume per connection (e.g., number of bytes), connection duration, rate, inter-packet delay, packet size, and direction.

The majority of statistical methods cannot identify flows early and they require reaching the end of the flow before taking any decision which is considered too late for most of network administrators, because it does provide no means to stop an Internet flow or to give it a special quality of service before its end. It's important for network administrators to identify Internet traffic correctly and on real time.

Another important challenge for network administrators is to detect and diagnose key network changes as a long-term congestion, a rerouting, a link failure or any other event causing a shift in network delays. In the literature there is a huge amount of anomaly detection methods but most of them require exhaustive measurements to function properly. Reducing the load of network-wide monitoring is always a vital need for network administrators. Recently, a new approach has emerged for Internet positioning having the main advantage of providing estimates for network delays between machines at a low measurement cost. This approach consists in building a coordinate system for the Internet [33]. The basic idea is to embed all the nodes of a given application (or overlay) in some Euclidean space, and to associate to each node specific coordinates in this space in such a way that the network delay between any two nodes can be approximated by the geometric distance separating them (Figure 6.1). These coordinate systems can be very useful if there is a way to use it for network changes detection.

## 1.2 Thesis Contributions

In this dissertation we describe the work are carried out during the thesis about Internet traffic identification and network anomaly detection.

In the first part of the thesis we aboard the traffic identification problem and we aimed developing fast and robust methods which identify Internet traffic applications on real time. We present our three methods for traffic identification that we developed during this thesis.

The first method (described in Chapter 3) is a new on-line iterative probabilistic method that identifies applications quickly and accurately by only using the size of the first N packets. Our method associates a configurable confidence level to the port number carried in the transport header of packets and is able to consider a variable number of packets at the beginning of a flow. By verification on real traces we can observe that even in the case of no confidence in the port number, a very high accuracy can be obtained for well known applications after few packets were examined.

The second method (described in Chapter 4) extends our iterative model to use the inter-packet time as feature to identify Internet traffic. To this end we had to introduce a model to isolate the noise due to the network conditions and extract the idle time generated by the application. We present our model to preprocess the inter-packet time and use the result as input to the learning process. We discuss the same iterative approach for the on-line identification of the applications and we evaluate our method on two different real traces. The results show that the inter-packet time can be transformed into an important parameter for the classification of Internet traffic.

Our third method (described in Chapter 5) is a new on-line method for traffic classification that combines the statistical and host-based approaches in order to construct a robust and precise method for early Internet traffic identification. We use the packet size as the main feature for the classification and we benefit from the traffic profile of the host (i.e. which application and how much) to decide in favor of this or that application. This latter profile is updated on-line based on the result of the classification of previous flows originated by or addressed to the same host. We evaluate our method on real traces using several applications. The results show that leveraging the traffic pattern of the host ameliorates the performance of statistical methods. They also prove the capacity of our solution to derive profiles for the traffic of Internet hosts and to identify the services they provide.

In the second part of this dissertation, we aboard the problem of network anomaly detection. We start by studying about the stability of Internet coordinate systems (especially Vivaldi). In a first stage, we confirm the fact that Vivaldi coordinates oscillate over time because of the adaptive nature of the system. However the variations of these coordinates are most of the time correlated with each other pointing to a stable cluster of nodes seen from inside the network. In a second stage, we present a new clustering algorithm based on the data mining Hierarchical Grouping Method to identify this cluster of stable nodes. Finally we highlight the utility of such finding with an application that tracks changes in network delays and tracking the presence of anomalies and the impacted machines. To this end, we propose to track a simple signal, which is the size of this biggest stable cluster. By changing artificially the network delays in different scenarios, we show that these changes are easily reflected by this body of stable nodes, hence allowing to obtain a global picture about the stability of the underlying network without the need for exhaustive delay measurements.

## 1.3 Problems encountered during the thesis

During this thesis we have encountered two main problems:

- The most important problem that we have encountered is the lack of Internet traffic traces with ground truth for the validation of our Internet traffic identification methods. This problem is a common problem for all academics working on this subject. One can find a lot of public traces for the validation like CAIDA [6], MAWI [37] and NLANR [44], but the major problem with these traces is that they only provides the Headers of packets and not the payload or the application behind each flow. Hence by using these traces we can only compare the results of our methods to the port number, and so we can't be sure about the verification of our methods without the ground truth. For this reason we decided not to use these well known public traces for the validation of our methods. As an alternative we managed to capture a real packet trace at the edge of the INRIA laboratory network, and we used two real traces captured at the edge of Brescia University in Italy, we thank the authors of [7] and [17] for giving us access to these traces.
- For the anomaly detection part, we run real experimentations on the Planetlab [47] network. The problem that we have found is the difficulty to create anomalies inside the Planetlab network as it used by thousand of researchers. To counter this problem we used simulations that implement the real topology of the Planetlab network.

## 1.4 Overview of Results

Our evaluation of the accuracy of our methods for traffic classification is facilitated by using three real traces with ground truth. The first trace is collected at the edge of the INRIA laboratory network and two traces collected at the edge of the Brescia university in Italy. The results show that we can classify Internet traffic into applications on real time and with very high precision exceeding 98%. We tested the performance of our methods on many client-server and peer-to-peer applications.

Concerning the anomaly detection subject, our experimentation results show that we are able to detect network changes by means of Vivaldi coordinates system by tracking only one signal which is the size of the biggest stable cluster of nodes formed by the clustering algorithm that we have developed for the purpose of this research.

## 1.5 Publications issued from this thesis

[1], Mohamad Jaber, Roberto Cascella and Chadi Barakat, ”Using host profiling to refine statistical application identification”, under submission. A preliminary version as INRIA Report.

[2], Mohamad Jaber, Roberto Cascella and Chadi Barakat, ”Boosting statistical application identification by flow correlation” in proceedings of the Euro-NF International Workshop on Trafic and Congestion Control for the Future Internet,Volos Greece, April 2011.

[3], Mohamad Jaber, Roberto Cascella, Chadi Barakat, ”Can we trust the inter-packet time for traffic classification?”, in proceedings of IEEE International Conference on Communications (ICC), Kyoto, Japan, June 2011.

[4], Mohamad Jaber, Cao-Cuong Ngo, Chadi Barakat, ”A view from inside a distributed Internet coordinate system”, in proceedings of the Global Internet Symposium at IEEE Infocom, San Diego, March 2010.

[5], Mohamad Jaber and Chadi Barakat, ”Enhancing Application Identification By Means Of Sequential Testing”, in proceedings of IFIP/TC6 Networking Conference, Aechan, Germany, May 2009.

## 1.6 Organization of dissertation

The remainder of this dissertation is organized as follows. In Chapter 2 we present a background on the Internet’s TCP/IP architecture, the transport protocols and applications. We also review prior work on Internet traffic classification and we give a background on clustering algorithms especially the K-Means algorithm that we used in our three methods. Chapter 3 describes our first method about traffic identification. We also explain the basics of our iterative approach that we have used throughout this research. We discuss the assumptions that we made and we will describe the datasets that we used in this thesis for the validation of our methods. In Chapter 4 we present our study about the inter-packet time and how we can managed to transform it into a second method for traffic classification. Chapter 5 describes our host based method. In chapter 6 we start the second part of this thesis by providing a background on

Internet anomaly detection and Internet coordinates systems, especially Vivaldi. Then we describe our experimental methodology and our new clustering algorithm. Finally we present our protocol for detecting network delays changes network changes and we show our experimental results for this subject. Chapter 7 presents conclusions and directions for future work.

# 2

# Background and Related Work

In this chapter we give an overview of the Internet protocols and applications together with the different traffic classification methods present in the literature. We complete the chapter with a background about the machine learning algorithms.

In Section 2.1 we give an overview of the TCP/IP Architecture and layers. In Sections 2.2, 2.3, and 2.4, we describe respectively the Network layer, the Transport layer where we detail TCP and UDP, and the Application layer where we give an overview of the main Internet applications categories. In Section 2.5 we explain the different headers of the Internet packets and the usefulness of informations extracted from these headers for the purpose of traffic classification. In Section 2.6 we detail the different traffic classification methods present in the literature and we discuss the advantages of our methods in comparison with the other methods. Finally we give an overview of the different types of machine learning algorithms in Section 6.5 with a particular focus on K-Means, the clustering algorithm that we used to develop our traffic classification methods.

## 2.1 TCP/IP Architecture

To measure the Internet properly and to correctly interpret the results of Internet measurements, it is essential to understand the Internet architecture. We explain in this section the TCP/IP Architecture of the Internet in order to give the reader an idea about how network traffic is transmitted across the Internet. The Internet is a packet switched network [49]. Instead of transmitting a message between two hosts as a single large message, the message is broken up into smaller pieces called packets. These packets are then separately delivered to the destined host. This allows the message to be transmitted and does not require all the com-

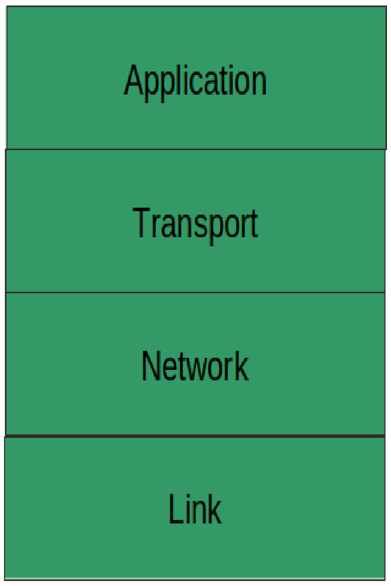

**Figure** 2.1: Protocol layers in the Internet.

munication links along the sender-to-receiver path to be reserved during message transmission. The use of packet switching has several advantages including: increasing the throughput of the communication links, increasing the robustness of the communication, and reducing the latency.

Using packet switching to provide reliable and efficient communication is a complex task. The complexity is managed by defining protocols communication protocols standards, for various subtasks, and by organizing those protocols into layers.

Protocols can be defined as a set of rules that govern the transfer of data between two hosts. The protocols used in the Internet are usually considered to be organized into four layers. Each layer has a particular responsibility, and provides a service to the layer above it. The four Internet layers are shown in Figure (2.1). Starting at the lowest layer, the link layer is responsible for physically interfacing with the underlying communication medium. The link layer's job is to actually move a packet from one location to another, and each link layer protocol is closely connected to the properties of the particular communication medium. As the top three layers are the most relevant to our work, we will explain them in more details, starting from the network layer.

## 2.2 Network Layer

The network layer is responsible for moving packets over sequences of links toward their destinations. The key protocol at this layer is the Internet Protocol (IP) [49]. In IP, every packet is independent and requires full addressing information to be included in each packet header. IP provides an unreliable service. It does the best job it can to deliver a packet to its final destination without providing any quality of service guarantee. For that reason, IP is known to provide a best effort service.

The IP layer also provides the addressing service based on a 32 bits field in IPv4, and 128 bits field when IPv6 will be fully deployed. When a packet is sent using IP, the address in the packet header is used by intermediate routers to determine on which path to send the packet. Every IP packet has a ”Time-To-Live” value. If a packet was not delivered, the routers will drop it when its ”Time-To-Live” expires. This ensures that undelivered packets do not circulate in the network indefinitely [49].

## 2.3 Transport Layer

The transport layer is concerned with providing a message service to applications that allows a flow of data between hosts. In principle, the application does not need to be aware that the communication is actually taking place in the form of packets; the transport layer can hide this fact from the application. The functions provided by the transport layer are mainly implemented in end systems. There are two key protocols at the transport layer, which differ in the type of service they provide to the application layer. TCP ([50]), the Transmission Control Protocol, provides a reliable flow of data between two hosts. The TCP sender divides the data passed to it from the application into packets and ensures that those packets are correctly delivered to the receiving system. The TCP receiver reconstructs the original data from its component packets and delivers the result to the receiving application. The other key transport protocol is UDP ([48]), the User Datagram Protocol. UDP provides a simpler service than TCP. UDP merely sends datagrams (small segments) of data that can fit into a single packet. It does not ensure that datagrams are correctly received, and there is no provision for sending data objects that are larger than what can fit into a single IP packet. Now let us explain the multiplexing of flows provided by the transport layer, and the functionalities supported by TCP and UDP.

### 2.3.1 Multiplexing Flows

One of the main functionalities of the transport layer is to allow hosts to have multiple and simultaneous data transfers with other hosts (even with the same one). The field of the transport layer (in TCP and UDP) responsible of this functionality is called port number. Thus, for each flow of packets there is a source port number field and a destination port number field that determine the port number of the source and the destination of this flow. The port number is used at the host to determine to which flow an arriving packet should be assigned. The flows of packets are identified by using the 5-tuple:

- source IP address,
- destination IP address,
- source port number,
- destination port number,
- transport layer protocol.

The source port number field in TCP and UDP is of 16 bits, the same for the destination port number, so the port numbers can take values from 0 to 65535. The port numbers from 0 to 1023 are reserved normally for well-known applications or protocols. For instance, a Web server uses the port 80 in general for incoming HTTP connections and an SMTP server uses the port 25. On the other hand, the port numbers ranging from 1024 to 65535 are dynamically assigned. We will explain more about the port number assignment in Section 2.6.1

### 2.3.2 Overview of TCP

TCP [50] is a connection-oriented Transport protocol. It provides a mechanism to ensure the smooth transfer of data. This ability is absolutely essential when applications must transmit large volumes of data reliably. TCP allows the establishment of a virtual circuit between the two points that exchange information. We can also say that TCP works in on-line mode (as opposed to UDP which is connection-less or datagram oriented).

- Before the transfer, the two applications establish the relation with their respective OS (Operating System), informing them of their desire to make or receive a call.
- Practically, one of the two applications must make a call and the other must accept.
- The protocols of the two OS then communicate by sending messages through the network to verify that the transfer is possible (allowed) and that both applications are ready for their roles.
- Once these steps accomplished, the protocol modules inform the respective applications that the connection is established and that the transfer can begin.
- During the transfer, the continuous dialog between the protocols permit to verify the correct routing of data.

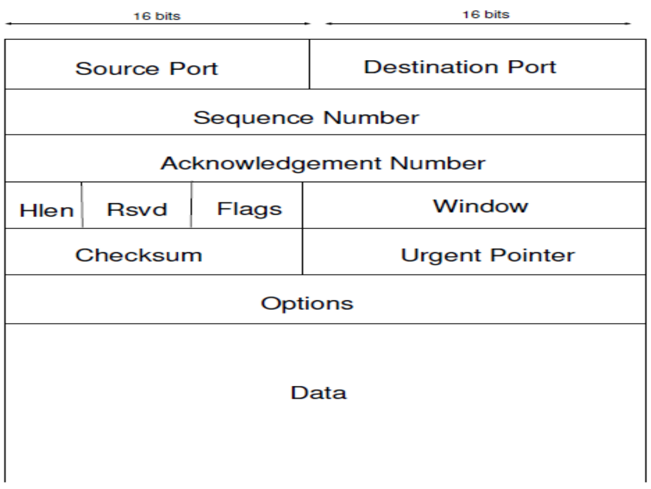

**Figure** 2.2: TCP Header.

Conceptually, to establish a connection (a virtual circuit) the elements of the 5-tuple cited in Section 2.3.1 are used.

The protocol allows the closing of the communication in one direction while the other continues to be active. The virtual circuit is broken when both parties have closed the stream. TCP is full duplex, so it permits that both end hosts send data independently. The general approach followed by TCP to ensure reliability is Acknowledgments. TCP breaks data up into segments, which usually correspond to IP packets; each segment has an associated TCP sequence number. When a sender transmits a segment, it sets a timer. When the segment is received at the destination, the receiver sends back an acknowledgment that specifies the next sequence number it expects to receive. If the sender's timer expires before the acknowledgment for the outstanding segment is received, the sender retransmits the segment. The TCP header (see Figure 2.2) consists of a fixed part and an optional part. Only the fixed part contains fields relevant to Internet measurements. These fields are:

- **The 16-bit source port:** it represents the port number of the local application.
- **The 16-bit destination port:** it represents the port number of the remote application.
- **The sequence number:** this is a number that identifies the position of data to be transmitted from the original segment. TCP gives a number for each byte transmitted.
- **The acknowledgment number:** it is a number that identifies the position of the last byte received in the incoming stream. It must be accompanied by the ACK flag.
- **SYN flag:** it is used for initiating a connection.

- **ACK flag:** it is set when the segment (the packet) is an acknowledgment.
- **PSH flag:** this is a notification of the transmitter to the receiver to indicate that all data collected must be transmitted to the application without waiting for any following data.
- **RST flag:** it is set to 1 when we want to re-initialize the connection.
- **FIN flag:** it indicates that the sender of the segment has finished transmitting.

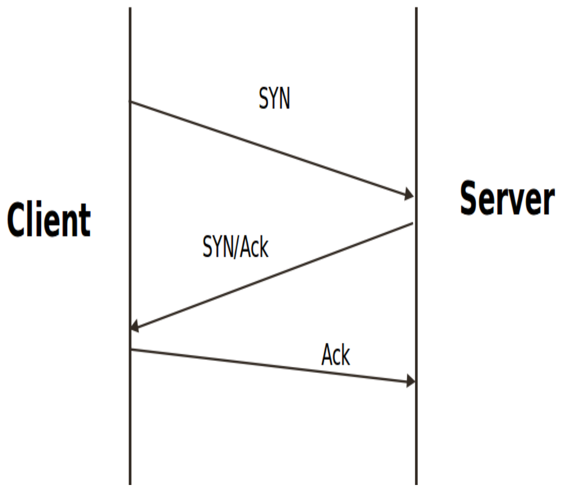

**Figure** 2.3: Connection scenario.

**Connection scenario (three-way handshake):**

The connection process (see Figure 2.3) is done as follows:

- The client sends a synchronization segment (SYN) containing its initial sequence number.
- The server receives the SYN packet, then it sends back to the customer a synchronization segment (SYN) together with an acknowledgment (ACK).
- The client receives the SYN/ACK packet and sends back an acknowledgment (ACK).
- The client notifies the opening of the connection.
- The server receives the ACK packet and notifies the opening of the connection.

**Disconnection scenario (three-way handshake)**

The disconnection process (see Figure 2.4) is done as follows:

- The server requests the close of the connection
- The server sends a FIN message informing the client that he will no longer send data.
- The FIN message is acknowledged (TCP level ACK) by the client.
- The client knows that the server closed the connection and requests the termination of the connection by sending a FIN message.
- The FIN message is acknowledged (TCP level ACK) by the server and the connection is declared closed.

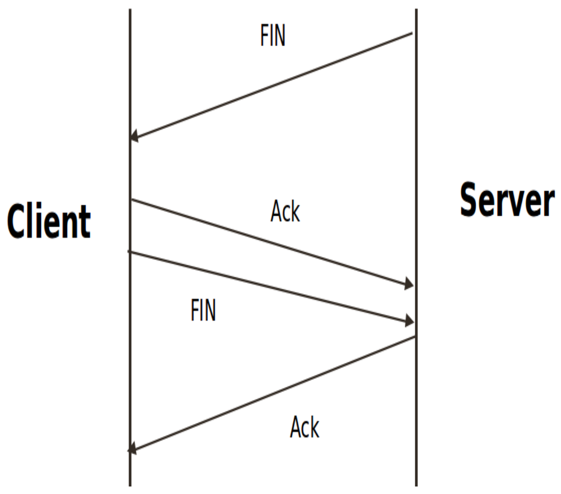

**Figure** 2.4: Disconnection scenario.

### 2.3.3 Overview of UDP

UDP (User Datagram Protocol) is a simple and connection-less Transport Protocol. It simply sends packets without waiting for acknowledgments. This protocol is transactional, and does not guarantee the delivery of the message, or its possible duplication. UDP provides a simple datagram service to applications. A datagram is a message that occupies only a single packet.

Many client-server applications that consist of one request and one response use UDP for transport. This is practical because, although UDP is itself unreliable, applications can themselves enforce reliability by setting timeouts for replies and retransmitting when necessary. For example, conventional queries in DNS are made using UDP. Since the service provided by UDP

**Table** 2.1: Main Applications.

| Applications Types | Applications-layer protocols | Transport protocol |
|---|---|---|
| Web | HTTP, HTTPS | TCP |
| Email | POP3, SMTP, IMAP | TCP |
| Chat | MSN, Yahoo, IRC | TCP |
| File tranfer | FTP | TCP |
| Peer-to-Peer | BitTorent, Edonkey, Gnutella... | TCP/UDP |
| Remote Computing | SSH, Telnet | TCP |
| Domain Name System | DNS | UDP |

is so similar to that provided by IP, the UDP header is quite simple. It provides little more than the source and destination ports. Needed to identify which application is associated with the data. For example, when a DNS (see Section 2.4.1) query is made, the destination port will be 53, the well-known port assigned to DNS.

The majority of traffic used for the validation of our classification techniques in this thesis is TCP traffic, but we have also used some UDP traffic. The details of the traces used will be explained in Chapter 3.

## 2.4 Application Layer

The application layer is concerned with implementing each particular application. Currently, the most important applications used in the Internet are the Email, the World Wide Web (Web), the peer-to-peer (P2P) programs (Bittorent, Edonkey...), the voice over IP programs like Skype, the on-line games, the multimedia streaming , the file transfer, and the Domain Name System (DNS). As the Transport layer, the Application layer implements end-to-end protocols, which means that they are only implemented in end systems, and that intermediate nodes (routers) do not participate in them. In the following subsections, we will describe the major application types. The described applications are listed in Table

### 2.4.1 DNS

The Domain Name System (DNS) [8] is a mechanism for translating an object into other object. It ensures the correspondence between a host name and the IP address, this translation is bidirectional [39]. Most applications use string versions of IP addresses (i.e. www.inria.fr) and these string versions are looked up in a database and converted into a.b.c.d IP address (i.e. 193.51.193.149). A DNS server usually listens on the default UDP port 53.

### 2.4.2 HTTP

HTTP (Hyper Text Transfer Protocol) [8] is a protocol for transferring data from one server to a client. It is the protocol mostly used in Internet. It was proposed as a protocol for exchanging requests between clients and servers on the World Wide Web. The HTTP protocol formalizes the way in which Web browsers send HTTP requests and receive responses from Web sites all over the world. Normally a client sends an HTTP request for a web page and receives the response which can be text, images, sound or videos. Although HTTP can make use of any underlying transport protocol to transmit HTTP messages between clients and servers, practically all known implementations use the Transmission Control Protocol (TCP) as their transport-level protocol.

### 2.4.3 P2P

The Peer-to-Peer (P2P) [8] is a model for computer networks different than the client-server model since each client is also a server. The Peer-to-Peer model can be centralized (connections through an intermediate server) or decentralized (to make direct connections). It can be used for file sharing, scientific computing or communication. The Peer-to-Peer model allows multiple computers to communicate over a network to share objects, files in most cases, but can also be a continuous media stream (streaming), a distributed computing, or a service such as telephony with Skype. The Peer-to-Peer model has permitted the decentralization of the system previously based on a few servers, allowing all computers to play the role of client and server. It has therefore facilitated the sharing of information. This has made the peer-to-peer tools an appropriate choice to decentralize services that must ensure high availability while providing low maintenance costs. However, these systems are more complex to design than client-server systems.

The most common application of peer-to-peer is file sharing. The advent of Internet connections (ADSL in particular) with no time limit has contributed to this growth. Each user is a peer of the network and the resources are files. Everyone can share files and download files from others. These systems are very effective even when it comes to exchange large volumes of data. Among the most commonly used applications one can distinguish BitTorrent , KaZaa and eMule.

### 2.4.4 Email

E-mail [8] or email is a service for transmitting messages electronically over a computer network. Email currently runs on top of SMTP (Simple Mail Transfer Protocol). SMTP connections use TCP as their transport protocol. An SMTP server on the sending side sets up a connection with the receiving SMTP server on the well-known port 25. It first exchanges identification information about itself, followed by the intended recipient of the mail.

POP (Post Office Protocol) is a protocol for retrieving e-mails located on a mail server. This protocol has evolutes across several versions, respectively POP1, POP2 and POP3. Currently, POP3, or Post Office Protocol Version 3 is used as standard. Its operation requires a connection via TCP over port 110.

Internet Message Access Protocol (IMAP) is from its side a protocol for retrieving e-mails on mail servers. Its purpose is similar to POP3. However it was designed to leave the messages on the server. IMAP uses TCP port 143.

### 2.4.5 FTP

The File Transfer Protocol (FTP) [8] allows users to copy files to and from machines. Until the early 1990s, FTP was the primary means for exchanging large files. As a result over half of the traffic on the Internet was due to FTP. The underlying transport-level protocol for FTP is TCP. During the last years, the use of the FTP protocol becomes increasingly scarce. FTP servers use the TCP port number 21.

## 2.5 Header

Every protocol needs to transmit control data to achieve its purpose. The layering of protocols naturally leads to the perpending of this control data in the form of headers. The basic idea is that when constructing a packet, each protocol takes what is provided by the higher layer, prepends its control information to the front of the packet (and occasionally to the end of the packet), and passes the result to the next lower layer protocol, as explained in Figure 2.5. The result is that packet headers are nested. When a packet is received, the lowest layer inspects and removes its header, and passes the resulting packet up to the next higher layer.

For an Internet packet the applicative header depends on the application, the transport header depends on the transport protocol used, while the IP header is the same for different applications and protocols. So the IP addresses (source and destination) of a packet can be found in the IP header while the port numbers (source and destination) and the protocol are in the transport header (TCP / UDP). Informations about applications are therefore at the application-level header knowing that when the data is encrypted, this applicative header is also encrypted. On the other hand, different applications have different statistical properties as the packet size, the total volume of data transfer and the number of packets transferred per connection. The statistical informations about these attributes are in the TCP-IP header, and therefore, even if the traffic is encrypted (provided that we can group the packets into flows),

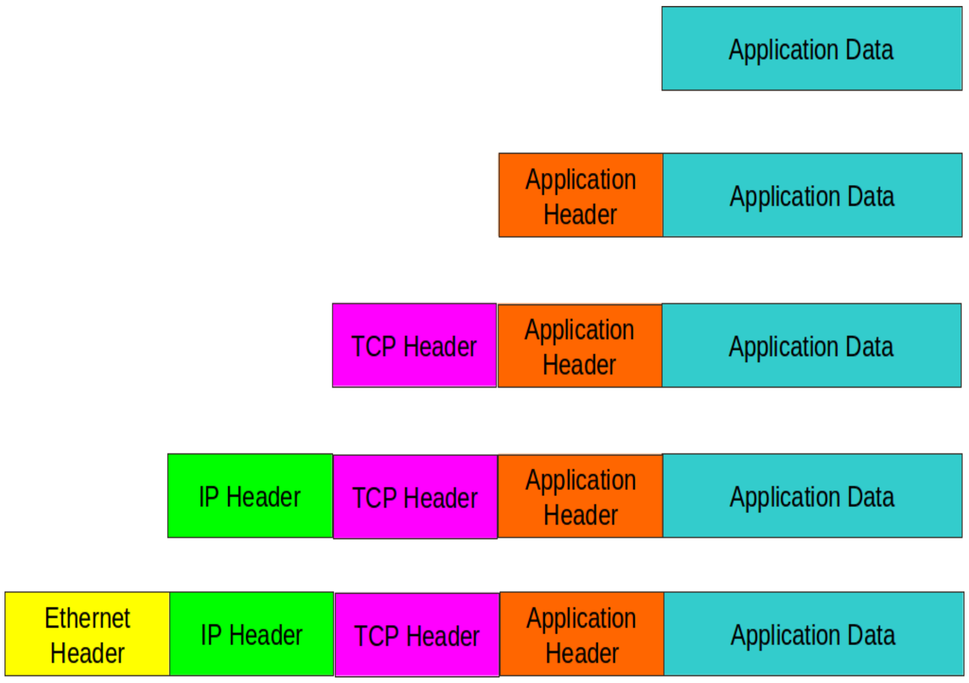

**Figure** 2.5: Different protocol header for a packet.

one can always find these informations, and thus can calculate these attributes.

## 2.6 Internet traffic identification methods

The identification of applications at the origin of Internet traffic is of major importance for network operators. On one side, this allows to treat flows in a different way based on their quality of service requirements. On the other side, it can serve for security reasons by blocking or looking closely at those users who run non legacy applications.

We can group the different methods of traffic identification into three main categories, which will be detailed in the following sections: 2.6.1, 2.6.2 and 2.6.3.

### 2.6.1 Port number based methods

The method of classification by port number is the most simple and the most traditional. The key point of this method is to associate flows to applications by using the destination port number in the Transport header (TCP or UDP). The correspondence between the port number and the sourcing application is defined by the IANA [16]. For example, DNS application commonly uses the destination port number 53 and the UDP protocol. IANA divides the port number ranges of TCP and UDP into three categories:

**Table** 2.2: Some known port numbers.

| Applications | Port number | Transport protocol |
|---|---|---|
| FTP | 21 | TCP |
| HTTP | 80 | TCP |
| HTTPS | 443 | TCP |
| POP3 | 110 | TCP |
| IMAP | 143 | TCP |
| SMTP | 25 | TCP |
| DNS | 53 | UDP |
| POPS | 995 | TCP |
| eMule | 4661-4662-4672 | TCP/UDP |
| Gnutella | 6346 | TCP/UDP |
| Bittorent | 6881 | TCP/UDP |
| SSH | 22 | TCP |
| Yahoo Messenger | 5010 | TCP |
| MSN | 1863 | TCP |

- From 0 to 1023, are the standard and well known port numbers.
- The range from 1024 to 49151 represents the registered port numbers.
- The rest from 49152 to 65535 represents the dynamic and / or private port numbers.

In Table 7.1 we cite the correspondence between the port number and some typical applications.

In theory, the destination port number (server side) allows the identification of applications. However, with the evolution of Internet services and applications (e.g. P2P applications) this method presenting several limitations and becomes unsatisfactory, as it is not always possible to associate an application to a port number for different reasons:

- The correspondence between the port number and application is not possible for some new applications such as P2P (eDonkey, Napster, Kazza,...), streaming, and others.
- Several implementations of TCP use port numbers from the registered port category. This could lead to classify erroneously these flows as belonging to the legacy application associated with the used port.
- The dynamic choice of the port number between the server and the client is sometimes possible. For example, FTP (passive) allows the dynamic negotiation of the port number

**Table** 2.3: Some protocol signatures.

| Applications | Signature |
|---|---|
| HTTP | ”http/1.” |
| SSH | ”SSH” |
| POP3 | ”+OK Password required for” |
| IRC | ”USERHOST” |
| Direct Connect | ”$MyN”,”Dir$” |
| MP2 | ”GO!!, MD5, SIZ0x2” |
| Fasttrac | ”Get /.hash” |
| eDonkey | ”0xe319010000” |
| Gnutella | ”GNUT”, ”GIV” |
| Bittorent | ”0x13Bit” |
| MSN Messenger | ”âPNGâ0x0d0a” |

used for data transfer. This port number is determined between the two ends during the initial control phase (using default port 20).

- Some applications are encapsulated in other well-known applications (they use port number associated to these latter one), such as streaming or peer-to-peer over HTTP.
- Some applications, including streaming over UDP, use port numbers that can possibly overlap the ranges used by other applications.

### 2.6.2 Deep packet inspection methods

Another popular approach is the deep packet inspection one. Methods following this approach [26, 14, 40, 9, 55, 35, 11] try to identify a characteristic bit string in the packet payload and compare it with a list of bit strings that potentially represents the control traffic of application protocols. For example ”http/1.” corresponds to the HTTP application and ”0xe319010000” corresponds to the eDonkey application. In table 7.2 we cite the correspondence between some applications and the corresponding bit strings.

In [35], Perenyi et al. develop a novel traffic classification method in order to identify P2P traffic. The method relies on a set of heuristics derived from the P2P traffic properties. They validate their method by means of one real data set obtained from one Internet service provider in Hungary. They show the high accuracy of the proposed method.

In [11], Dewes et al. take the problem of Chat Systems traffic. They develop a method that uses both the port number and the payload inspection in order to separate chat traffic from the other applications. They start by filtering some flows that have some known port numbers, then they use the signatures of the chat applications in order to separate them from the other applications.

In [26], Karagiannis et al develop a method to identify P2P traffic based on flow connection patterns without relying on packet payload. This method relies on another deep packet inspection method for the purpose of validation. They concentrate on the nine most popular P2P protocols. They argue that their method can identify more than 95% of the P2P traffic even when the deep packet inspection method fails in identifying correctly the traffic.

In [55], Sen et al. develop a deep packet inspection method to identify accurately P2P applications especially the most used ones: Bittorent, eDonkey, Gnutella, KazaA and Direct Connect. This method identifies Internet traffic by using payload signature at the application level. The authors try to found payload signatures which are highly accurate, scalable and robust. They then validate their method on two full packet traces collected in November 2003. They managed to check correctly the majority of the traffic by inspecting only few packets. They also show that a big ratio of P2P traffic use the well known port numbers (of other standard applications) and so one cannot rely on the port number in order to identify Internet P2P traffic.

In [14], Haffner et al. develop a new method based on payload inspection while developing automatically the accurate signatures for individual applications. They use three machine learning algorithms to learn application signatures, the naive Bayes, AdaBoost and Regularized Maximum Entropy. They validate their method over two traces (one for the training in order to find signatures and one to test the signatures) which contain the following applications: FTP, SMTP, POP3, IMAP, HTTPS, HTTP, and SSH. The algorithms give a very good accuracy when only 64 to 256 bytes of the payload were used.

In [40], Moore et al. develop a new payload inspection method to identify Internet traffic. They validate their method by means of a full payload packet trace from an Internet site. They showed that the classification based on well-known port numbers is not efficient and leads to a high amount of the overall traffic being unknown or misclassified.

In [9], Antoniades et al. develop Appmon, a new method for traffic classification. Appmon uses three different approaches to affect flows to applications. In a first time, they search in the application messages for characteristic application patterns. For certain applications that dynamically negotiate the port numbers, they decode the application packets to identify the new, dynamically generated port numbers and then tracks traffic flows through these ports. Finally, well-known applications using standard port numbers are identified by using the port number.

The deep packet inspection technique is an online method and quasi deterministic able to identify different types of applications in a mixture of traffic, but it also has several limitations:

- The descriptive information of the character string of the application, or the version of the application are not always available.

- With the significant increase in secured applications where the packet payload is encrypted, the method does not always recognize applications.
- The format and type of packets are different from one application to another. This information is not easy to extract, since it depends in particular on the protocol used.
- Some services that have different requirements can be encapsulated in traditional applications, such as streaming over HTTP.

### 2.6.3 Behavioral-based and statistical methods

The statistical techniques [38, 41, 67, 63, 7, 54, 45, 43, 13, 12, 5, 2, 46, 3, 59, 25] seem to be today an interesting alternative to the previous two approaches. They allow to recognize and to classify the applications according to their statistical signatures. These signatures can be volume (number of bytes) per connection, the connection duration, rate, the inter-packet delays, the packet sizes, and direction.

Most of the techniques that use statistical features require a machine learning algorithm to perform the classification of flows into applications. The machine learning algorithm used in the statistical methods can be divided into two broad categories:

- **Supervised algorithms:** They require knowledge of the application of each flow before applying the algorithm in the training phase.
- **Unsupervised algorithms:** They require no previous knowledge.

Similarly, each supervised or unsupervised algorithm consists of two main phases:

- **The learning phase;** where flows belonging to the training data set are affected to different classes according to the values of statistical criteria for each flow, and then classes and applications are associated to each other.
- **The classification phase;** where each new connection is classified to one of the classes already defined in the learning phase (each connection is assigned to the class where the distance between its attributes and attributes of the center class is the closest) .

In [38], McGregor et al. show the utility of using clustering algorithms for the identification of the traffic. They propose to use an unsupervised machine learning, called Expectation Maximization, and the following statistical criteria: packet size statistics (min, max, quartiles,etc.), inter-arrival time statistics, byte count, connection duration, the number of transitions between transaction mode and bulk transfer mode, and the idle time. All these statistical criteria are calculated over the entire flows. For the validation of their method they used two public traces from NLANR and Waikato. They considered the following applications: HTTP, SMTP, FTP (control), NTP, IMAP, DNS.

In [41], Moore et al. use the Baysian technique (Naive Bayes and Naive Bayes with Kernel Estimation and Fast Correlation-Based Filter method) in order to classify Internet traffic. They use a total of 248 statistical features among them are: flow duration, TCP port, packet inter-arrival time statistics, payload size statistics, effective bandwidth based upon entropy, Fourier transform of packet inter-arrival time. All these features are calculated on full flows, so the classification is offline. For the validation process they use a large range of database, P2P, Buck, Mail, and services Traffic.

In [67], Zander et al. develop a new statistical method which uses the AutoClass algorithm to identify Internet traffic. They use the following statistical criteria: packet size statistics (mean and variance in forward and backward direction), inter-arrival time statistics (mean and variance in forward and backward direction), flow size and flow duration. All these criteria are calculated on full flows, so the identification is off-line and not on the fly. For the validation of the method they use three public traces from NLANR (Auckland-VI, NZIX-II and Leipzig-II) which contain the following applications: Half-Life, Napster, AOL, HTTP, DNS, SMTP, Telnet and FTP. They use a port number method for the ground truth of the flows and find an average classification accuracy of 86.5 %.

In [63], Nigel et al. use four supervised clustering algorithms (Naive Bayes, C4.5 decision Tree, the Bayesian network and the Naive Bayes Tree) to classify Internet traffic. They make a comparison between two sets of statistical features in order to improve the accuracy of the classification. The first features set includes the number of packets in forward direction, the maximum forward packet length, the minimum forward packet length, the mean forward packet length, the standard deviation of forward packet length, the minimum backward packet length and the transport protocol used. The second features set includes the number of packets in forward direction, the maximum forward packet length, the mean backward packet length, the maximum backward packet length, the minimum forward inter-arrival time, the maximum forward inter-arrival time, the minimum backward inter-arrival time, the maximum backward inter-arrival time and the flow duration. They use two small public data set for the validation of their method together with a port based method for the comparison of the results. They start by using a big number of features and after wards they reduce the number of features using a correlation-based and consistency-based feature reduction algorithm. They confirm that a similar level of classification accuracy can be obtained when using several different algorithms with the same set of features and training/testing data, the accuracy is in the order of 93.76 % for the first data set and 93.14 % for the second data set.

In [7], Crotti et al. classify Internet traffic and especially the TCP traffic. They characterize each flow by an ordered sequence of $N$ pairs $P_i = (s_i, \Delta T_i)$, while $s_i$ is the size of the packet $i$ and

$\Delta T_i$ the inter-arrival time between the packet $i$ and the packet $(i-1)$. They build a PDF vector for each known application, and for each unknown flow they assign it to the application whose PDF (Probability Density Function vectors) describes it the better. They use a classification algorithm that is built mostly in an empirical way without solid statistical foundations. For the validation of their method they collected a trace with payload at the edge of the Brescia University in Italy (the trace contains HTTP, SMTP and POP3 applications). They use a payload analysis method for the result comparison. We will use this trace for the validation of our method; the trace will be described later in this thesis. The method can reach an accuracy of 90%.

In [54], Roughan et al. develop a new statistical method based on the Nearest Neighbor, the Linear Discriminate Analysis and the Quadratic Discriminant Analysis. They use packet level, flow level, connection level, intra-flow/connection, and multi-flow features. They calculate these features on full flows. For the validation of their method they use the Waikato public trace which contains the Telnet, FTP, KaZaa, Real Media Streaming, DNS and HTTPS applications.

In [45], Park et al. try to identify Internet traffic using the Naive Bayes with Kernel Estimation, the Decision Tree J48 and the Reduced Error Pruning Tree algorithms. The authors use the following features: the flow duration, the initial advertised window bytes, the number of actual data packets, the number of packets with the option PUSH, the packet sizes, the advertised window in bytes, the packet inter-arrival time and the size of the total burst of packets. For the validation they use traces from NLANR, USC/ISI and CAIDA which contains the following applications: HTTP, Telnet, Chat, FTP, P2P, multimedia, SMTP, POP, IMAP, NDS, Oracle and X11.

In [43], Nguyen and Armitage use the supervised Naive Bayes algorithm to classify Internet traffic. The authors use packet length statistics (min, max, mean, and standard deviation) and packet inter-arrival time statistics (min, max, mean, standard deviation). All these criteria are calculated over a small number (25 packets) of consecutive packets (classification window) taken at various points of the flow lifetime, where the changes in flow characteristics are significant. They validate their method by using traces collected at an on-line game server in Australia and provided by University of Twente, Netherland, which contains the following applications: On-line Game (Enemy Territory) traffic, with many other applications (HTTP, HTTPS, DNS, NTP, SMTP, Telnet, SSH, P2P ...).

In [13], Erman et al. use three unsupervised clustering algorithms (K-Means, DBSCAN and Auto Class) to classify traffic. They use the following criteria: the total number of packets, the mean packet length, the mean payload length excluding headers, the number of bytes transfered (in each direction and combined) and the mean packet inter-arrival time. For the validation they use one public trace (NLANR) and a self-collected 1-hour trace from the University of Calgary which contains the HTTP, P2P, SMTP, IMAP, POP3, MSSQL and other applications.

The authors found an accuracy of 84 % while using K-Means, 74 % for DBSCAN and 91 % for the Auto-class Algorithms. The authors show that K-Means is a very good algorithm in terms of precision and simplicity.

In another work by Erman et al. about traffic classification [12], the authors use the same statistics criteria used in [13], but they compare between two other machine learning algorithms, the Naive Bayed and the Expectation Maximization (EM). They show that the EM classifier outperforms the Naive Bayes classifier in terms of classification accuracy. The accuracy for EM can achieve more than 90 % on the data set used by the authors for the validation. They analyze the time required to build the classification models for both approaches as a function of the size of the training data set.

In [5], Bonfiglio et al. propose an approach specific to identify Skype traffic by recongnizing specific characteristics of Skype. They use the Naive Bayes and Pearso's Chi-Squate test algorithms, as well as the message size (the length of the message encapsulated into the transport layer protocol segment) and the average inter-packet gap. For the validation they use two real and self collected traces.

In [2], Bermolen et al. propose a novel methodology to classify accurately P2P-TV applications. The authors use the count sof packets and bytes exchanged among peers during small time-windows: they argue that these two counts convey a wealth of useful information, concerning several aspects of the application and its inner working, such as signaling activities and video chunk size. For the validation of their method they use two real traces, the first one was collected at the edge of the university campus and the second at the edge on one ISP in Italy.

In [46], Pietrzyk et al. evaluate the statistical classification methods as a complementary tool to deep packet inspection methods. They compared the performance of three classification algorithms, the Naive Bayes Kernel Estimation, the Bayesian Network and the C4.5 Decision Tree. They show that statistical methods can offer a high performance when they are applied to the same site where they are trained, and that by means of these methods we can discriminate between applications. On the other hand they demonstrate that these methods suffer when the training is made on traces where the ground truth is available, then used later to classify traffic in other networks. For the validation they use a big data set collected from the ADSL network of France Telecoms which contains a big set of applications (WEB, P2P, GAMES, CHAT, STREAMING etc.). They show that the C4.5 Tree Decision algorithm offers the best performance in terms of accuracy and precision.

The majority of the previously described statistical methods cannot identify flows early and they require reaching the end of the flow before taking the decision which could be too late for some applications related to network administration. The following methods do not make this assumption.

In [3], Bernaille et al. test three clustering algorithms (K-Means, Gaussian mixture model, and the Spectral clustering). The input features to assign flows to applications are the size and the direction of the first four packets (without counting the three-way handshake packets) jointly used. Bernaille et al. were the first to introduce the notion of real time Internet traffic identification, they were able to classify Internet flows directly after receiving the fourth packet. Their 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. But the method requires the first four packets of every flow in the correct order and so it cannot identify flows that do not have more than four packets (without counting the three-way handshake packets). In addition if only one of the first four packets was not collected the method cannot identify the flow.

In [32], Li et al. develop a new statistical method which tries to identify traffic on-line. They use the C4.5 Decision Tree algorithm. In order to classify traffic on-line they calculate features over a period of the flow and not over the complete flow. They discuss that there is a trade-off between accuracy, latency and throughput of the method. This trade-off depends on the choice of subset of features and size of the observation window and the classification algorithm used. They use finally 12 features (server port, client port, count of all packets with PUSH bit set in TCP header (server to client), the total number of bytes sent in initial window, (client to server  server to client), average segment size: data bytes divided by the number of packets. (client to server), median of total bytes in IP packet (client to server), count of packets with at least 1 byte of TCP data payload (server to client), variance of total bytes in packets (client to server), minimum segment size observed. (server to client), total numbers of RTT samples found (server to client), Count of all packets with push bit set in TCP header (client to server) ) calculated over 5 packets in maximum or over 5 seconds if there are no more than 5 packets. They show that by calculating a small number of features, over a small number of packets or a short duration of a traffic flow, their approach can provide good balance between accuracy, throughput and latency. The major problem with this method is the complexity of the calculation of features which presents a bottleneck for the time of classification.

In BLINC [25], Karagiannis et al. develop a new method that doesn't treat each flow as a distinct entity but considers the role of the host focusing on the source and destination hosts of the flows. The host behavior is studied across three levels: (i) social that detects the host popularity and communities of hosts, (ii) functional that identifies the functional role of a host (offered services, used services) and (iii) application that refines the classification while using other criteria (protocol, packet size), this method makes it possible to identify traffic without payload analysis and can achieve a good accuracy more than 80 %. But the method cannot identify flows without information about the IP address, hence it presents problems in case of NAT.

In [59], Trestian et al. characterize the role and type of traffic of an end-point by collecting publicly available information on the web based on the IP address of the host.

### 2.6.4 Discussion

In this section we discuss our proposed methods for traffic classification in comparison to the other traffic classification approaches in the literature. The three methods that we propose in this thesis all belong to the statistical approach and so they do not suffer from the problems of the port based method (small accuracy) and the deep packet inspection method (in case of encrypted traffic). We use a simple machine learning algorithm, K-Means described in the next section. Our methods are on-line, so we have a big advantage on the majority of methods that use statistical features calculated over the full flow (cannot identify traffic before the end of flows). The statistical metrics that we use are the packet size and the inter-arrival time between packets, and so these parameters can be extracted directly from the packet header without need for heavy processing operations. We develop a new iterative model which permits us to simplify the training and the classification phases and to start giving decision about the application when we receive the first packet of a flow. The confidence about this decision will increase with every new packet received. This iterative model allows us to decrease the time required for the classification process, hence satisfying to the real time requirement. Furthermore, in our third method we propose a new on-line method for traffic classification that combines the statistical and host-based approaches, this combination between the both approaches allows benefiting from the useful information provided by both approaches and hence to reduce the problems caused by using them separately.

In the next section we will give a general overview of clustering algorithms, especially the K-Means one that we use in our methods.

## 2.7 Clustering algorithms

Clustering is the process of organizing a big set of objects into sets of similar objects (see Figure 2.6). Clustering methods are generally based on a notion of distance between objects to analyze. They identify groups of similar objects and, in particular within these groups, patterns and interesting correlations. All clustering methods therefore seek to cut the data space into different parts (clusters) such that all points of a given cluster are closer together than any other point of another cluster [15]. Therefore, a cluster is a set of objects which are similar between them and dissimilar to the objects belonging to other clusters.

We distinguish four main categories of clustering methods:

- the partitioning algorithms;
- the hierarchical algorithms;

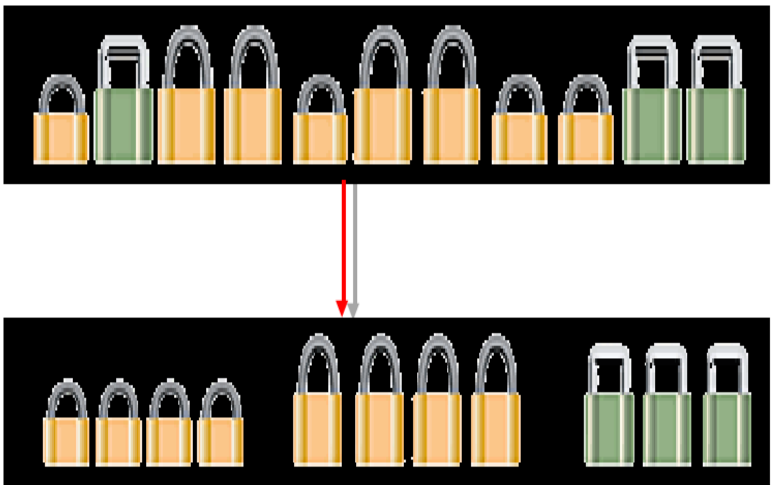

**Figure** 2.6: Example of clustering.

- the grid-based algorithms;
- the density based algorithms.

### 2.7.1 The partitioning algorithms

The basic idea of these algorithms is to divide the studied space into distinct regions. In fact, they try to get the best partitioning that optimizes a certain criterion through an iterative procedure [15]. The most known portioning algorithms are K-Means, PAM, CLARA and CLARANS. We explain them by the means of the most famous partitioning algorithm K-Means.

#### K-Means

The K-means algorithm [15] is the most popular and simplest clustering algorithm (see Figure 2.7 and 2.8) . The functioning of K-Means is summarized by the following steps:

- first, it specifies K points as initial centroid;
- then, each cluster is associated with a centroid (center point);
- the centroid is (typically) the mean of the points in the cluster;
- each point is assigned to the cluster with the closest centroid;
- closeness is measured by Euclidean distance, cosine similarity, correlation, etc;
- here K-Means recalculates the center of gravity of each class;

- then it repeats the previous steps until all clusters stabilize.

Note that for K-Means, one should specify as input the number of classes he wants. The accuracy of K-Means is based on the number of classes if the statistical criteria used are significant. K-Means then assigns connections to classes according to the similarity based on Euclidean distance. At the end, each connection must be assigned to a cluster. We cannot have points that do not belong to any cluster. The convergence to the final model needs only some iteration of the algorithm.

The complexity of K-Means is in order of O (n * K * I * d), where: n is the number of points, K is number of clusters, I is the number of iterations and d is the number of attribute.

### 2.7.2 The hierarchical algorithms

The hierarchical algorithms create a hierarchy of clusters which can be represented in a tree structure. The hierarchical methods are grouped into two categories:

- the agglomerative clustering algorithms which start with the points as individual clusters and at each step, merge the closest pair of clusters until only one cluster (or k clusters) left.
- the divisive clustering algorithms which start with one, all-inclusive cluster and at each step, split a cluster until each cluster contains a point (or there are k clusters).

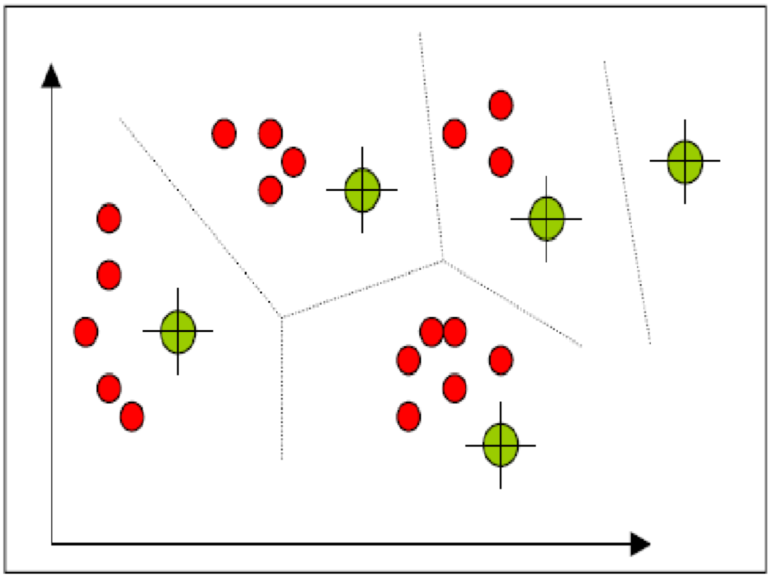

**Figure** 2.7: KMeans, step 1: assigning each object to the closest cluster.

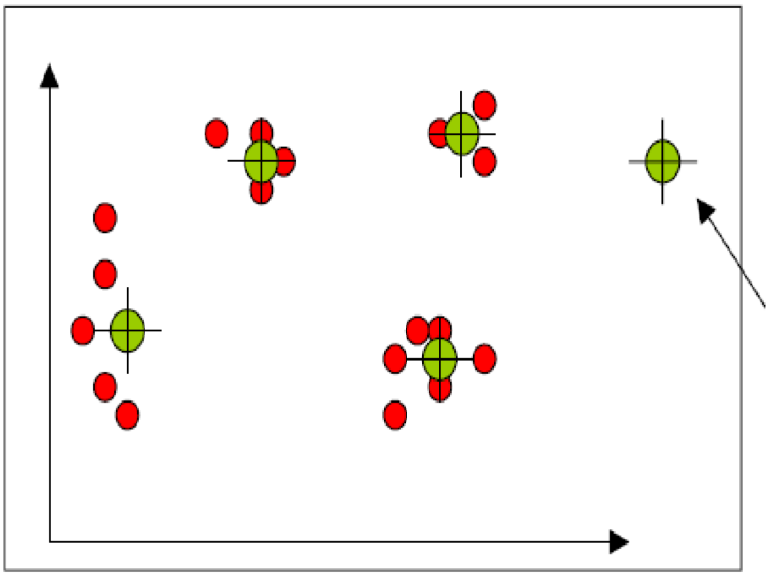

**Figure** 2.8: KMeans, step 2: recalculating clusters.

### 2.7.3 The grid-based algorithms

The methods using grids [15] perform a segmentation of space into cells and cell aggregation based on some criterion. A cell is called an elementary cell unit. These methods are generally used for spatial data. The main advantage is that the models they produce are insensitive to the order of the data. The most representative algorithm of this category is STING

### 2.7.4 The density based algorithms

The methods using the notion of density [15] require concepts of density, connectivity and border. For each subject, at least M neighboring objects are counted within a radius E-neighborhood. E-neighborhood is an input parameter. An object A is called directly density-reachable for an object B if the distance between A and B is less than the E-neighborhood radius and the number of points M for the object B is greater than MinPts, where MinPts is an entry parameter. The main advantage of methods using a notion of density is their ability to discover clusters of arbitrary shapes, as well as noise management. The most famous density based algorithm is DBSCAN.

## 2.8 Conclusions

In this chapter, we presented an overview of the TCP/IP protocol Architecture. We explained the different layers; while detailing in particular the transport layer and the application layer.

Then we described the different approaches for Internet traffic classification present in the literature. Finally we gave an overview about clustering algorithms, especially the K-Means algorithm that we use along this thesis. We will discuss more about that in the next chapter. The next chapter describes the traces we used for the validation of our methods, in addition to a presentation and motivation of the statistical criteria that we used. However, we explain our iterative model and the first method that we developed.

# 3 Enhancing Application Identification By Means Of Sequential Testing

In this chapter we will describe our on-line iterative probabilistic method that identifies applications quickly and accurately by only using the size of packets. The iterative model is a general model that can be applied to any other statistics in packets. In this Chapter we will present our first method, we will use the size and the direction of the first N packets as statistical parameters. Our method associates a configurable confidence level to the port number carried in the transport header and is able to consider a variable number of packets at the beginning of a flow. By verification on real traces we observe that even in the case of no confidence in the port number, a very high accuracy can be obtained for well known applications after few packets were examined. First of all we will explain the assumption that we made and the motivation of our work, then we explain our method and our iterative approach that we used for our three methods for traffic classification developed in this thesis. Then we will present the traces that we used for the validation of our methods and finally we present the experimental results for our first method described in this chapter. The results of this chapter were published in [19].

## 3.1 Introduction

Recently there has been a trend to use traffic statistics for the identification of applications. Indeed, it is known that different applications are governed by different end-to-end protocols and consequently they generate packets of different sizes and with different inter-packet times. Promising results have been recently found in this direction but we believe there is still room for more research to understand the capacity of the approach and of its limitations. Note that the majority of these methods use statistical parameters calculated over the complete flows and so they can't identify the traffic on-line.

In [3], Bernaille et al. test three clustering algorithms (K-Means, Gaussian mixture model, and the Spectral clustering) and the input features to assign flows to applications are the size and the direction of the first four packets (without counting the three-way handshake packets) jointly used. Bernaille et al. were the first to introduce the notion of real time Internet traffic identification, they were able to classify Internet flows directly after receiving the fourth packet. Their results show that the first four packets of a TCP connection are sufficient to classify known applications with accuracy over 90% and to identify new applications as unknown. But the method requires the first four packets of every flow in the correct order and so it can't identify flows that don't have more than four packets (without counting the three handshake packets). In addition if only one of the first four packets was not collected, the method can't identify the flow. This joint consideration of packet sizes prohibits the method from extending to more packets, otherwise the space of observations becomes complex to handle. Our idea in this work is to separate the observations to allow more generality.

In this thesis, we consider packets separately from each other, which has the main advantage of reducing the problem complexity at the expense of a small loss in performance caused by the correlation that might exist among packets. This separation is necessary to be able to consider more packets than the very few ones at the beginning of a flow. We introduce a new function that is able to quantify for each new flow the probability that a classification decision is wrong. We do this for different possible applications and for variable number of packets per flow. The function can be easily updated as long as new packets pop up from each flow. The classification is then simply to tag the flow with the most probable application. In this way we are able to evaluate with a high precision the probability of wrong or false decisions which allows a better understanding of the power of this approach and as a consequence, the proposition of more meaningful classification methods.

Another contribution of our work is that we associate a confidence level to the port number carried in the transport header of packets. This confidence level is to be set by the administrator. A low value of confidence level can be set in case of no confidence in the port number or a higher value can be set when the port number is a reliable information. Unfortunately this information on the port number has been largely overlooked in the literature. The port number has been either completely ignored or used partially as in [3] to identify flows after a first classification step. In our work, we choose to model this information by a configurable confidence level that can be set by administrators based on their experience about the traffic. The validation will show that even for a very low confidence level in the port number, we can get a very high accuracy. Clearly the accuracy increases when more confidence is associated to the port number.

The main contribution of this chapter is then in the proposition of a new statistical method for application identification that is able to scale to more than the very few packets at the beginning of a flow. As we will show, this scaling is necessary since the more packets are monitored from a flow, the higher the precision of the classification. Our observations are made on real traces that we collected ourselves on the network of INRIA Sophia Antipolis in Spring 2008. We also consider two real traces from the Brescia University for further validation. Over all these traces, one can notice the high performance of our method that is able, for example, to reach an accuracy of 98% for the first eight packets even without any confidence in the port number. Clearly, higher accuracy can be obtained if more weight is given to the value in the port number field. This accuracy is higher than what has been noticed so far in the literature.

The remainder of the chapter is structured as follows. In Section 3.2 we give more motivations to our work and we discuss our assumptions. In Section 3.3 we explain our methodology and we present our probability function. In Section 3.4 we describe the traces used to evaluate our method and in Section 3.5 we provide the validation results and discussions. The chapter is concluded in Section 6.9.

## 3.2 Method Assumptions

In this section we explain the details of our study and the assumptions we made. We are targeting a new statistical method for the identification of applications in the Internet traffic, which is able to classify flows early, on the fly, and with very high precision. A flow in our context is a TCP or a UDP connection defined by the 5-tuple information (IP addresses, port numbers and protocol). We want to be safe when our method affects a flow to an application while being able to identify the application before the end of the flow. We start by exploring the interesting method developed in [3] where the main idea was to identify traffic based on the size and the direction of the first four packets considered jointly. A precision of around 90% was announced in [3], but it has been also observed that if one uses more than four packets together, this will cause a loss in the identification accuracy. So we depart from the model in [3] but while considering more packets in order to understand the limitations of on-line statistical methods. Then, we make some further assumptions that will allow us to consider more packets while continuously increasing the accuracy. We couple this with the proposition of a new probability function to classify flows in a more accurate way. Our function is calculated on the fly, after a calibration phase by machine learning techniques to account for different application characteristics. In the following sections, we will start by studying the packet size distribution, then we will study the joint consideration of packet size and the correlation between the different packets of a flow. Then we will explain our iterative method.

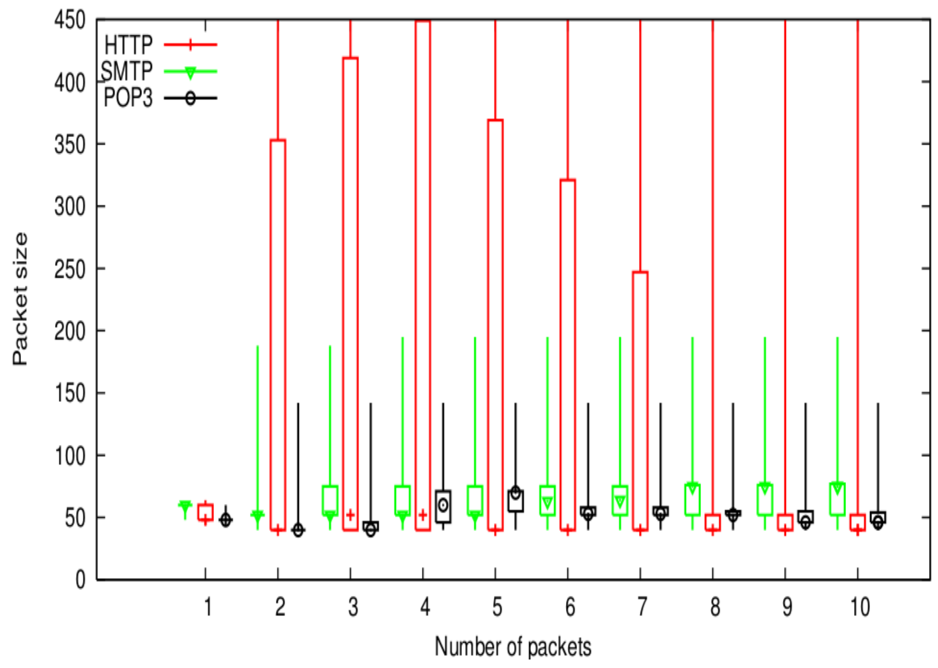

**Figure** 3.1: Packet size distribution: traffic sent by hosts inside the Brescia university campus.

### 3.2.1 Packet size distribution

We initially analyze the first 10 packet sizes of the tested flows of trace I (described in Table 3.1) to understand how well the packet size characterizes an application. Fig.( 3.1) and Fig ( 3.2) show the distributions for each packet number, where the number indicates the order in a flow; we limit the plot to 450 bytes. The results from the packets sent and received by the hosts inside the Brescia campus are shown respectively. For each packet number, the plots show the median, and the bars indicate the quartiles, and the 2nd and 98th percentiles. From the distribution we first notice that all the tested flows are initiated by the hosts (inside the Brescia university campus) with the same IP prefix because there are no first packets of a flow in Fig. 3.2.

Fig. 3.1 shows that the first packet for the three applications has the same size in almost all

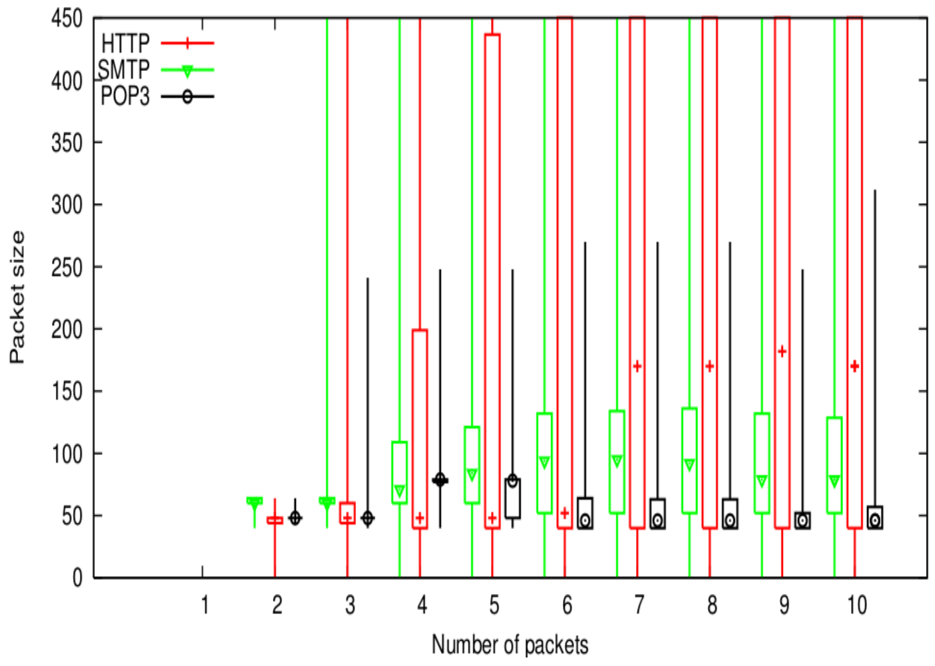

**Figure** 3.2: Packet size distribution: traffic received by hosts inside the Brescia university campus.

the flows. In particular, the SMTP packet is larger than the others. POP3 and HTTP have the same value for the median but the latter spans more values for the size of the packet, shown by the 75th percentile. The second packet has a similar distribution for the three applications. In Fig 3.2, the second packet has a size equal to the median and POP3 and HTTP have a similar distribution. By analyzing the first packets we can conclude that it is possible to distinguish an SMTP application from a POP3 or HTTP flow. From the 6th packet, the three applications have different distributions for the packet size. This means that it is possible to differentiate one application from the others both for packets sent or received by the hosts. We can conclude from these two figures that using the packet size can be a very good parameter to differentiate between Internet applications.

### 3.2.2 Joint consideration of packets

We begin to study the classification of traffic while using the size and the direction of the first N packets together (i.e. the first four together, the first five together, etc). By together we mean that the space in which we put ourselves is a multidimensional space where one dimension is associated to the size of each packet (+ and - to model the direction). Clusters are formed and associated to applications in this multidimensional space and new flows are classified accordingly. In this section, we anticipate the description of the clustering and classification procedure and of our traces used for validation to highlight an important limitation of a model considering packets jointly as in [3]. Then, we make and support the claim that studying packets separately from each other allows statistical traffic classification to scale to more packets and to provide more accuracy. We plot in Figure 3.3 the global classification accuracy as a function of the number of packets considered per flow. This figure is an average over several standard applications existing in the trace I (described in Table 3.1) and for two cluster number values as input for the K-Means algorithm. It shows that the precision of the classification increases with the number of packets until it reaches a maximum (90% while using 80 clusters and 94% while using 400 clusters) for four packets used for the classification. At that point, the precision begins to decrease until it reaches 82% with 80 clusters and 88% with 400 clusters after using 10 packets jointly for the classification. After looking closely at the numerical results to understand the reasons behind this decrease in accuracy beyond four packets, and as we observed in Section 3.2.1, we believe that this decrease is not because the packets five, six, etc are not distinctive of the different types of applications, but rather because we are using more dimensions during the classification and so the forming of clusters in the multidimensional space becomes more challenging. On one side, it is hard to find the optimal number of clusters to be used (the figure shows the results for 80 and 400). And on the other side, increasing the number of dimensions should be accompanied by an exponential increase in the number of clusters, which can become larger than what clustering algorithms like K-Means can handle in practice. This is the main reason for which we propose to consider packets separately from each other as if they come from independent observations. Each packet (first, second, third, etc) is studied separately in its own low dimensional space, then the flow is classified using a probability function (kind of likelihood function) that combines the different observations resulting from its different packets. This assumption is supported next by the low level of correlation existing between packets of a flow. The main advantage of our approach is that it reduces the complexity of the multidimensional space needed for learning packet size characteristics when packets are considered jointly. We replace this multidimensional space by a separate low dimensional space per packet (one dimension per packet if the direction is represented by signs + and -). The benefit is clearly a less complex and a less erroneous learning method and a classification accuracy that keeps increasing as long as we add more packets from each flow (for instance we refer to Fig. 3.18). In fact, the gain one gets from reducing the space complexity is much more important than what is lost by ignoring correlation among packets.

### 3.2.3 On the auto-correlation of sizes of packets within a flow

We don't claim that packet sizes are uncorrelated or that they form independent observations. We only assume this independence to ease the classification of flows provided we have learned the individual characteristics of their packet sizes. Nevertheless, the low level of auto-

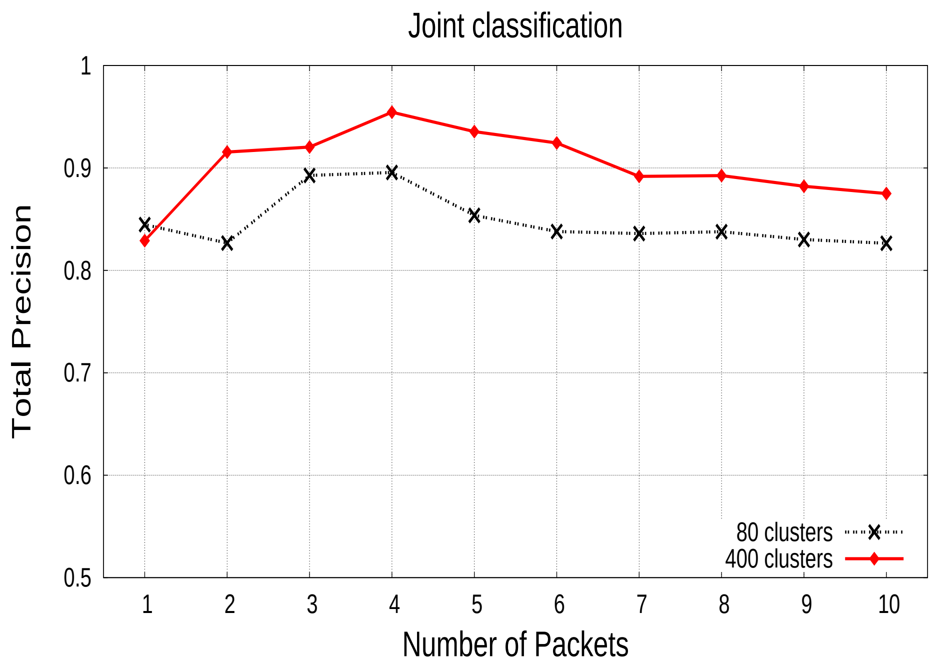

**Figure** 3.3: Precision of traffic classification when using jointly the sizes of the first N packets of each flow.

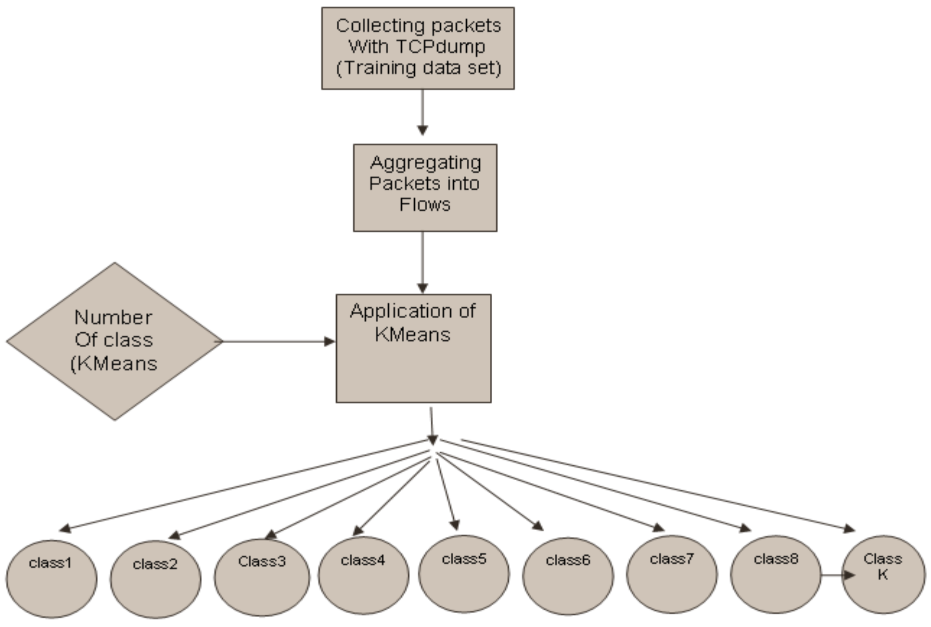

**Figure** 3.4: Model building phase.

correlation in the packet size process would help us making this assumption and would make our method even stronger. This is what we are going to check in this section.

We can evaluate the correlation between two random variables X and Y using the following correlation coefficient : $R(X, Y) = \frac{COV(X,Y)}{\sigma(X)*\sigma(Y)}$. COV is the covariance function and σ is the standard deviation. The common practice is to suppose a strong correlation between X and Y when $|R(X, Y)| \geq 0.7$ and a weak correlation when $|R(X, Y)| \leq 0.3$. We measure the value of this coefficient for the first ten packets in each flow of Internet traffic. Several applications are considered : HTTP, HTTPS, SSH, IMAP, SMTP, and POP3. Figures 3.5 and 3.6 show the correlation coefficient values between every two packets among the first ten. The X axis in the figures represents the lag. For example for lag 1, we plot the correlation value between the sizes of every two consecutive packets (packet 1 and 2, packet 2 and 3, etc). For lag 3 we consider all packets which are separated by two other packets from the same flow (packet 1 and 4, packet 2 and 5, etc), and so on. We can clearly see in these figures that the correlation value between any pair of packets, for all applications and for all lag values, is often smaller than 0.3 and in most cases even close to 0. This means that we can safely develop our method with the assumption that packet sizes are independent of each other, even if we know that this is not absolutely true. The correlation is low enough to make our method more scalable and more efficient than when considering packet sizes jointly in the learning and classification phases.

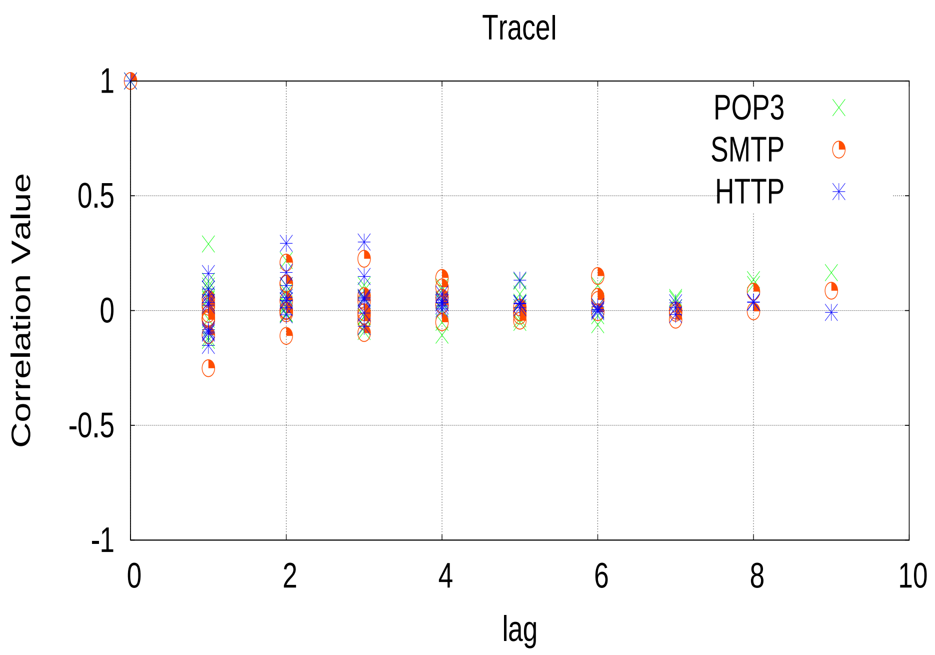

**Figure** 3.5: Correlation values between the first ten packets (POP3, SMTP, HTTP).

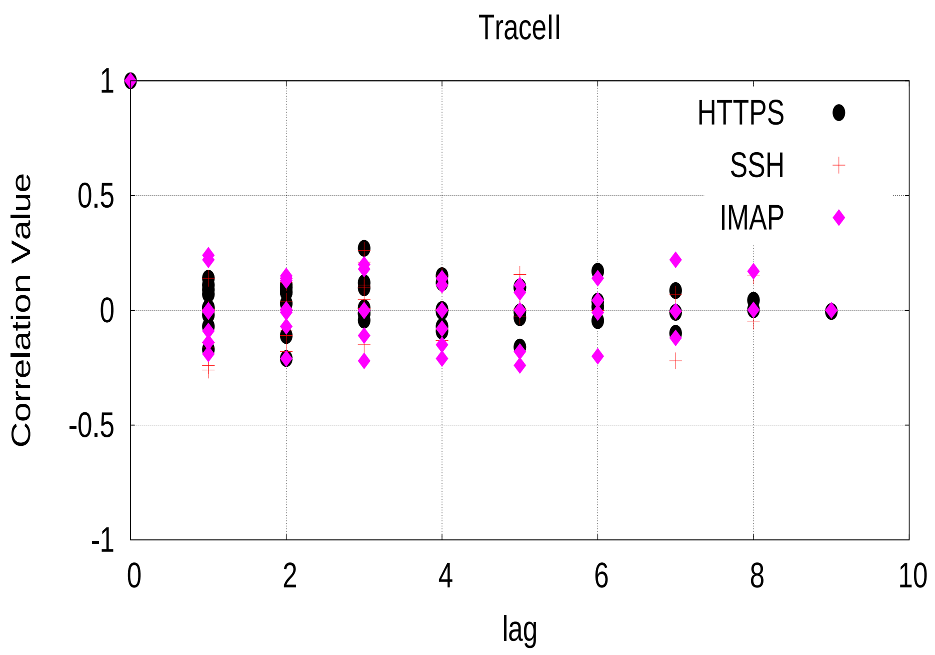

**Figure** 3.6: Correlation values between the first ten packets (HTTPS, SSH, IMAP).

## 3.3 Method Description

In order to benefit from information carried by the first N packets of a flow and to avoid problems during the clustering phase caused by the use of many parameters, we resort to an iterative packet-based approach where we use the size and the direction of every packet individually to calculate the likelihood of originating from this or that application, then we merge the results from all packets of a flow together using a probability function that we introduce to associate the flow to the most probable application. Note that we will also consider the time between packets, more details will be presented in the next chapter with our second method.

Our approach consists of three main phases: the model building phase, the classification phase and the application probability and labeling phase.

### 3.3.1 Model building phase

Also called learning phase, this phase is very important in our work. It construct a set of clusters or models (using a training data set) which we use later in the traffic classification phase. As we study each packet individually, we build a separate model for each packet using all flows in the training data set (model for the first packet, model for the second packet, etc). These models describe how the size of a packet, say the first one, varies over the different applications and how it occupies the different parts of the packet size space.

Let us describe the method that we use to generate models. We begin by explaining how we create good learning traces. Then, we describe how we represent spatially the flows in these traces. To cluster the flows in the space, we use the K-Means algorithm [64].

We choose the training traces such that they are representative of the applications to identify. To this end, we take a similar number of flows from all applications because if the number of flows is not the same, applications that predominate may bias the clustering and the classification afterwards. The number of flows is made equal by random selection from applications having more flows than necessary. To build the model for a given packet size (say the size of the i-th packet from a flow), we represent each flow as a point on an axis. This point has a positive coordinate if the packet is sent by the client and a negative coordinate if the packet is sent by the server. The coordinate of this point is equal to the size of the packet. When the time between packets is used, a flow is represented by two coordinates on two axis, one for the packet size and one for the time between this packet and the previous one (or the next one). Without loss of generality, we consider a maximum of ten packets per flow. At the end of the model building phase, we get ten models for the first ten packets from any flow. In each model (see Figure 3.4), one can have foe example 200 classes (or clusters). Note that 200 was shown to be a good trade-off between complexity and precision (see results later). Nevertheless, the performance of our method is of low dependence on this number of classes as we use only one dimension for the training. Then, for each application and for each class we associate a specific probability proportional to the number of flows from this application present in the class. This probability models the likelihood that a packet from this application falls in this class in general. As we have the same number of flows from all applications in the training trace, we define the probability of obtaining class $i$ knowing the application $A$, noted by $Pr(i/A)$, as the number of flows that belong to the application $A$ in class $i$, $F_{A,i}$, divided by the total number of flows belonging to the application $A$ in the training trace, $F_A$. We have $Pr(i/A) = \frac{F_{A,i}}{F_A}$. For example, in a class where we have 400 flows (300 WEB, 60 HTTPS and 40 SMTP), and we have a total of 8000 flows for each application in the training trace, we associate this class to these three applications following these probabilities:

$Pr(i/WEB) = \frac{300}{8000}$, $Pr(i/HTTPS) = \frac{60}{8000}$, $Pr(i/SMTP) = \frac{40}{8000}$.

### 3.3.2 Classification Phase

Consists in using the models built in the learning phase to classify traffic online. Note that here we affect flow packets to classes and not to applications. Each time there is a new packet from a flow, it is classified independently of the other packets using the model corresponding to its position within the flow. For example, when we capture the first packet of a new flow, we affect the packet (and hence the flow) to one class of the model built for all packets which are first in their flows. The same for the second packet, and so on.

To carry this association flow-class in a given model, we calculate the Euclidean distance that separates the point corresponding to the new flow in the model space from the center of each class and we affect it to the closest one. We repeat this classification for all packets from a flow until we are satisfied or we reach some threshold. The way the satisfaction is measured is done via a new probability function to be described later. This function uses the per-packet and per-class probabilities calculated in the model building phase and identified during the classification phase. Clearly, the classification of a flow is different and independent from one packet to another, hence the result of the classification. For example, a new flow can be affected to the class number 5 using the first packet and to the class number 19 using the second one, and so on. It is this set of classification results that will decide on the most probable application to which the flow would belong.

### 3.3.3 Application probability and labeling phase

We affect here flows to applications while relying on the results of the classification phase. The classification phase indicates the probability that each packet of a flow belongs to this or that application. These are the functions $\mathrm{Pr}(i/A)$ obtained from the classes in which the packets fall. We combine (on the fly) these probabilities together to obtain a new value evaluating how well we do if we associate the entire flow to this or that application. This leads to an online iterative application probability function that we use for assignment and identification. Let us define the following variables to be used next :

- $i$: used to denote classes.
- $A$: used to denote applications.
- $N$: the maximum number of packets tested from a flow.
- $k$: the test number $k$ (packet 1, packet 2, etc).
- $NA$: the total number of applications.
- $\alpha_A$: the value (between 0 and 1) affected by the administrator to the confidence in the standarized port number relative to application $A$.
- $F_{A,i}$: the number of flows belonging to application $A$ and class $i$ in the training trace.
- $F_A$: the total number of flows belonging to application $A$ in the training trace.
- $\mathrm{Pr}(A/\mathrm{Result})$: the probability that a flow belongs to application $A$ knowing the results of the classification phase (i.e. class $i(1)$ for the first packet, class $i(2)$ for the second packet and so on).

- $Pr(i(k)/A)$: the probability that the k-th packet of a flow from application $A$ falls in class $i$. This can be calculated from the training trace as $Pr(i(k)/A) = \frac{F_{A,i}}{F_A}$.
- $Pr(A)$: the probability that any flow randomly selected comes from application $A$, independently of any information on its packet sizes and times. The sum of these probabilities over all applications under consideration must be equal to 1. We integrate in this probability the confidence in the port number carried in the transport header of each packet. For example, for a given flow, and if the port number is equal to 80 (the standard WEB port number), we give $Pr(WEB)$ the value $\alpha_{80}$ set between 0 and 1 (specified by the network administrator as a function of how confident he is in port 80 being only web). This value models the probability that a flow carrying the port number 80 belongs to the WEB application. For all other applications we give $Pr(A)$ the same value that is equal to : $\frac{(1-\alpha_{80})}{(NA-1)}$.
  Note that the administrator might not give any weight to the port number. This can be the case when he does not trust this information. Here, we give the $Pr(A)$ the same value for all applications independently of what is carried in the port number field. This specific value is equal to: $\frac{1}{NA}$.

We aim at calculating the probability that a flow belongs to an application $A$ given the results of the classification phase applied to the first, let's say $N$, packets of the flow. Take for example the first two packets and their corresponding classes $i(1)$ and $i(2)$. The probability we are looking for can be written as follows:

$$\begin{aligned}
Pr(A/(i(1) \cap i(2))) = \frac{Pr(A \cap (i(1) \cap i(2)))}{Pr(i(1) \cap i(2))} &= \frac{Pr(A) * Pr((i(1) \cap i(2))/A)}{Pr(i(1) \cap i(2))} \\
&= \frac{Pr(A) * Pr(i(1)/A) * Pr(i(2)/A)}{Pr(i(1) \cap i(2))} \\
&= \frac{Pr(A) * \prod_{k=1}^{2} Pr(i(k)/A)}{\sum_{A=1}^{NA} Pr(A) * \prod_{k=1}^{2} Pr(i(k)/A)}
\end{aligned}$$

Now, we can generalize this expression to calculate the probability that a flow belongs to an application $A$, given the classification results for the first $N$ packets :

$$Pr(A/Result) = \frac{Pr(A) * \prod_{k=1}^{N} Pr(i(k)/A)}{\sum_{A=1}^{NA} Pr(A) * \prod_{k=1}^{N} Pr(i(k)/A)}$$

We call this probability the assignment probability and we use it to decide on how well the profile of packet sizes of a new flow fits some application $A$. For each new flow and when we capture the first packet (except the SYN packet), we first classify the flow according to this packet and we calculate the probability that it belongs to each application. Then, we take the highest assignment probability and we compare it with a threshold $th$ specified by the network administrator. If this probability is greater than the threshold $th$, we label the flow by the application, otherwise we take and classify the next packet and we recalculate the assignment probability using the results of the classification phase obtained for the first and second packets separately. We check again the resulting probability and we keep adding more packets until the threshold is exceeded or a maximum allowed number of tests is reached.

## 3.4 Trace Description

In this section we will explain the three real traces (described in Table 3.1) that we use for the validation of our methods. The first trace, noted Trace I, is collected at the edge gateway of Brescia University's campus network in Italy (used and described in [7]). This trace is made up of three standard applications: HTTP (WEB), SMTP and POP3. The second trace, noted Trace II, is collected by us at the edge of INRIA Sophia Antipolis network in France during Spring 2008. Trace II is made up of five standard applications: HTTP (WEB), HTTPS, IMAP, SSH and SMTP. We divide every trace into a training trace and a validation trace. The training part is used to construct the models and the validation part is used to evaluate how well our iterative packet-based method behaves in identifying the application behind each flow. Note that we made sure that there were enough flows from each application in each trace so that our learning phase can provide representative models and our validation phase meaningful results.

To well calibrate and evaluate our classification method, we need to know the real application associated with each flow. For the first trace, the authors use a method based on deep packet inspection to infer real applications. For the INRIA trace, we use tcpdump ([58]) to measure each application separately at the interface of the server reserved by INRIA to this application (for instance we collect the HTTP flows from the interface of INRIA's WEB server, and so forth). Since servers at INRIA are dedicated to unique applications, we are sure this way about the real application behind each collected flow. Traffic coming from the different servers is then mixed together to form one large trace.

The third trace, noted trace III [17], was collected on three consecutive working days during fall 2009 at the edge gateway of Brescia University's campus network in Italy (described in [17]). This trace is made up of four applications: HTTP (WEB), HTTPS, EDONKEY and BITTORENT. For the ground truth of applications, a deep packet inspection method is used.

In the learning phase we use a training set extracted from these traces, which consists of an equal number of flows per application to ensure that there is no bias in our learning.

**Table** 3.1: Traces Description.

| Trace name | Source | Date | Applications | Ground Truth validation |
|---|---|---|---|---|
| Trace I [7] | Brescia University | April 2006 | HTTP<br>SMTP<br>POP3 | Deep packet inspection |
| Trace II | INRIA Laboratory | Spring 2008 | HTTP<br>SMTP<br>HTTPS<br>SSH<br>IMAP | collected from servers |
| Trace III [17] | Brescia University | Fall 2009 | HTTP<br>HTTPS<br>EDONKEY<br>BITTORENT | Deep packet inspection |

## 3.5 Experimental Results

In this section, we evaluate the overall effectiveness of our method. The metrics used for the evaluation are:

- *False Positive (FP) rate* is the percentage of flows of other applications classified as belonging to an application $A$.
- *True Positive (TP) rate* is the percentage of flows of application $A$ correctly classified.
- *Precision* is the ratio of flows that are correctly assigned to an application, $TP/(TP + FP)$. The overall precision is the weighted average over all applications given the number of flows per application.

We present the results of our method as a function of the number of packets classified per flow and the weight affected to the port number (specified by the network administrator). For instance a port number weight equal to $0.5$ means that the chance that the flow comes from the standard application associated to this port number, $\Pr(A)$, is equal to 50% while the chance that the flow does not come from this standard application is also 50% (see definition of $\Pr(A)$ in Section 3.3.3). Note that one can also reflect in $\Pr(A)$ the proportion of flows from each application. Indeed, if the WEB for example forms the majority of the traffic, there is a high chance that a new flow belongs to WEB. In our validation, we don't account for this factor and we only calculate $\Pr(A)$ using information on the port number.

To associate flows to applications we set the threshold $th$ to the maximum value equal to 1 and we fix a maximum number of tests that we vary. When the maximum number of tests is reached, we associate the flow to the application having the maximum assignment probability. This guarantees that we will wait the maximum number of tests to decide and thus to have the real result for every packet number value. In practice the administrator can set the threshold to a value less than 1 if he wants to leave the classification when the method reaches this value for its probability function.

### 3.5.1 Number of clusters

As we discussed before, one should give to the K-Means algorithm the number of clusters that he needs. In order to choose the best number of clusters, we run our method with different values and we calculate the total precision for Trace I. In Fig. 3.7 we plot the total precision of our method as a function of the number of packet used for the classification. The different lines represent the result for a different value for the number of clusters used for K-Means. We can observe clearly that the precision of the classification increases when the number of clusters used increases. The best precision for our method can be obtained when 400 clusters are used. When we use 200 clusters, we obtain a precision close to the precision obtained for 400 clusters, the difference is around 1% approximately. On the other hand the complexity of our method increases with the number of clusters used, Whether in the training or the classification phase. The complexity in the training phase means the time to generate the model, and in the classification phase the time to affect a flow to a class as we need to compute the Euclidean distance to the center of all clusters. Throughout this thesis, we will work with 200 clusters, we choose this value as a good trade-off between the precision and the complexity.

### 3.5.2 True positive ratio

In Figure 3.8, we present the true positive ratio for the HTTP application (Trace I) and this is for several values of the port number weight. We can observe that without using any weight for the port number ($\alpha_A$ set to $1/NA$), we can obtain a true positive ratio that exceeds 99% at the tenth packet. The true positive ratio is around 64% when we use only the first packet, and 65% after using the second packet, around 94% after using four packets. We can also observe that when we give some weight to the port number we get more precision for the first three packets. By using more packets the nature of our iterative model makes the true positive ratio for all the values of $\alpha_A$ increase, and converge together to a high precision.

In Figure 3.9, we present the true positive ratio for the POP3 application (Trace I) for different values of $\alpha_A$. We notice that without using any weight for the port number the true positive ratio for the POP3 is around 57% for the first packet, then it keeps increasing when

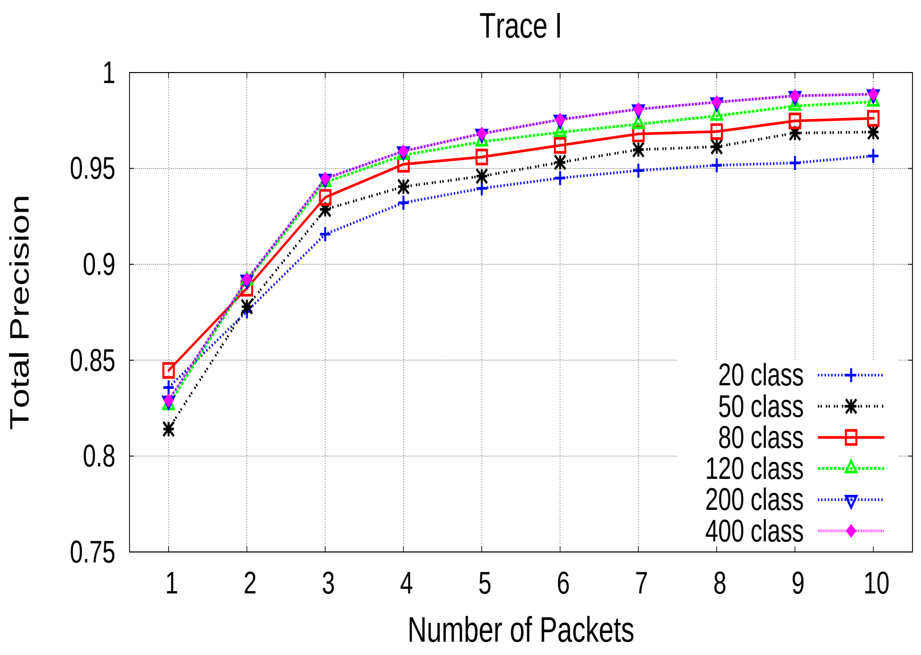

**Figure** 3.7: Average total precision for different clusters number (Trace I).

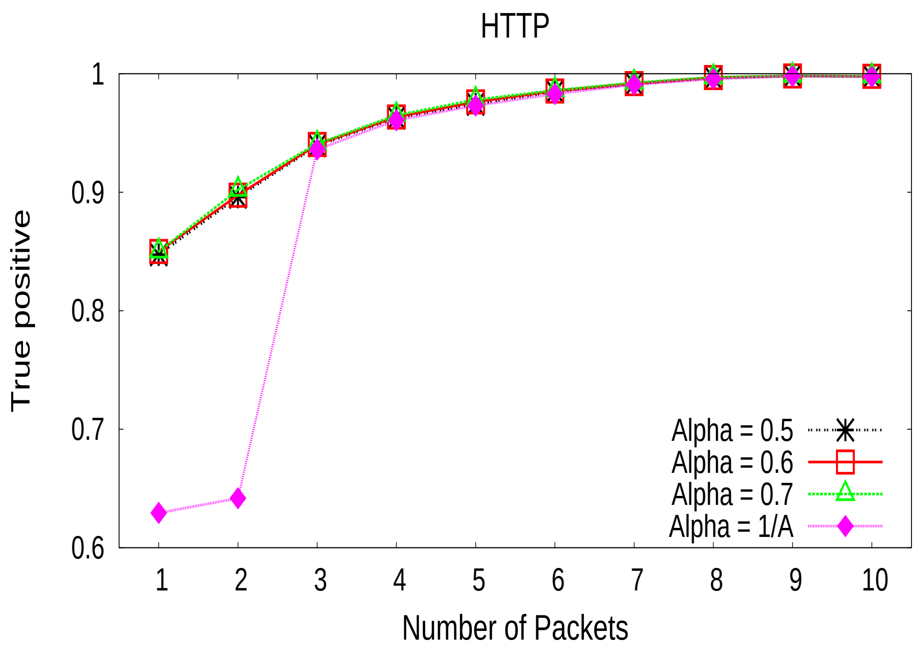

**Figure** 3.8: Average True positive ratio for The HTTP application (Trace I).

we use more packets. This precision reaches 90% after using four packets and 98% after 10 packets. When we use more weight for the port number the true precision starts around 90% after only the first packet and keep increasing to reach 98% after 10 packets. This increase in the precision when we give more weight to the port number is normal as we use legacy applications that respect the standard port number.

The true positive ratio of the SMTP application (trace I) is plotted in Figure 3.10. It starts around 90% after the first packet for the different values of $\alpha_A$.

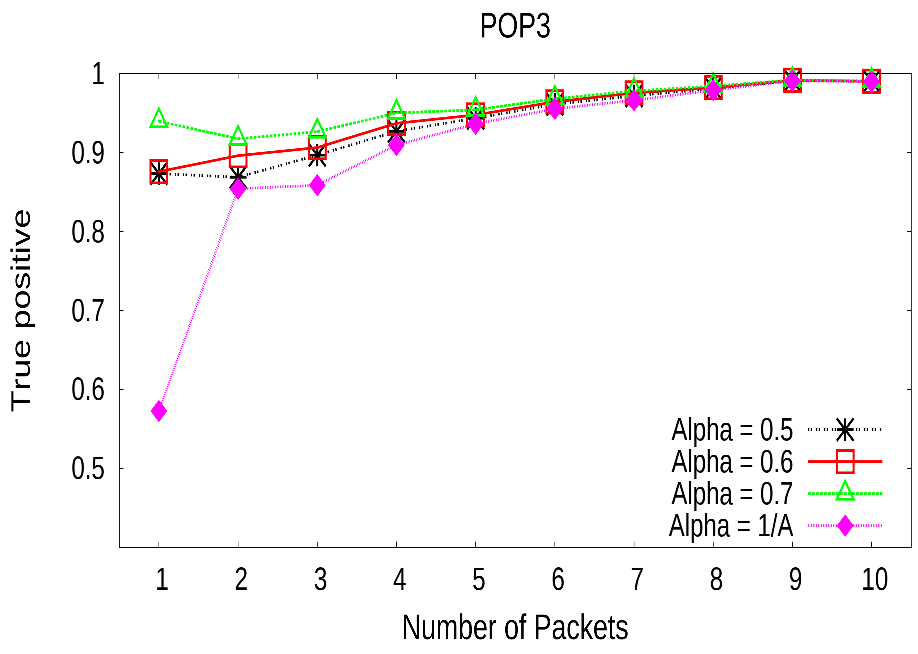

**Figure** 3.9: Average True positive ratio for The POP3 application (Trace I).

In Figure 3.11 we present the true positive ratio for the HTTP, HTTPS, IMAP, SSH and SMTP applications (Trace II). All the results are for $\alpha_A$ equal to $1/NA$, so without any weight for port number. The true positive ratio for all the applications keep increasing with each new packet used for the classification. After 10 packets, this precision reaches 97% approximately for all the applications.

In Figure 3.12 we present the true positive ratio for the HTTP, HTTPS, EDONKEY and BITTORENT applications (Trace III). All the results are for $\alpha_A$ equal to $1/NA$, so without any weight for port number. As before the true positive ratio for all the applications keep increasing with each new packet used. For the HTTP, the true positive ratio is more than 95% after the first packet and keeps increasing to reach 99% after 10 packets. For the HTTPS application, the true positive is around 85% for the first packet, and reaches 93% approximately after 10 packets. For the EDONKEY application, the true positive ratio starts with 64% for the first packet, reaches

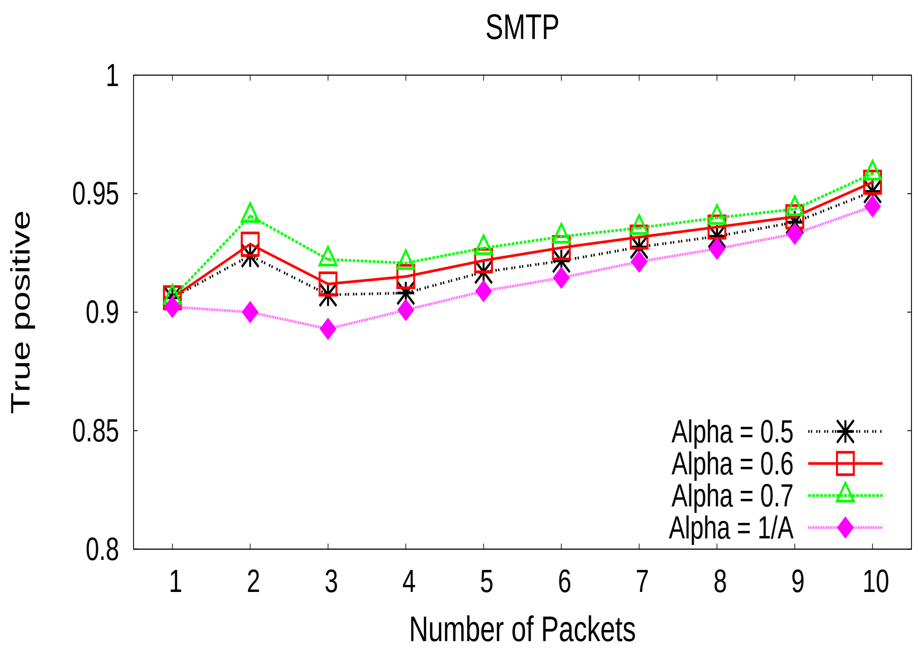

**Figure** 3.10: Average True positive ratio for The SMTP application (Trace I).

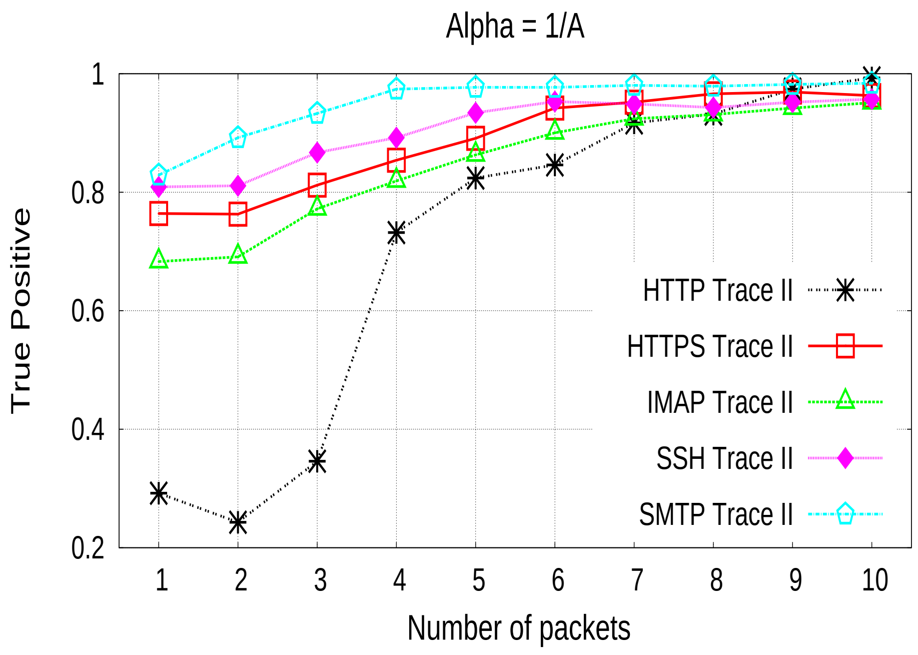

**Figure** 3.11: Average true positive ratio for the different applications (Trace II).

95% after five packets and 98% after 10 packets. For the BITTORENT application we have a very good precision which exceeds 97% even when we use only one packet for the classification.

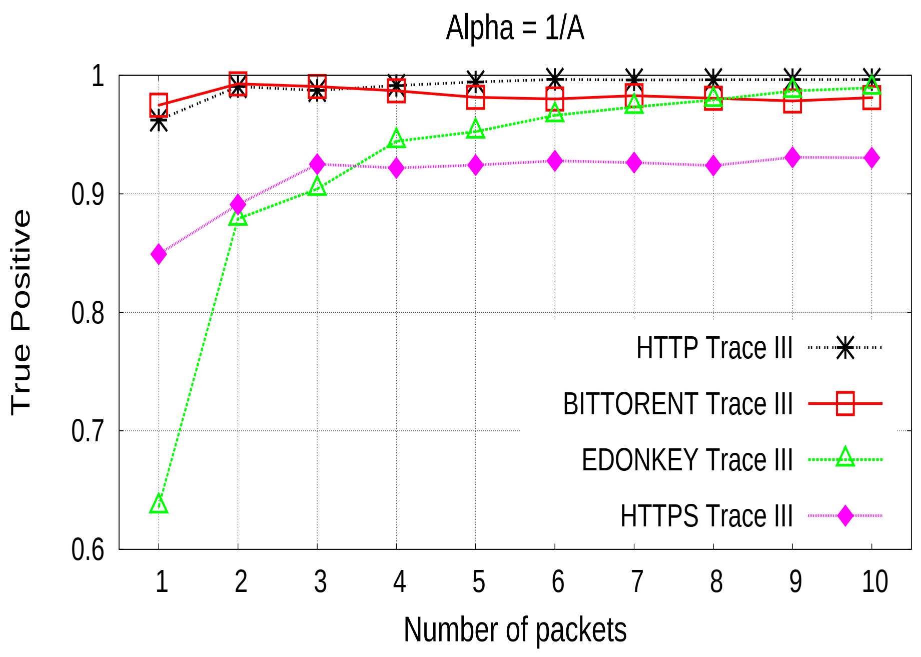

**Figure** 3.12: Average true positive ratio for the different applications (Trace III).

We can conclude from these results that we can classify the different applications with very high accuracy, without using any weight for the port number. This increases the robustness of our method because relying strongly on the port number can decrease the precision when an application uses the port number of another application.

### 3.5.3 False positive ratio

In Figure 3.13, we present the false positive ratio for the HTTP application (Trace I) and this is for several values of the port number weight. We can observe that without using any weight for the port number ($\alpha_A$ set to 1/NA), the false positive ratio starts with 20% for the first packet, and keeps decreasing until it reaches 5% after the second packet and finishes by converging to 0% approximately after the sixth packet.

In Figure 3.14, we present the false positive ratio for the POP3 application (Trace I) and this is for several values of the port number weight. We can observe that without using any weight for the port number ($\alpha_A$ set to 1/NA), the false positive ratio starst with 10% for the first packet, it increases a litle bit for the second packet and then it keeps decreasing until it converges to 4% after 4 packet . We can observe the same results for the other weights of the

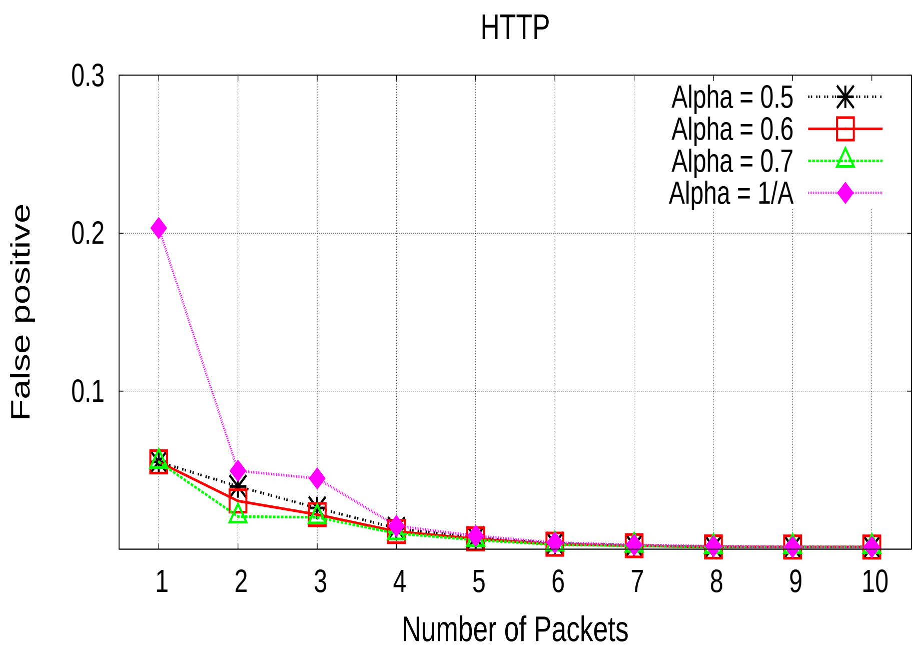

**Figure** 3.13: Average False positive ratio for The HTTP application (Trace I).

port number.

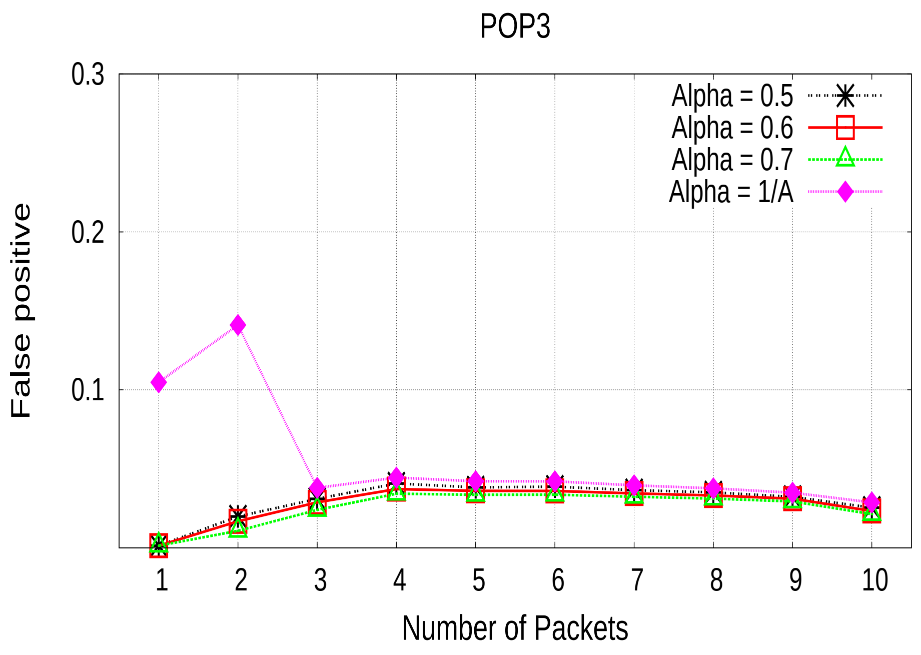

**Figure** 3.14: Average False positive ratio for The POP3 application (Trace I).

For the SMTP application (Trace I) and for all the values of $\alpha_A$ we can see clearly in Figure 3.15 that the false positive ratio is around 4% after four packets, then it decreases and reaches 1% approximately after 9 packets.

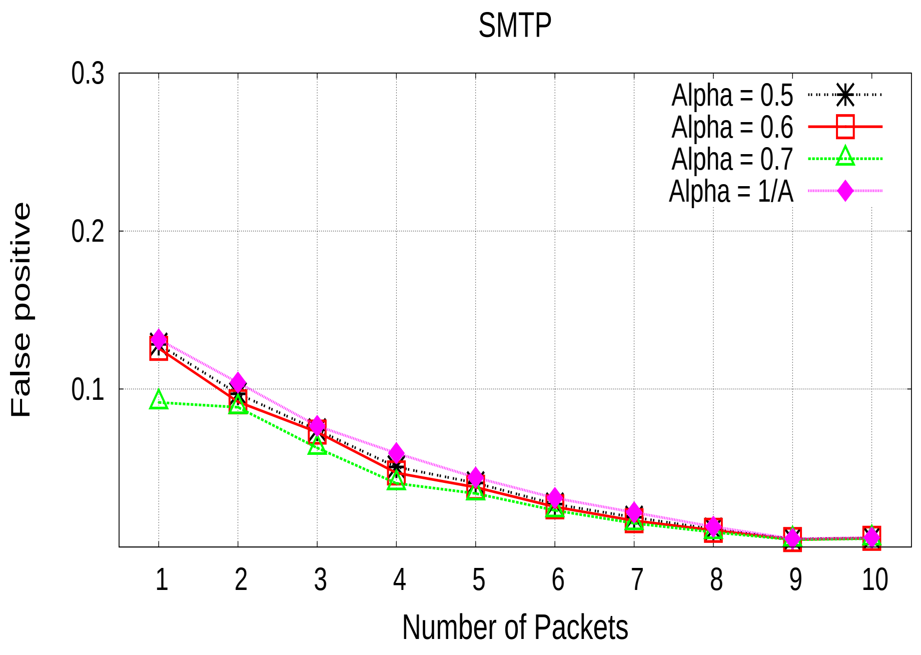

**Figure** 3.15: Average False positive ratio for The SMTP application (Trace I).

In Figure 3.16 we present the false positive ratio for the HTTP, HTTPS, IMAP, SSH and SMTP applications (Trace II). All the results are for $\alpha_A$ equal to $1/NA$, so without any weight for port number. The false positive ratio for all the applications keep decreasing with each new packet used for the classification. After using 10 packets, the false positive ratio is around 3% approximately for all the applications.

In Figure 3.17 we present the false positive ratio for the HTTP, HTTPS, EDONKEY and BITTORENT applications (Trace III). All the results are for $\alpha_A$ equal to $1/NA$. The false positive ratio for all the applications keep decreasing with each new packets used for the classification. This false positive ratio end up by being less than 4% for all the applications.

We can conclude from these results that our method is very accurate, especially when we use more and more packets for the classification. The iterative approach always permits to increase the precision with every new packet added.

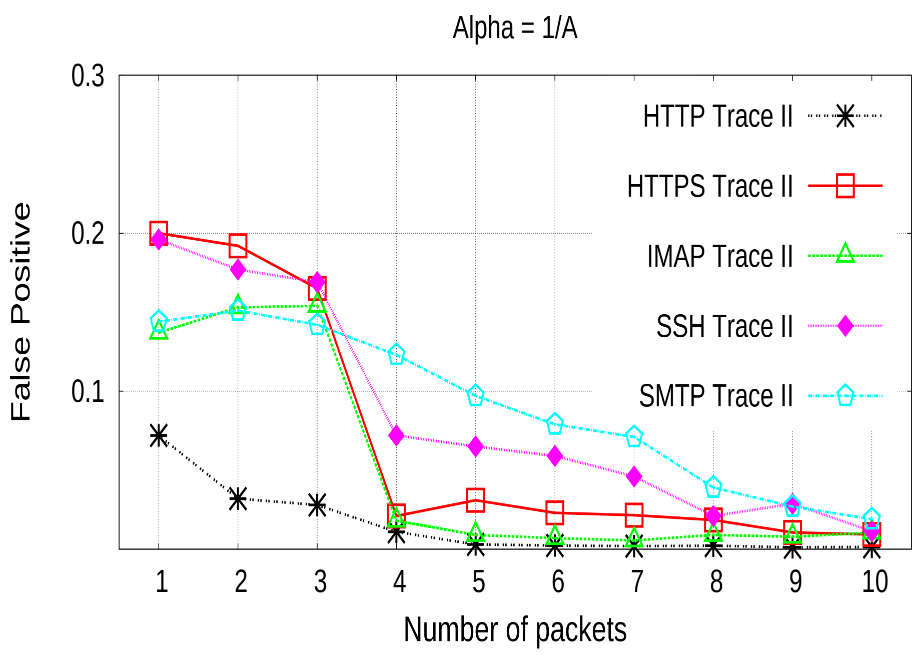

**Figure** 3.16: Average false positive ratio for the different applications (Trace II).

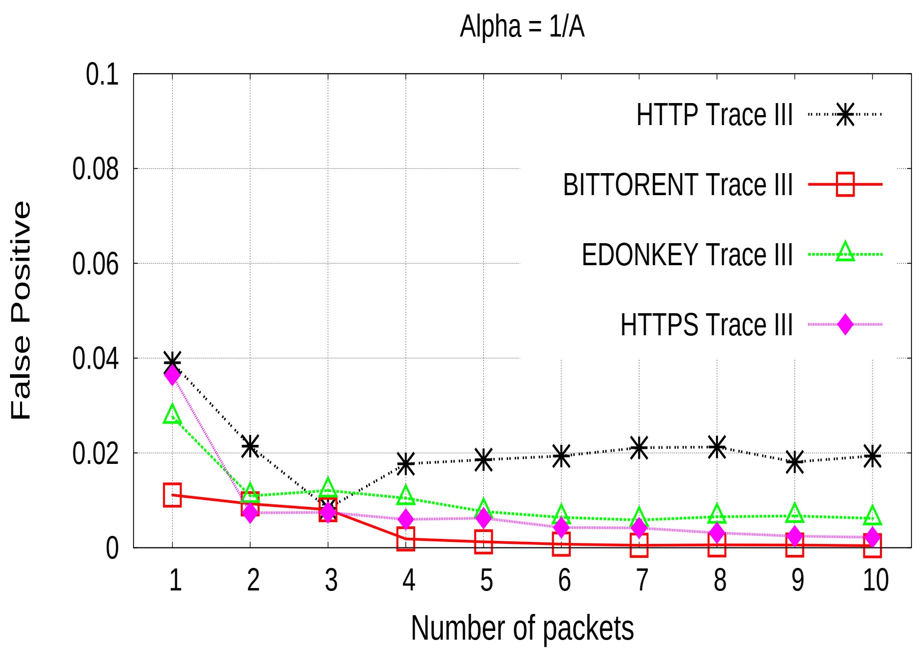

**Figure** 3.17: Average false positive ratio for the different applications (Trace III).

### 3.5.4 Total precision

In Figure 3.18 we present the average total precision of our method over for the Trace I and for several values of the port number weight. We can see that without using any weight for the port number, the precision of our method increases over 94% after the third packet and reaches 99% approximately after ten packets. Clearly, one can get higher precision by assigning more weight to the port number especially for the first three packets.

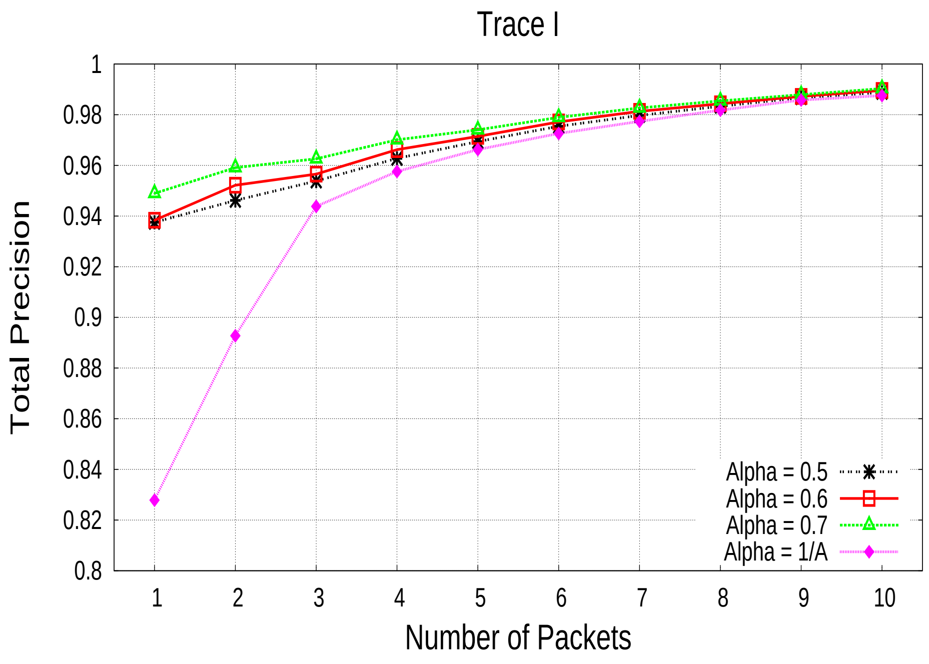

**Figure** 3.18: Average total precision for different values of Alpha (Trace I).

## 3.6 Conclusions

In this chapter we have proposed and applied a statistical traffic classification technique based on learning and analysis the size and the direction of the first packets of flows. By considering packets separately from each other and with the help of a new probability function that combines the observations made on the different packets from a flow, we were able to obtain a high classification accuracy that kept improving by adding more packets from each flow, and that was able to reach high levels of around 98% and even more. Eight applications were considered and validation was done on three real traces. In the next chapter we will apply our iterative model while using the inter-packets time as parameter.

# 4

# Can we trust the inter-packet time for traffic classification?

In this chapter we make a complete study about the inter-packet time to prove that it is also a valuable information for the classification of Internet traffic. We discuss how to isolate the noise due to the network conditions and extract the time generated by the application. We present a model to preprocess the inter-packet time and use the result as input to the learning process. We discuss an iterative approach for the on-line identification of the applications and we evaluate our method on two different real traces previously described in Chapter 3. The results show that the inter-packet time can be indeed transformed into an important parameter to classify Internet traffic. The results of this chapter were published in [21].

## 4.1 Introduction

The majority of statistical techniques [3, 66, 65, 29] in the literature rely on the packet size to identify the applications. They discard the inter-packet time for not being a good information to differentiate between applications as it depends on the network status. [3] shows that using the size and the direction of the first four packets is a good method to differentiate between applications and that we cannot rely on the inter-packet time because of its dependency on the network conditions. [66] also discusses that the parameters based on the packet size are preferred to the parameters based on the inter-packet time. [29] shows that using the inter-packet time does not cause a significant increase in the precision of the classification. In our iterative method described in 3, we find that the precision of our iterative classification method decreases when we use the inter-packet time jointly to the packet size. This is the starting point of the second part of our work about traffic classification, described in this Chapter, where we study the inter-packet time and analyze the causes of the decrease in performance. Our intent is to extrapolate relevant information from the inter-packet time and use it as a feature to help the classification of the Internet traffic. We believe that any data, such as inter-packet time, packet size, and direction of the packets, is relevant to identify an application if we can properly extract the information that characterize the behavior of the application.

In this chapter we present a complete study about the inter-packet time. We first propose a model for the inter-packet time to analyze which factors contribute to the inter-arrival time between two packets. We then propose a solution to distinguish the network delay introduced by the application. We use our iterative method developed in Chapter 3 to classify the Internet traffic on line. Our statistical method is able to extend the classification to any number of inter-packet times per flow, compared to the majority of previous works that require to reach the end of the flow before taking the decision, which could be too late for some applications related to network administration. We consider inter-packet times separately from each other which has the main advantage of reducing the problem complexity at the expense of a small loss in performance caused by the correlation that might exist among inter-packet times. This separation is necessary in order to consider more inter-packet times than the very few ones at the beginning of the flow.

We evaluate our method on two different real traces and we discuss the results when we preprocess the inter-packet times without filtering the network delays and when they are filtered out. We show that the inter-packet time is a meaningful parameter to identify applications and that the precision of the classification increases from 80% to 98% for all applications after filtering the noise coming from the network.

The rest of the chapter is organized as follows. Section 4.2 discusses the inter-packet time and presenst our model. Section 5.3 reviews our method and its application to the inter-packet time. Section 5.4 and Section 5.5 describe the traces and the evaluation results respectively. Section 5.6 concludes the chapter.

## 4.2 Model Description

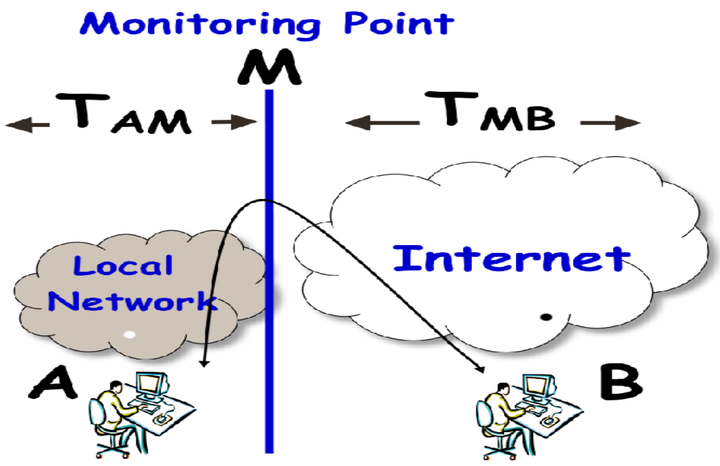

**Figure** 4.1: The system.

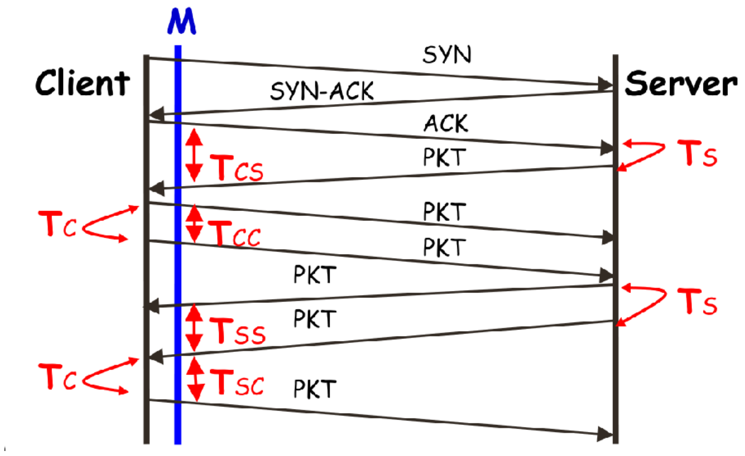

**Figure** 4.2: Inter-packet times.

In this section we analyze the inter-packet time and its use as parameter to classify Internet applications. Most of the recent literature in traffic classification [3, 66, 67, 65, 29] argues that the inter-packet time is not an informative parameter to characterize and distinguish between applications. Indeed, the timing between subsequent packets is not only function of the application data availability, but also of the size of the TCP congestion window and the network conditions. Starting from these observations, we interpret the inter-packet time and model it to filter out the noise and to extract the time introduced by an application. This latter component is specific to each application and should resist better to network conditions. We present our model for a TCP connection, which can also be applied to UDP Internet traffic. We discuss possible differences at the end of the section.

In our model and without loss of generality we consider a monitoring point at the edge of the network, located in the ISP network, as shown in Figure 4.1. The monitor passively captures the flows between any two hosts; a flow consists of the packets with the same 5-tuple (IP source and destination, port source and destination, IP protocol). For each flow, we consider a client and a server, and we assume that the client is the user who initiates the connection, e.g., by sending the SYN packet for TCP. The packets of this same flow are inspected to extract the statistical properties to identify the application. In the following we model the system by considering the position of the user with respect to the monitor. Then we analyze the different cases when the host close to the monitor (A in Figure 4.1) acts as a client or as a server.

Figure 4.1 shows the three entities and their relative positions. A is the user behind the local network, B is an Internet user far from the local network, and M is the monitoring point. We define $T_{AM}$ as the time taken by a packet to travel between the local user A and the monitoring point M and $T_{MB}$ as the time between the monitoring point M and the user B. These times are shown in Figure 4.1.

The inter-packet time is characterized by the network, the size of the congestion window, and the time required by the application to generate and push the data to the transport layer. The inter-packet time computed at the monitor between any two packets of the same flow can take one of these four different forms based on which host generates the packets, as shown in Figure 4.2:

- $T_{CC}$: two consecutive packets generated by the client;
- $T_{CS}$: packet from the client followed by a packet from the server;
- $T_{SC}$: packet from the server followed by a packet from the client;
- $T_{SS}$: two consecutive packets generated by the server.

We can now define the time taken by the application at the hosts A and B to generate the packets: $T_C$ and $T_S$ are the time due to the application at the client and server side respectively. The time due to the variations of the network conditions, e.g., variable queueing time, between subsequent packets is accounted as $\epsilon$.

Let's first consider the communication between a client and a server, when the client is the entity A in Figure 4.1, i.e., located close to the monitor. From Figure 4.2, we can calculate the inter-packet times. $T_{CC}$ is equal to the time $T_C$, taken by the application to generate the data, plus the time $\epsilon_C$ due to possible changes of the network conditions between the client and the monitoring point. $T_{CS}$ is equal to the application time at the server $T_S$ plus twice $T_{MB}$, the time for the packet to travel between the monitoring point and the server, and $\epsilon_S$, which accounts for possible variations in the network conditions. The times are computed as follows:

$$\begin{aligned} T_{CC} &= T_C + \epsilon_C \\ T_{CS} &= T_S + (2 * T_{MB}) + \epsilon_S \\ T_{SC} &= T_C + (2 * T_{AM}) + \epsilon_C \\ T_{SS} &= T_S + \epsilon_S \end{aligned} \tag{4.1}$$

When the monitor is located close to the server and the client is the entity B of Figure 4.1, the inter-packet times are:

$$\begin{aligned} T_{CC} &= T_C + \epsilon_C \\ T_{CS} &= T_S + (2 * T_{AM}) + \epsilon_S \\ T_{SC} &= T_C + (2 * T_{MB}) + \epsilon_C \\ T_{SS} &= T_S + \epsilon_S \end{aligned} \tag{4.2}$$

The equations (4.1) and (4.2) show that many components contribute to the inter-packet time. This increases the complexity in creating a statistical signature of an application solely on the inter-packet time. Indeed, the time required by an application to generate and transfer packets to the transport layer is masked by the fact that additional time is added due to the network conditions and the TCP layer.

We are now interested in isolating the time due to the application, which we have identified as $T_C$ and $T_S$ in equations (4.1) and (4.2). We assume that the time due to the changes of network conditions between two consecutive packets $\epsilon_C$ or $\epsilon_S$ is negligible. We are aware of this error since the network is not stable and the queueing time at the routers might change or consecutive packets might follow different paths, but we assume that the times, that add or subtract between, compensate each other.

Finally we quantify the time between the entity $A$ and the monitoring point, $T_{AM}$, and the time between the entity B and the monitoring point, $T_{MB}$. We only discuss the case when the monitoring point is located close to entity $A$; the other case is similar. If we consider that the monitoring point is located close to the gateway router of the ISP, then $T_{AM}$ is half the local round-trip time that a connection experiences within the components inside the ISP. $T_{MB}$ is half the remote round-trip time over the wide area Internet from the monitoring point to the server [36]. Now the final question remains the estimation of the local and remote RTT to compute $T_{AM}$ and $T_{MB}$. We compute the remote RTT from the TCP three-way handshake to establish the session. We use the time between the SYN and the SYNACK packets, as this time is independent from the application, and we assume that it is constant for the duration of the session. Possible variations of the remote RTT are accounted in $\epsilon_S$. The local RTT can be estimated from the SYNACK and ACK packets (or DATA packet in case of piggybacking).

Assuming that $\epsilon_C$ and $\epsilon_S$ are negligible, we can filter any possible noise. Indeed, we can compute $T_C$ and $T_S$ from equations (4.1) and (4.2) and use then instead of the inter-packet time to characterize the applications and to classify the Internet traffic. Note that the same model applies to UDP traffic when we can estimate the local and remote RTT for a connection between the client and the server.

## 4.3 Method Description

Our purpose for the classification of Internet traffic is to detect online which flow belongs to which application. We use a statistical and iterative method that computes the probability that packets are generated by an application. We have defined and used this method to classify Internet traffic based on the packet sizes in Chapter 3. When applied to the inter-packet time, the method allows an iterative classification of the flows for each inter-packet time independently. It considers more and more inter-packet times until the classifier reaches a predefined threshold. Each flow corresponds to a sequence of $N + 1$ packets $Pkt_k$, where $k$ indicates the position of the packet in the flow independently of its direction. $IPT_k$ with $1 \leq k \leq N$ represents the inter-arrival time between $Pkt_k$ and $Pkt_{k+1}$.

In this section we first propose an overview of our method and then we detail its application and extension to the inter-packet time, which we use as a feature for determining an application signature. The method consists of three main phases which are detailed in the following sections: the model building phase, the classification phase, and the application probability or labeling phase.

### 4.3.1 Model building and classification phase

We use K-Means as supervised machine learning algorithm to partition the input features in a predefined number of clusters. Given the number of clusters, K-Means assigns each input feature to a cluster so as to minimize the Euclidean distance of each input to the centroid of the corresponding cluster.

$IPT_k$ denotes the inter-packet time, i.e., the observations, and for each inter-packet time we train separately K-Means to obtain different set of classes. Each observation is pre-processed to determine the features in accordance with the 4 different types of inter-packet time defined in Section 4.2: $T_{CC}$, $T_{CS}$, $T_{SC}$, and $T_{SS}$. Figure 4.3 shows the observations as points in a two

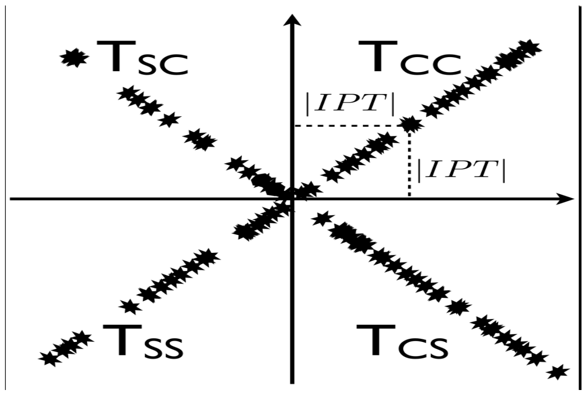

**Figure** 4.3: Types of inter-packet time $IPT_k$ as input to K-Means.

dimensional plane, where the X and Y coordinates indicate the first packet $Pkt_k$ and the second packet $Pkt_{k+1}$ that determine the inter-packet time respectively. The absolute value of the point coordinates is equal to $IPT_k$. The sign of each coordinate depends on the type of inter-packet time: positive sign when one of the two packets that determines the inter-packet time is sent by the client; negative sign in case the packet is originated by the server. As a result of this processing phase, each feature is a 2-dimensional vector.

In the learning algorithm, every class is affected to all applications with different probabilities proportional to the number of flows from each application present in the class. Hence, each class is characterized by the probability that the elements within this class are generated by the different applications.

The model building consists of constructing these sets of classes (clusters) by using a training data set, described in Section 5.4. This learning phase is used to compute $Pr(i|A)$, the per-class $(i)$ probability knowing the application $(A)$. We build a separate model, i.e., set of classes, for every inter-packet time noted by $IPT_k$ and we use these classes for the classification phase.

The classification consists of using the classes defined in the learning phase to test and assign the Internet flows to a class. The test is performed by computing the Euclidian distance that separates the point defined by the feature extracted from the inter-packet time $IPT_k$ and the centroid of each class determined for the $k$-th inter-packet time. We affect the point to the closest class. The test is repeated for all the inter-packet times of a flow iteratively until we reach a predefined threshold. The classification output consists in the probability that the $IPT_k$ identifies an application and is given as input to a labeling function described in the following section.

**Table** 4.1: Traces used for Inter-arrival time validation.

| Trace name | Source | Date | Applications | Ground Truth validation |
|---|---|---|---|---|
| Trace I [7] | Brescia University | April 2006 | HTTP<br>SMTP<br>POP3 | Deep packet inspection |
| Trace II | INRIA Laboratory | Spring 2008 | HTTP<br>SMTP<br>HTTPS<br>SSH<br>IMAP | collected from servers |

### 4.3.2 Application probability or labeling phase

In the labeling phase we assign a flow to an application knowing the result of the classification. We combine iteratively the results of the classification for each single inter-packet time and we calculate the probability ($Pr(A|Result)$) that a flow belongs to an application $A$ given the classification results of the first $N$ inter-packet times (i.e., class $i(1)$ for the first inter-packet time, class $i(2)$ for the second inter-packet time and so on).

$$Pr(A|Result) = \frac{Pr(A) * \prod_{k=1}^{N} Pr(i(k)|A)}{\sum_{A=1}^{N_A} Pr(A) * \prod_{k=1}^{N} Pr(i(k)|A)} \tag{4.3}$$

.

$Pr(A)$ is the probability that any flow randomly selected comes from application $A$. The network administrator can set this value if he wants to put confidence on the classification derived by other techniques, such as port number classification. In our study, we consider this probability to be uniform for all applications. $Pr(i(k)|A)$ is the probability that $IPT_k$ of a flow belongs to the class $i$ knowing the application $A$. $N_A$ is the total number of applications. We call this probability the assignment probability and we use it to decide on how well the profile of an inter-packet time of a new flow fits an application $A$. We calculate this probability for every inter-packet time computed after capturing packets from a flow. We stop this iterative process when the highest assignment probability over all applications is above a predetermined threshold or the maximum allowed number of tests is reached. This way the threshold is seen as a way to leave the classification phase earlier when we are sure about the flow.

## 4.4 Trace Description

For the validation of our method about inter-packet time we use two real of the three traces described in Chapter 3 (see Table 4.1). We evaluate of our method by using the applications

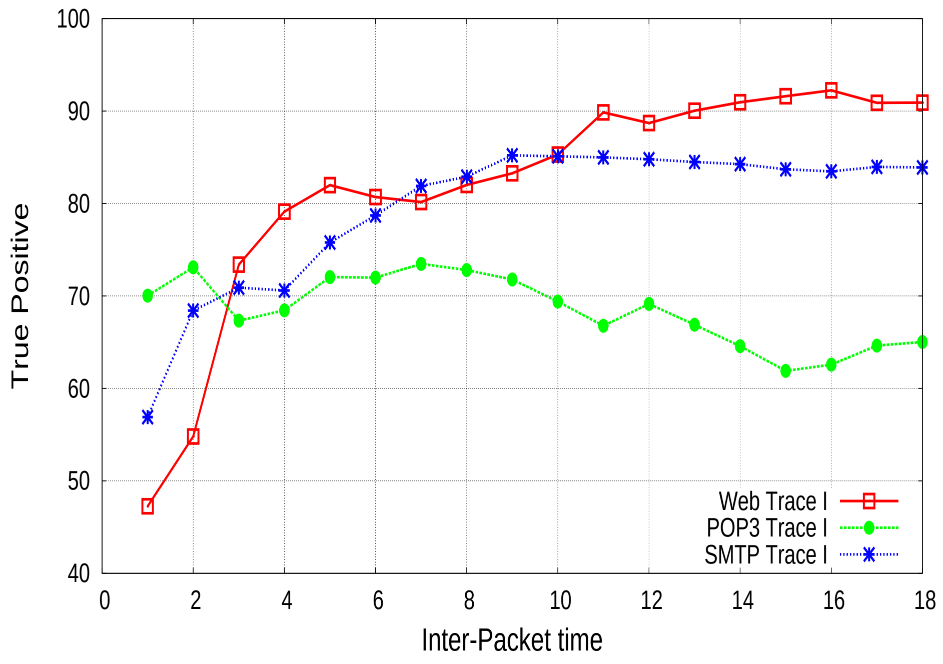

**Figure** 4.4: TP rate without filtering (Trace I).

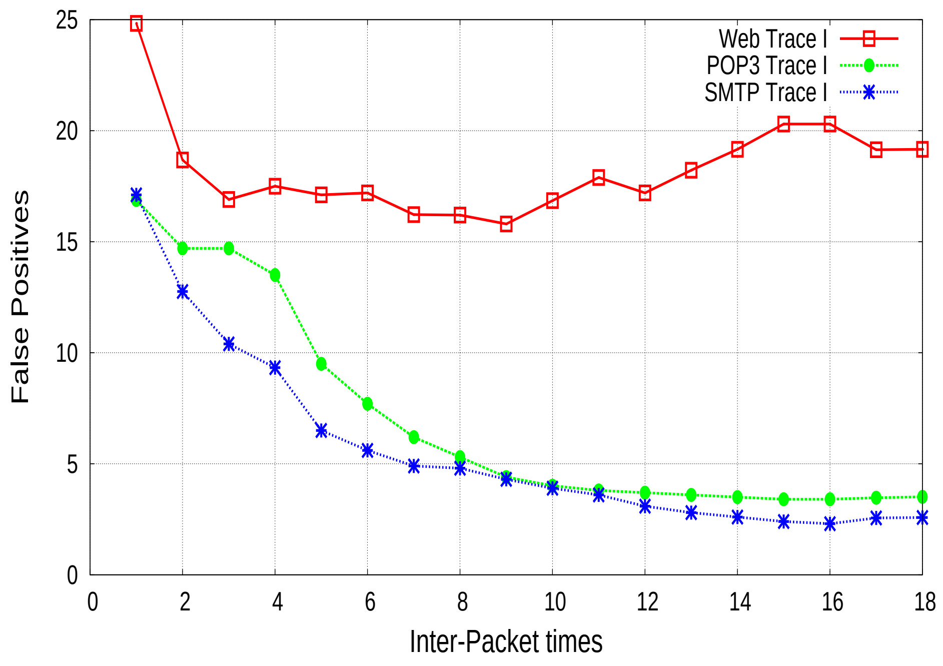

**Figure** 4.5: FP rate without filtering (Trace I).

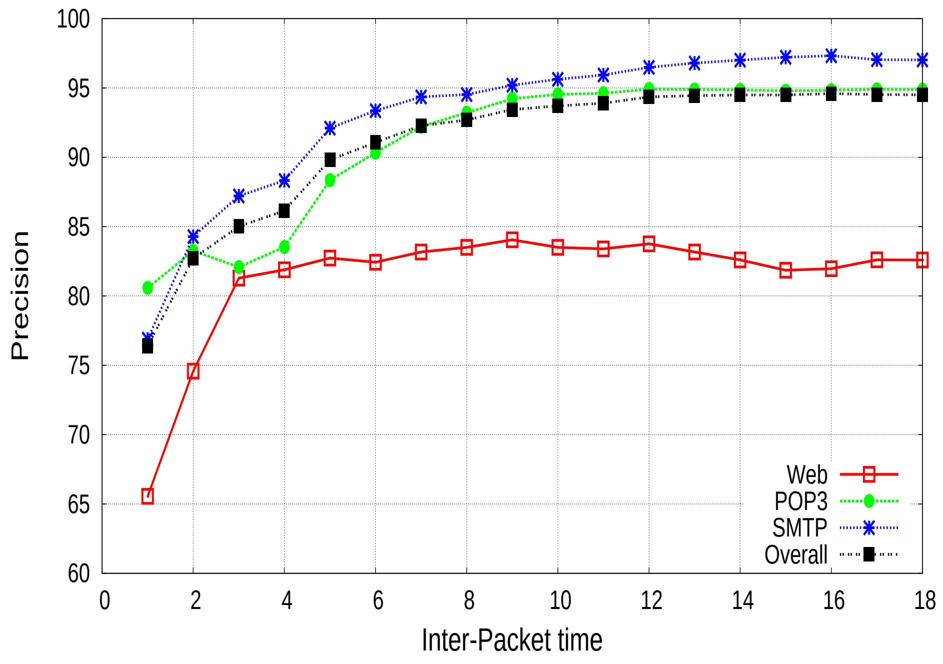

**Figure** 4.6: Precision without filtering (Trace I).

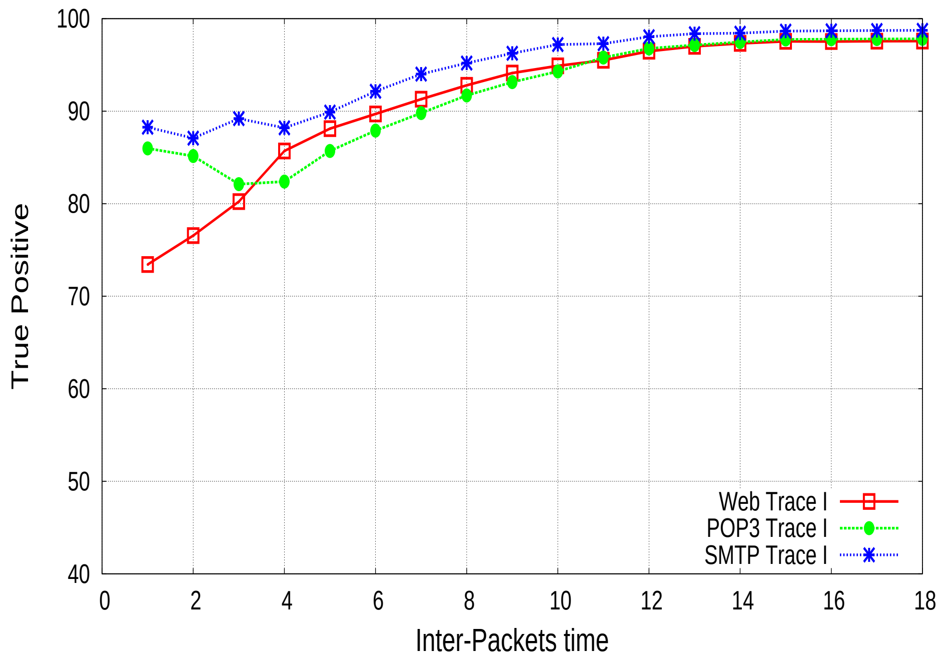

**Figure** 4.7: TP rate after filtering RTT (Trace I).

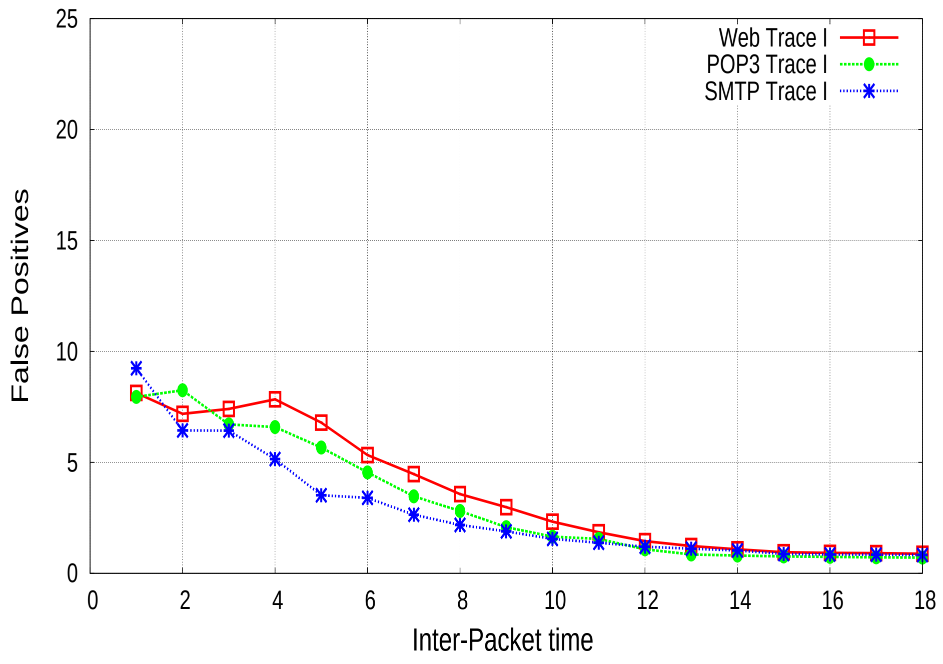

**Figure** 4.8: FP rate after filtering RTT (Trace I).

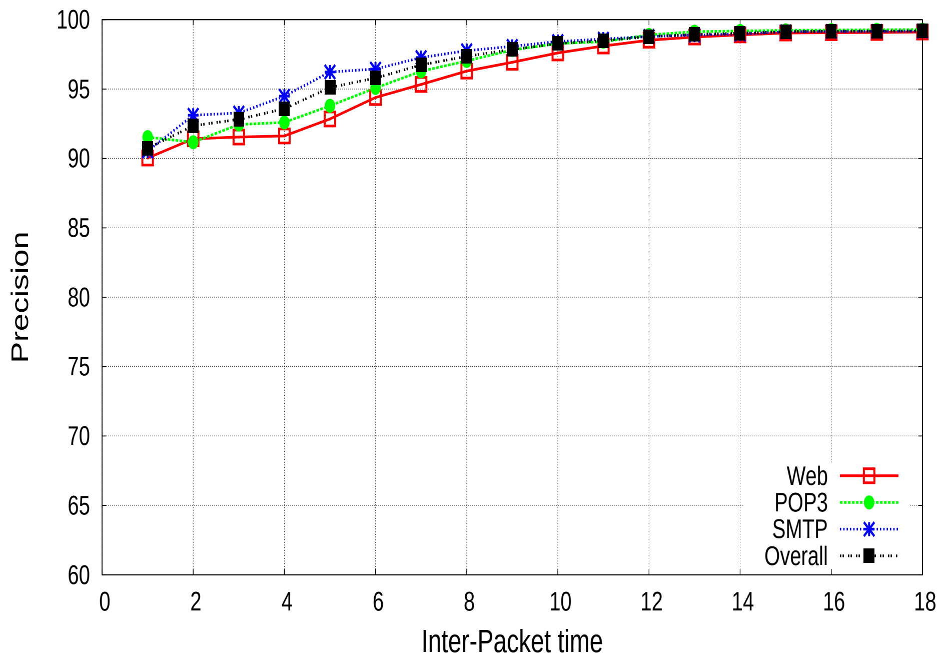

**Figure** 4.9: Precision after filtering RTT (Trace I).

available in these traces, reported in Table 4.1. However, our model is general and it can be applied to any application. As explained in Chapter 3, the flows of the traces are divided into a training and a testing set. The training set is used in the learning phase to construct the model and the testing set is used to evaluate how well our iterative method behaves in identifying the application. Note that we use the same number of flows per application to ensure that there is no bias in our learning phase.

## 4.5 Experimental Results

In this section, we evaluate the overall performance of our method while using the inter-packet time as a feature to classify the Internet traffic. We use the traces described in Section 5.4 and we model the inter-packet time as discussed in Section 4.2. The monitoring point is close to the client for Trace I and close to the server for Trace II, see Figure 4.2. We initially test the inter-packet time without filtering the noise and then we compare these results with the ones obtained by filtering the remote round-trip time from the $T_{CS}$ and the $T_{SC}$. We consider the local round-trip time to be negligible because the traces are collected at the campus router for Trace I and at the servers for Trace II. The metrics used for the evaluation are:

- *False Positive (FP) rate* is the percentage of flows of other applications classified as belonging to an application $A$.
- *True Positive (TP) rate* is the percentage of flows of application $A$ correctly classified.
- *Precision* is the ratio of flows that are correctly assigned to an application, $TP/(TP + FP)$. The overall precision is the weighted average over all applications given the number of flows per application.

We run the method for all the available inter-packet times to test their significance as a feature for identifying applications. We set the number of clusters equal to 200 for K-Means. We have tested the supervised machine learning algorithm with different number of clusters and 200 gave the best results as it allowed to group the features in small clusters and account for possible noise in the observations. The number of flows per application used for training and testing the algorithm are reported in Table 4.1. We report the detailed results of the test conducted on Trace I and only a summary of the results on Trace II for comparison.

Figure 4.4 shows the TP rate for the HTTP, POP3, and SMTP applications (Trace I) as a function of the number of inter-packet times considered for the classification without filtering the remote RTT. We can notice that the TP rate increases as we use more inter-packet times for the identification of the application for HTTP and SMTP traffic. However, the TP rate for POP3 traffic does not show any improvement, the results are worse after the $IPT_8$. The variability of the TP rate for the first packets is associated with the noise added to the inter-packet time by the network conditions and the TCP behavior, as explained in Section 4.2.

In Figure 4.5 we plot the FP rate for the traffic of Trace I as a function of the number of inter-packet times. We can observe that the FP rate drops below 5% when we use more IPTs for POP3 and SMTP traffic, and it remains around 20% for the HTTP traffic. This means that some of the traffic of other applications is classified as HTTP. In particular, we can conclude from Figures 4.4 and 4.5 that part of the POP3 traffic is classified as HTTP traffic. Indeed, the $IPT_k$, for $k \geq 8$, does not add significant information and the distribution of the inter-packet time for HTTP might have similar characteristics as for the POP3 traffic.

Now we preprocess the inter-packet time to filter the remote RTT and to eliminate part of the noise caused by the network. We test the method on the same traces (Trace I). Figure 4.7 shows the TP rate and Figure 4.8 the FP rate. We can clearly see that the TP rate keeps increasing for all the applications when we add more IPTs to the classification and approaches 97% after 12 IPTs. There is a significant improvement for the POP3 application already from the first few IPTs and after 6 IPTs we classify correctly more than 90% of the flows.

The FP rate also improves significantly for all applications and it equals 5% already after the $IPT_6$, see Figure 4.8. If we use more inter-packet times for the classification then the FP rate keeps decreasing and approaches 1% for all applications when we use all available IPTs. This confirms the efficiency of our filtering operation. Thus, we can conclude from this preliminary analysis that filtering the network noise from the inter-packet time is an important function to differentiate between applications.

In Figures 4.6 and 4.9 we plot the precision of our method for Trace I before and after filtering the RTT respectively. We first compute the precision per application and then we calculate the overall precision by weighting the single precision by the number of flows per application. Figure 4.6 shows that the precision of our method approaches 80% for HTTP traffic while it is around 95% for SMTP and POP3 traffic. This is justified by the low false positive rate obtained for these two applications (see Figure 4.5). After filtering the RTT we achieve a precision of 99% in all cases, see Figure 4.9.

We conclude the evaluation by testing our method on Trace II, see Table 4.1. We plot in Figure 4.10 and 4.11 the precision of our method before and after filtering the RTT value. The results confirm the strength of our model in extracting relevant information from the inter-packet time to identify Internet applications. Finally, we can notice that filtering the RTT improves significantly the classification and we are able to have a precision above 90% already after few IPTs.

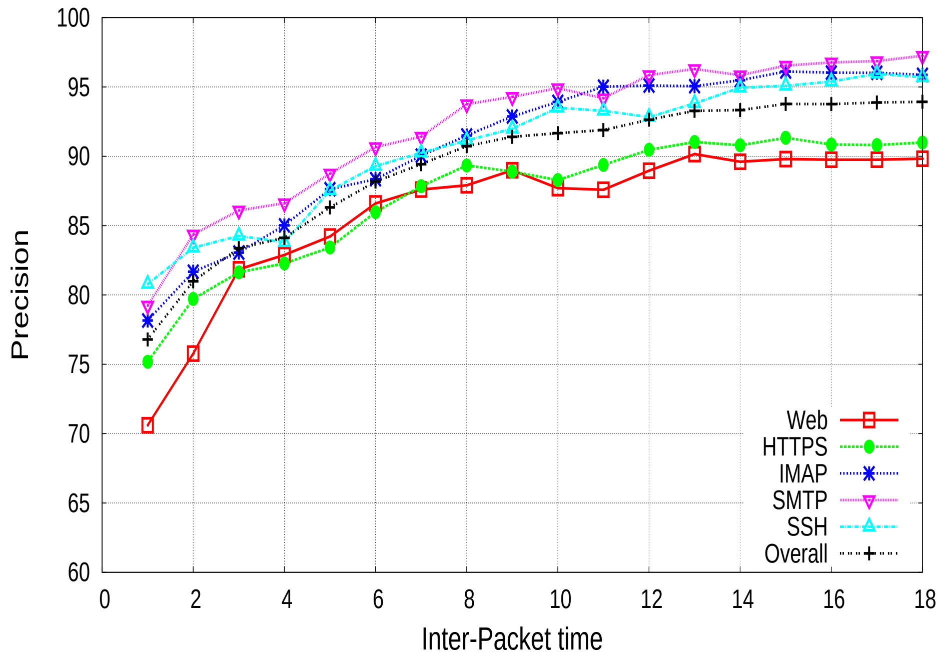

**Figure** 4.10: Precision Without filtering (Trace II).

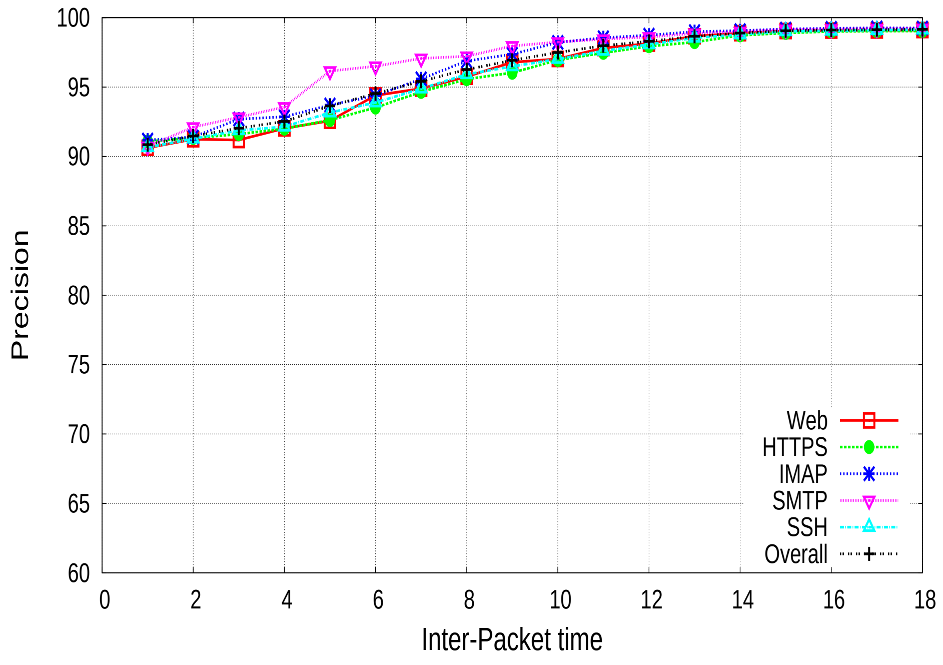

**Figure** 4.11: Precision After filtering RTT (Trace II).

## 4.6 Conclusion and future works

In this chapter we describe our study about the inter-packet time and we analyze how it can be used to identify Internet applications. We model the different components of the inter-packet time and we propose to filter the noise due to the network delay to extract relevant features for the classification. We then introduce our iterative method explained in Chapter 3 already used for the classification based on the packet size and apply it to the inter-packet time.

We evaluate our solution on two different real traces and the results show that the inter-packet time can be transformed into a relevant feature to identify Internet traffic after some appropriate processing. In particular, this processing is important to highlight the interactive characteristics of each application at the client and server. The results show that our method reaches a total precision of 99% for the classification of all applications, when we filter the network noise from the inter-packet time. The possibility of using the inter-packet time as a classification feature parameter increases our choice and makes our iterative method more robust. In the next chapter we will present our third method which classify Internet traffic while combining the iterative statistical method and host based information. We make this combination in order to make our classification method even more robust with an additional service able to profile hosts.

5

# USING HOST PROFILING TO REFINE STATISTICAL APPLICATION IDENTIFICATION

In this chapter, we describe our new on-line method for traffic classification that combines the statistical and host-based approaches in order to construct a robust and precise method for early Internet traffic identification. We use the packet size as the main feature for the classification and we benefit from the traffic profile of the host (i.e. which application and how much) to decide in favor of this or that application. This latter profile is updated on-line based on the result of the classification of previous flows originated by or addressed to the same host. We evaluate our method on real traces using several applications. The results show that leveraging the traffic pattern of the host ameliorates the performance of statistical methods. They also prove the capacity of our solution to derive profiles for the traffic of Internet hosts and to identify the services they provide. The results of this chapter were published in [22] and [20].

## 5.1 Introduction

As we discussed before the statistical methods are preferred to port-based ones and deep packet inspection since they don't rely on the port number and they also work for encrypted traffic. These methods combine the statistical analysis of the application packet flow parameters, such as packet size and inter-packet time, with machine learning techniques. Other approaches rely on the way the hosts communicate and their traffic patterns to identify applications.

The common feature of statistical methods is that they classify every flow independently of each other using the pattern of its packets (size, time, and direction). Indeed, they don't use any information about the traffic pattern of the originating host or the type of services that run on the destined server. The same thing applies to peer-to-peer communications. We believe that the classification of previous flows sharing the same IP address either as source and/or destination is important to refine the classification of future flows and hence Internet applications in general. For instance, a host browsing the web is more prone to open several consecutive HTTP connections. A machine hosting a POP3 mail server is very likely to receive POP3 flows. In general, hosts have profiles for their flows either because of the behavior of users or the services run on them, and these profiles can help in the identification of flows in which they are implied. In this chapter, we propose to build the traffic profile of hosts, based on the result of the classification of previous flows, and then to use this information to refine the classification of subsequent flows. On one hand these profiles help in flow classification and on the other hand they point to the behavior of the users behind them and on the network services they deploy.

Our solution differs from the previous works that consider the role of the host [59, 25] by the fact that we rely mainly on the statistical criteria, that we refine with the host information. In [59], Trestian et al. characterize the role and type of traffic of an end-point by collecting publicly available information on the web based on the IP address of the host. BLINC [25] is a solution for Internet traffic identification that considers the role of the host. It focuses on the source and destination of the flows to determine the host behavior, which is studied across three levels: social to account for the host popularity and communities of hosts (groups of communicating hosts), functional to identify the functional role of a host (offered services, used services), and protocol patterns of the host. The main difference, which is also the strength, of our approach consists of only considering the flows sent and received by the monitored hosts and in crossing the information between flows of the same host so as to build profiles and reach better classification. We construct and leverage the profiles of the communicating hosts simultaneously and on the fly without requiring the traffic monitor to maintain a detailed history of their interaction.

Our contribution in this work can be summarized as follows. First, we define the host profile and we determine the host-based probability that a flow is of a given application in both the incoming and outgoing direction. We develop a new method that relies on the result of the classification of flows from the same host to determine the profiles of hosts and to use these profiles later as an initial guess before the classification of future flows. The main idea is to combine the statistical properties of a flow with the traffic profile of the end-points to better associate flows to applications. The host profiles are updated after each classification using an exponential weighted moving average filter to absorb any transient behavior; the way the profile accounts for past classified flows depends on some discounting parameter, which can be decided by the network administrator. Once described, we use two real traces to test our method and to show how to characterize the traffic pattern of each host in the traces.

The rest of the chapter is organized as follows. Section 5.2 introduces and discusses the host profiling. Section 5.3 explains our classification method. Section 5.4 and Section 5.5 describe the traces and the evaluation results, respectively. We conclude the chapter in section 5.6.

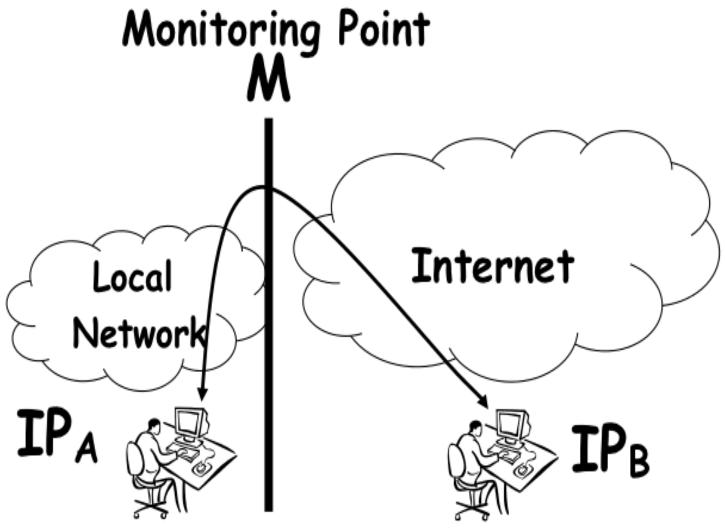

**Figure** 5.1: The monitored system.

## 5.2 Host traffic profile

In this section we discuss how we determine the traffic profile of a host and what are the benefits of using this information to refine the classification of Internet applications. The methodology herein described is general and it can be integrated to any classification method transparently. In our model and without loss of generality we consider a monitoring point at the edge of the network, located in the ISP network, as shown in Fig. 5.1. The monitor passively captures the flows between any two users; a flow consists of the packets with the same 5-tuple (IP source and destination, port source and destination, IP protocol). For each flow, we consider the two end points: a host located inside the ISP network ($IP_A$ in Fig. 5.1) and a destination host downstream the monitoring point ($IP_B$ in Fig. 5.1). We don't assign any specific role to the hosts, $IP_A$ and $IP_B$, which can act indifferently as client or server during a session. The monitor inspects the packets of each flow and extracts statistical information, such as packet size, inter-packet time, direction of the packet, etc. This information is used to create the signature of the flow and to assign the flow to the application that matches the signature. We will discuss in Section 5.3 the definition of this signature when we present the classification procedure.

The traffic profile of a host is defined based on the type of previous flows. This requires that the monitor collects statistical information about a flow, classifies the flow, and stores the result of the classification to track the activity of a host. In a real setting, we assume that the monitor logs only the traffic for the hosts inside the ISP network, or those of interest, and might store information about some IP addresses that runs dedicated services. The traffic profile, so computed, gives an indication of the applications that run at â each specified host.

The novelty of our approach consists of using this traffic pattern to predict future flows that involve the same host. In this section, we first discuss how a monitor computes the probability that a flow of packets between two hosts is of a certain application solely using the traffic patterns of these hosts. Then we discuss how the monitor computes and updates the host profile.

### 5.2.1 Host based probability of a flow

Let $F$ denote a function that associates a packet flow between a source $S$ and destination $D$ to an application $A(i)$, with $1 \leq i \leq N_A$ and $N_A$ the number of monitored applications. Let $P(F_S = A_S|S)$ (or $P(F_D = A_D|D)$) be the probability that, given the host traffic profile, the flow is of application $A_S$ for the source (or $A_D$ for the destination). Then, the probability $P(F = A(i))$ that the flow is of application $A(i)$ is computed as follows:

$$\begin{aligned} P(F = A(i)) &= P((F_S = A(i)_S) \cap (F_D = A(i)_D)|A_S = A_D) \\ &= \frac{P(F_S = A(i)|S) * P(F_D = A(i)|D)}{\sum_{j=1}^{N_A} P(F = A(j)|S \cap D)} \\ &= \frac{P(F_S = A(i)|S) * P(F_D = A(i)|D)}{\sum_{j=1}^{N_A} P(F_S = A(j)|S) * P(F_D = A(j)|D)} \end{aligned} \tag{5.1}$$

Equation (5.1) means that we compute the probability by considering the cases when the prediction for each host is in accordance with that of the other host of the flow. Equation (5.1) also holds when the monitor only records the traffic profile of one of the two hosts. In fact, if we assume a uniform probability for the other host, e.g., $P(F_D = A_D|D) = \frac{1}{N_A}$, then, equation (5.1) simplifies to $P(F = A(i)) = P(F_S = A(i)|S)$.

### 5.2.2 Host profile definition and update

We now discuss how the monitor computes the host profile and updates this information. Each host can be source or destination for different flows and this depends on whether it does send the first packet of the flow or receive it simply. The monitor captures the flows and decides about a flow by using the source or destination profile of the host. This results in two traffic profiles for the same host. In the rest of the section, we discuss a generic host and the computation of the source profile for this host; the destination profile is defined in the same way.

Let S denote the generic source host of a flow and $F_S$ the function that maps the flow to an application. The monitor computes the host profile by using previous classified flows, in this case when the host is the source of the first packet. The profile $P(\mathcal{A}|S)$ is thus defined as the prior distribution for the flows in the domain $\mathcal{A}$, which defines the applications $A(i), 1 \leq i \leq N_A$. If the monitor has not any information about previous traffic of a host, then, the monitor considers a uniform prior distribution. The prior distribution is updated after each classification of a new collected flow.

The profile update works as follows. Let $P_{(n-1)}(A(i)|S)$ be the prior probability for application $A(i)$ at host S computed from the past $(n-1)$ flows that the monitor affects to the application $A(i)$ equal to the probability $P(F_S = A(i)|S)$. Then the posterior probability for each application is computed as follows:

$$\begin{aligned} P_{(n)}(A(i)|S) = \lambda * P_{(n-1)}(A(i)|S) + \\ + (1-\lambda) * P(F_S(n) = A(i)|S) \end{aligned} \tag{5.2}$$

$P(F_S(n) = A(i)|S)$ is the result of the classification of flow $n$ and $\lambda$, $0 \leq \lambda \leq 1$, represents the discounting factor for past classifications. When $\lambda$ is close to $0$, the profile is computed by associating a higher weight to the most recent flows. When $\lambda$ is close to 1 the profile is calculated over a longer period, which means that the profile is determined in equal measure by all previous classified flows. When $\lambda = 1$ the profile corresponds to the initial prior distribution, which in our case assigns a uniform probability to all applications (case of Chapters 3 and 4). The best choice of $\lambda$ depends on the traffic pattern of the host and on the performance of the classifier. We will discuss more about $\lambda$ in Section 5.5.

The amount of information that the monitor maintains to update the profile of the host is limited to the two prior distributions. Table 5.1 shows an example of the source and destination profiles of a host. In this example we compute the profile of a host for five applications: FTP,

**Table** 5.1: Example of a traffic profile of a host.

| Applications: | FTP | HTTP | POP3 | SMTP | SSH |
|---|---|---|---|---|---|
| Source: | 0.02 | 0.76 | 0 | 0.2 | 0.02 |
| Destination: | 0.22 | 0 | 0.1 | 0.23 | 0.45 |

HTTP, POP3, SMTP and SSH. In this example we can observe that the profile of this host is in favor of the HTTP application, and so if we have a flow generated by this host, we will consider that there is more chance that it belongs to the HTTP application.

## 5.3 Method Description

Our purpose for the classification of Internet traffic is to detect on-line which flow belongs to which application. We use a statistical and iterative method that computes the probability that packets are generated by an application. We have defined and used this method to classify Internet traffic based on the size of the packets in Chapter 3. The method allows an iterative classification of the flows for each packet size independently, and keeps adding more packet sizes for the identification of an application until the classifier reaches a predefined threshold. Each flow corresponds to a sequence of $N$ packets $Pkt_k$, where $k$ indicates the position of the packet in the flow independently of its direction.

In this section we first recall our iterative method and then we detail how the method uses the host profile to refine the classification. The method consists of three main phases which are detailed in the following sections: the model building phase, the classification phase, and the application probability or labeling phase.

### 5.3.1 Model building and classification phase

We use K-Means as supervised machine learning algorithm. Given the number of clusters $N_C$, K-Means assigns each input feature to a cluster so as to minimize the Euclidean distance to the centroids of the clusters.

$Pkt_k$ denotes the packet size, i.e., the observation. For each packet size we train separately K-Means to obtain different set of classes. The input feature corresponds to the value of the size of the packet associated with a sign that represents the direction of the packet. A positive sign corresponds to a packet from the source to the destination. In the learning algorithm, every class is affected to all applications with different probabilities proportional to the number of flows from each application present in the class. Hence, each class defines the probability that the elements within this class are generated by the applications.

The model building phase consists of constructing these sets of classes (clusters) by using a training data set, described in Section 5.4. This learning phase is used to compute $P(C(j)|A(i))$, i.e., the per-class probability, knowing the application $A(i)$. The probability is computed for all clusters $C(j)$, where $1 \leq j \leq N_C$ and $N_C$ is the number of clusters. We build a separate model, i.e., set of classes, for every packet size noted by $Pkt_k$ and we use these classes for the classification phase.

The classification consists of using the classes defined in the learning phase to test and assign the Internet flows to a class. The test is performed by computing the Euclidean distance between the input feature from $Pkt_k$ and the centroid of each class determined for the $k$-th packet size. We affect the point to the closest class. The test is repeated for all the packet sizes of a flow iteratively until we reach a predefined threshold. The classification result consists in the probability that the $Pkt_k$ identifies an application and it is given as input to a labeling function described in the following section.

### 5.3.2 Application probability or labeling phase

In the labeling phase we assign a flow to an application knowing the result of the classification and the probability computed from the profiles of the source and destination, as discussed in Section 5.2. We combine iteratively the results of the classification for each single packet size and we calculate the probability ($P(A(i))$ that a flow belongs to an application $A(i)$ given the prediction from the host profiles and the classification results of the first $N$ packet sizes (i.e., class $C(j(1))$ for the first packet size, class $C(j(2))$ for the second packet size and so on).

$$\begin{aligned} P(A(i)) &= P(A(i)|Result \cap P(F = A(i))) \\ &= \frac{P(F = A(i)) * \prod_{k=1}^{N} P(C(j(k))|A(i))}{\sum_{i=1}^{N_A} [P(F = A(i)) * \prod_{k=1}^{N} P(C(j(k))|(A(i))]} \end{aligned} \tag{5.3}$$

$P(F = A(i))$ is the probability that a flow between a source and a destination comes from application $A(i)$ based on their traffic profiles. $P(C(j(k))|A(i))$ is the probability that $Pkt_k$ of a flow belongs to the class $C(i)$ knowing the application $A(i)$. $N_A$ is the total number of applications. We call this probability the assignment probability. It combines the result of the classification, obtained with the K-Means clustering method, and the pattern of the hosts, which gives an indication of the next type of application flow. The assignment probability is computed when the monitor captures each packet of the same flow. Thus, we do not require to wait for a given number of packets to start the classification procedure. We stop this iterative process when the highest assignment probability is above a predetermined threshold or the maximum allowed number of tests is reached. This way the threshold is seen as a way to leave the classification phase earlier when we are sure about the flow. The monitor updates the profiles of the hosts, both the source and the destination, when the classification ends and it

**Table** 5.2: Traces used for host based method validation.

| Trace name | Source | Date | Applications | Ground Truth validation |
|---|---|---|---|---|
| Trace I [7] | Brescia University | April 2006 | HTTP<br>SMTP<br>POP3 | Deep packet inspection |
| Trace III [17] | Brescia University | Fall 2009 | HTTP<br>HTTPS<br>EDONKEY<br>BITTORENT | Deep packet inspection |

has assigned a flow to a given application. The update of the profiles is done as described in Section 5.2.

## 5.4 Trace Description

In this chapter we use two real traces for the evaluation of our host based method. In Table 5.2 we give a refreshment about the two traces used and detailed in Chapter 3. The two traces have been collected at the edge gateway of the Brescia University's campus network. We will not use the trace that we collected at the edge of INRIA laboratory as all collected flows are from INRIA servers (HTTP server, IMAP server etc.), and so for every IP adress we will only find one application making the profiles deterministic and the classification straight forward. The first trace, noted trace I [7], was collected during April 2006 and the second trace, noted trace III [17], was collected on three consecutive working days during fall 2009. Every trace consists of two sets, a training set and a testing set, and the type of applications associated with each flow is determined with a deep packet inspection method.

In the learning phase we use the training set, which consists of an equal number of flows per application to ensure that there is no bias in our learning. During this phase we do not compute the host profile, the application flows are only used to construct the classes in K-Means. The host profile is automatically computed during the testing phase. We initially consider a uniform prior distribution for the source and destination profiles of an unknown host. Then, we update the profiles once flows of this host are collected. The testing set is used to evaluate how well our iterative method behaves in identifying the application.

## 5.5 Experimental results

In this section we present the evaluation results of our method when the traffic profile of the hosts is used to refine the classification. We consider the case of a monitor that maintains the profile of the hosts located inside the ISP domain, since it is interested to understand what is the usage of the network by the ISP customers. In a real setting, the monitor might also decide to maintain the information about popular Internet servers in order to facilitate the Internet identification (for example if one tracks the IP address of a facebook server, he can directly identify flows without the need for more analysis). Indeed, it can use these profiles to compute the probability that a flow is originated by an application, as we discuss in Section 5.2. We use the traces described in Section 5.4 and we profile the hosts with the same IP prefix, i.e., those inside the Brescia campus. For addresses outside the Brescia campus We have counted an average of 10 flows per IP address, therefore we have decided not to show the results since there is not a sufficient number of flows per IP to compute the profile.

The flows are all TCP connections and the hosts within the campus initiate the connection. The metrics used for the evaluation are:

- *False Positive (FP) rate* is the percentage of flows of other applications classified as belonging to an application $A$.
- *True Positive (TP) rate* is the percentage of flows of application $A$ correctly classified.
- *Precision* is the ratio of flows that are correctly assigned to an application, $TP/(TP + FP)$. The overall precision is the weighted average over all applications given the number of flows per application.

We run the test for all the available packet sizes to test its significance as a feature for identifying applications. We set the number of clusters equal to 200 for K-Means. We have tested the supervised machine learning algorithm with different number of clusters and 200 has shown the best results (as discussed in Chapter 3) as it allows to group the features in small clusters and account for possible noise in the observations. It is also a good trade-off between precision and speed of classification.

### 5.5.1 Classification results

In this section we discuss the performance of the classification method when the host profile is used to refine the probability that a flow is of a given application type.

**Total Precision**

Fig. 5.2 and 5.3 plot the total precision for trace I and trace III respectively versus the number of packets used for the classification. Our method classifies a flow at each packet iteratively, as we discussed in Section 5.3. The different lines in the plot correspond to the precision of the classifier when different values of the discounting factor $\lambda$ are used. The value of $\lambda$ determines the weight assigned to the last classification results. When $\lambda = 0.1$, the most recent classification results determine the profile of the host. When $\lambda = 0.9$, the host profile is computed over a longer period. The value of $\lambda = 1$ means that a uniform probability is associated to each application, thus, the host profile is not used, as we have discussed in Section 5.2.2. The results show that the precision of the classifier improves considerably when the profile of the hosts is used to decide in favor of this or that application, especially for the first four packets.

For Trace I, we can observe in Fig. 5.2 that a value of $\lambda = 0.9$ gives the best performance for the classifier. We obtain a precision of 96% already after two packets, 98% after four packets and 99.9% when 10 packets are used for the classification. For $\lambda = 0.1$ and 0.5 the classifier predicts with less accuracy the applications. With this value of $\lambda$ the classifier is more sensitive to recent flows. Thus, it is more prone to a wrong classification when the host has a uniform traffic behavior over all applications. For this trace we have a big number of flows that belong to two different applications generated uniformly by the same host. Thus, the method classifies the applications with less precision for small values of $\lambda$. For $\lambda = 0.1$ we have approximately the same precision for all the packets which approach 95%. For $\lambda = 0.5$ we obtain a very good precision for the first five packets, then the precision decrease and converge to 95% after the sixth packet. For $\lambda = 1$ which is the case when a uniform probability is associated to each application, we obtain a precision around 89% after two packets, this precision keep increasing to reach 95.5% after four packets and 98% after 10 packets. For Trace III and for all the selections of $\lambda$, we have better performance compared to the classification without host profile information ($\lambda = 1$). We can observe in Fig. 5.3 that for $\lambda$ equal 0.1, 0.5, and 0.9 the precision increases already after the first packet. However, after the fifth packet the precision for all the values of $\lambda$ converge around 99%. And so we can see clearly that the use of the host information permit us to increase the precision (in comparison of the classification without host information) of the classifier especially for the first four packets. We can conclude from these results that the profile of the host gives an early characterization of a flow because of the traffic pattern of the host. For instance, we can consider that a host that is browsing the web is more prone to have a sequence of HTTP connections. And so the use of information about the host profile can help our statistical methods to give better performance.

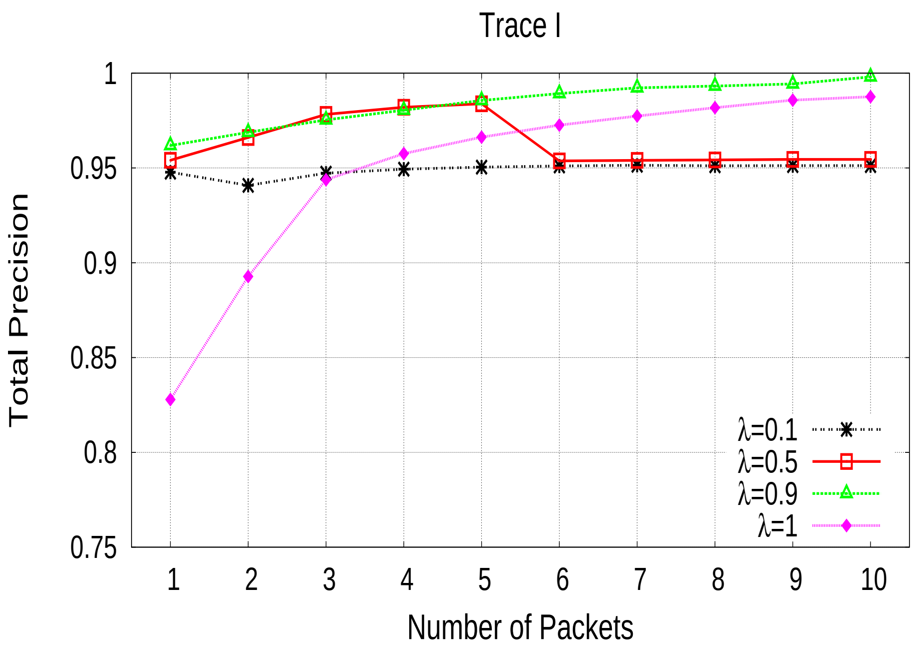

**Figure** 5.2: Total precision versus the number of packets (Trace I).

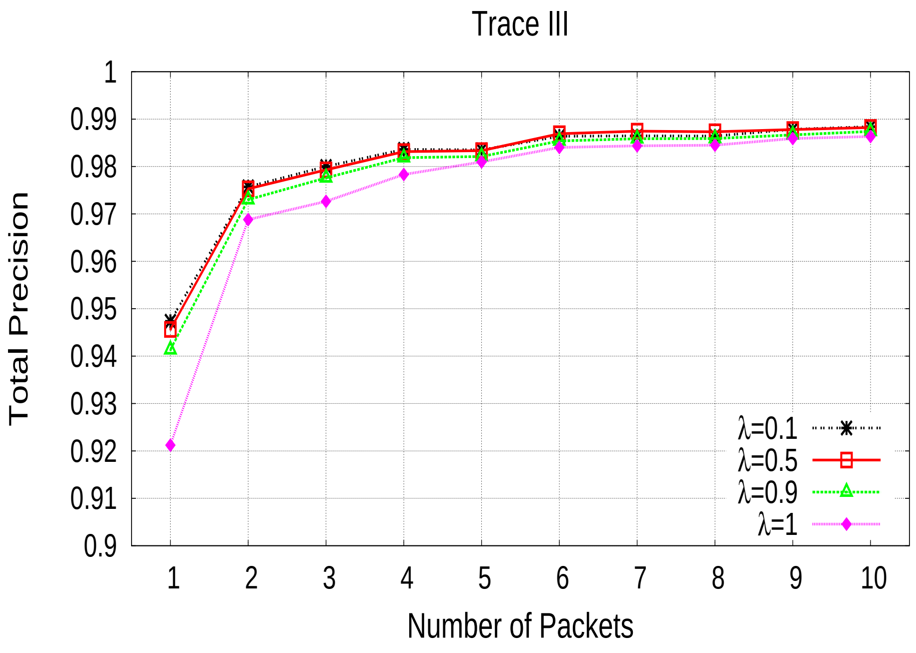

**Figure** 5.3: Total precision versus the number of packets (Trace III).

**True Positive**

Fig. 5.4 shows the True Positive (TP) ratio for the POP3 application (Trace I) as a function of the number of packets used for the classification. We can see that for $\lambda = 0.9$ the TP ratio start with 97% with the first packet and keeps increasing when we use more packets for the classification, until it reaches a value of 99% with ten packets. In comparison, when the host information is not taken into account, the TP ratio start with 56% after the first packet and approaches 90% after using four packets, then keeps increasing to reach a maximum value of 99% after 10 packets. The TP ratio drops when $\lambda = 0.5$ and 6 packets are used for the classification. Similar performance can be observed for $\lambda = 0.1$ at the second packet. This means that the classifier fails to identify correctly POP3 flows as they are assigned to other applications. We will confirm this behavior when we analyze the False Positive ratio to understand what is the output application of our classification method. The fact that this happens only for these small values of $\lambda$ means that we have some hosts that are generating flows belonging to different applications.

Fig. 5.5 and Fig. 5.6 plot the True Positive (TP) ratio for HTTP and SMTP as a function of the number of packets respectively. Fig. 5.5 shows that the TP ratio increases for all the values of $\lambda$, even when we do not use any host information. The classifier has better performance with $\lambda = 0.1$, which means that there are consecutive HTTP flows in general. We can notice clearly the importance of using host information for the first four packets. This use permits us to obtain a very high accuracy even when we only use the first packets. From Fig. 5.6 it is interesting to notice that the TP ratio is 99% for any number of packets of SMTP when we use the profile of the host to refine the classification. The fact that the precision is very high for all values of $\lambda$ means that the SMTP traffic is predominant in some hosts. For $\lambda = 1$ we can observe that the precision of SMTP is around 91% approximately which means that the use of host profile information is very useful for the SMTP application.

In Fig. 5.7, 5.8, 5.9 and 5.10 we plot the True Positive ratio for different applications (HTTP, HTTPS, EDONKEY and BITTORENT) of Trace III. We can clearly observe that for all the values of $\lambda$ we have better performance compared to the classification without host profile information ($\lambda = 1$). The True Positive ratio keeps increasing when more packets are used for the classification. For all the applications we can observe that we have better performance when we use a small value for $\lambda$, which means that we don't have a lot of changes in the traffic pattern of these hosts. For HTTP, we obtain an excellent precision for all values of $\lambda$. For $\lambda = 0.1$, $0.5$ and $0.9$ the true positive ratio is around 98% after the first packet, after the second packet it reaches 99% and stabilizes around that. In comparison for $\lambda = 1$ the precision is smaller for the first four packets and converges also to 99% after the fifth packet. For HTTPS, we obtain a good precision for all the values of $\lambda$ even for $\lambda = 1$. This precision is around 85% after the first packet, 90% after the second and converges to 94% approximately after six packets.

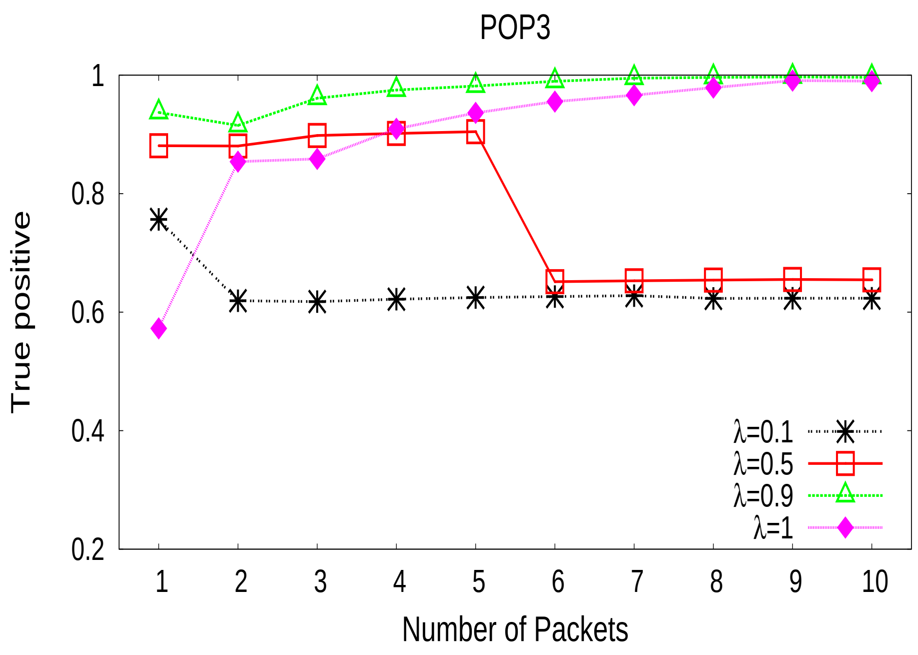

**Figure** 5.4: True positive ratio for POP3 application (Trace I).

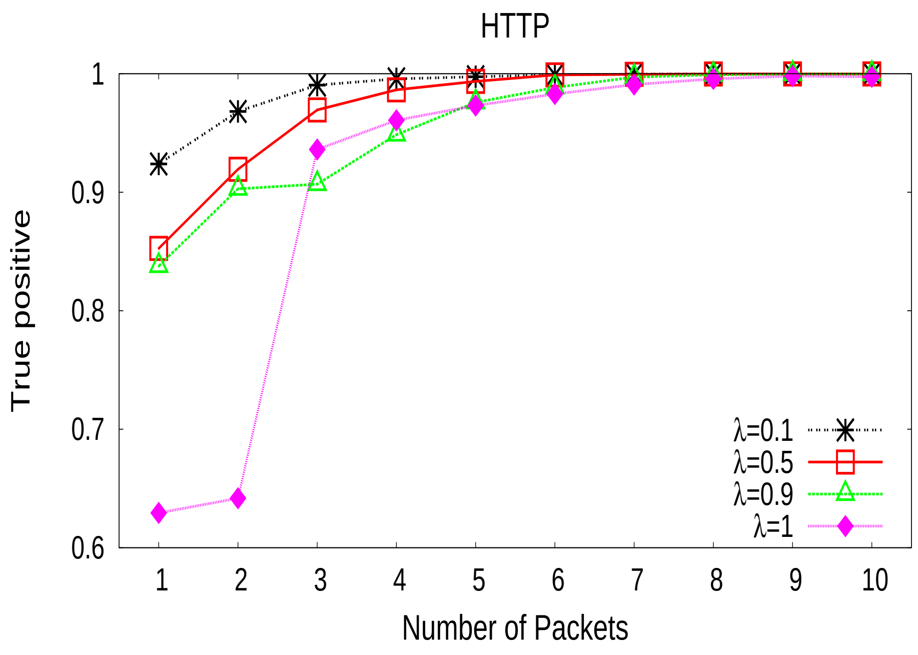

**Figure** 5.5: True positive ratio for HTTP application (Trace I).

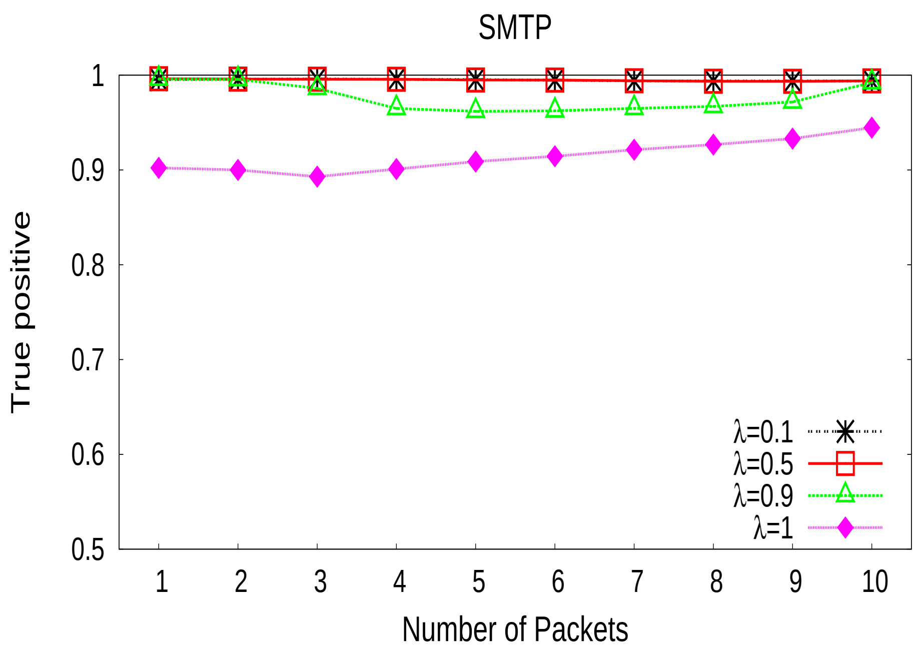

**Figure** 5.6: True positive ratio for SMTP application (Trace I).

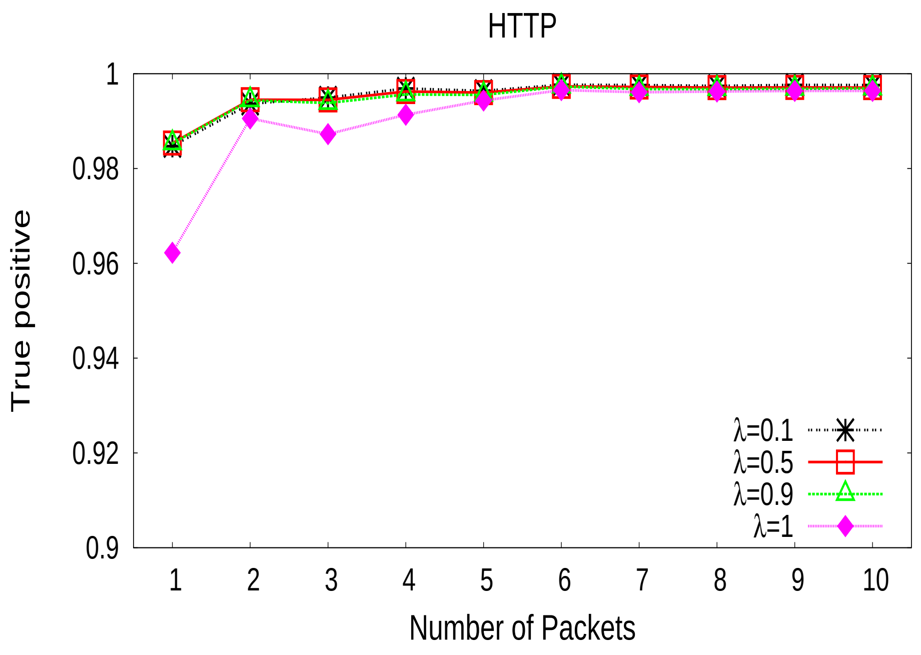

**Figure** 5.7: True positive ratio for HTTP application (Trace III).

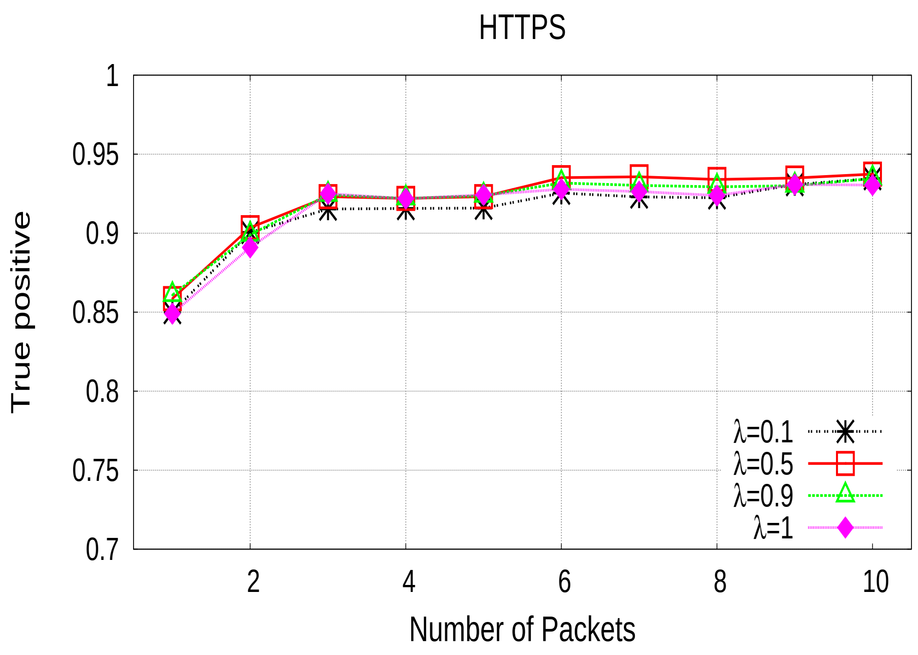

**Figure** 5.8: True positive ratio for HTTPS application (Trace III).

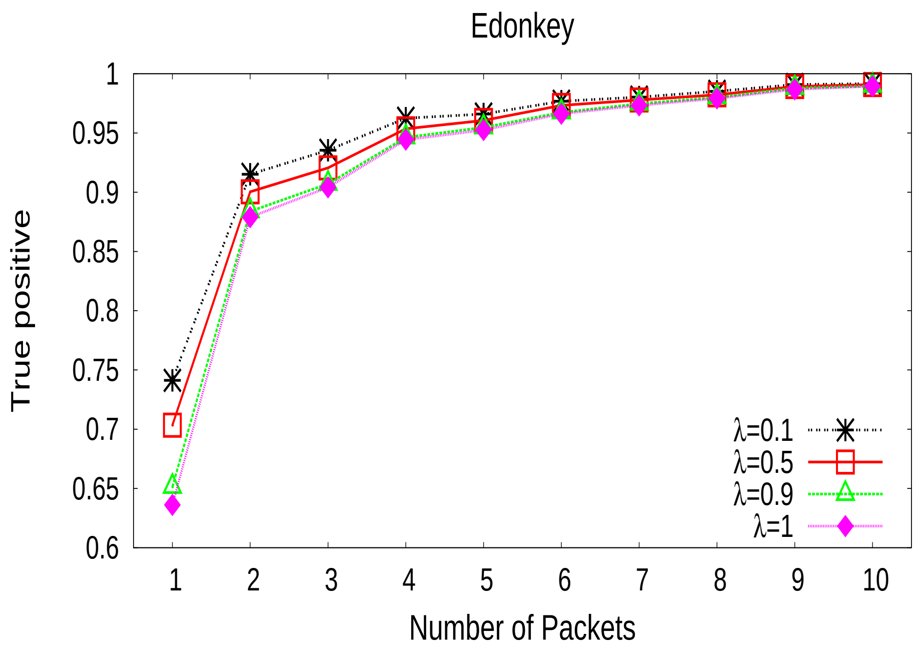

**Figure** 5.9: True positive ratio for Edonkey application (Trace III).

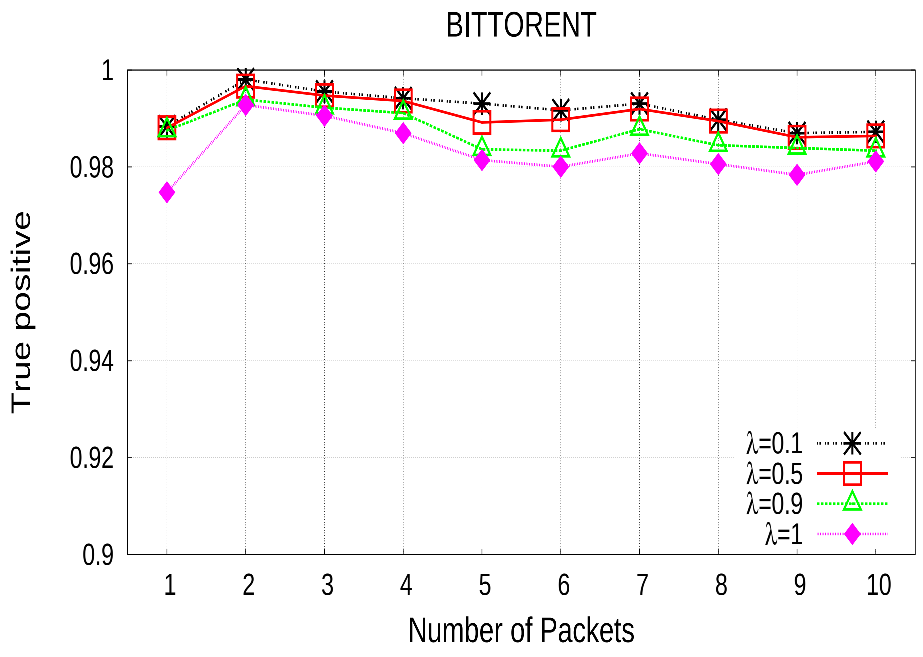

**Figure** 5.10: True positive ratio for Bittorent application (Trace III).

For the EDONKEY application the true positive ratio starts from 65% approximately while using the first packet for $\lambda = 1$ and $\lambda = 0.9$. It increases to 87% after the second packet, to 95% after the fifth packet and keeps increasing to reach 99% after ten packets. In comparison, the true positive ratio is higher for the first four packets for $\lambda = 0.1$ and $\lambda = 0.5$. After the fifth packet it converges to same precision approximately for all the values of $\lambda$.

For BITTORENT, we get a very high precision starting from the first packet. The true positive ratio is around 98% for $\lambda = 1$ and $\lambda = 0.9$ while using any number of packets for the classification. This precision is a little bit higher for the small values of $\lambda$, around 99%. We can explain this higher precision for all the values of $\lambda = 1$ by the fact that the distribution of the size of the first packets is very different compared to the one of other applications, and so that the packet size is very effective to classify BITTORENT traffic.

We can conclude from the results of the true positive ratio that the use of the host information increases the performance of the classification especially for the first four packets. The efficiency of our iterative model permits us to obtain a very good precision even without the host information after the fifth packet.

**False Positive**

Now we move to the results of the False Positive ratio for Trace I and Trace III to confirm our previous conclusions.

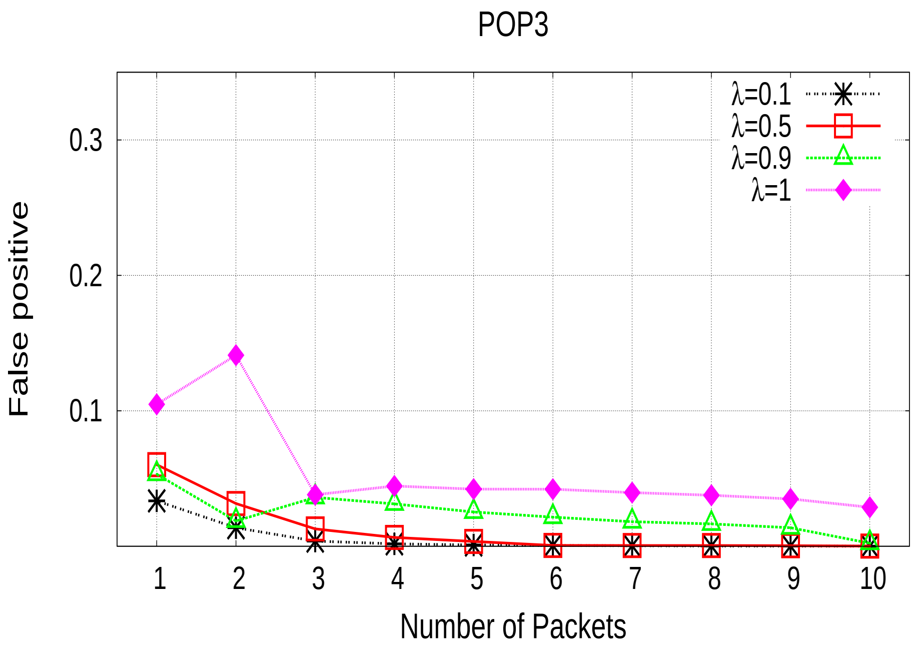

**Figure** 5.11: False positive ratio for POP3 application (Trace I).

For Trace I we can immediately notice in Fig. 5.11 that the percentage of misclassified flows of other applications, assigned to POP3, drops significantly after 4 packets. This false positive ratio is around 5% when we don't use the host information ($\lambda = 1$), around 3% for $\lambda = 0.9$ and around 0% for $\lambda = 0.1$ and $\lambda = 0.5$. Thus, the True Positive ratio of POP3(Fig. 5.4) indicates the correctly classified POP3 flows. Fig. 5.12 also confirms that most of the POP3 flows that have not been detected are indeed classified as HTTP traffic. This is shown for values of $\lambda = 0.1$ and $\lambda = 0.5$. For the higher values of $\lambda$ the false positive ratio drops to 0% after the fifth packet.

Fig. 5.13 shows that the classification for most of the SMTP traffic is indeed correct when the classification of recent flows has more weight. When $\lambda = 0.9$, the classifier labels other flows as SMTP, which means that some hosts have SMTP flows that interleave with the ones of other applications. The false positive ratio drop to 0% after the seventh packet for the small values of $\lambda$ and after the tenth packet for the large ones.

In Fig. 5.14, 5.15, 5.16 and 5.17 we plot the False Positive ratio for the different applications (HTTP, HTTPS, EDONKEY, BITTORENT) of Trace III. We can observe that for all values of $\lambda$ we get better performance in comparison with the classification without host profile information.

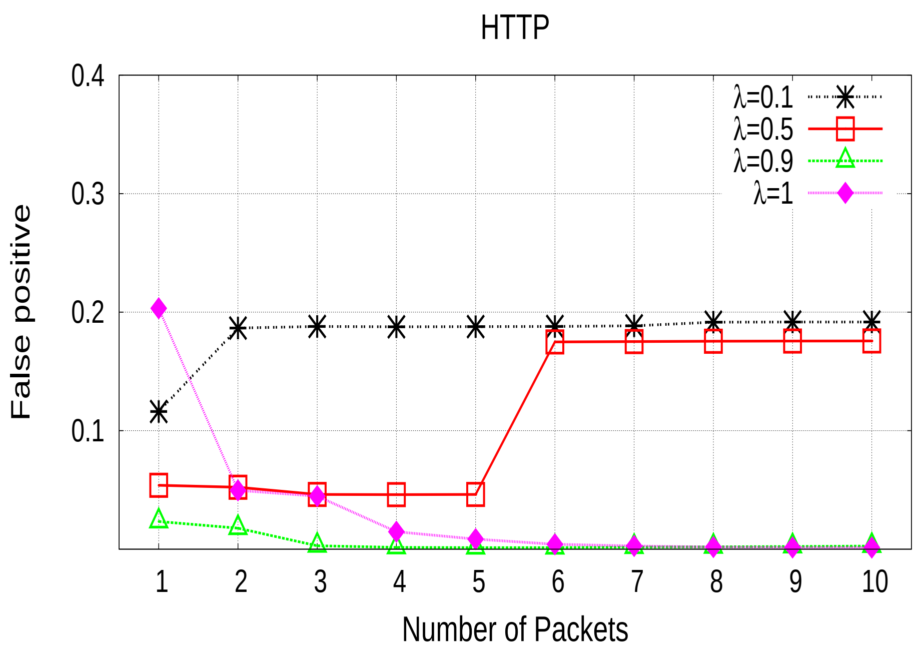

**Figure** 5.12: False positive ratio for HTTP application (Trace I).

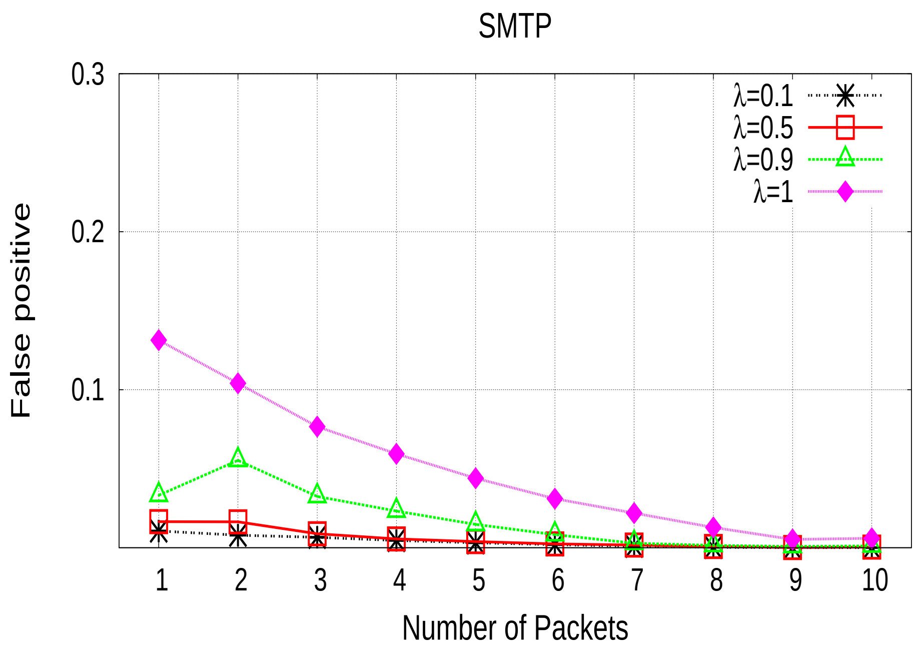

**Figure** 5.13: False positive ratio for SMTP application (Trace I).

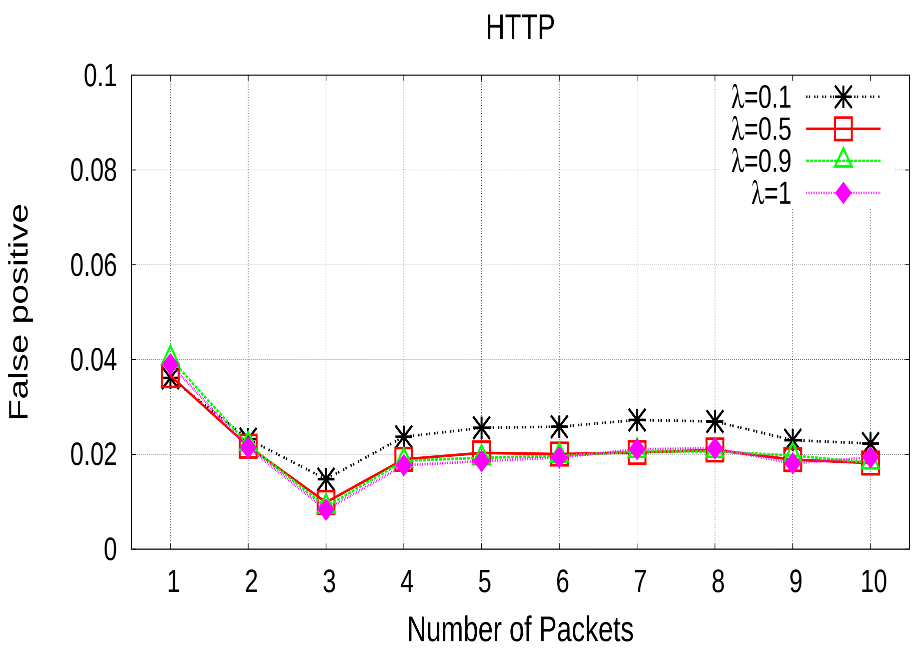

**Figure** 5.14: False positive ratio for HTTP application (Trace III).

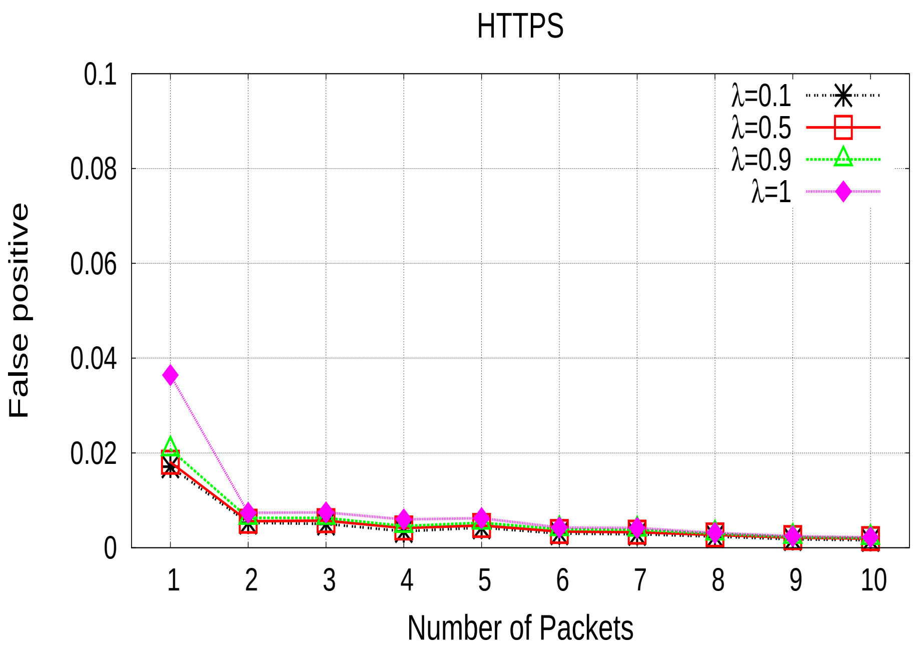

**Figure** 5.15: False positive ratio for HTTPS application (Trace III).

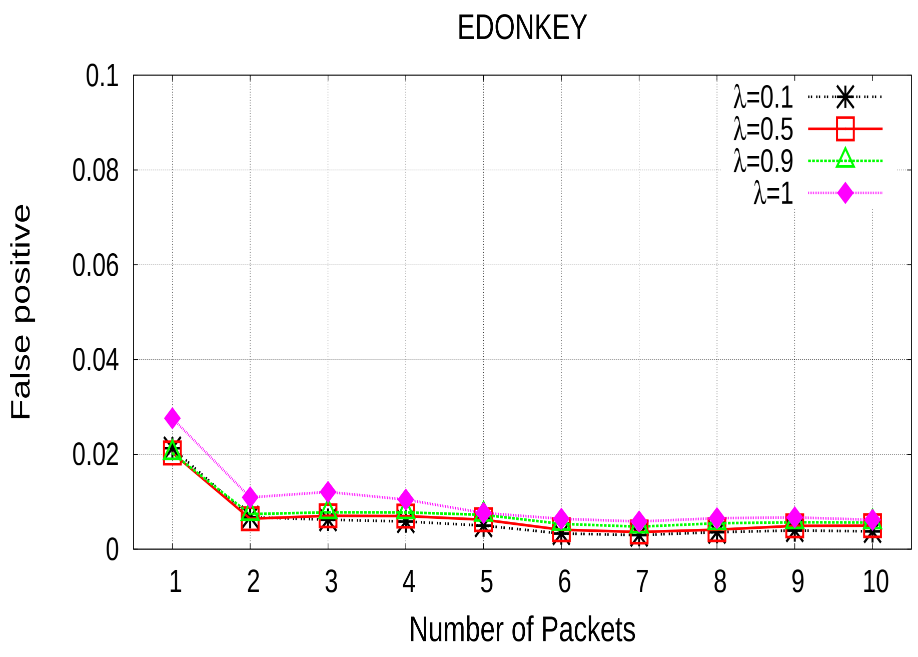

**Figure** 5.16: False positive ratio for Edonkey application (Trace III).

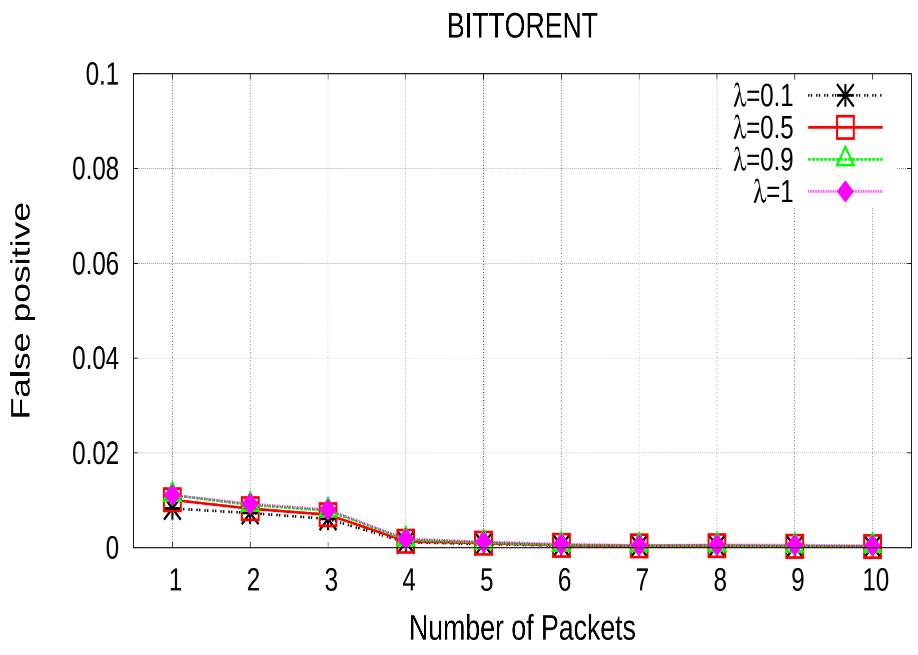

**Figure** 5.17: False positive ratio for Bittorent application (Trace III).

For HTTP the False Positive ratio reaches 2% for all values of λ only after two packets. We can notice the same behavior for HTTPS application where the false positive ratio drop to 0.5%. For BITTORENT, the False Positive ratio is always around 0% after the fourth packet and for all λ values. This observation can be explained by the fact that the size of the first packets for the BITTORENT application is different from the other applications. Finally for the EDONKEY application the False Positive ratio is around 3% for the first packets and continuously decreases until it reaches 1% after 7 packets.

The next section discusses the impact of λ for the classification. Then we analyze the profile of one host to understand which services it runs and determine its traffic pattern.

### 5.5.2 Importance of the discounting factor λ

Fig. 5.18 plots the total precision for Trace I as a function of λ. The discounting factor λ determines how previous flows are considered for the classification of a new one. We recall that for λ close to 1 the host profile is computed over a longer period and previous flows have similar weights. The opposite case is when λ is close to 0. A precision above 90% for all values of λ prove that our iterative method is effective in classifying the applications. However, we can see that the precision is a function of λ. In particular we notice that when few packets are used for the classification (4 in the plot), the profile of the user helps for an early detection of the Internet traffic. In this case, the machine learning algorithm does not have much information about a flow and it associates to applications comparable classification probabilities. Thus, for these first packets the profile of the host has higher influence on the output of the classification.

However, a correct λ is important to improve the classification even if more information is available. From Fig. 5.18, we can notice that high values perform better for the total precision. The host profile prediction of the application is more accurate because we associate a probability to the application of the next flow based on the fraction of the past traffic due to this application. In order to study the best λ value for every application we will plot the true positive ratio for the different applications.

In Fig. 5.19 we study the impact of λ on the true positive ratio of the HTTP application. We can see clearly that while using ten packets for the classification all values of λ give a good precision for the HTTP application. However, when we use a smaller number of packets for the classification a small value of λ gives better performance.

Fig. 5.20 shows that with any number of packets used for the classification, thus the true positive ratio of the SMTP application is higher when we use a value of λ less then 0.9. Thus we can use any value of λ from 0 to 0.9 if we are interested to classify the SMTP application with very high precision.

In Fig. 5.21 we plot the true positive ratio for the POP3 application as a function of λ. For this application and for any number of packets used for the classification we can notice that the

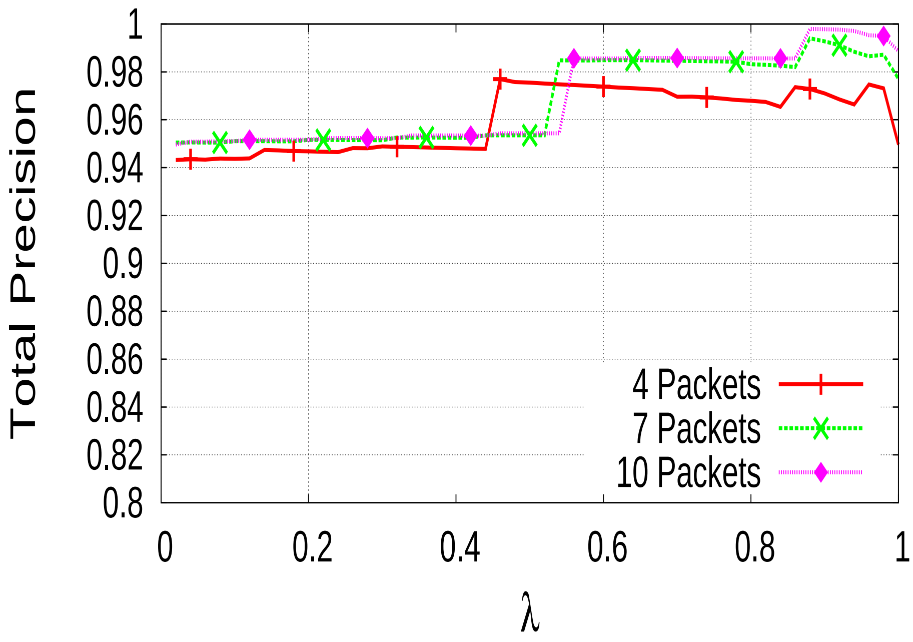

**Figure** 5.18: Total precision computed after 4, 7, and 10 packets as a function of λ.

better performance is when we use a large value of λ, especially a $\lambda = 0.88$. If we use a small value of λ, the performance of the classifier decreases a lot and we will have a considerable ratio of POP3 flows classified as HTTP. We can confirm these analysis when we observe the false positive ratio of the HTTP application in Fig. 5.22. The figure show that the false positive ratio of HTTP is high and around 20% for small values of λ. In the next section, we characterize this traffic and how the flows are distributed, in order to analyze more the reasons of this decrease in the true positive ratio of POP3 application.

### 5.5.3 Traffic pattern of a host

In the Trace I, described in Table 5.2, we have identified different types of hosts. In particular, there are some hosts within the Brescia campus that are dedicated to single services and other hosts that use all the three applications. In this section, we analyze in details the traffic of the latter type of hosts and we determine their profile (as example we analyze one host generating a big number of flows). This will shed light on the importance of λ for the identification of the Internet traffic. The total number of flows generated by the IP, noted IP1, is 14, 151 flows, which is subdivided as follows: 7, 101 HTTP flows, 7, 014 POP3 flows, and only 35 SMTP flows.

Fig. 5.23 plots the cumulative distribution (CDF) of consecutive flows of the same type of

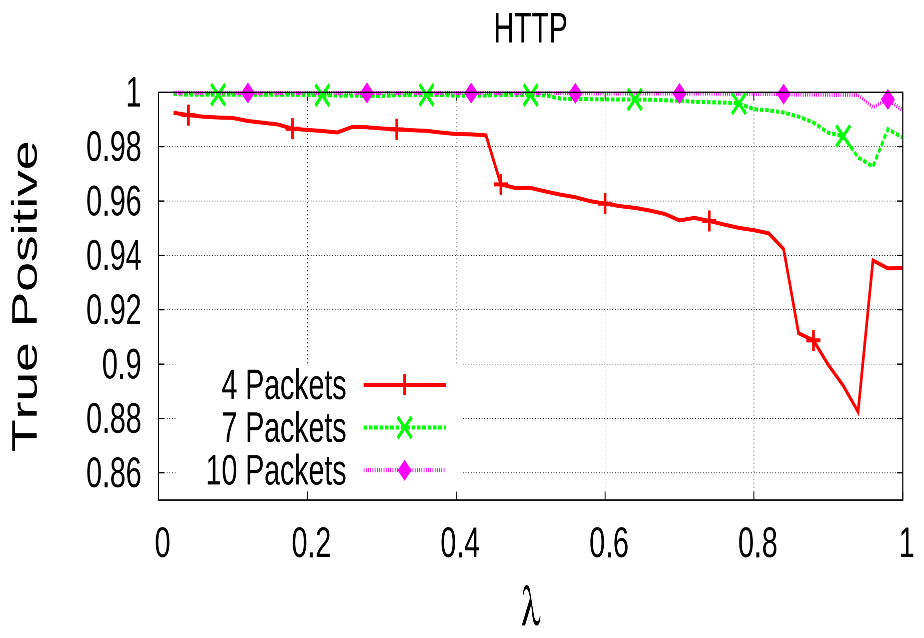

**Figure** 5.19: True positive for HTTP computed after 4, 7, and 10 packets as a function of λ.

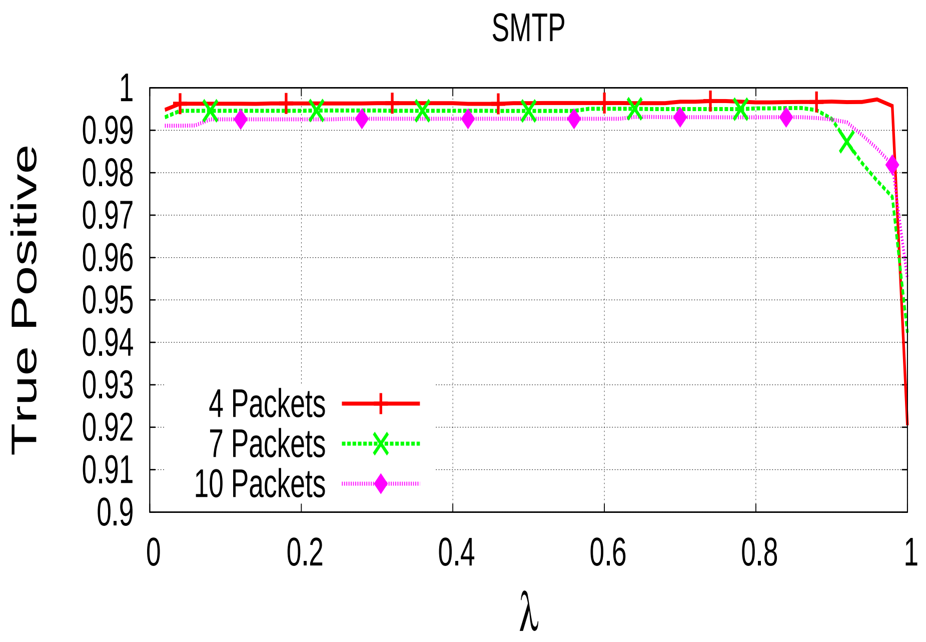

**Figure** 5.20: True positive for SMTP computed after 4, 7, and 10 packets as a function of λ.

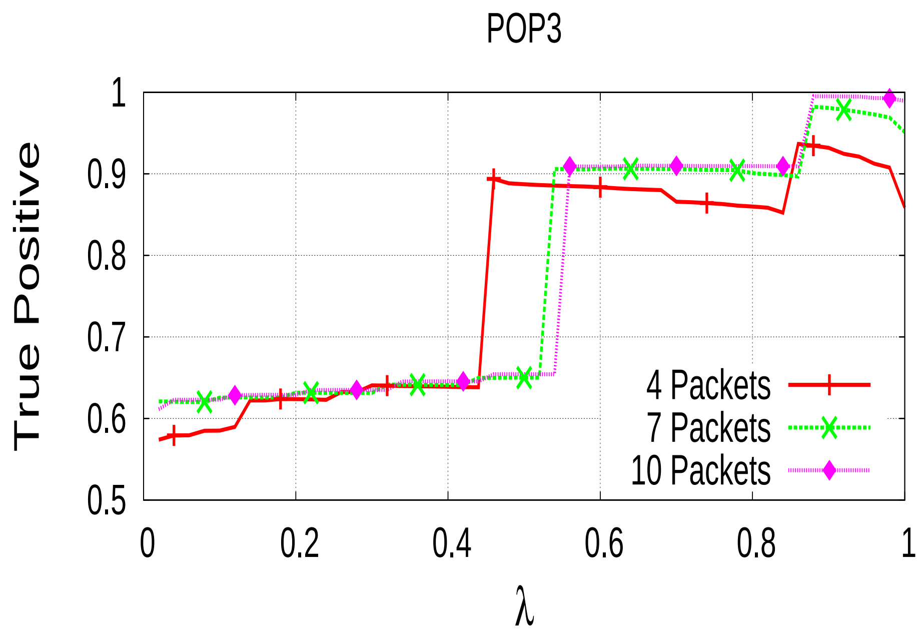

**Figure** 5.21: True positive for POP3 computed after 4, 7, and 10 packets as a function of λ.

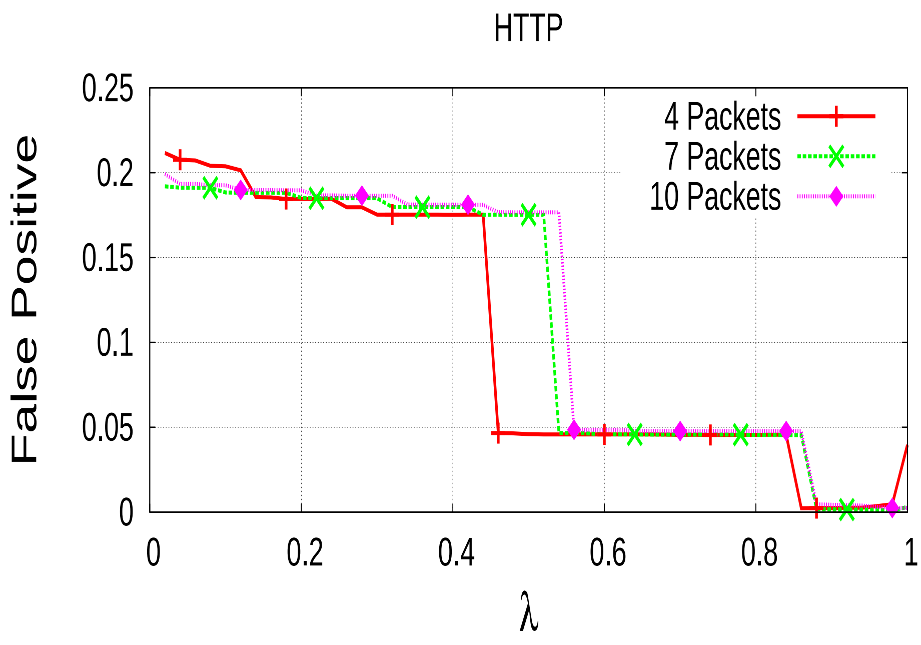

**Figure** 5.22: False positive for HTTP computed after 4, 7, and 10 packets as a function of λ.

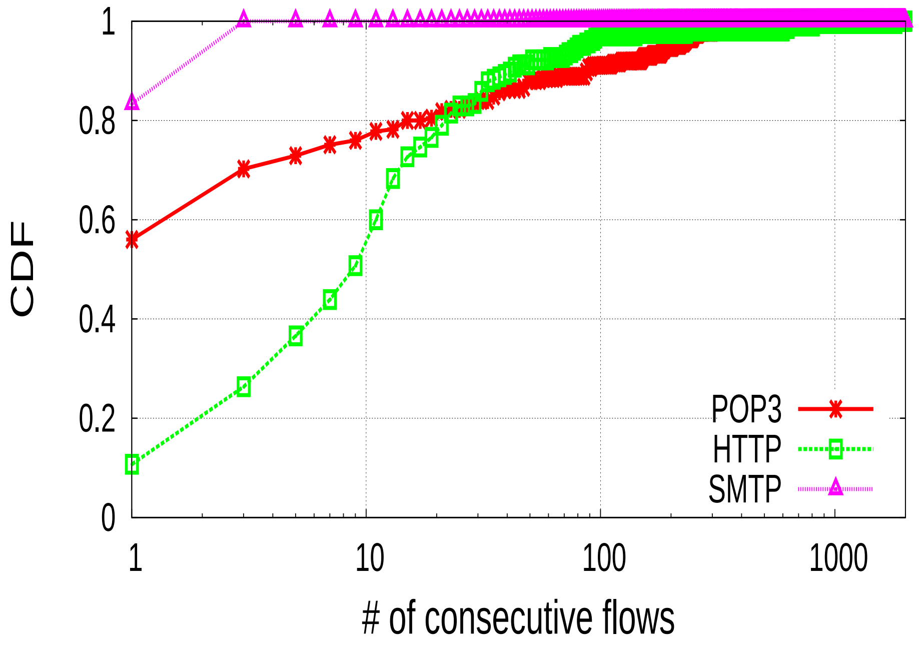

**Figure** 5.23: IP1: number of consecutive application flows.

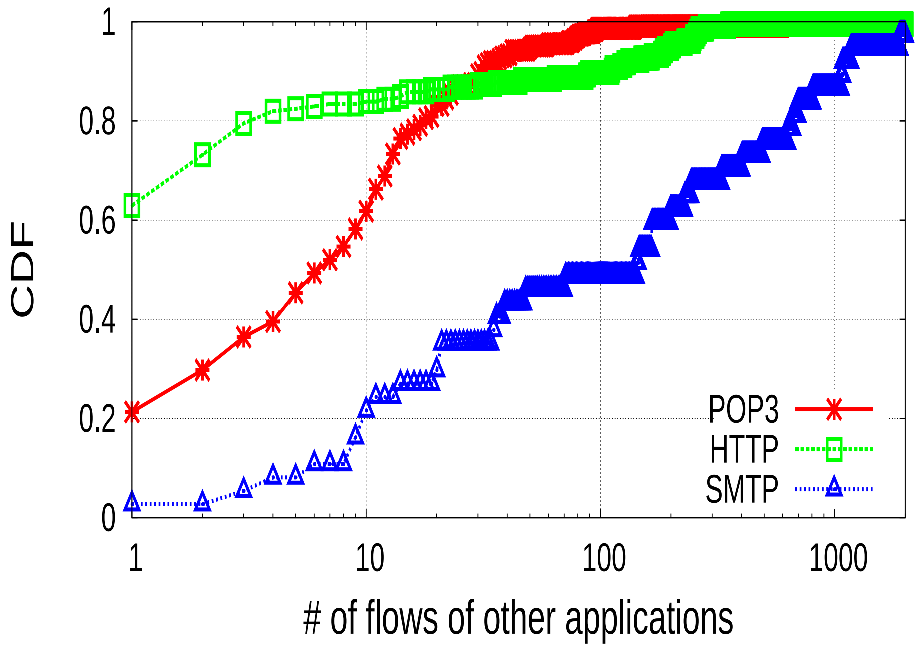

**Figure** 5.24: IP1: number of flows of other applications.

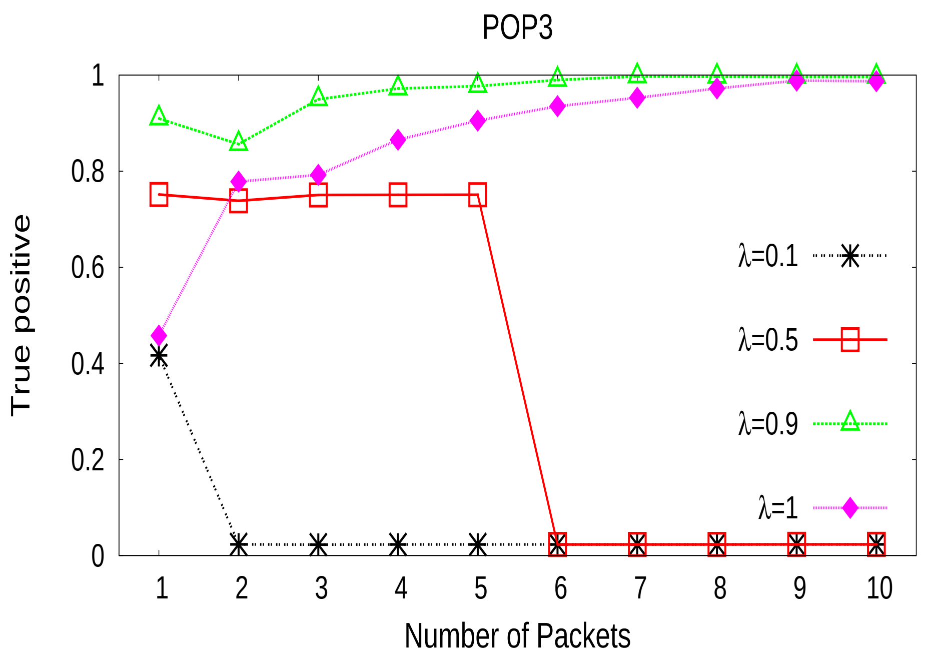

**Figure** 5.25: IP1: True positive ratio for POP3 application.

application in a semi-log scale. Since there are only 35 SMTP flows, most of these flows are isolated so that there are only sequences of 1 to 3 consecutive flows (burst). Almost 60% of the POP3 traffic consists of a single flow. The rest of the POP3 traffic consists of single larger bursts and the largest one consists of almost 700 flows. The number of small HTTP consecutive flows is 10% of the total number of HTTP bursts and 80% of the bursts consist of less than 100 flows. This corresponds to a typical browsing activity of a user, who surfs from one web site to another by following the hyper-links.

The fact that the medium size burst of HTTP traffic get interleaved with POP3 has evident impact on our profiling method when we use a small value of λ. In particular, since there are more consecutive HTTP flows than POP3, the host based classification with a small value of λ is more sensitive to the presence of a new burst. Moreover, the POP3 and HTTP packet size has similar distribution for certain packet numbers, as discussed before in Chapter 3, making the host profile decide of the classification.

To conclude the characterization of the profile, Fig. 5.24 plots the cumulative distribution (CDF) of flows of other applications that separate two flows of the same application. There are more than 60% of HTTP flows burst separated by only 1 flow of another application and 80% of HTTP flow bursts separated by less than 10 consecutive flows. As for POP3, there are 20% and 60% of bursts respectively. The presence of small bursts of other applications between two HTTP flows justifies the high value of false positive ratio (see Fig. 5.12). In general, this

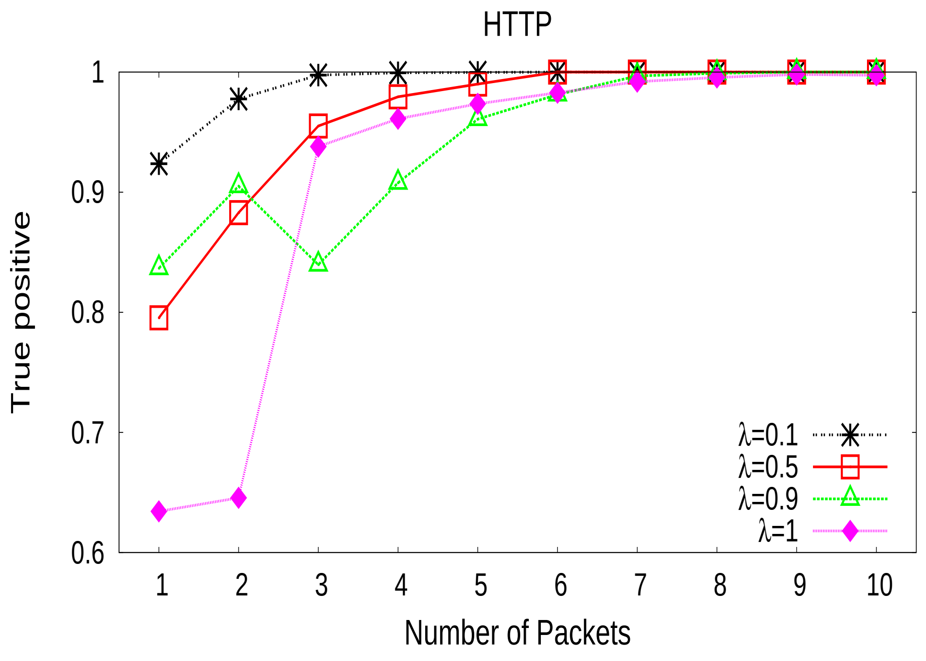

**Figure** 5.26: IP1: True positive ratio for HTTP application.

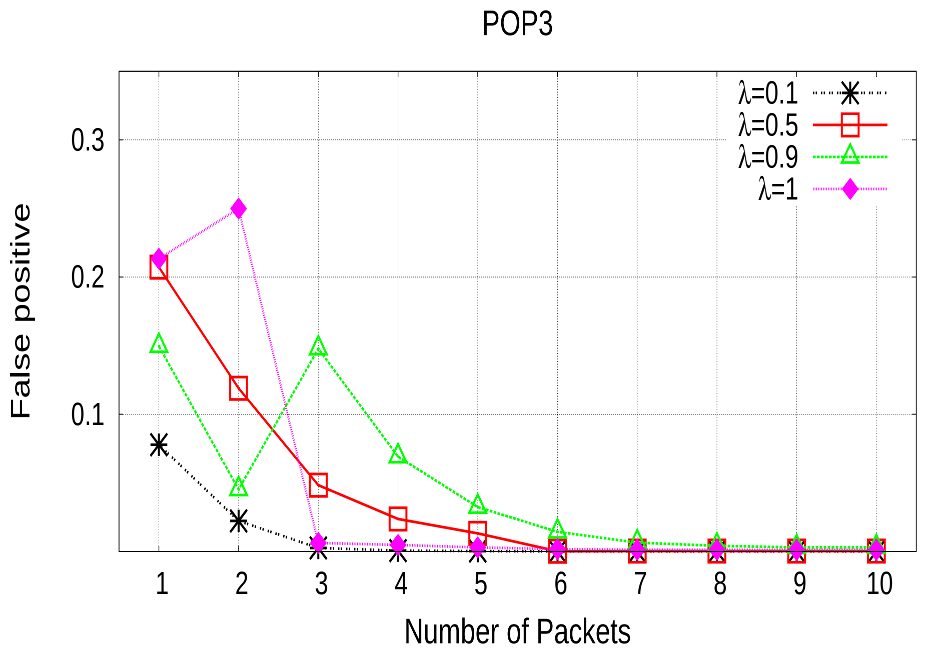

**Figure** 5.27: IP1: False positive ratio for POP3 application.

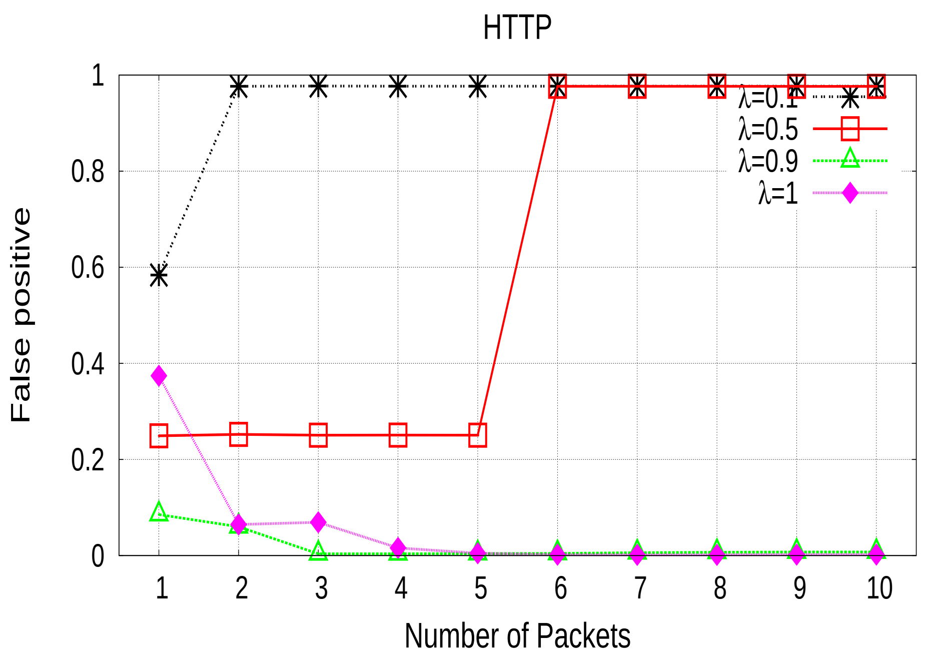

**Figure** 5.28: IP1: False positive ratio for HTTP application.

is more pronounced with small values of λ. In Fig. 5.25, 5.26, 5.27 and 5.28 we show the classification results for IP1. We can clearly see that the true positive of the POP3 application drops to 0 approximately for small λ where us for large values of λ it keeps increasing. At the same time we observe the increase in the false positive ratio of the HTTP application, so POP3 flows are classified as HTTP because of the HTTP flow burst. The results confirm the analysis of Trace I, and the impact of the profile of the host on the classification.

### 5.5.4 Trace aggregation

For the validation of our method we use two different traces, every trace contains a set of applications. The training phase was done by using only the applications present in each trace. As we don't have a large number of applications in each trace, we notice that our classification results can be biased by the fact that the training phase is done only for few applications. In order to confirm our results with a larger set of applications we decided to test our method by means of an aggregated model. We do a joint training for the entire set of applications present in the traces, and we build an aggregated traffic model, then we classify the traffic based on this model. We construct our model by using ten known applications present in our data set. The applications that we used are: SKYPE, BITTORENT, EDONKEY, FTP, HTTP, SMTP, SSH, HTTPS, IMAP, and POP3. We used the same number of flows for each application and we use as input for the K-Means algorithms 200 clusters. In the following we present the results of classification of trace III while using the aggregated model obtained from the three traces together using the sizes of packets.

In Fig. 5.29 we present the total precision of the trace III. We can see clearly that we still have a very good precision, even after the use of the aggregated model. We can observe that for $\lambda = 0.1$, $0.5$ and $0.9$ the performance of the classifier is better in comparison with $\lambda = 1$ (without host information) especially for the first five packets. The aggregated model leads to slightly lower accuracy when compared to using a separate model learned from trace III ( see Figure 5.3). For $\lambda = 1$ we have a total precision of 82% for the first packet, this precision increases and reaches 96% after five packets and 98% after 8 packets. For the smaller values of $\lambda$ the precision is better. We start from 86% for $\lambda = 0.9$, and 88% for $\lambda = 0.5$ and $0.1$. For all the values of $\lambda$ the precision converges after the sixth packet to 98%.

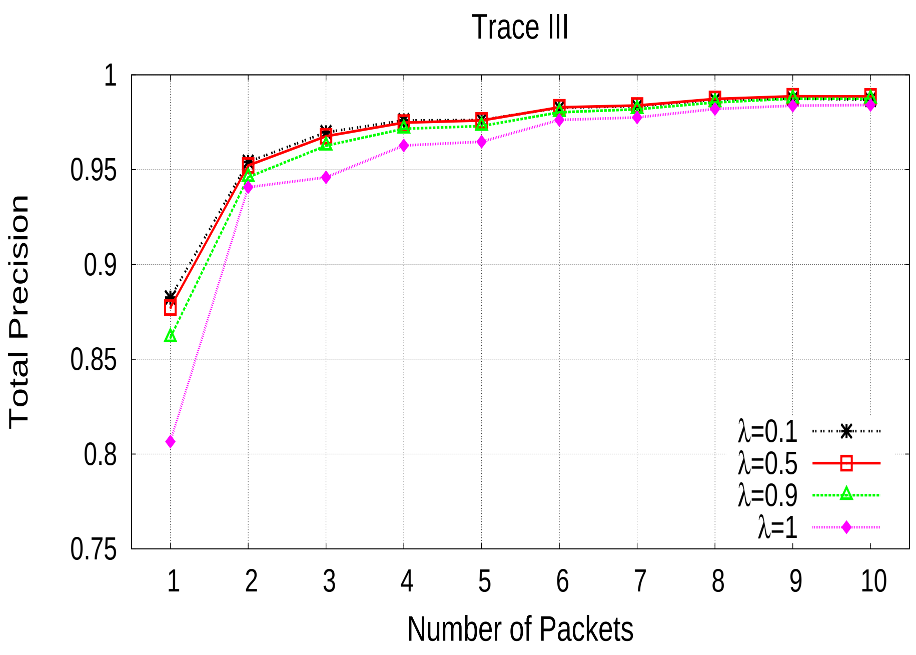

**Figure** 5.29: Total precision After trace aggregation (Trace III).

In Fig. 5.30, 5.31, 5.32 and 5.33 we present the true positive ratio for the HTTP, HTTPS, EDONKEY and BITTORENT applications. For the HTTP application, the precision is very good for all the values of $\lambda$, after one packet this precision is 85% for $\lambda = 1$ and around 94% for the other values of $\lambda$. After four packets the true positive ratio reaches 97% approximately for all the values of $\lambda$. For the HTTPS application the true positive ratio after one packet used for the classification starts with 63% for $\lambda = 1$ and 70% approximately for the other values of $\lambda$. The true positive ratio keep increasing with every new packet and reaches 93% after six packets and

95% after 8 packets. For the EDONKEY application the true positive ratio starts with 54% for $\lambda = 1$ and 0.9 and around 63% for $\lambda = 0.5$ and 0.1. After the first packet, the true positive ratio increases for all the values of $\lambda$ and reaches 93% after four packets and 97% after 8 packets. For the BITTORENT application we have an excellent result for all the values of $\lambda$, we can observe a little decrease of 1% in comparison with the previous result with the separated model, but we still have a true positive ratio larger than 97% for all packets.

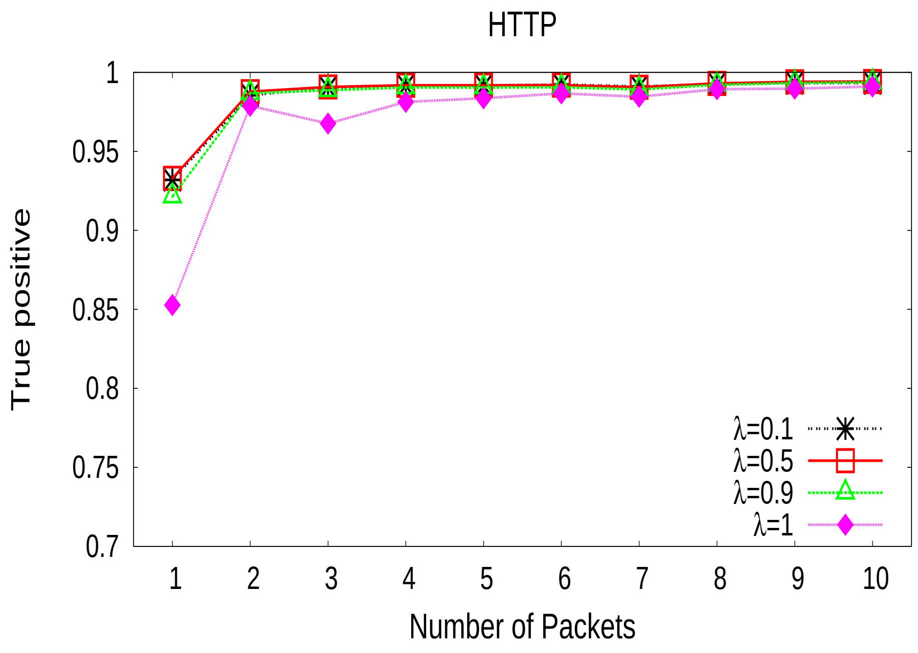

**Figure** 5.30: True positive ratio for HTTP application after trace aggregation (Trace III).

In Fig. 5.34, 5.35, 5.36 and 5.37 we present the false positive ratio of the HTTP, HTTPS, EDONKEY and BITTORENT applications. As for the separated model, we can see clearly that the false positive ratio for all the applications is very small and drops below 2% after using 2 packets for the classification. From all these results we can confirm our claims about the useful use of the host information to refine the classification of Internet traffic when exploiting the statistical features of packets as a means.

## 5.6 Conclusions

In this chapter we present our new method for Internet traffic identification that combines the statistical and host-based approaches. The statistical parameters that we use are the size and direction of the first N packets. The novelty of our approach consists in leveraging the host profile to refine the classification. First we define the profile of the host and how it is

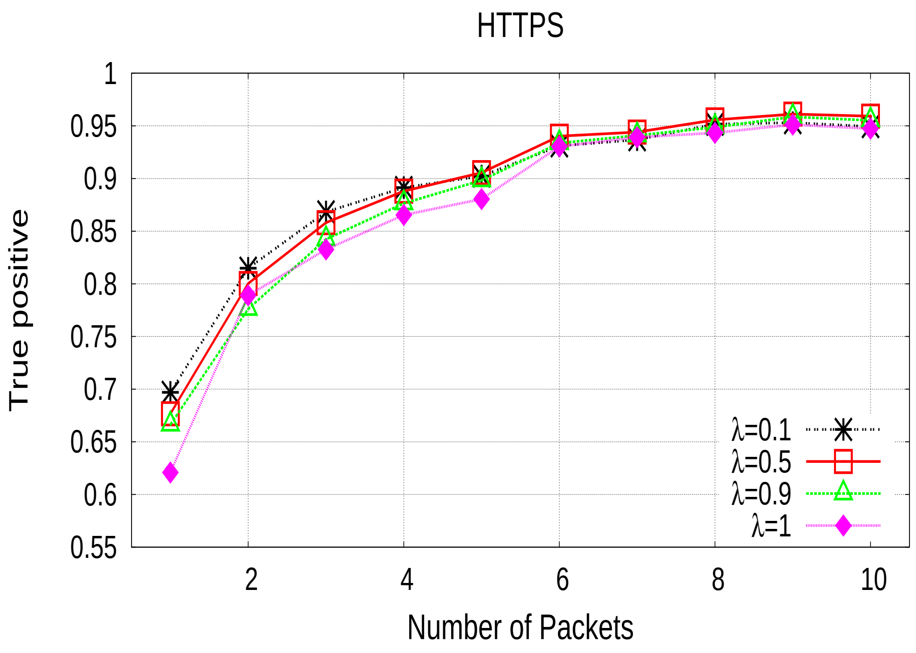

**Figure** 5.31: True positive ratio for HTTPS application after trace aggregation (Trace III).

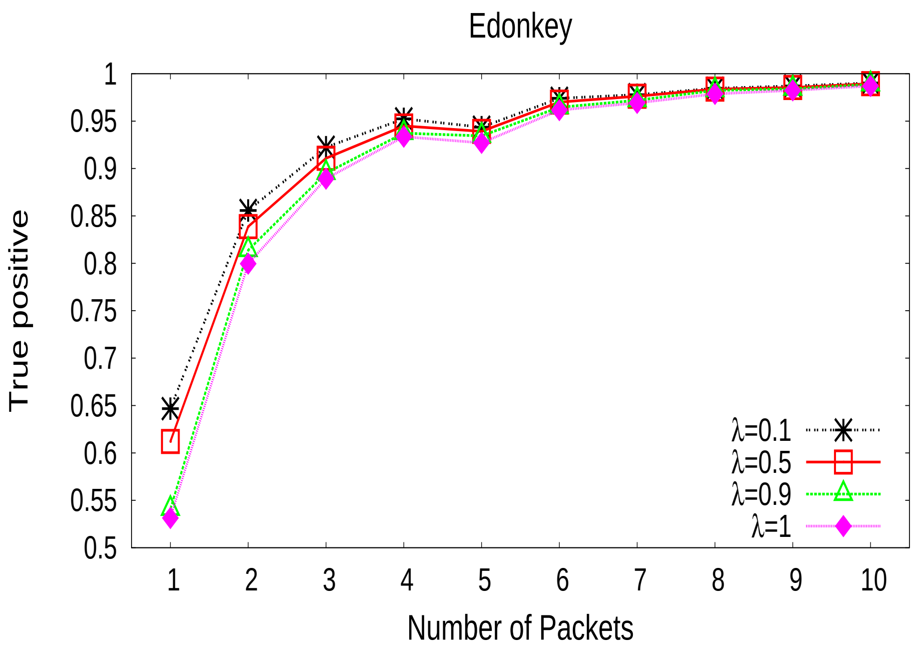

**Figure** 5.32: True positive ratio for Edonkey application after trace aggregation (Trace III).

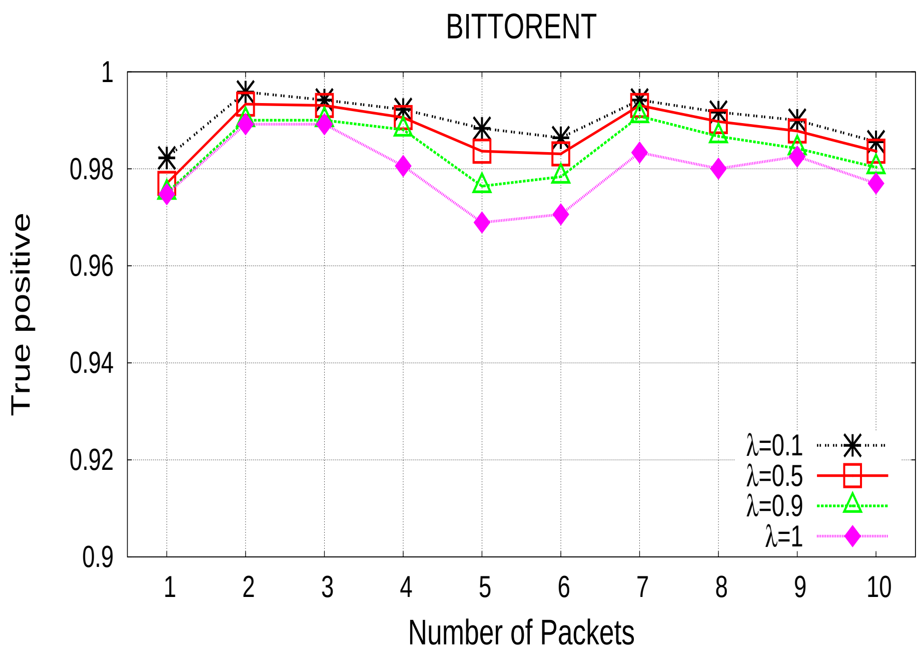

**Figure** 5.33: True positive ratio for Bittorent application after trace aggregation (Trace III).

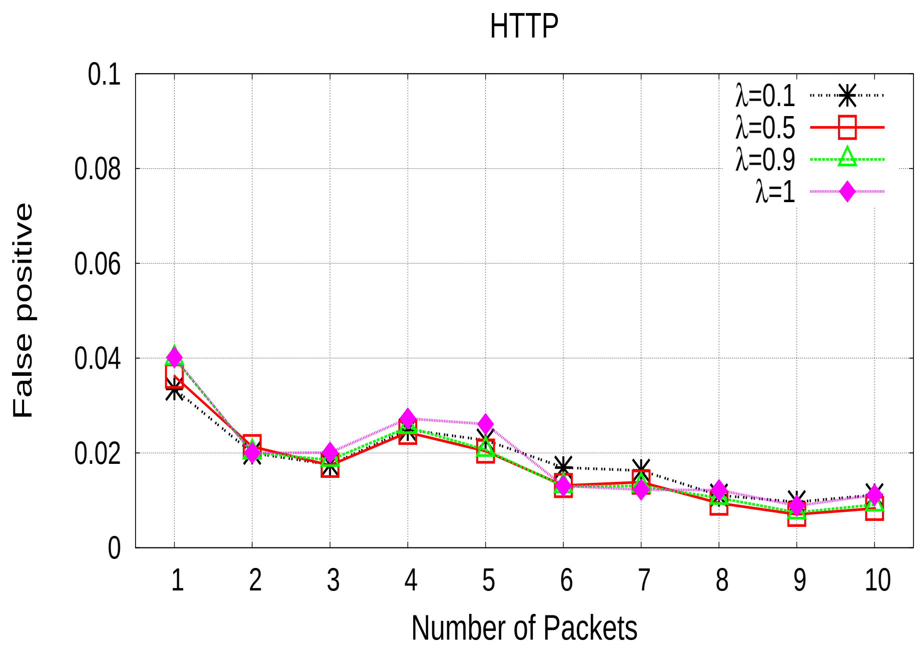

**Figure** 5.34: False positive ratio for HTTP application after trace aggregation (Trace III).

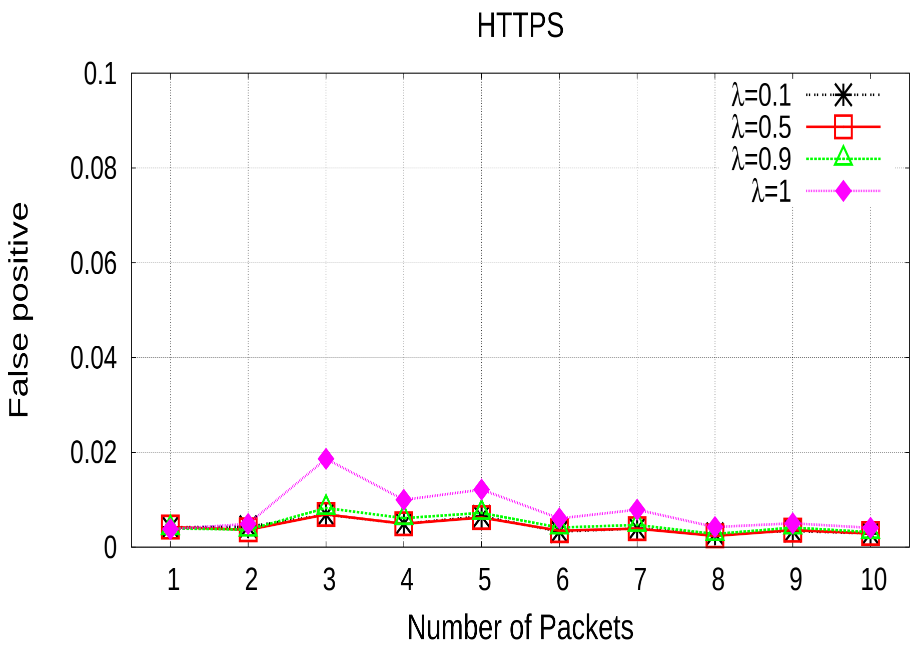

**Figure** 5.35: False positive ratio for HTTPS application after trace aggregation (Trace III).

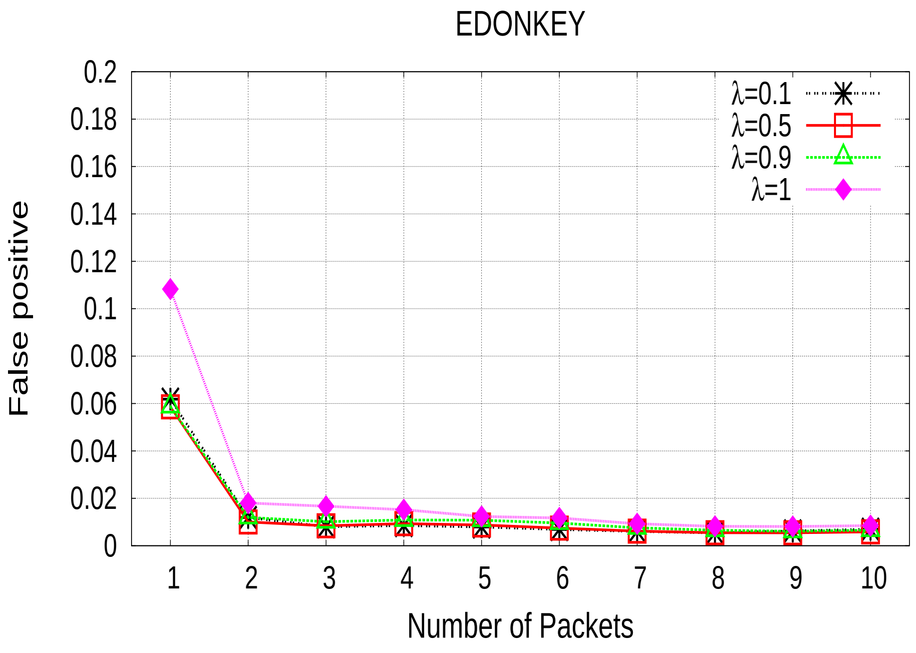

**Figure** 5.36: False positive ratio for Edonkey application after trace aggregation (Trace III).

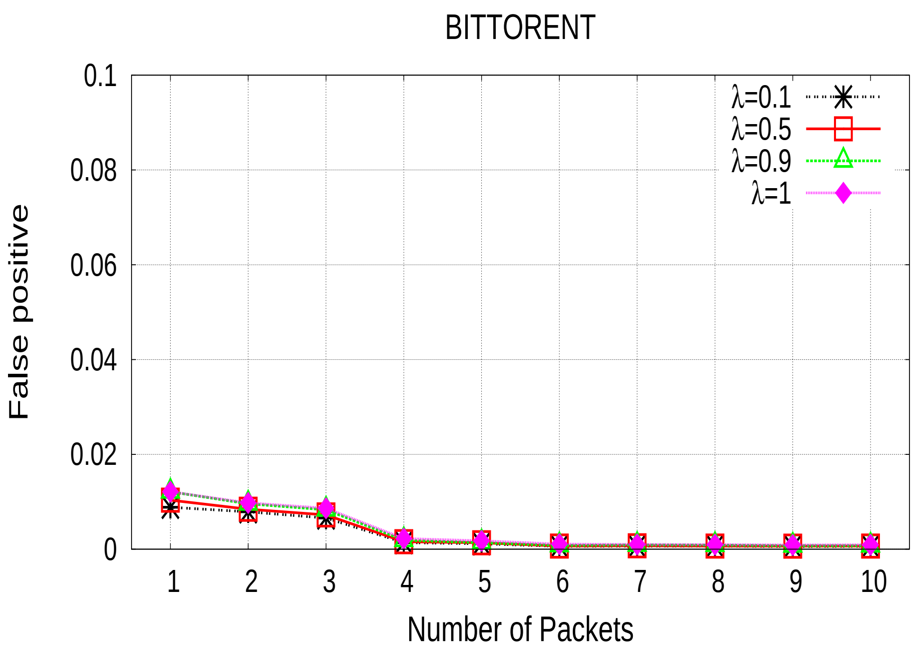

**Figure** 5.37: False positive ratio for Bittorent application after trace aggregation (Trace III).

updated. Then we show how the profiles of the source and destination hosts are used to assign a prediction probability to the new flow.

We evaluate our solution on two real traces and we profile the hosts with the same IP prefix. We test our method for different values of the discounting factor $\lambda$ and discuss the optimal choice based on the traffic pattern of the host. The results show a great improvement for the classification of applications when the host profile is used. In particular, the classifier reaches a precision of 98% after using 10 packets for the classification. Finally, we characterize the host profile and show the distribution of the flows, i.e., the traffic pattern of a representative host. Next we start the second part of this thesis where we explain the work we did about anomaly detection using the host coordinate and without the need for high cost measurements.

# 6

# Can we detect network changes by using Vivaldi coordinates?

The Vivaldi algorithm [10] is known to be one of the most interesting approaches for the calculation of Internet coordinates. It is a fully distributed, light-weight and adaptive algorithm, requiring no fixed network infrastructure and no distinguished nodes. Recent studies show that host coordinates in the Vivaldi system are not stable and they drift rapidly even when the network delays do not change. In this chapter, we observe that, despite the instability of Vivaldi coordinates in their absolute values, there is still a stable internal structure that can better reflect the stability of the underlying network. We proceed for this study by extensive simulations and experimentations. In the first stage, we confirm the fact that Vivaldi coordinates oscillate over time because of the adaptive nature of the system. The variations of these coordinates however are most of the time correlated with each other pointing to some stable cluster of nodes seen from inside the network. In a second stage, we present a new clustering algorithm based on the Hierarchical Grouping method [62] to identify a cluster of stable nodes once the network and the host coordinates reach their stationary regime. The metric that we use to cluster nodes in the system is the amount of variation of their Euclidean distances with respect to each other. Our main finding is that a stable cluster of nodes always exists and that this cluster groups most of the nodes. We highlight the utility of such finding with an application that tracks changes in network delays. To this end, we propose to track a simple signal, which is the size of this biggest stable cluster. By changing artificially the network delays in different scenarios, we show that these changes are easily reflected by this body of stable nodes, hence allowing to obtain a global picture about the stability of the underlying network without the need for exhaustive delay measurements. The results of this chapter were published in [23].

## 6.1 Introduction

During the last years, the Internet has evolved from a small academic network to a huge network interconnecting tens of thousands of autonomous systems (AS). This rapid growth of the Internet infrastructure, coupled with the flexibility of the end-to-end service it provides, has led to a vigorous development of Internet usage and the appearance of a wide variety of applications. Many of these applications such as skype [56], KaZaA [28], RON [53], Akamai [1], and BitTorrent [4], are based on the overlay technology, where a set of hosts and proxies collaborate together to provide a global service to end users. For these applications, the positioning of each host with respect to the entire overlay is important for its configuration and for the dissemination of content through it. It is also important for the tracking of any change in network conditions and for the reconfiguration of the overlay accordingly. For example, a BitTorrent peer gains by exchanging data with neighboring peers [27] and a regular web client gains in transfer time by connecting to a close web server [52]. The positioning in the Internet can be achieved by exhaustive and regular measurements between all hosts involved in the overlay communication, however given the large size of these overlays, this solution becomes unfeasible. One must add to this difficulty the impracticality of a solution per application and the interest of a universal positioning service.

Recently, a new approach has emerged for Internet positioning having the main advantage of providing estimates for network delays between machines at a low measurement cost. This approach consists in building a coordinate system for the Internet [33]. The basic idea is to embed all the nodes of a given application (or overlay) in some Euclidean space, and to associate to each node specific coordinates in this space in such a way that the network delay between any two nodes can be approximated by the geometric distance separating them (Figure 6.1). We can distinguish two main groups of network coordinate systems: the Centralized coordinate systems like GNP [42] and the distributed coordinate system like Vivaldi [10]. In GNP we have a set of nodes named landmarks that are used as references to compute the coordinates of the other nodes. For the distributed coordinate systems like Vivaldi all nodes are similar and the coordinates are calculated in a distributed way without any central unit. Both groups aim to minimize the global error between network delays and geometric distances resulting from the embedding. The strong point of these groups is their ability to calculate coordinates with a subset of the global delay matrix, allowing then to infer the rest of the network delays with satisfactory error. Their problem however is in the difficulty to embed paths violating the triangular equality as such violation is possible in the Internet whereas it cannot be modeled by the Euclidean metric.

The Vivaldi algorithm [10] is known to be one of the most promising approaches for Internet coordinates. It is a fully distributed, light-weight and efficient algorithm, requiring no fixed

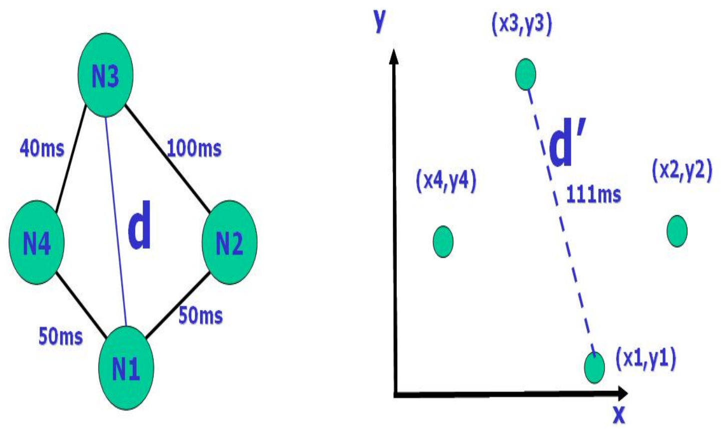

**Figure** 6.1: Delay embedding and estimation by an Internet coordinate system.

network infrastructure and no distinguished nodes. It is based on an emulation of paths by springs, where the initial lengths of springs is equal to network delays and the actual lengths of springs to geometric distances. It follows that the positions of the nodes that minimize the potential energy of the springs also minimize the embedding error. This path-spring emulation has inspired the authors of Vivaldi to propose a fully distributed algorithm that uses network measurements to adjust coordinates towards their optimal values. According to the Vivaldi algorithm, by only measuring the network delay to few neighbors, and knowing the coordinates of these neighbors, a node can adjust its coordinates in steps until reaching the stable regime. The low cost of Vivaldi and the ease of its deployment have motivated us to run exhaustive measurements and simulations to understand the behavior of node coordinates and the extent to which they can be used. In particular, we are interested in evaluating the stability of these coordinates. Such an evaluation shows that Vivaldi can be leveraged to monitor a large network and to detect and diagnose any change such as a long-term congestion, a rerouting, a link failure, or any other event causing a shift in network delays. In the literature, Internet coordinates have been often studied from the perspective of estimating network delays. Here, we invert the problem and we study them from the perspective of network monitoring, given their light-weight feature and their distributed nature.

Vivaldi coordinates are largely covered in the literature. Several papers, Kaafar et al. [24] and Wang et al. [60] study the problem of coordinates security. In [30], [31], [61] the authors show that network coordinates in the Vivaldi system are not stable and the coordinates drift rapidly even when the network delays do not change. In this work we show that, despite the instability of Vivaldi coordinates in their absolute values, there is still a stable internal structure that can better reflect the stability of the underlying network. Our contributions can be summarized in three main points.

1. We confirm the fact that Vivaldi coordinates oscillate over time because of the adaptive nature of the system. The variations of these coordinates however are most of the time correlated with each other, pointing to some stable cluster of nodes seen from inside the system.

2. We propose a new clustering algorithm based on the data mining Hierarchical Grouping method [18] to identify this cluster of stable nodes once the network and the host coordinates are in their stationary regime. The metric that we use to cluster nodes in the system is the amount of variation of their Euclidean distances with respect to each other. One important observation is that such stable cluster of nodes always exists and is grouping most of the nodes.

3. We highlight the utility of the stable internal structure of Vivaldi coordinates with an application that tracks changes in network delays. To this end, and instead of tracking jointly the global matrix of network delays, we propose to track a simple signal being the size of this biggest stable cluster.

The details of our anomaly detection protocol will be described in Section 6.7. By changing artificially the network delays in different scenarios, we show that these changes are easily reflected by this body of stable nodes, hence allowing to obtain a global picture about the stability of the underlying network without the need for exhaustive delay measurements.

The remaining of this chapter is organized as follows. In Section 6.2 we give a background about network coordinate systems and we focus on Vivaldi as we use it in this work. Section 6.3 explains our experimental methodology and 6.4 presents first results to motivate the work. Section 6.5 presents our clustering algorithm. Section 6.6 summarizes the experimental results confirming the presence of a stable internal structure and how network changes are reflected by this stable structure. In Section 6.7 we present our anomaly detection protocol and Section 6.8 explain the validation results. Finally, we conclude the chapter in Section 6.9 with some perspectives on our future research.

## 6.2 Background on Network Coordinates

In the last years several approaches have been proposed based on the definition of Internet as a geometric space, see Figure 6.2. According to this definition, a few measurements can be sufficient to predict the end-to-end delay of networking elements. The key idea is to characterize the position of the node in the space with its coordinates in such a way that the geometric

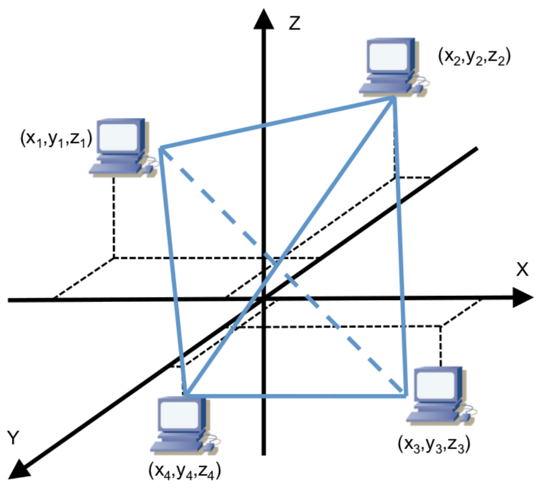

**Figure** 6.2: Geometric space model of the Internet.

distance of any two nodes approximates their network delay. This implies that the RTT (Round-Trip Time) measurements are only used to determine the exact position of every node in the space. The approach can scale well in large networks because all the end-to-end delays can be obtained by computing the geometrical distance without the need to do exhaustive delay measurements.

We can distinguish two main approaches for network coordinates which will be discussed in the following sections:

**Centralized** coordinate systems are build with the assumption that there exists a centralized component, i.e., a set of nodes named landmarks, that are used to compute the coordinates of every node.

**Distributed** coordinate systems leverage the concept of peer-to-peer networks and compute the coordinates of the nodes in a controlled and distributed way.

### 6.2.1 Centralized coordinate systems

The first approach for landmark-based coordinate system is the Global Network Positioning (GNP) [42] which has been later extended to improve the computation and the accuracy of the end-to-end measured delay. GNP starts by computing the coordinates of a set of nodes selected to be the landmarks, i.e., the reference nodes for the others. This is accomplished in a centralized way by first collecting the RTT between any pair of landmarks and determining their coordinates by minimizing the error between the measured RTT and the predicted geometric distance computed on the coordinates themselves. In a second step, each node retrieves the coordinates of the landmarks and uses them as a basis to compute its own coordinates by minimizing the error between the geometric distance and the measured distance to the landmarks.

The number of the landmarks is an important parameter to reduce the error of the end-to-end delay prediction. In [42], the authors show that it is feasible to consider a low dimensional Euclidean space to embed the network coordinates of the nodes. In fact, with 7 landmarks, 50% of the distance predictions have less than 10% error, 90% of them have less than 50% error.

Other approaches that extend GNP have been presented in the literature. Tang and Crovella [57] have proposed the use of ”virtual” landmarks to determine the coordinates of the nodes. Every node determines its networking position with respect to a set of landmarks by using the Lipschitz embedding: the coordinates are the end-to-end delays to the set of landmarks. Thus, if there are $n$ landmarks, the coordinates are expressed in a $n$-dimensional vector. The authors then reduce the dimensional space by compressing the delay matrix, using the Principal Component Analysis (PCA) method. The idea is to use a linear transformation to map each vector in another space with reduced dimensions $m << n$ in such a way that the computed distance in the new space is close to the original distance. Thus, the new coordinates of a node can be expressed as the networking distance to a set of $m$ virtual landmarks. In this way nodes that are close still maintain their distance relationship in the new dimensional space.

### 6.2.2 Distributed-based coordinate systems

In this section, we discuss the distributed approach that leverages the concept of peer-to-peer to use either any existing node as landmarks or by directly computing the distance without requiring an infrastructure.

Practical Internet Coordinates (PIC) [34] does not require a specific set of landmarks to compute the coordinates since a node can select as landmarks nodes whose coordinates have been already computed. PIC uses the Simplex Downhill method to minimize the objective distance error function (sum of relative errors of the distances to the set of landmarks). The discovery of the nodes with computed coordinates can be achieved by picking randomly nodes, or selecting those close, or by a hybrid selection.

Since PIC does not use a fixed set of landmarks, it can be exposed to malicious attacks from nodes which provide false coordinates. To overcome this issue, PIC eliminates landmarks that do violate the *triangle inequality*: given three nodes, $a$, $b$ and $c$ then $d(a, c) \leq d(a, b) + d(b, c)$, where $d(a, b)$ is the distance between node $a$ and $b$. However, triangular inequalities might occur because of IP routing irregularities even without the presence of malicious nodes.

The most used distributed approach to compute network coordinates is Vivaldi [10]. It does not require any fixed network infrastructure or landmarks to determine the coordinates. Indeed any node can compute its own coordinates after collecting end-to-end delay to other nodes. The key idea for computing the coordinates is the simulation of a spring, which is placed between any pair of nodes and the length of the spring in its rest position is the estimated distance of the nodes in the coordinate space. The potential energy of the spring is proportional to the square of the shift from its rest position and it gives the error function that is minimized at each iteration.

In Vivaldi, the coordinates can be computed on Euclidean spaces of any dimension by using the same principle, where the higher is the size of the space, the lower is the error in predicting the RTT between nodes. In addition, Vivaldi introduced the concept of height as additional dimension representing the access link of the node while the Euclidean space resembles the core of the Internet.

### 6.2.3 Vivaldi description

Vivaldi is a fully distributed, light-weight and efficient algorithm, requiring no fixed network infrastructure and no distinguished nodes. It is able to estimate network delays accurately while minimizing the load on the network. The basic ideas of Vivaldi can be summarized as follows:

- Every node has an estimation of its coordinates.
- It picks a random node and asks for its coordinate estimation.
- It computes the Euclidean distance between itself and this selected node and compares it to the network delay between them.
- It computes the stretch of the path and decides on the change to introduce into its coordinates.
- All nodes keep applying this algorithm in a decentralized manner until all the system stabilizes.

In Vivaldi each node calculates its own coordinates by performing periodic measurements of Round-Trip Time (RTT) with a small number of other nodes called neighbors chosen arbitrarily. After each measurement made with one of these neighbors, the node performs an update of its coordinates: it approaches or recedes from its neighbor in an incremental way minimizing the difference between the RTT estimated using the coordinates and the measured RTT. In summary, when node $i$ communicates with its neighboring node $j$, it looks for three inputs:

- The round-trip time $RTT_{m,i,j}$ between the node $i$ and the node $j$.
- The coordinates of the node $j$, $x_j$.
- The estimated error reported by $j$, $e_j$.

After receiving these inputs, the node $i$ performs the following steps:

- It estimates the network delay between $i$ and $j$ by the Euclidean distance, i.e., $RTT_{e,i,j} = \|x_i - xj\|$
- It associates a weight $w$ for this sample which depends on the estimated local error of $i$, $e_i$, and that of its neighbor $j$, $e_j$: $w = e_i/(e_i + e_j)$.
- It computes the relative error of this sample, $e_s$:

$$e_s = \frac{\|\|x_i - xj\| - RTT_{m,i,j}\|}{RTT_{m,i,j}}.$$

- It updates its local error as follows: $e_i = e_j * w + e_i * (1 - w)$.
- It calculates the adaptive step $\delta$ which defines the amount by which the node is allowed to move toward its position indicated by the measurement: $\delta = C_c * w$, where $(0 < C_c < 1)$.
- Finally, the node updates its coordinates in the following way:

$$x_i = x_i + \delta * (RTT_{m,i,j} - RTT_{e,i,j}) * u(x_i - x_j),$$

  where $u(x_i, x_j)$ is a unit vector indicating the direction in which the node $i$ must move. This is typically the straight line in the Euclidean space connecting it to its neighbor.

## 6.3 Experiment methodology and motivation

### 6.3.1 Experiment description

We use the PlanetLab platform [47] for our experiments. Planetlab contains more than 1000 nodes spread across the world in approximately 489 sites. We use Pyxida [51] an implementation of Vivaldi for our experiments. We successfully managed to get meaningful coordinates from 145 PlanetLab nodes. The other nodes are simply eliminated from the experiments. Our experiments were carried out between February and September 2009 with the following parameters for the Vivaldi algorithm. The number of neighbors per node is set to 32 and every node chooses its neighbors arbitrarily. We use an Euclidean space of four dimensions plus the height to calculate the coordinates of each node. Every 10 seconds, each node pings (using UDP) one of its 32 neighbors and updates its coordinates. This leads to approximately one tour over all the neighbors every 320 seconds. For coordinate measurements and collection, every node sends its coordinates to a collection server every 10 seconds where they are stored and used afterwards for our analysis.

**Table** 6.1: Network changes scenarios.

| Scenario name | Number of nodes affected |
|---|---|
| Scenario I | 50 |
| Scenario II | 20 |
| Scenario III | 10 |
| Scenario IV | 5 |

### 6.3.2 Simulations description

One problem with these experimentations however is the impossibility to provoke network delay changes (anomalies) inside the PlanetLab network, because we do not control the intermediate routers and links. This clearly limits the evaluation of the capacity of our algorithm to detect network changes.

To counter the above limitation of PlanetLab experimentations, we run extensive simulations. To be close to reality, we simulate a real topology of PlanetLab provided by the iPlane [18] archive. IPlane gives us all the data about the routes between sources and destinations on PlanetLab, the list of routers on the path, the delays between all routers, and all routers' interfaces. From this data we extract the PlanetLab topology with all detailed paths and routers between 200 PlanetLab nodes. We thus generate the various types of network delay changes that we want to study. In this chapter we show the results for some scenarios changing the network delay. We leave the complete study of network delay anomalies to a future work. As we experience a convergence time of coordinates around 2000 seconds (less than one hour similar results were found in [30]), we fix the duration of all simulations to 30000 seconds.

### 6.3.3 Network changes scenarios

In this section we describe the four different scenarios of network delay changes that we carried out for the validation of this work. Among the 200 nodes that we have, we create network delay changes influencing $n$ nodes (a different number of nodes for each scenario). For that, we increase the delay of the access links connecting these $n$ nodes from their original values to a value equal to 30 times the original values (this can be seen as a link outage or a severe congestion on those links). From time 0 s to time 5000 s we run the simulation with the normal IPlane delay settings, then we provoke the delay anomalies from time 5000 s to time 15000 s. From 15000 s till 30000 s (the end of the simulation) we return to the normal settings. We are interested in studying the impact of these delay anomalies on Vivaldi coordinates and whether they can be detected and impacted nodes identified. More details about the different scenarios are available in Table 6.1.

## 6.4 Stability of Vivaldi coordinates

As Vivaldi coordinates form an effective way for estimating network delays, one can assume that a link exists between network delay conditions and network coordinates. To confirm this assumption and to evaluate the existence of such a link, we start by observing the stability of coordinates in their absolute values over a network that does not present important shifts in its delays. Unfortunately, such coordinates oscillate so much that it makes it impossible to link them to any change in network conditions. This is the main motivation for the clustering algorithm that will come next.

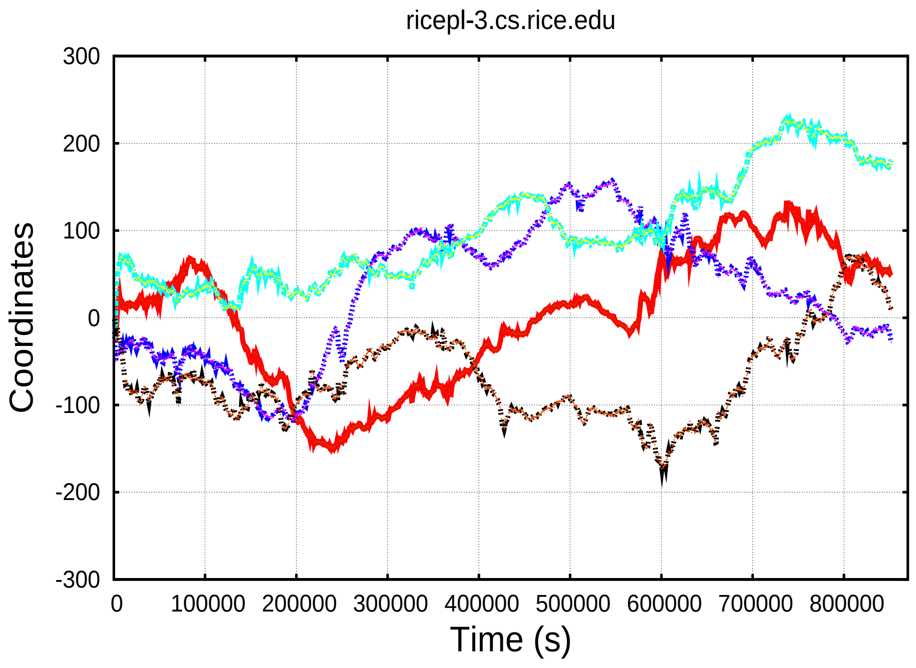

**Figure** 6.3: Coordinate variations (Rice university).

In Figures 6.3, 6.4, 6.5 and 6.6 we present the variations of machine coordinates as a function of time for two PlanetLab nodes which have a stable RTT between them and with most of the other nodes (Figures 6.3 and 6.4) and for two other PlanetLab nodes (Figures 6.5 and 6.6) which experience an important variability in the RTT between them and all the other nodes. We can clearly observe the large variations of the four dimensions coordinates, for both the normal and the abnormal nodes. This is mainly caused by the Vivaldi adaptive algorithm and the absence of any fixed reference point in the Euclidean space. We can conclude that by tracking simply the coordinates in their absolute form, we cannot differentiate between a shift due to a normal behavior of the system or a shift caused by a change in network delay conditions (anomaly).

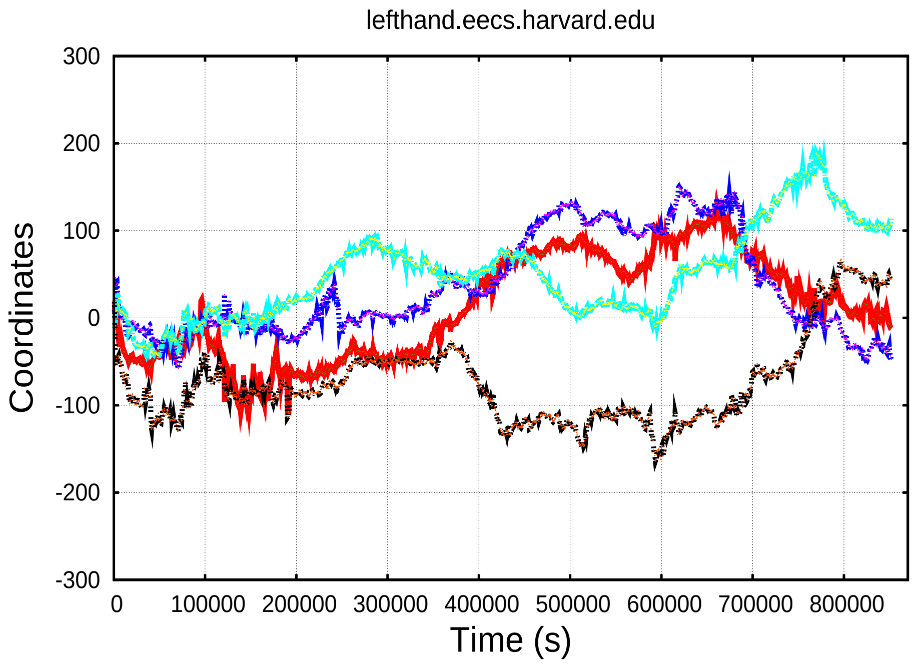

**Figure** 6.4: Coordinate variations (Harvard university).

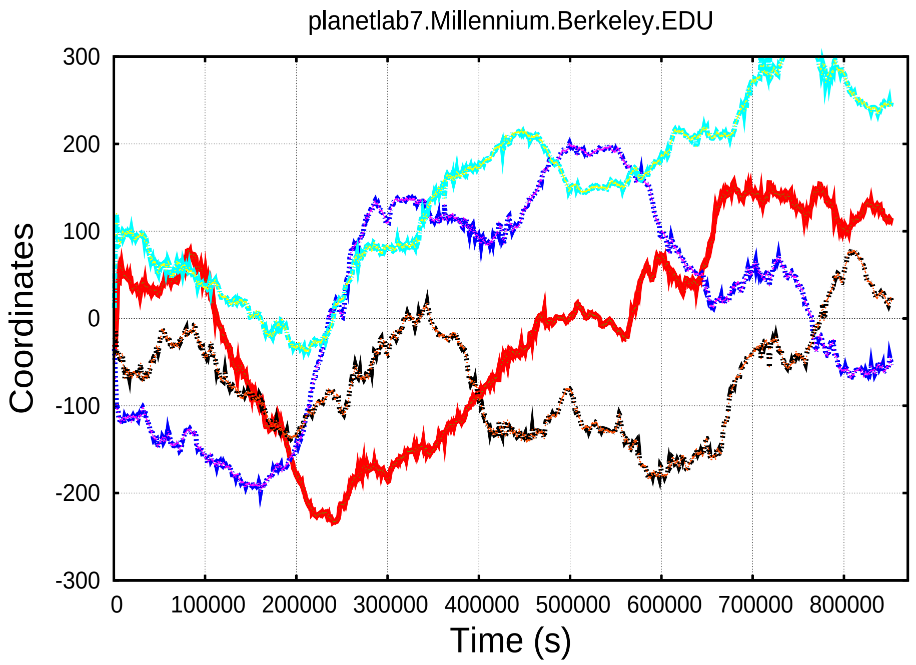

**Figure** 6.5: Coordinate variations (Berkeley university).

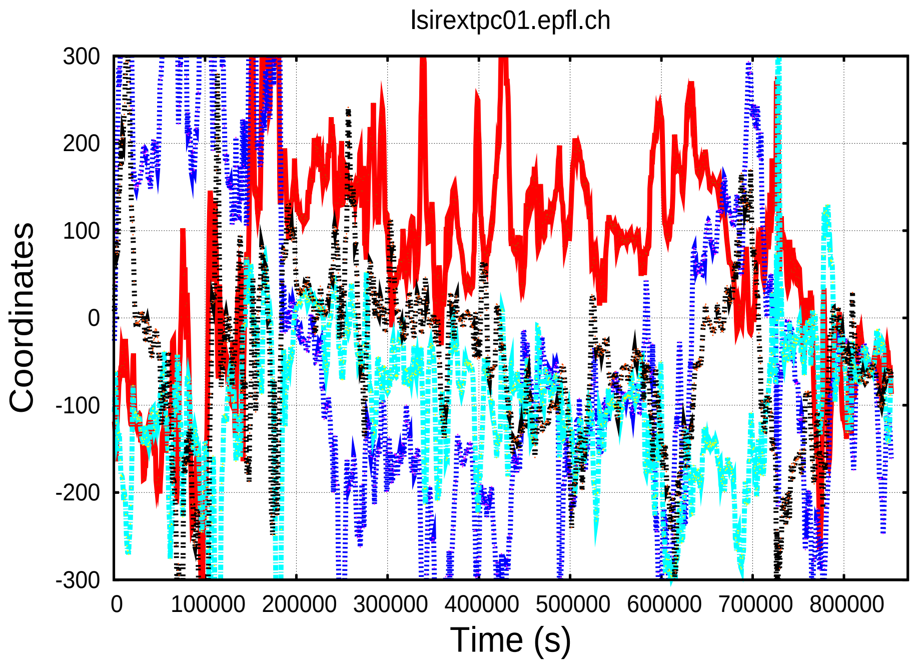

**Figure** 6.6: Coordinate variations (EPFL university).

Here we ask several questions about coordinates.

1. How do coordinates change?
2. Can we find a correlation between the coordinate variations of a set of nodes?
3. What is the metric to apply to coordinates to be able to detect whether there is a network problem or not?
4. Can we detect a change in network conditions by only tracking coordinates?

While observing coordinates, we discover a strong relation between the shifts of all nodes. In particular, we notice that in general nodes move together, which means that all the system shifts in a synchronized way. We also notice that some particular nodes shift more than others because of the instability of their paths. This pushes us to introduce a clustering technique with the main goal to differentiate between nodes according to the value of their shifts with respect to each other.

## 6.5 Clustering algorithm

We present a new clustering algorithm based on the Hierarchical Clustering Method [62]. The main idea of our clustering algorithm is to start by putting all participating nodes in the same cluster. Then, in the second step, all nodes that are far (variation larger than certain threshold) from at least one other node in the cluster are excluded from the cluster. These excluded nodes will pass another round to form their own cluster. Finally, each cluster contains nodes which are all mutually close to each other. The metric that we use to cluster nodes in the system is the amount of variations of their Euclidean distances with respect to each other. The larger the variations of the Euclidean distance of two nodes, the farther they are from each other.

### 6.5.1 Definitions

- $K$: Number of participating nodes.
- $N$: Number of dimensions of the Vivaldi Euclidean space.
- $\vec{x}_i(t)$: The coordinates of node $i$ at time interval $t$.
- $E_{ij}(t)$: The Euclidean distance of two nodes $i$ and $j$ at time interval $t$:

$$E_{ij}(t) = \sqrt{\sum_{d=1}^{N} (x_i(t)[d] - x_j(t)[d])^2}.$$

- $\Delta E_{ij}(t) = |E_{ij}(t) - E_{ij}(t-1)|$: The shift in the Euclidean distance between two nodes $i$ and $j$ at time interval $t$ compared to the previous interval.
- In Vivaldi, each node has 32 neighbors. Every 10s it sends a ping to one of its neighbors and updates its coordinates. Thus, we use $t = 10s$ as the frequency with which our clustering algorithm is executed.
- $\epsilon$: This is the threshold of allowed variations of Euclidean distances (expressed in ms). Nodes with variations in their Euclidean distances to each other below $\epsilon$ (with some confidence level) will be in the same cluster. We have tested our algorithm with different values of $\epsilon$ over the different experiments in our work.
- Violation indicator $s_{ij}(t)$: The membership of two nodes $i$ and $j$ to the same cluster at time interval $t$ is violated when the shift in the Euclidean distance between them is larger than the threshold $\epsilon$: $s_{ij}(t) = 1$ if $\Delta E_{ij}(t) > \epsilon$ and $s_{ij}(t) = 0$ otherwise.
- $C$, or the confidence level, denotes the ratio of membership tests that the nodes should pass for them to be declared in the same cluster. Differently speaking, two nodes do not belong to the same cluster, i.e., $v_{ij} = 1$, if the number of violations of their Euclidean

distances satisfies:

$$\sum_{t_i=t-I+1}^{t} s_{ij}(t_i)/I > 1 - C.$$

Otherwise, we set the variable $\nu_{ij}$ summarizing the result of the test to 0.

- I: Number of past time intervals of 10s to be considered for the clustering. After trying several values for I, we specify it to 100 in this work. This means that at any time t, the clustering is performed over the 100 past intervals of 10 s each (i.e. past 1000 s).

### 6.5.2 Clustering algorithm description

The main goal of our algorithm is to find the maximum number of nodes in each cluster that satisfy the threshold $\epsilon$ for their Euclidean distance shifts. The algorithm starts from a sorted list of nodes according to their number of violations with respect to all other nodes, and it ends up with a set of clusters that contain nodes without any violation with each other (with confidence level C). The details of our algorithm are as follows:

- We calculate the number of violations from each node to all other nodes, i.e., $V_i = \sum_{j=1}^{K} \nu_{ij}$.
- We sort nodes by their decreasing number of violations, $V_i$. The algorithm here starts with all nodes in the same cluster.
- To find the first cluster, we exclude the node with the largest number of violations, then we recalculate the number of violations for the remaining nodes. We continue excluding nodes with large number of violations until there is no violation between the nodes in the cluster. We are sure that the distance between any two nodes in this cluster shifts by less than the threshold $\epsilon$ within the confidence level C.
- We repeat the above steps to form a second cluster with the excluded nodes.
- We stop the algorithm when we are left without any excluded node.

## 6.6 Clustering results

We run our clustering algorithm every 10s over both PlanetLab and the iPlane traces. Next we study the distribution of the number of clusters and their sizes. We recall that a cluster is composed of nodes that are relatively stable (between them) according to the $\epsilon$ used and the confidence level C. Roughly speaking, this means that the shift of the Euclidean distance between any node i in a cluster and any other node j of the same cluster, $\Delta E_{ij}(t) = |E_{ij}(t) -$

$E_{ij}(t-1)|$, does not exceed $\epsilon$ in more than $((1-C)*100)\%$ of the 100 past measurement intervals.

### 6.6.1 Cluster size distribution with different $\epsilon$ values

In this subsection we study the impact of $\epsilon$ on the output of our clustering algorithm. To this end, we fix the confidence level C to 90% (10 violations are allowed among 100 intervals), and we perform the clustering while changing the value of $\epsilon$. We present in Figures 6.7, 6.8, 6.9, 6.10, 6.11 and 6.12 the distribution of the sizes of the first four clusters that we found for different values of $\epsilon$ and as a function of time.

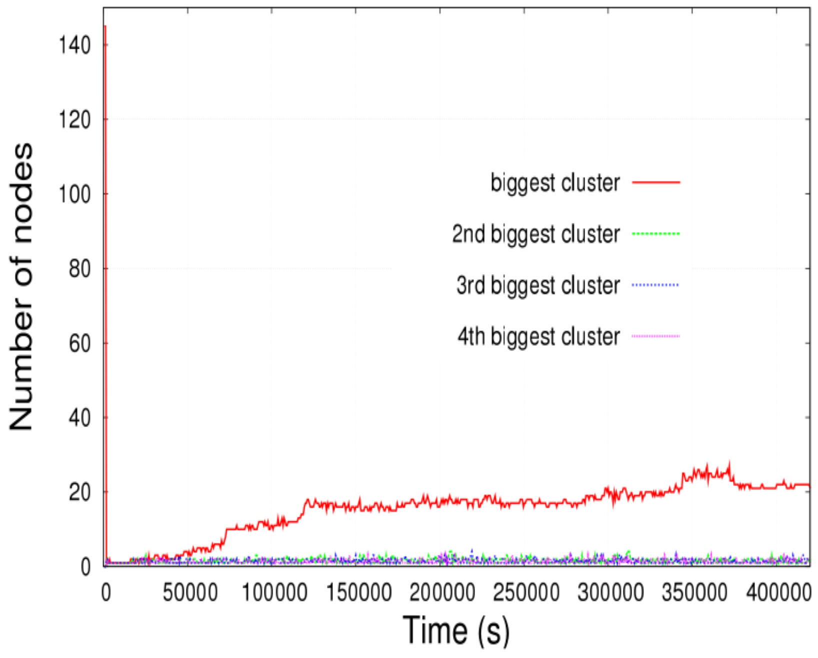

**Figure** 6.7: Cluster size for $\epsilon = 2$ (145 PlanetLab nodes).

In Figures 6.7, 6.8 and 6.9, we observe that the size of the biggest cluster is around 20, 37 and 85 nodes respectively among the 145 PlanetLab nodes. This relatively small size of the biggest cluster is normal because we set the value of $\epsilon$ to 2ms (in Fig. 6.7), 4ms (in Fig. 6.8), 8ms (in Fig. 6.9) respectively, which means that we only allow a small shift in Euclidean distances compared to the shift inherent to Vivaldi dynamics. For the other clusters, we always have a small number of nodes, smaller than 3, 7 and 10 nodes respectively. We can say that nodes belonging to the biggest clusters for these values of $\epsilon$ are really stable and can be used to detect small shifts in the network delays between them. We can also observe that the size of the biggest cluster itself changes over time and that we have some nodes that leave and enter this

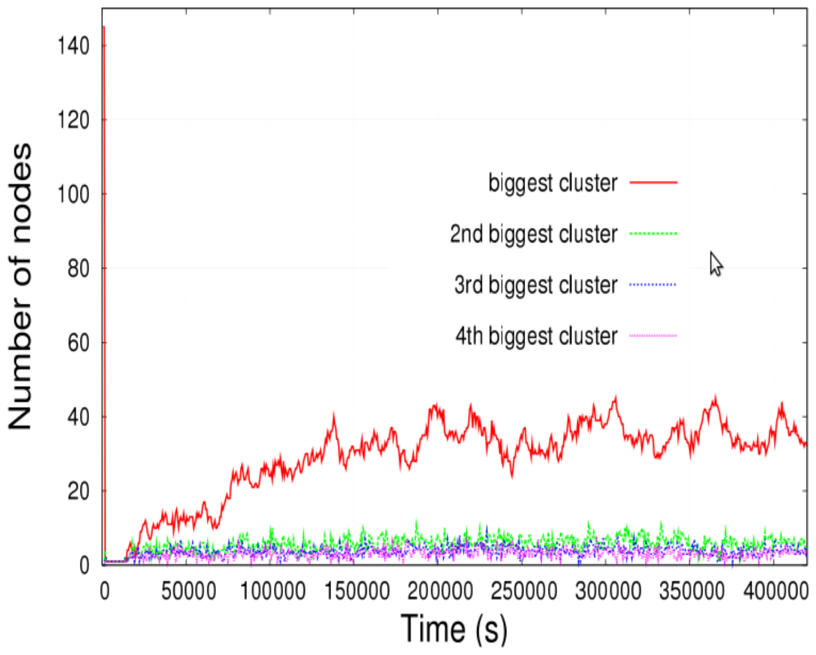

**Figure** 6.8: Cluster size for $\epsilon = 4$ (145 PlanetLab nodes).

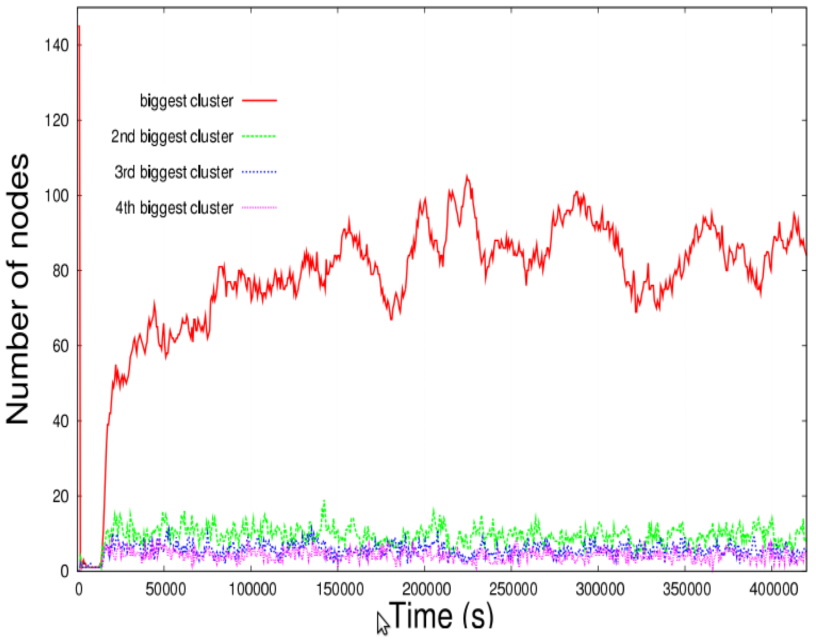

**Figure** 6.9: Cluster size for $\epsilon = 8$ (145 PlanetLab nodes).

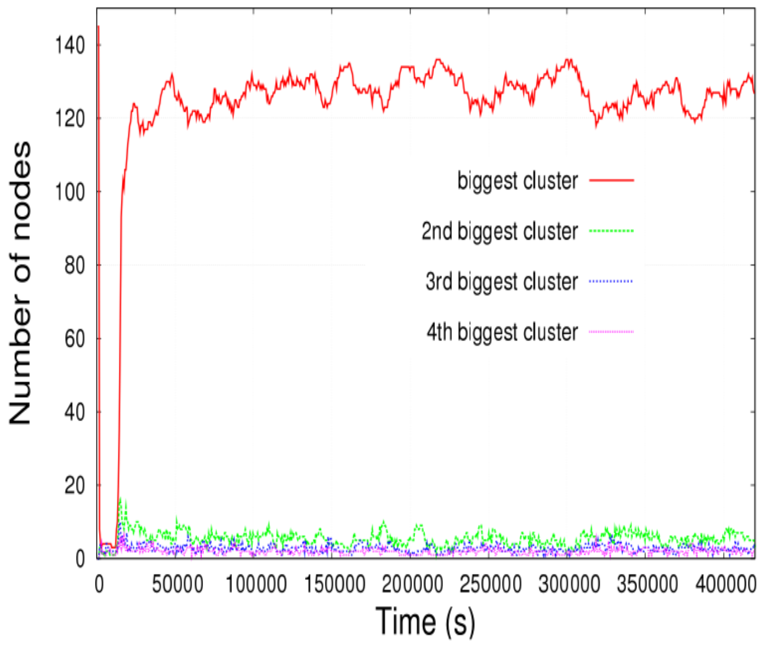

**Figure** 6.10: Cluster size for $\epsilon = 16$ (145 PlanetLab nodes).

cluster for these small values of $\epsilon$. This is the result of variations in Euclidean distances that can be caused either by Vivaldi dynamics or by small shifts in network delays.

In Figure 6.10 and for $\epsilon$ equal to 16ms, we notice that the size of the biggest cluster is around 125 nodes among 145 PlanetLab nodes in total. We also notice that the biggest cluster is more stable (compared to Figures 6.7, 6.8 and 6.9) and that the sizes of the other clusters are always smaller than 5.

In Figures 6.11 and 6.12 we observe that the size of the biggest cluster is around 140 (Fig. 6.11) and 142 (Fig. 6.12) nodes among 145 nodes in total. Here, we use an $\epsilon$ equal to 32ms and 64ms respectively, which means that we allow a shift in Euclidean distances of 32ms (Fig. 6.11) and 64ms (Fig. 6.12). The sizes of the other clusters than the biggest one is always smaller than 2 and 1 respectively. Here we can say that we have all the nodes approximately in the same cluster, and that this biggest cluster is relatively stable, with only few nodes that change their cluster.

From the above figures we can confirm the presence of a big and stable cluster. When we use a small $\epsilon$ value, we obtain a small cluster which contains very stable nodes that are almost of constant delay with respect to each other. However, when we use a large $\epsilon$, we obtain one main (biggest) cluster that contains approximately all the nodes. The distance shifts between these nodes is now larger, which limits the capacity of the algorithm to track small changes in network delays.

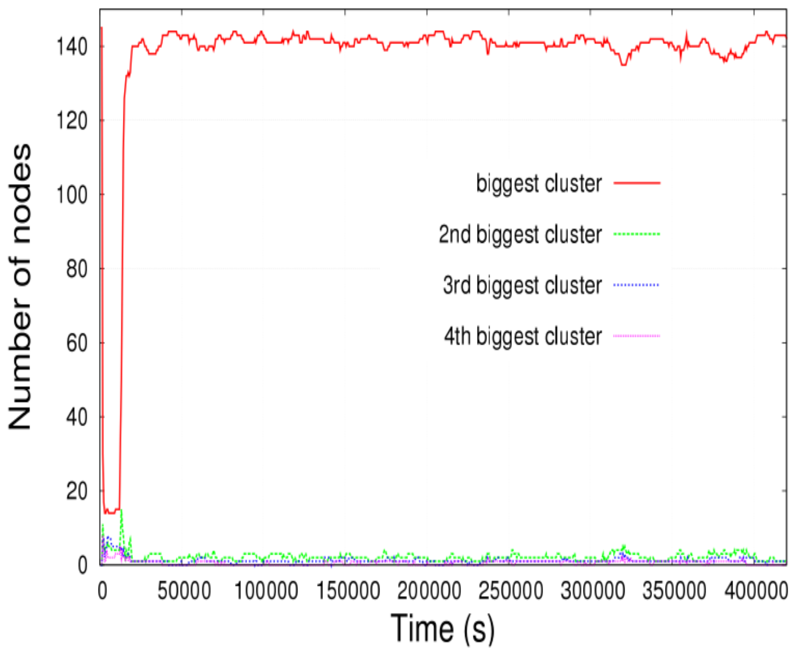

**Figure** 6.11: Cluster size distribution for $\epsilon = 32$ (145 PlanetLab nodes).

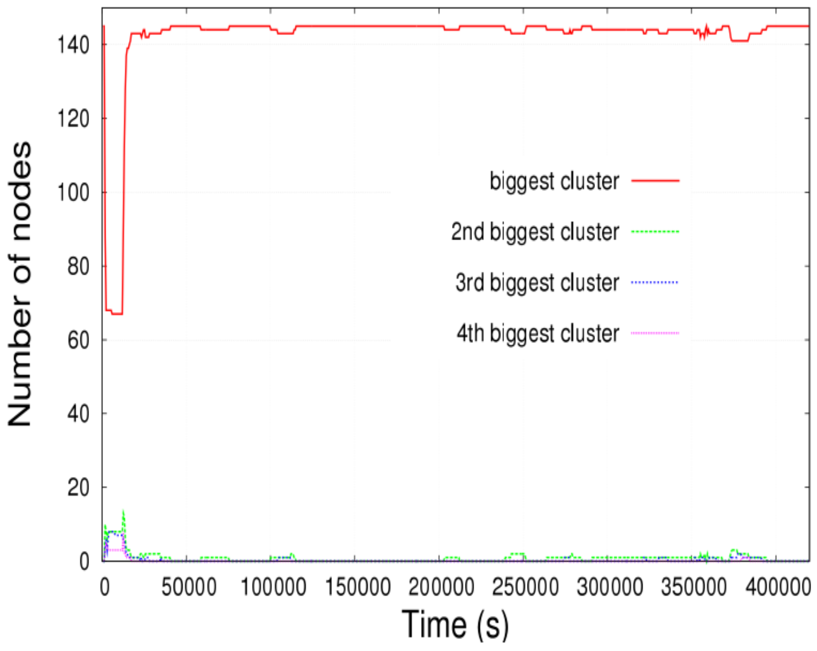

**Figure** 6.12: Cluster size distribution for $\epsilon = 64$ (145 PlanetLab nodes).

### 6.6.2 Biggest cluster stability for different C values

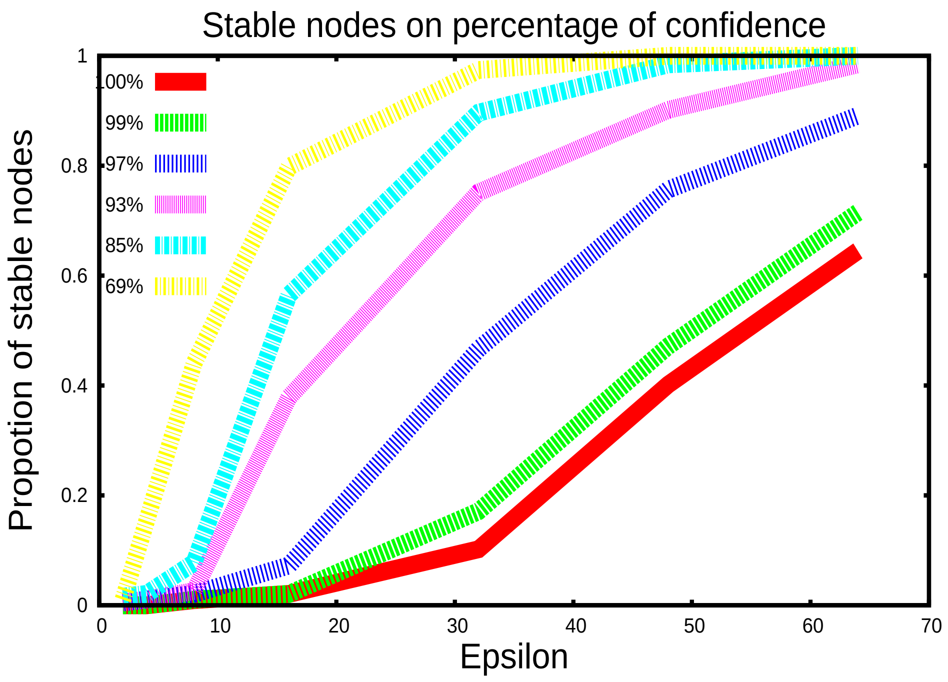

**Figure** 6.13: Proportion of stable nodes for different C values as a function of $\epsilon$.

In Figures 6.13 we study the stability of nodes in the biggest cluster as a function of the confidence level C and the value of $\epsilon$ used. We present the proportion of stable nodes as a function of $\epsilon$ and for different confidence levels. We can clearly observe that when we use a low confidence level and for all $\epsilon$ values, the biggest cluster becomes more stable, especially when we use large $\epsilon$ values. For example, if we use a confidence level of 85% and an $\epsilon$ of 64ms, we obtain a very stable biggest cluster where 99.5% of nodes are inside. We can conclude from this figure that in the normal case and for a large $\epsilon$ value and a low confidence level, we have a stable biggest cluster that reflects well the state of the participating nodes. Unfortunately, the biggest cluster in this latter case has limited capacity to detect shifts in network delays since most of the shifts below $\epsilon$ will pass undetected. The capacity to detect larger shifts than $\epsilon$ is however still effective.

### 6.6.3 Impact of network delay changes on the size distribution of clusters

In this subsection, we study the impact of a change in network delays on the distribution of the biggest clusters' sizes. To achieve this goal, we use the scenario I for network delay change described in Section 6.3.3 (50 nodes affected by a network change) and we study the impact of this scenario on the coordinate system by making a comparison with the normal case. We fix the confidence level C to 90% for all the results in Figures 6.14, 6.15, 6.16, 6.17, 6.18 and 6.19.

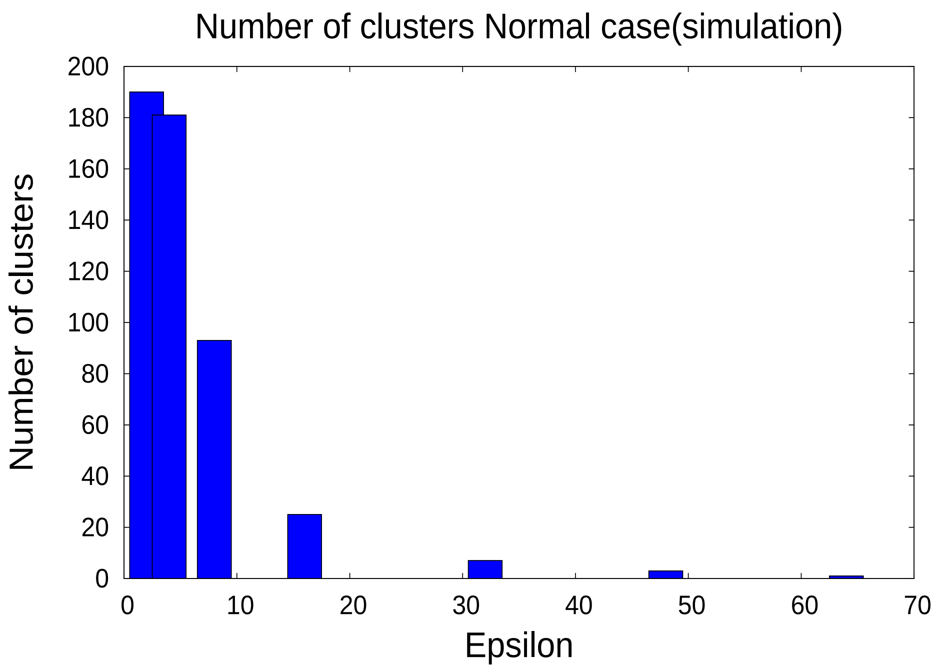

**Figure** 6.14: Number of clusters in the normal case as a function of $\epsilon$ (simulation).

In Figures 6.14 and 6.15 we compare the number of clusters as a function of $\epsilon$ between the normal case and the abnormal case. In the abnormal case (Fig. 6.15), the number of clusters clearly increases for all the values of $\epsilon$. For example for $\epsilon$ equal to 32 ms, we have 8 clusters in the normal case and 21 clusters in the abnormal case. Indeed, the shift in the delay makes some nodes far from each other causing shifts in their Euclidean distances, and hence the appearance of more clusters. Once the delays of the network stabilize, the clustering should be back to its normal situation.

In Figures 6.16 and 6.17 we present the average proportion of nodes in the biggest cluster as a function of $\epsilon$ for the normal case (Fig. 6.16) and the abnormal case .(Fig 6.17). The proportion of nodes in the biggest cluster clearly decreases during the abnormal case for all the values of $\epsilon$, and this decrease becomes more and more clear when we use a larger $\epsilon$. For example, for $\epsilon$ equal to 64ms, the proportion of nodes in the biggest cluster decreases from 99% in the normal case to 78% in the abnormal case.

In Figures 6.18 and 6.19, we present the average and the standard deviation of the proportion of nodes in the biggest cluster. The difference between the normal and the abnormal case is clear; during the abnormal case we have a large standard deviation caused by the change in the composition of the biggest cluster due to the anomaly. We can conclude from the above figures that the number of nodes in the biggest cluster is a good parameter to differentiate between a

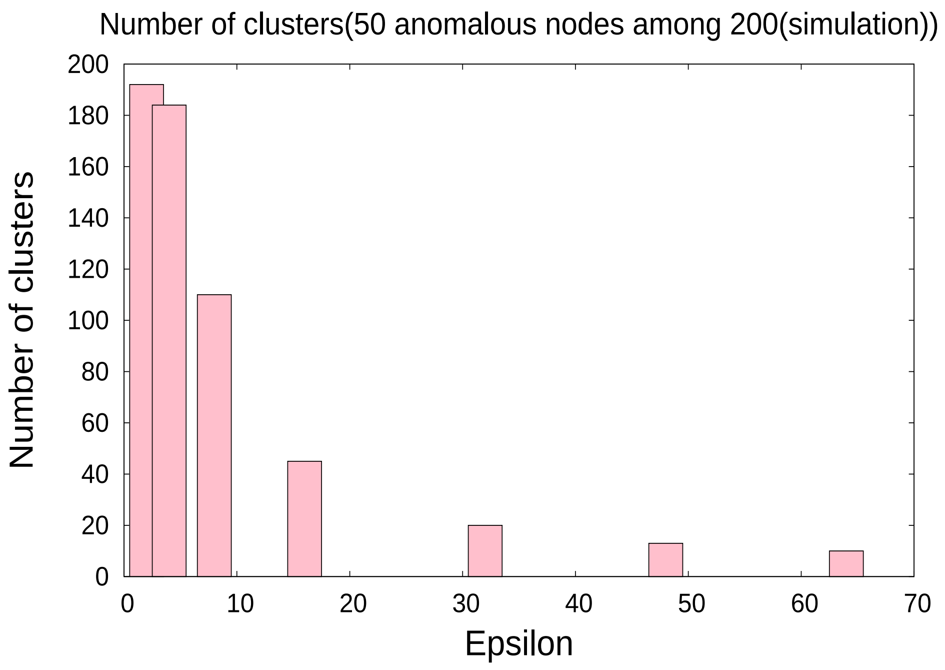

**Figure** 6.15: Number of clusters in one abnormal case as a function of $\epsilon$ (simulation, 50 abnormal nodes among 200).

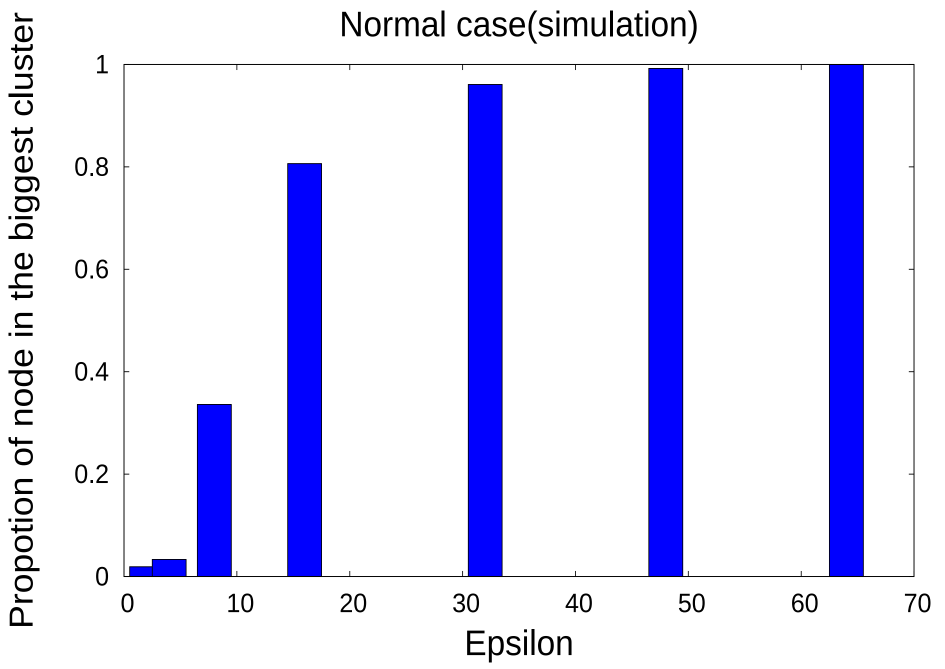

**Figure** 6.16: Number of nodes in the biggest clusters in the normal case as a function of $\epsilon$.

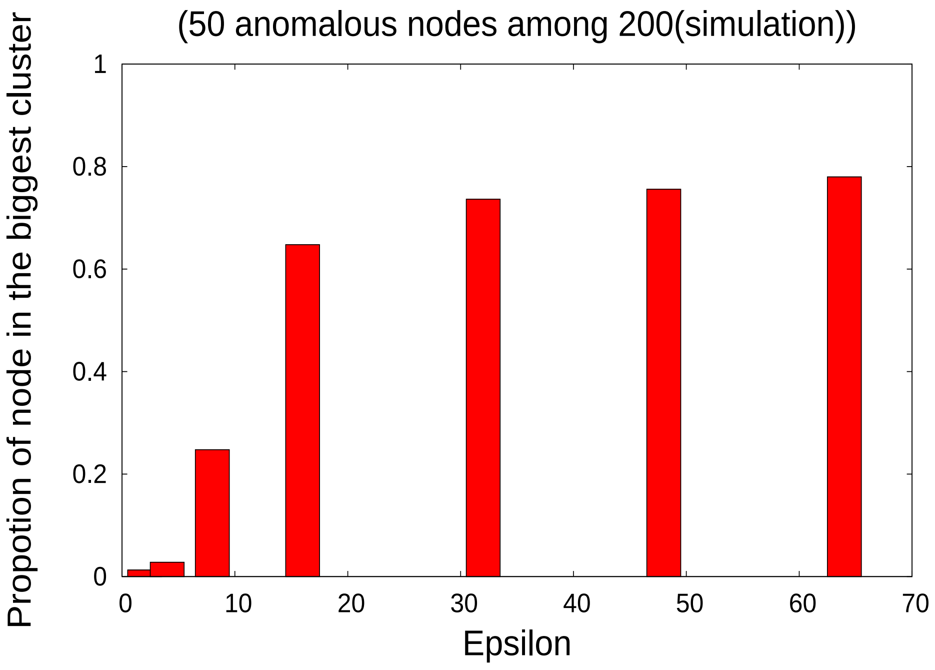

**Figure** 6.17: Number of nodes in the biggest clusters in the normal case as a function of $\epsilon$(simulation, 50 abnormal nodes among 200).

normal and an abnormal network scenario. This can be done by monitoring the movement of stable nodes that are all the time in the biggest cluster when everything is normal.

## 6.7 Anomaly detection protocol

To detect anomalies we decide to track the number of nodes staying in the biggest cluster. As we have seen in the previous sections, during a normal case we have always a stable big cluster with some small variations in its composition, so we decided to calculate the mean number of nodes in the biggest cluster $\mu_{nb}$ together with the standard deviation $\sigma_{nb}$ during the normal case. The signal that we will track all the time is the number of nodes in the biggest cluster that we will compare to the interval $[\mu_{nb} - \sigma_{nb}, \mu_{nb} + \sigma_{nb}]$. A signal outside this interval means that we have some change in the network according to the $\epsilon$ and the confidence level C used. In order to do that, we perform a kind of training phase where we calculate, for every value of $\epsilon$ and C the interval $[\mu_{nb} - \sigma_{nb}, \mu_{nb} + \sigma_{nb}]$ during the normal case.

The network administrator should set the different parameters of our clustering algorithm based on which changes he expects to detect:

- the amount of changes to detect (set $\epsilon$),

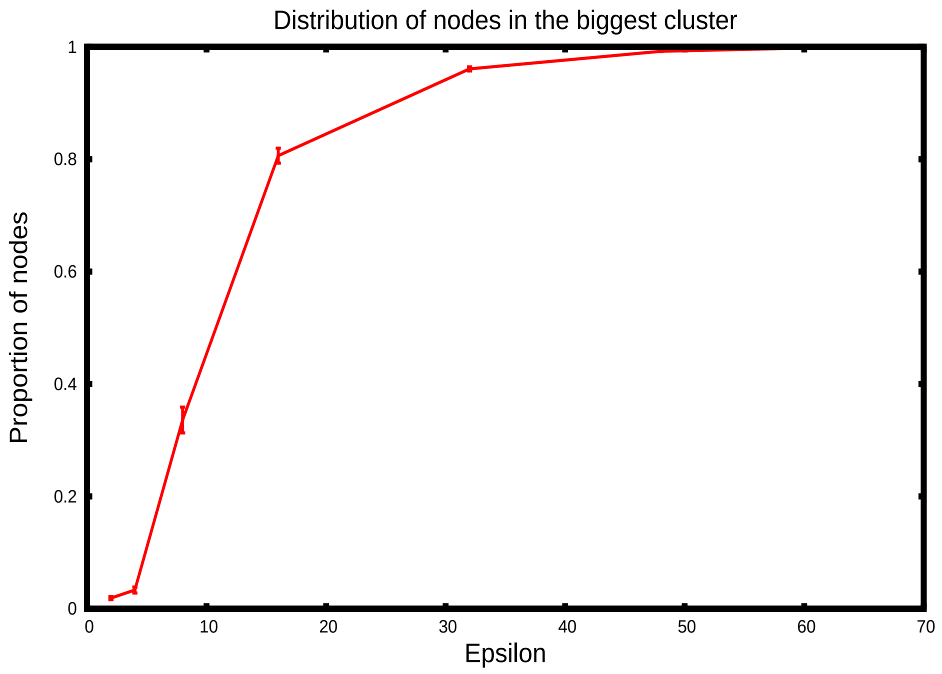

**Figure** 6.18: The average number of nodes (with the standard deviation) in the biggest cluster(simulation, 200 normal nodes).

- the number of nodes to track (set $\epsilon$),
- the sensitivity threshold (set C),
- the time history to consider (set I).

In the training phase we determine the stable nodes to track (nodes always in the biggest cluster), and we calculate the mean number and the standard deviation of nodes staying in the biggest cluster (a separate training phase for every value of $\epsilon$ and C). The main idea of the protocol is to track stable nodes (nodes staying in the big cluster during a normal situation). If some nodes exit the big cluster at any moment making its size outside the tolerated interval, we can conclude that we have a network change that affects these nodes.

To differentiate between a congestion and a link down we should take into consideration the duration of the changes. For example if the changes persist, we can conclude that we have a link outage. If the duration of the changes is small (less than certain threshold), we can have a small congestion.

Two trade-off are to be taken into consideration:

- If a small $\epsilon$ is used, a small change in the network can be detected, but the number of stable nodes (nodes that we can track) will be also small. So we should choose a good

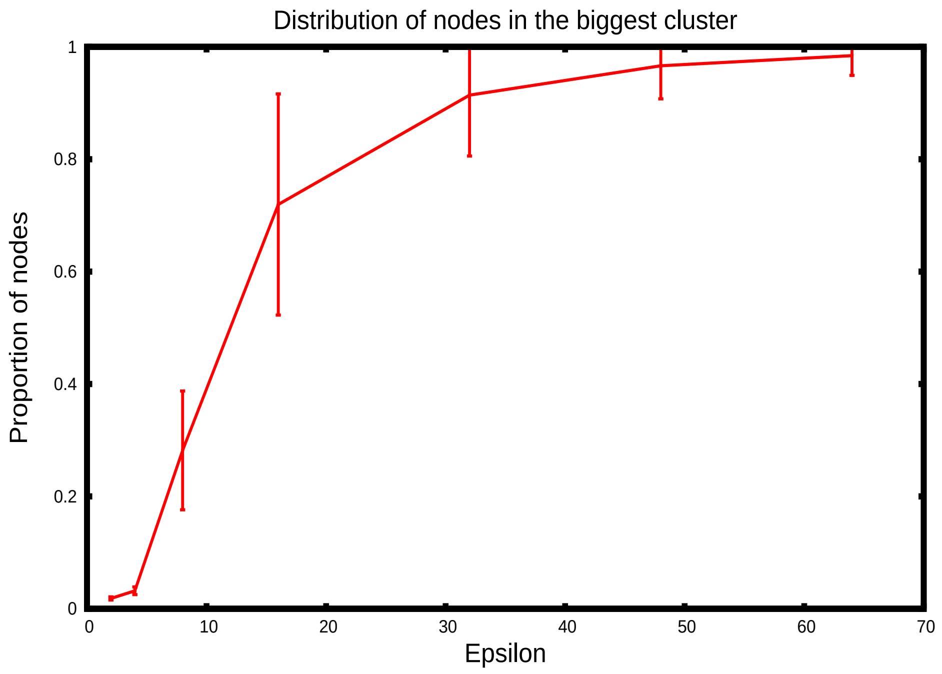

**Figure** 6.19: The average number of nodes (with the standard deviation) in the biggest cluster (simulation, 50 abnormal nodes among 200).

value of $\epsilon$ in order to have the capability to track the larger possible set of nodes and to detect the threshold of network changes that we consider as important for our work. For example, if we take an $\epsilon$ of 4 $ms$, we will be able to detect any change in the network bigger than 4 $ms$ but we will be able to track only some nodes among 200 nodes forming our simulations scenario.

- If we use a high confidence level C, we can detect short-term changes (changes that persist for small duration), but we can not track all the nodes that we have. So again we should choose a good value of C in order to track the larger possible set of nodes and to detect changes that persist to a time that we consider important for our network.

The details of our protocol are as follows:

- For every $\epsilon$ and for every C, we do the training phase in a normal case and we calculate the interval $[\mu_{nb} - \sigma_{nb}, \mu_{nb} + \sigma_{nb}]$.
- The number of nodes in the biggest cluster for every value of C for a given $\epsilon$ will be considered as a signal.
- The set of signals for every $\epsilon$ will define the input for a network change equivalent to the value of $\epsilon$ in ms.

- Choose $\epsilon$ and start by observing the signal for different values of C.
- A signal will be considered as abnormal if its value is outside the interval $[\mu_{nb} - \sigma_{nb}, \mu_{nb} + \sigma_{nb}]$.
- For the signal with larger C, changes with small duration can be detected (for a small set of nodes), while with signals with smaller C changes that persist for longer duration can be detected (approximately for all the nodes).

## 6.8 Anomaly detection protocol results

In this section we present the results of our anomaly detection protocol for the five network changes scenarios described in Section 6.3.3. An exhaustive study for all possible scenarios will be done in future work. For the results that we present in this section we take two values of C, 70% and 85% and two values of $\epsilon$, 16 and 32 ms. We define:

- NA: the number of nodes affected by the network changes.
- NT: the number of nodes that can be tracked.
- NAT: the number of nodes affected that can be tracked.
- ND: the number of affected nodes detected.
- NF: the number of affected nodes that we fail to detect.

In Table 6.2 we present the results of our anomaly detection protocol for the four scenarios while using two values of $\epsilon$ and two values of C. We can clearly observe that our anomaly detection protocol is too efficient when we can track the affected nodes. On the other hand the main problem that we have is that we are not able to track all the nodes for small values of $\epsilon$ and C. There is clearly a trade-off between the number of nodes that can be tracked and the amount of changes that can be detected ($\epsilon$) together with the duration of the variations detected (C).

## 6.9 Conclusions and future work

In this chapter, we studied the Vivaldi coordinate system from inside and we showed that despite the instability of these coordinates, we can still use them to track network changes. We started by developing a new clustering algorithm that permitted us to group the stable nodes together following the amount of variations of their coordinates. The clustering pointed out to the presence of a stable biggest cluster which contains most of the nodes. We showed that the

**Table** 6.2: Anomaly detection results.

| Scenario | C | $\epsilon$ | NA | NT | NAT | ND | NF |
|---|---|---|---|---|---|---|---|
| Scenario I | 70% | 16 | 50 | 170 | 45 | 45 | 5 |
| Scenario I | 70% | 32 | 50 | 198 | 50 | 50 | 0 |
| Scenario I | 95% | 16 | 50 | 160 | 43 | 43 | 7 |
| Scenario I | 95% | 32 | 50 | 195 | 49 | 49 | 1 |
| Scenario II | 70% | 16 | 20 | 170 | 18 | 18 | 2 |
| Scenario II | 70% | 32 | 20 | 198 | 20 | 20 | 0 |
| Scenario II | 95% | 16 | 20 | 160 | 17 | 17 | 3 |
| Scenario II | 95% | 32 | 20 | 195 | 19 | 19 | 1 |
| Scenario III | 70% | 16 | 10 | 170 | 9 | 9 | 1 |
| Scenario III | 70% | 32 | 10 | 198 | 10 | 10 | 0 |
| Scenario III | 95% | 16 | 10 | 160 | 8 | 8 | 2 |
| Scenario III | 95% | 32 | 10 | 195 | 9 | 9 | 1 |
| Scenario IV | 70% | 16 | 5 | 170 | 5 | 5 | 0 |
| Scenario IV | 70% | 32 | 5 | 198 | 5 | 5 | 0 |
| Scenario IV | 95% | 16 | 5 | 160 | 5 | 5 | 0 |
| Scenario IV | 95% | 32 | 5 | 195 | 5 | 5 | 0 |

size of this biggest cluster reflects well the state of the network, and so a node that belongs to this biggest cluster is a stable and normal node. By tracking the number of nodes in the biggest cluster, one can detect main changes in network delays. The amount of changes to detect and the confidence level of tracking can be set by the network administrator. With our algorithm, coordinates can then become an efficient way for a light-weight network diagnosis that does not require the deployment of a complex monitoring infrastructure. In our future work, we will push the study further by designing a distributed protocol for machine clustering and for the detection and localization of network delay anomalies, by the help of Vivaldi coordinates. We are also planning to conduct an exhaustive experimental study to validate the performance of such protocol by considering a large set of scenarios for changing network delays.

# 7

# CONCLUSIONS

## 7.1 Summary of the thesis

The identification of Internet traffic applications is very important for ISPs and network administrators to protect their resources from unwanted traffic and prioritize some major applications. Statistical methods are preferred to port-based ones and deep packet inspection since they don't rely on the port number and they also work for encrypted traffic. These methods combine the statistical analysis of the application packet flow parameters, such as packet size and inter-packet time, with machine learning techniques. Other approaches rely on the way the hosts communicate and their traffic patterns to identify applications. The majority of these statistical methods identify traffic off-line and classify every flow independently. Another important challenge for network administrators is to detect and diagnose key network changes as a long-term congestion, a rerouting, a link failure or any other event causing a shift in network delays. In the literature there is a huge amount of anomaly detection methods but most of them require exhaustive measurements to function properly. Reducing the load of network-wide monitoring is always a vital need for network administrators.

In this thesis we developed three new statistical methods in order to identify Internet traffic on the fly. Together with, we developed a new light-weight method for Internet anomaly detection which use the Vivaldi coordinate system. Section 7.2 presents the main contributions of this thesis and we present future research perspectives in Section 7.3.

## 7.2 Thesis contributions

Three new methods were developed for traffic identification follows an iterative approach and relies on the learning and the analysis of the size and the direction of the first packets of flows. By considering packets separately from each other and with the help of a new probability function that combines the observations made on the different packets from a flow, we are able to obtain a high classification accuracy that kept improving by adding more packets from each flow, and that was able to reach high levels of precision around 98% after 10 packets to be used for the classification. The main advantage of this method is that it permits to start the identification process directly after the arrival of the first packet in the flow. With every new packet the decision taken is confirmed. Unlike the joint consideration of the packet size, the iterative model of our method permits to use any number of packets while increasing the precision of the classification with every new packet. Note that the complexity of our method is lower than the complexity of the joint model.

As a second contribution in this thesis we carry out a study about the inter-packet time and we analyze how it can be used to identify Internet applications. We model the different components of the inter-packet time and we propose to filter the noise due to the network delay to extract relevant features for the classification. We then use the same iterative method while using the preprocessed inter-packet time as a parameter. We evaluate our solution on real traces and the results show that the inter-packet time is a relevant parameter to identify Internet traffic after some appropriate processing. In particular, this processing is important to highlight the interactive characteristics of each application at the client and server. The results also show that our method reaches a total precision of 99% for the classification of all applications, when we filter the network noise from the inter-packet time (the first 20 inter-packet time are used). The inter-arrival time can then be used to identify Internet traffic. The possibility of using the inter-arrival time as a parameter increases our choice and makes our method more robust.

As a third contribution in this thesis we develop a new method for Internet traffic identification that combines the statistical and host-based approaches. The statistical parameters that we use are the size and direction of the first N packets. The novelty of our approach consists in leveraging the host profile to refine the classification. First we define the profile of the host and how it is updated. Then we show how the profiles of the source and destination hosts are used to assign a prediction probability to the new flow. We evaluate our solution using real traces. The results show a great improvement for the classification of applications when the host profile is used. In particular, the classifier reaches a precision of 98% after using the first 10 packets of a flow for the classification. Using the host information permits to ameliorate the performance of our method especially for the first few packets which makes the identification more effective.

Concerning our contribution in the anomaly detection subject we develop a new light-weight method for Internet anomaly detection while using the Vivaldi coordinates system. Firstly, we study the Vivaldi coordinate system from inside and we show that despite the instability of these coordinates, we can still use them to track network changes. For this research we develop a new clustering algorithm that permits us to group the stable nodes together following the amount of variations of their coordinates. The clustering points out the presence of a stable biggest cluster which contains most of the nodes. We show that the size of this biggest cluster reflects well the state of the network, and so a node that belongs to this biggest cluster is a stable and normal node. By tracking the number of nodes in the biggest cluster, one can detect main changes in network delays. The amount of changes to detect and the confidence level of tracking can be set by the network administrator. With our algorithms, coordinates can then become an efficient way for a light-weight network diagnosis that does not require the deployment of a complex monitoring infrastructure.

## 7.3 Future works

Concerning the future research perspectives for the traffic identification subject we plan to:

- Validate our methods on other applications and traces, especially ADSL traces.
- Develop a network traffic generator which can simulate real Internet traffic in order to help researchers working in this domain to counter the problem of the lack of traces for the validation of their methods.
- Collect a big data set and make it public in order to test and compare the performance of different methods present in the literature. As all methods are tested over different data sets, there is no common support to compare results.

For the anomaly detection subject we will push the study further by designing a distributed protocol for machine clustering and for the detection and localization of network delay anomalies, by the help of Vivaldi coordinates. We are also planning to conduct an exhaustive experimental study to validate the performance of such protocol by considering a large set of scenarios for changing network delays.

# BIBLIOGRAPHY

[1] Akamai. Akamai technologies. `http://www.akamai.com/html/misc/requirements.html`, 2009. 114

[2] Paola Bermolen, Marco Mellia, Michela Meo, Dario Rossi, and Silvio Valenti. Abacus: Accurate behavioral classification of p2p-tv traffic. *Computer Networks*, 55(6):1394–1411, 2011. 4, 25, 28, 157

[3] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *CoNEXT*, Lisboa, Portugal, 2006. 4, 25, 28, 38, 39, 42, 62, 63, 157, 158

[4] BitTorrent. Bittorrent. `http://bitconjurer.org/BitTorrent/protocol.html`, 2009. 114

[5] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing skype traffic: when randomness plays with you. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 37–48, New York, NY, USA, 2007. ACM. 4, 25, 28, 157

[6] Caida. The cooperative addociation for internet data analysis. `http://www.caida.org/home/`, 2010. 6

[7] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. In *ACM-Sigcomm CCR*, volume 37, pages 5 –16, January 2007. 4, 6, 25, 26, 49, 50, 68, 84, 157

[8] Marc Crovella and Balachander krishnamurthy. *Internet Measurement*. John Wiley Sons, Ltd, 2007. 18, 19, 20

[9] S. Antonatos E. Markatos S. Ubik D. Antoniades, M. Polychronakis and A. ÃslebÃ¸. Appmon: An application for accurate per application network traffic characterization. In *IST BroadBand Europe*, December 2006. 4, 23, 24, 156

[10] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, New York, NY, USA, 2004. ACM. 113, 114, 118

[11] Christian Dewes, Arne Wichmann, and Anja Feldmann. An analysis of internet chat systems. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, IMC '03, pages 51–64, New York, NY, USA, 2003. ACM. 4, 23, 156

[12] J. Erman, A. Mahanti, and M. Arlitt. Internet traffic identification using machine. In *49th IEEE GLOBECOM*, San Francisco, USA, 2006. 4, 25, 28, 157

[13] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, MineNet '06, pages 281–286, New York, NY, USA, 2006. ACM. 4, 25, 27, 28, 157, 158

[14] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Acas: automated construction of application signatures. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, MineNet '05, pages 197–202, New York, NY, USA, 2005. ACM. 4, 23, 24, 156

[15] Mark A. Hall Ian H. Witten, Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. 2005. 30, 31, 33

[16] IANA. Internet assigned numbers authority. `http://www.iana.org/assignments/port-numbers`. 4, 21, 155

[17] Trace II. Brescia university. `http://www.ing.unibs.it/ntw/tools/traces/`, 2009. 6, 49, 50, 84

[18] Iplane. Iplane, an information plane for distributed services. `http://iplane.cs.washington.edu/`, 2009. 121

[19] Mohamad Jaber and Chadi Barakat. Enhancing application identification by means of sequential testing. In *IFIP/TC6 Networking Conference*, Aechan, Germany, May 2009. 37

[20] Mohamad Jaber, Roberto Cascella, and Chadi Barakat. Boosting statistical application identification by flow correlation. In *Euro-NF International Workshop on Trafic and Congestion Control for the Future Internet*, Volos Greece, April 2011. 77

[21] Mohamad Jaber, Roberto Cascella, and Chadi Barakat. Can we trust the inter-packet time for traffic classification? In *IEEE International Conference on Communications (ICC)*, Kyoto, Japan, June 2011. 61

[22] Mohamad Jaber, Roberto Cascella, and Chadi Barakat. Using host profiling to refine statistical application identification. In *IEEE INFOCOM Mini-Conference*, Orlando, FL, USA, March 2012. 77

[23] Mohamad Jaber, Cao-Cuong Ngo, and Chadi Barakat. A view from inside a distributed internet coordinate system. In *the Global Internet Symposium at IEEE Infocom*, San Diego, USA, March 2010. 113

[24] Mohamed Ali Kaafar, Laurent Mathy, Chadi Barakat, Kave Salamatian, Thierry Turletti, and Walid Dabbous. Securing internet coordinate embedding systems. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 61–72, New York, NY, USA, 2007. ACM. 115

[25] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. Blinc: Multilevel traffic classification in the dark. In *SIGCOMM*, New York, USA, August 2005. 4, 25, 29, 78, 157

[26] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, pages 121–134, New York, NY, USA, 2004. ACM. 4, 23, 24, 156

[27] Thomas Karagiannis, Pablo Rodriguez, and Konstantina Papagiannaki. Should internet service providers fear peer-assisted content distribution? In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 6–6, Berkeley, CA, USA, 2005. USENIX Association. 114

[28] KaZaA. Kazaa media dekstop. `http://www.kazaa.com/`, 2009. 114

[29] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *ACM CoNEXT*, Madrid, Spain, 2008. 62, 63

[30] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network coordinates in the wild. In *Proceedings of the Fourth USENIX Symposium on Network Systems Design and Implementation (NSDI)*, April 2007. 115, 121

[31] Jonathan Ledlie, Peter Pietzuch, and Margo Seltzer. Stable and accurate network coordinates. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 74, Washington, DC, USA, 2006. IEEE Computer Society. 115

[32] W. Li and A. W. Moore. A machine learning approach for efficient traffic classification. In *Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 310–317, Washington, DC, USA, 2007. IEEE Computer Society. 29

[33] Hyuk Lim, Jennifer C. Hou, and Chong-Ho Choi. Constructing internet coordinate system based on delay measurement. *IEEE/ACM Trans. Netw.*, 13(3):513–525, 2005. 4, 114

[34] A. Rowstron M. Costa, M. Castro and P. Key. Practical internet coordinates for distance estimation. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, Tokyo, March 2004. 118

[35] A. Gefferth S. MolnÃir M. PerÃ©nyi, T. Dinh Dang. Identification and analysis of peer-to-peer traffic. *JOURNAL OF COMMUNICATIONS*, 1, November 2006. 4, 23, 156

[36] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On dominant characteristics of residential broadband internet traffic. In *IMC*, 2009. 65

[37] Mawi. The mawi working group of the wide project. `http://mawi.wide.ad.jp/mawi/`, 2009. 6

[38] Anthony Mcgregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. In *In PAM*, pages 205–214, 2004. 4, 25, 157, 158

[39] P. Mockapetris and K. J. Dunlap. Development of the domain name system. *SIGCOMM Comput. Commun. Rev.*, 18(4):123–133, 1988. 18

[40] A. Moore and K. Papagiannaki. Toward the accurate identification of network applications. In *the 6th PAM*, pages pages 41–54, October 2005. 4, 23, 24, 156

[41] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. *SIGMETRICS Perform. Eval. Rev.*, 33:50–60, June 2005. 4, 25, 26, 157, 158

[42] T.S.E NG and H. ZHANG. Predicting internet network distance with coordinates-based approaches. In *IEEE Infocom '02*, pages 170–179. IEEE, 2002. 114, 117, 118

[43] Thuy T. T. Nguyen and Grenville Armitage. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. In *in Proceedings of the IEEE 31st Conference on Local Computer Networks*, 2006. 4, 25, 27, 157

[44] NLANR. Nlanr pma data project. `http://labs.ripe.net/datarepository/data-sets/nlanr-pma-data`, 2009. 6

[45] Junghun Park, Hsiao-Rong Tyan, and C C Kuo. Internet traffic classification for scalable qos provision. *2006 IEEE International Conference on Multimedia and Expo*, pages 1221–1224, 2006. 4, 25, 27, 157

[46] Marcin Pietrzyk, Jean-Laurent Costeux, Guillaume Urvoy-Keller, and Taoufik En-Najjary. Challenging statistical classification for operational usage : the adsl case. In *IMC 2009, 9th ACM SIGCOMM Internet Measurement Conference, November 4-6, 2009, Chicago, IL, USA*, 11 2009. 4, 25, 28, 157

[47] Planetlab. Planetlab. `http://www.planet-lab.org/`, 2009. 6, 120

[48] J. Postel. User datagram protocol, 1980. 13

[49] Jon Postel. Dod standard internet protocol. *SIGCOMM Comput. Commun. Rev.*, 10:12–51, October 1980. 11, 12, 13

[50] Jon Postel. Dod standard transmission control protocol. *SIGCOMM Comput. Commun. Rev.*, 10:52–132, October 1980. 13, 14

[51] Pyxida. Pyxida: An open source network coordinate library and application. `http://pyxida.sourceforge.net/`, 2009. 120

[52] S. Ratnasamy, M. Handly, and S. Shenker. Topologically-aware overlay construction and server selection. In *IEEE Infocom '02*, New York, NY, USA, 2002. IEEE. 114

[53] RON. Resilient overlay network. `http://nms.csail.mit.edu/ron/`, 2009. 114

[54] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, pages 135–148, New York, NY, USA, 2004. ACM. 4, 25, 27, 157

[55] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *WWW*, Philadelphia, USA, May 2004. 4, 23, 24, 156

[56] Skype. Skype. `http://skype.com/`, 2009. 114

[57] L. Tang and M. Crovella. Virtual landmarks for the internet. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, Miami Beach, FL, USA, 2003. ACM. 118

[58] TCPdump. Collection of various patches that have been floating around for lbl's tcpdump and libcap programs. `http://www.tcpdump.org/`, 2007. 49

[59] Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovi, and Antonio Nucci. Unconstrained endpoint profiling (googling the internet). In *ACM SIGCOMM*, SIGCOMM '08, pages 279–290, Seattle, WA, USA, 2008. ACM. 4, 25, 29, 78, 157

[60] Guohui Wang and T. S. Eugene Ng. Distributed algorithms for stable and secure network coordinates. In *Proceedings of the ACM/USENIX Internet Measurement Conference (IMC'08)*, Oct 2008. 115

[61] Guohui Wang, Bo Zhang, and T. S. Eugene Ng. Towards network triangle inequality violation aware distributed systems. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 175–188, New York, NY, USA, 2007. ACM. 115

[62] Joe H. Ward. Hierarchical grouping to optimize an objective function. *journal of the American Statistical Association*, pages 236–244, 1963. 113, 124

[63] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *SIGCOMM Comput. Commun. Rev.*, 36:5–16, October 2006. 4, 25, 26, 157

[64] I. Witten and E. Frank. Data mining: Practical machine learning tools and techniques. In *Morgan Kaufmann*, San Francisco, USA, 2005. 46

[65] Charles Wright, Fabian Monrose, and Gerald M. Masson. Hmm profiles for network traffic classification. In *ACM VizSEC/DMSEC*, 2004. 62, 63

[66] Sebastian Z, Thuy Nguyen, and Grenville Armitage. Self-learning ip traffic classification based on statistical flow characteristics. In *the 6th PAM*, 2005. 62, 63

[67] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. Automated traffic classification and application identification using machine learning. In *LCN*, 2005. 4, 25, 26, 63, 157, 158

# Présentation des Travaux de Thèse en Francais

## 7.4 Introduction

Internet a été créé comme un réseau simple, ouvert et flexible, où le seul service offert aux utilisateurs est le meilleur effort ”best efort”. Ce choix était délibéré pour la raison simple de faciliter l'interconnexion de réseaux, la création d'applications et l'amélioration de services d'Internet. Pendant les trente dernières années, Internet s'est développé à un réseau géant connectant des dizaines de milliers de réseaux autonomes. Au début, l'utilisation d'Internet a été limitée à l'échange d'informations par ”E-mails” et ”newsgroup”. Le besoin de trouver et d'organiser l'information quelques années plus tard a mené au développement des premières pages Web utilisant l'hypertexte. Aujourd'hui, des centaines de millions de personnes utilisent Internet, stimulant le développement d'une large variété d'applications qui continuent à apparaître tous les mois. Un principe fondamental d'Internet est que le réseau devrait rester aussi simple comme possible et donc l'intelligence devrait être dans les machines des utilisateurs qui exécutent les opérations standard comme des ”error recovery”, la localisation de ressources, etc. C'est le principe célèbre de bout en bout. Ce principe permet à n'importe quelle application Internet d'envoyer des paquets pour n'importe quelle adresse IP et n'importe quel numéro de port. Le réseau fait alors son mieux pour guider les paquets à cette destination correctement. Malheureusement, rien n'est gratuit dans la vie, l'ouverture du réseau au trafic rend la tâche de l'administration d'Internet et son ingénierie compliquée et mène par conséquent à un comportement nuisible pour les utilisateurs ainsi que les opérateurs du réseau. Dans cette thèse on essaye de fournir des solutions pour certains de ces problèmes rencontrés dans l'Internet d'aujourd'hui.

### 7.4.1 Motivation

Dans le but d'intégrer la qualité de service dans les réseaux (réseaux de nouvelle génération, NGN, donner une qualité de service pour le voix sur IP par exemple), ainsi que pour pouvoir arrêter des connexions Internet appartenant à un type d'application bien définie (i.e. connexion peer-to-peer) et enfin pour la détection d'anomalies et l'analyse de ses raisons, les opérateurs réseaux ont de plus en plus besoin d'identifier les différents types de trafic Internet (Web, MSN, DNS, peer-to-peer, etc.). Donc l'identification du trafic Internet est considérés aujourd'hui comme l'un des défis les plus importants pour les administrateurs réseau et les ISPs afin de pouvoir protéger leurs ressources contre le trafic indésirable et de prioriser certaines applications majeures. Cette identification peut aussi aider ces opérateurs pour le bon dimensionnement de leurs réseaux. Par contre il ne suffit pas d'identifier tout simplement le trafic, mais il faut l'identifier le plus tôt et le plus précisément possible. Aujourd'hui cette reconnaissance des applications devient de plus en plus complexe. Historiquement, cette reconnaissance a été basée sur les numéros de port. Cette méthode qui est considérée comme 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 numéros de 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 numéro de port est de moins en moins efficace.

Les techniques actuelles de l'inspection approfondie des paquets ”Deep packet inspection” (DPI) permettent d'aller plus loin dans l'identification des applications mais ils exigent une exploration complète et coûteuse de la charge utile des paquets. Mais le problème avec ces méthodes est qu'ils sont trop lentes sur les liens grands vitesse puisqu'elles incitent une charge importante des administrateurs, ainsi qu'ils ne sont pas pratiques quand les paquets sont cryptés.

Les méthodes statistiques sont préférées à celles basées sur le numéro de port et l'inspection approfondie des paquets (Deep packet inspection), car elles sont robustes au changement malveillant du numéro de port et fonctionnent avec le trafic crypté. Ces méthodes combinent l'analyse des paramètres statistiques des flux de paquets, tels que la taille des paquets et le temps les séparant, avec des techniques issues de la théorie d'apprentissage (machine learning). La majorité des méthodes statistiques ne peuvent pas identifier les flux applicatifs en temps réel et elles ont besoin d'atteindre la fin des flux avant de prendre une décision sur leur nature. Ceci est considéré comme trop long pour la plupart des administrateurs réseau, puisqu'il ne permet pas de bloquer un flux Internet indésirable à son début ni de lui donner en amant une qualité particulière de service.

La majorité de méthodes statistiques ne peut pas identifier des flux tôt (en temps réel) et ils exigent le fait d'atteindre la fin du flux avant la prise de n'importe quelle décision. Or cette attente jusqu'au la fin du flux est considérée inefficace par la plupart d'administrateurs réseau, parce qu'il ne fournit vraiment aucun moyen pour arrêter un flux ou lui donner une qualité spéciale de service avant sa fin. Donc c'est considéré comme très important pour les administrateurs de réseau d'utiliser une méthode efficace qui permet d'identifier le trafic Internet correctement et en temps réel.

Un autre défi important pour des administrateurs de réseau est de détecter et diagnostiquer les anomalies et les changement dans le réseau ainsi que leur raison que ça soit une congestion à long terme, une déviation, un échec de liaison ou un autre événement. Dans la littérature il y a un grand nombre de méthodes pour la détection d'anomalie mais la plupart d'entre eux exigent que des mesures exhaustive et périodique soient faites (des mesures de délai aller-retour RTT de multi-point vers multi-point). La réduction de la charge dans le réseau est considéré toujours un besoin essentiel pour les administrateurs de réseau. Récemment, une nouvelle approche a fait son apparition, les systèmes de coordonnée Internet, ces systèmes utilisées pour le positionnement des noeuds dans l'Internet ayant l'avantage principal de fournir les délai Aller-retour entre n'importe qu'elle deux noeuds dans le réseau et sans besoin de faire des mesures exhaustive.

Cette approche consiste la construction d'un système de coordonnée pour Internet. L'idée de base est de transmettre les délais mesurés entre quelques noeuds a un certain espace Euclidien et associer à chaque noeud des coordonnées spécifiques dans cet espace d'une telle façon que le délai de réseau entre n'importe quels deux noeuds peut être rapproché par la distance géométrique (Euclidien) les séparant. Ces systèmes de coordonnée peuvent être très utiles s'il y a une façon de les utiliser pour la détection des anomalies et de changements dans le réseau.

### 7.4.2 Contributions de la Thèse

Dans cette dissertation nous avons décrit les travaux effectués pendant cette thèse sur le sujet de l'identification de trafic Internet ainsi sur la détection des anomalies dans les réseaux. Dans la première partie de la thèse nous abordons le problème d'identification de trafic et nous avons visé le développement d'une méthode rapide et robuste qui permet d'identifier les applications en temps réel et avec grande précision. Nous présentons les trois méthodes que nous avons développés et qui permettent d'identifier le trafic Internet. La première méthode (décrite dans le Chapitre 3) est une nouvelle méthode probabiliste itérative qui permette d'identifier les applications en ligne, en temps réel et avec une précision très élevés (plus de 99%) et en utilisant seulement la taille des premiers paquets N. Notre méthode associe un niveau de confiance configurable au numéro de port porté dans l'entête de paquets (couche transport) et peut considérer un nombre variable de paquets au début d'un flux. Par la vérification sur des traces réelles nous pouvons observer que même dans le cas ou l'information sur le numéro de port n'est pas utilisé, une précision très élevée de classification peut être obtenue pour les applications existant dans notre trace après que quelques paquets ont été examinés.

La deuxième méthode (décrite dans le Chapitre 4) complète notre modèle itératif pour utiliser le temps entre paquet comme caractéristique pour identifier le trafic Internet. Pour arriver à cette fin nous avons dû présenter un modèle pour isoler le bruit introduit par le réseau et extraire le temps produit par l'application. Nous présentons notre modèle pour pré-traiter le temps entre-paquet et utiliser le résultat comme paramètre pour notre méthode itérative. Nous discutons ensuite la même approche itérative et nous évaluons notre méthode sur deux traces réelles différentes. Les résultats montrent que le temps entre-paquet peut être transformé à un paramètre important pour la classification de trafic Internet (après l'élimination du bruit).

Notre troisième méthode (décrite dans le Chapitre 5) est une nouvelle méthode itérative pour la classification de trafic en temps réel qui combine les approches statistiques avec et les approches qui profilent les utilisateurs afin de construire une méthode robuste et précise pour l'identification du trafic Internet. Nous utilisons la taille de paquet comme la caractéristique (paramètre) principale pour la classification et nous profitons du profil de trafic de l'utilisateur (adresse IP) (c'est-à-dire qu'elle l'application et combien) pour nous décider en faveur de ceci ou cette application. Ce dernier profil est mis à jour en ligne basé sur le résultat de la classification de flux précédents produits par ou adressé au même hôte (utilisateur). Nous évaluons notre méthode sur des traces réelles en utilisant plusieurs applications. Les résultats montrent que l'utilisation des information sur les hôtes (utilisateurs) permet d'améliorer la performance des méthodes statistiques. Ils prouvent aussi la capacité de notre solution de tirer des profils pour le trafic d'hôtes (utilisateurs) Internet et d'identifier les services qu'ils fournissent.

Dans la deuxième partie de cette dissertation, nous abordons le problème de détection des anomalies dans les réseaux. Nous commençons par étudier la stabilité de systèmes de coordonnée Internet (particulièrement Vivaldi). Dans une première étape, nous confirmons le fait que les coordonnées de Vivaldi oscillent au fil du temps en raison de la nature adaptative du système. Toutefois, les variations de ces coordonnées sont dans la plupart du temps en corrélation les unes avec les autres, pointant par conséquent vers un cluster de noeuds stables vu de l'intérieur du réseau. Dans un deuxième temps, nous présentons un nouvel algorithme de clustering basé sur des méthodes de groupement hiérarchique afin d'identifier ce cluster de noeuds stables. Enfin, nous soulignons l'utilité d'une telle constatation avec une application qui permet de détecter les changements dans le réseau. En changeant artificiellement les délais du réseau dans différents scénarios, nous montrons que ces changements sont reflétées par ce corps de noeuds stables, permettant ainsi d'obtenir une image globale de la stabilité du réseau sans avoir besoin de mesures exhaustives des délais.

### 7.4.3 Problèmes rencontrés pendant la thèse

Pendant cette thèse nous avons rencontré deux problèmes principaux :

- Le problème le plus important que nous avons rencontré est le manque de traces de trafic Internet avec "ground truth" pour la validation de nos méthodes d'identification de trafic Internet. Ce problème est un problème commun pour tous les scientifiques travaillant sur ce sujet. On peut trouver beaucoup de traces publiques pour la validation comme CAIDA, MAWI et NLANR, mais le problème majeur avec ces traces est qu'ils fournissent seulement les En-têtes de paquets et pas la charge utile ou l'application derrière chaque flux. Donc en utilisant ces traces nous pouvons seulement comparer les résultats de nos méthodes avec la classification utilisant le numéro de port et donc nous ne pouvons pas être sûrs de la vérification de nos méthodes sans le "ground truth". Pour cette raison nous avons décidé de ne pas utiliser ces traces publiques pour la validation de nos méthodes.

  Comme une alternative nous avons réussi à capturer une trace de paquet réelle au bord (edge) du réseau de laboratoire INRIA et nous avons utilisé deux traces réelles capturées au bord (edge) de l'Université de Brescia en Italie.

- Concernant la partie sur détection des anomalie, nous avons fait des expérimentations réelles sur le réseau Planetlab. Le problème que nous avons trouvé est la difficulté de créer des anomalies à l'intérieur du réseau Planetlab comme celle-ci est utilisé par des milliers des chercheurs autour du monde. Pour résister à ce problème nous avons utilisé les simulations qui mettent en oeuvre la topologie réelle du réseau Planetlab.

**Table** 7.1: Quelques numero de port standard.

| Applications | Numero de Port | Protocole de Transport |
|---|---|---|
| FTP | 21 | TCP |
| HTTP | 80 | TCP |
| HTTPS | 443 | TCP |
| POP3 | 110 | TCP |
| IMAP | 143 | TCP |
| SMTP | 25 | TCP |
| DNS | 53 | UDP |
| POPS | 995 | TCP |
| eMule | 4661-4662-4672 | TCP/UDP |
| Gnutella | 6346 | TCP/UDP |
| Bittorent | 6881 | TCP/UDP |
| SSH | 22 | TCP |
| Yahoo Messenger | 5010 | TCP |
| MSN | 1863 | TCP |

## 7.5 Etat de l'art

On peut regrouper les différentes méthodes d'identification de trafic en trois grandes classes:

### 7.5.1 Les méthodes d'identification par numéro de port

Ces méthodes sont les plus simples et les plus classiques où il suffit de regarder le numéro de port du destinataire dans l'entête de chaque paquet pour l'affecter à une application précise. Un enregistrement central, IANA (Internet Assigned Numbers Authority [16]), est responsable de définir le numéro de port pour chaque service standard. Par exemple, les serveurs de Web, en utilisant le protocole de HTTP pour transférer des pages Web, devraient fonctionner sur le port 80.

En principe, le numéro de port destinataire (côté serveur) permet l'identification de l'application, le tableau 7.1 donne un exemple de cette correspondance. Cependant, avec l'évolution des services Internet et l'apparition de nouvelles architectures réseaux (P2P par exemple), la classification basée sur le numéro de port présente des limitations, et elle est devenue dans plusieurs cas de figure insatisfaisante. La première limitation est qu'il n'est pas toujours possible d'associer une application à un numéro de port pour différentes raisons :

- La correspondance entre le numéro de port et l'application n'est pas toujours possible, et exclut en particulier certaines nouvelles applications; c'est le cas par exemple des applications P2P (Skype, Napster, Kazza,...), streaming, et d'autres.
- Plusieurs implémentations de TCP emploient des numéros de ports dans la catégorie des ports en-

registrés. Ceci pourrait conduire à classifier de manière erronée la connexion comme appartenant à l'application associée à ce port.

- Le choix dynamique du port de communication entre le serveur et le client est parfois possible. Par exemple, le ftp (passif) permet la négociation dynamique du numéro de port utilisé pour le transfert de données. Ce numéro est déterminé entre les deux extrémités durant la connexion de contrôle initiale (utilisant par défaut le port 20).
- Différentes applications sont encapsulées dans des applications bien connues, comme par exemple le streaming ou la messagerie instantanée sur http.
- Certaines applications, notamment le streaming sur UDP, utilisent des plages de ports qui peuvent éventuellement chevaucher des plages utilisées par d'autres applications.

La deuxième limitation est due au fait que les politiques de gestion de la QoS sont impactées par l'existence de services utilisant les mêmes numéros de ports, mais ayant des exigences de QoS différentes. Ainsi, des trafics élastiques ayant moins de contraintes en temps de transfert de paquets peuvent être confondus avec des services temps-réel ayant par contre de fortes contraintes en temps de transit.

Une étude récente sur le trafic pair à pair effectuée par CAIDA confirme qu'il est de plus en plus difficile d'identifier des applications de pair à pair en utilisant les numéros de ports.

### 7.5.2 Les méthodes d'identification par "deep packet inspection"

Une approche alternative est d'inspecter la charge utile de chaque paquet en recherchant des signatures spécifiques pour chaque protocole. Par exemple, une signature simple pour le trafic eDonkey serait ?0xe319010000?. Donc avec ces méthodes, chaque paquet qui contient cette signature est marqué comme trafic eDonkey.

Ces méthodes [26, 14, 40, 9, 55, 35, 11] appelle "Deep packet inspection methods" analysent les charges utiles des paquets et les comparent avec les signatures connues de chaque application (see table 7.2). Ces techniques sont des méthodes de classification quasi déterministe pour identifier les différents types d'applications dans un mélange de trafic, mais elles possèdent également de nombreuses limitations:

- L'information descriptive de la chaîne de caractères de l'application, ou de la version de l'application, n'est pas toujours disponible.
- Avec l'augmentation importante des applications sécurisées où le payload du paquet est crypté, cette technique ne permet pas de reconnaître l'application d'un paquet où le trafic est crypté, et par suite elle ne fonctionne pas.
- Les formats ainsi que les types de paquets sont différents d'une application à l'autre. Ces informations ne sont donc pas faciles à extraire, puisqu'elles dépendent en particulier des protocoles adjacents employés.
- Cette technique ne fonctionne pas avec les nouvelles applications car on n'a pas d'information sur leur signature.

**Table** 7.2: Les signatures de quelques applications.

| Applications | Signature |
|---|---|
| HTTP | "http/1." |
| SSH | "SSH" |
| POP3 | "+OK Password required for" |
| IRC | "USERHOST" |
| Direct Connect | "$MyN","Dir$" |
| MP2 | "GO!!, MD5, SIZ0x2" |
| Fasttrac | "Get /.hash" |
| eDonkey | "0xe319010000" |
| Gnutella | "GNUT", "GIV" |
| Bittorent | "0x13Bit" |
| MSN Messenger | "?PNG?0x0d0a" |

- Avec ces méthodes, on a besoin d'une très grande capacité de stockage car il faut stocker la charge utile de chaque paquet.
- Des services qui ont des exigences différentes peuvent être encapsulés dans des applications classiques, par exemple le streaming sur HTTP.

### 7.5.3 Les méthodes statistiques

Puisque les classifications utilisant le numéro de port ne sont pas toujours efficaces, et vu que la recherche des signatures précises dans les charges utiles des paquets est limitée, il y a eu une nouvelle tendance de développer des techniques alternatives de classification dans la communauté de recherches. Ces méthodes partagent une base commune : au lieu de regarder les ports ou le contenu du paquet, ils analysent le comportement des connexions TCP. Plusieurs papiers [38, 41, 67, 63, 7, 54, 45, 43, 13, 12, 5, 2, 46, 3, 59, 25] ont démontré l'intérêt de distinguer les différents types de trafic à partir des critères statistiques (par exemple la taille des paquets et le temps entre paquet) et en utilisant des méthodes de classification supervisées ou non supervisées. Expliquons tout d'abord l'approche du "machine learning" qu'on peut diviser en deux grandes catégories:

- Les algorithmes supervisés: qui nécessitent la connaissance de l'application de la connexion avant d'appliquer l'algorithme dans la phase d'apprentissag
- Les algorithmes non supervisés: qui ne nécessitent aucune connaissance précédente.

De même, chacun de ces algorithmes supervisés ou non supervisés est constitué de deux phases principales:

- La phase d'apprentissage où l?on groupe les connexions d'un "training data set" dans des classes différentes suivant les valeurs des critères statistiques de chaque connexion, puis on associe chaque classe à une application précise.

- La phase de classification où l?on classifie chaque nouvelle connexion dans une des classes déjà définies dans la phase d'apprentissage (on affecte chaque connexion dans la classe dont la distance entre ces valeurs et les valeurs du centre de classe est la plus proche).

Dans [38], McGregor et al. ont montré l'utilité d'employer les algorithmes de classifications pour l'identification du trafic tout en en utilisant un algorithme non supervisé appelé ”Auto Class”, et en utilisant les critères statistiques suivantes : taille de paquet, temps entre arrivées, nombre d'octets et durée de connexion.

De même Zander et al. ont utilisé l'auto class algorithme dans [67] et en utilisant des critères statistiques sur les paquets et les connexions.

Dans [41], Moore et al. utilisent un algorithme supervisé appelé Naïve Bayes pour classifier le trafic de TCP, et ils essayent de trouver le meilleur ensemble de critères statistiques à utiliser pour classifier ce trafic.

Dans [13], Erman et al. ont utilisé trois algorithmes non supervisés (KMeans, DBSCAN et Auto Class) pour classifier le trafic TCP. Cette méthode peut identifier les protocoles avec une bonne précision.

En [3], Laurent Bernaille et al. ont développé une méthode qui permet de classifier le trafic TCP en temps réel en utilisant seulement la taille et le sens de chacun des quatre premiers paquets, mais cette méthode utilise des paquets précis, c'est-à-dire qu'elle ne peut pas identifier les connexions lorsque les quatre premiers paquets ne sont pas captés, ainsi que les connexions qui ont un nombre de paquets inférieur à quatre

## 7.6 Description de notre méthodes iterative

Pour profiter d'informations portées par les N premiers paquets d'un flux et éviter des problèmes pendant la phase d'apprentissage et de classification causée par l'utilisation de beaucoup de paramètres, nous recourons à une approche à base de paquet itérative où nous utilisons la taille et la direction de chaque paquet individuellement pour en calculer la probabilité d'appartenir à chaque application, en deuxième phase nous fusionnons les résultats de tous les paquets d'un flux ensemble en utilisant une fonction de probabilité (approche Bayésienne) que nous présentons pour associer le flux à l'application le plus probable. Notez que nous considérerons aussi le temps entre-paquets après l'avoir filtré des bruits dut au réseau. Plus de détails seront présentés dans les chapitres suivant. Notre approche consiste en trois phases principales : la phase de construction du modèle, la phase de classification et la phase d'étiquetage (on associe les flux aux applications).

### 7.6.1 La phase de la construction du modèle ou d'apprentissage

C'est une phase très importante dans notre travail ou on construit les modèles (ensemble de classes) en utilisant un jeu de données spécial, (on connait en avance l'association entre les flux et les applications pour cette jeu de données). Ces modèles construit vont être utilisés plus tard dans la phase de classification.. Comme nous étudions chaque paquet individuellement, nous construisons un modèle séparé pour chaque paquet en utilisant tous les flux dans le jeu de données (un modèle pour le premier paquet, un modèle pour le deuxième paquet, etc). Ces modèles décrivent comment la taille d'un paquet, (le premier paquet par exemple), varie entre le différents applications) et comment la taille de paquet occupe les différentes parties de l'espace. On a utilisé un algorithme de "Datamining" bien connus appelle K-Means pour construire les modèles dans la phase d'apprentissage. Nous avons choisit les traces pour l'apprentissage de façon qu'ils sont représentatifs des applications à identifier. Pour cette fin, nous avons pris un nombre semblable de flux de toutes les applications parce que si le nombre de flux n'est pas le même, les applications qui prédominent peuvent influencer et biaiser les modèles et par suite la classification. Pour construire le modèle pour une taille de paquet donnée (disent la taille du paquet i-th d'un flux), nous représentons chaque flux comme un point sur un axe. Ce point a une coordonnée positive si le paquet est envoyé par le client et une coordonnée négative si le paquet est envoyé par le serveur. La coordonnée de ce point est égale à la taille du paquet. Quand le temps entre des paquets est utilisé, un flux est représenté par deux coordonnées sur deux axe, un pour la taille de paquet et un pour le temps entre ce paquet et le précédent (ou le suivant). Sans perte de généralité, nous considérons un maximum de dix paquets par flux. À la fin du modèle construisant la phase, nous obtenons dix modèles pour les dix premiers paquets de n'importe quel flux. Dans chaque modèle, on a utilisé 200 classes, comme il faut préciser le nombre de classe pour K-Means. Notez que le choix de nombre de classe a été basé sur des expérimentations, 200 classes a montré qu'il est un bon compromis entre la complexité et la précision.

### 7.6.2 La phase de classification

Dans cette phase nous allons utiliser les modèles construits dans la phase d'apprentissage pour classifier le trafic en ligne. Notez qu'ici nous affectons des paquets de flux aux classes et pas aux applications.

À l?arrivé de chaque nouveau paquet d'un flux, il est classifié indépendamment des autres paquets en utilisant le modèle correspondant à sa position dans le flux. Par exemple, quand nous capturons le premier paquet d'un nouveau flux, nous affectons le paquet (et de là le flux) à une classe du modèle construit pour tous les paquets qui sont premiers dans leurs flux. Le même pour le deuxième paquet, et cetera. Pour associer un flux a une classe précis, nous calculons la distance Euclidienne qui sépare le point correspondant au nouveau flux dans l'espace modèle du centre de chaque classe et nous l'affectons au classe le plus proche. Nous répétons cette classification pour tous les paquets d'un flux jusqu'à ce que nous soyons satisfaits ou nous atteignons un certain seuil. La satisfaction est mesurée est fait via une nouvelle fonction de probabilité décrit plus tard. Clairement, la classification d'un flux est différente et indépendante d'un paquet à un autre, de là le résultat de la classification. Par exemple, un nouveau flux peut être affecté au classe numéro 5 en utilisant le premier paquet et à la classe numéro 19 en utilisant le deuxième, et cetera. C'est l'ensemble de résultats de classification qui choisiront finalement l'application la plus probable à laquelle le flux appartiendrait.

### 7.6.3 La phase d?étiquetage ou d'association aux applications

Nous affectons dans cette phase les flux aux applications en comptant sur les résultats de la phase de classification. La phase de classification indique la probabilité que chaque paquet d'un flux y appartient cette application ou pas. Ceux-ci sont obtenus des classes dans lesquelles les paquets tombent pendant la phase de classification. Nous combinons (instantanément) ces probabilités ensemble pour obtenir une nouvelle évaluation pour décider si nous y associons le flux entier à cette une application ou une autre. Ceci mène à une fonction de probabilité itérative que nous utilisons pour l'attribution des flux aux applications. Laissez-nous définir les variables suivantes à être utilisé ensuite :

- $i$: utilisé pour dénoter des classes.
- $A$: utilisé pour dénoter des applications.
- $N$: le nombre maximal de paquets évalués d'un flux.
- $k$: le numéro de paquet testé $k$ (paquet 1, paquet 2, etc).
- $NA$: le nombre total d'applications.
- $\alpha_A$: la valeur (entre 0 et 1) affecté par l'administrateur à la confiance en numéro de port standard pour l'application $A$.
- $F_{A,i}$: le nombre de flux appartenant à l'application $A$ et classe $i$ dans la trace d'apprentissage.
- $F_A$: le nombre total de flux appartenant à l'application $A$ dans la trace d'apprentissage.
- $Pr(A/Result)$: la probabilité qu'un flux appartient à l'application $A$ en connaissant les résultats de la phase de classification (c'est-à-dire la classe i (1) pour le premier paquet, la classe i (2) pour le deuxième paquet et cetera).
- $Pr(i(k)/A)$: la probabilité que le paquet d'ordre $k$ d'un flux appartenant a l'application $A$ chutes dans classe $i$. Ceci peut être calculé de la trace d'apprentissage comme $Pr(i(k)/A) = \frac{F_{A,i}}{F_A}$.

- $Pr(A)$: la probabilité que n'importe quel flux aléatoirement choisi vient de l'application $A$, indépendamment de n'importe quelles informations sur ses tailles de paquet et le temps entre-paquets. La somme de ces probabilités sur toutes les application doit être égale à 1. Nous intégrons dans cette probabilité la confiance donné par l'administrateur du réseau au numéro de port porté dans l'en-tête de transport de chaque paquet. Par exemple, pour un flux donné et si le numéro de port est égal à 80 (le numéro de port WEB standard), nous donnons au $Pr(WEB)$ une valeur $\alpha_{80}$ entre 0 et 1 (cette valeur est donnée par l'administrateur comme une fonction de la confiance qu'il a dans le port 80 en étant seulement le du Web). Ces modèles de valeur la probabilité qu'un flux portant le port numéro 80 appartient à l'application WEB. Pour toutes les autres applications nous donnons au $Pr(A)$ la même valeur qui est égale : $\frac{(1-\alpha_{80})}{(NA-1)}$.
  Notez que l'administrateur peut décider de ne pas donner de poids au numéro de port. Ceci peut être le cas quand il n'a pas confiance en ces informations. Ici, nous donnons $Pr(A)$ la même valeur pour toutes les applications indépendamment au numéro de port. Cette valeur spécifique est égale $\frac{1}{NA}$.

Nous visons à calculer la probabilité qu'un flux appartient à une application $A$ étant donné les résultats de la phase de classification appliquée au premier $N$ paquets du flux. Prenez par exemple les deux premiers paquets et leurs classes correspondante $i(1)$ and $i(2)$. La probabilité que nous cherchons peut être écrite comme suit ::

$$\begin{aligned}
Pr(A/(i(1)\cap i(2))) = \frac{Pr(A\cap(i(1)\cap i(2)))}{Pr(i(1)\cap i(2))} &= \frac{Pr(A)*Pr((i(1)\cap i(2))/A)}{Pr(i(1)\cap i(2))} \\
&= \frac{Pr(A)*Pr(i(1)/A)*Pr(i(2)/A)}{Pr(i(1)\cap i(2))} \\
&= \frac{Pr(A)*\prod_{k=1}^{2}Pr(i(k)/A)}{\sum_{A=1}^{NA}Pr(A)*\prod_{k=1}^{2}Pr(i(k)/A)}
\end{aligned}$$

Maintenant, nous pouvons généraliser cette expression pour calculer la probabilité qu'un flux appartient à une application $A$, étant donné les résultats de classification pour les premiers $N$ paquets :

$$Pr(A/Result) = \frac{Pr(A)*\prod_{k=1}^{N}Pr(i(k)/A)}{\sum_{A=1}^{NA}Pr(A)*\prod_{k=1}^{N}Pr(i(k)/A)}$$

Pour chaque nouveau flux et quand nous capturons le premier paquet (sauf le paquet SYN), nous classifions d'abord le flux selon ce paquet et nous calculons la probabilité qu'il appartient à chaque application.

Alors, nous prenons la probabilité d'attribution la plus haute et nous le comparons avec un seuil $th$ indiqué par l'administrateur de réseau. Si cette probabilité est plus grande que le seuil $th$, nous étiquetons le flux par l'application, autrement nous prenons et classifions le paquet suivant et nous recalculons la probabilité d'attribution en utilisant les résultats de la phase de classification obtenue pour le premiers et deuxièmes paquets séparément. Nous vérifions de nouveau la probabilité résultante et nous continuons à ajouter plus de paquets jusqu'à ce que le seuil soit excédé ou le maximum permis de nombre de tests est atteint.

## 7.7 Conclusion

Trois nouvelles méthodes pour l'identification de trafic ont été développées, ces méthodes suivent une approche itérative et utilisent comme paramètres statistique la taille, la direction et le temps entre les premiers paquets de flux. En considérant les paquets séparément l'un de l'autre et avec l'aide d'une nouvelle fonction de probabilité qui combine les observations faites sur les différentes paquets d'un flux, nous pouvons obtenir une haute précision de classification qui continuent à s'améliorer en ajoutant plus de paquets de chaque flux et cela a pu atteindre un niveaux de précision autour de 98% après l'utilisation de 10 paquets pour la classification. L'avantage principal de cette méthode consiste en ce qu'il permet de commencer le processus d'identification directement après l'arrivée du premier paquet dans le flux. La décision prise est confirmée avec chaque nouvelle paquet reçus. Différemment aux autres méthodes qui utilisent la taille des 1er ensemble d'une manière jointe, le modèle itératif de notre méthode permet d'utiliser n'importe quel nombre de paquets en augmentant la précision de la classification avec chaque nouveau paquet. Notez que la complexité de notre méthode est inférieure que la complexité du modèle joint. Comme une deuxième contribution dans cette thèse nous avons effectué une étude sur le temps entre-paquet et nous analysons comment on peut l'utilisé comme paramètre pour identifier les applications Internet. Nous avons crée une modèle pour les différents composants du temps entre-paquet et nous proposons de filtrer le bruit causé par le réseau pour extraire des caractéristiques appropriées pour la classification. Particulièrement ce traitement est important de mettre en évidence les caractéristiques interactives de chaque application (client et serveur). Les résultats montrent aussi que notre méthode atteint une précision totale de 99% pour la classification de toutes les applications quand nous filtrons le bruit de réseau du temps entre-paquet (le 20 premier temps entre-paquet est utilisé). Le temps entre-paquets peut alors être utilisé pour identifier le trafic Internet. La possibilité d'utiliser le temps entre-paquets comme un paramètre augmente notre choix (au niveau des paramètres) et fait notre méthode plus robuste.

Comme une troisième contribution dans cette thèse nous développons une nouvelle méthode qui combine les approches statistiques et ceux profilant les utilisateurs. Les paramètres statistiques que nous utilisons sont la taille et la direction des premiers paquets N. La nouveauté de cette approche consiste dans l'utilisation du profil de l'utilisateur pour affiner(raffiner) la classification. Nous évaluons notre solution en utilisant des traces réelles. Les résultats montrent une grande amélioration pour la classification des applications quand le profil des utilisateurs est utilisé (surtout pour les 4 premiers paquets). Particulièrement le classificateur atteint une précision de 98% après l'utilisation de 10 premiers paquets d'un flux pour la classification.

Concernant le sujet de la détection d'anomalie nous avons développé une nouvelle methode qui permet de détecter les anomalie en utilisant le système de coordonnées Vivaldi. Dans une première étape, nous confirmons le fait que les coordonnées de Vivaldi oscillent au fil du temps en raison de la nature adaptative du système. Toutefois, les variations de ces coordonnées sont dans la plupart du temps en corrélation les unes avec les autres, pointant par conséquent vers un cluster de noeuds stables vu de l'intérieur du réseau. Dans un deuxième temps, nous présentons un nouvel algorithme de "clustering" basé sur des méthodes de groupement hiérarchique afin d'identifier ce cluster de noeuds stables. Enfin, nous soulignons l'utilité d'une telle constatation avec une application qui permet de détecter les changements dans le réseau. En changeant artificiellement les délais du réseau dans différents scénarios, nous montrons que ces changements sont reflétées par ce corps de noeuds stables, permettant ainsi d'obtenir une image globale de la stabilité du réseau sans avoir besoin de mesures exhaustives des délais.

## 7.8 Travaux futures

Concernant les perspectives de recherche futures nous planifions de:

- Valider nos méthodes sur d'autres types d'applications et des traces, particulièrement des traces ADSL.
- Développer un générateur de trafic de réseau qui peut simuler le trafic Internet réel pour aider les chercheurs travaillant dans ce domaine à résister au problème du manque de traces pour la validation de leurs méthodes.
- Rassembler un grand jeu de données et faites le public pour tester et comparer la performance des différentes méthodes existant dans la littérature. Comme toutes les méthodes sont évaluées sur des jeux de données différents, il n'y a aucun support commun pour comparer des résultats.
- Conduire une étude expérimentale complète pour valider la performance du notre protocole de détection d'anomalie en considérant un grand jeu de scénarios de changement de l'état de réseau.