Network Traffic Classification: From Theory to Practice

Valentín Carela-Español

Advisor: Dr. Pere Barlet-Ros
Co-Advisor: Prof. Josep Solé-Pareta

Ph.D. in Computer Science

Universitat Politècnica de Catalunya BarcelonaTech
Department d'Arquitectura de Computadors

Barcelona, September 2014

Abstract

Since its inception until today, the Internet has been in constant transformation. The analysis and monitoring of data networks try to shed some light on this huge black box of interconnected computers. In particular, the classification of the network traffic has become crucial for understanding the Internet. During the last years, the research community has proposed many solutions to accurately identify and classify the network traffic. However, the continuous evolution of Internet applications and their techniques to avoid detection make their identification a very challenging task, which is far from being completely solved.

This thesis addresses the network traffic classification problem from a more practical point of view, filling the gap between the real-world requirements from the network industry, and the research carried out in the network traffic classification field. In this work, we identify several problems that hinder the introduction of new proposals for traffic classification in production networks. First, proposed techniques usually do not meet the special requirements of production networks, with massive amounts of traffic and limited resources to process it, which make them very difficult to deploy. Second, classification techniques need to be updated regularly in order to maintain its accuracy. The cost of this maintenance process is usually unfeasible in networks with hundreds or thousands of routers. Last but not least, there is a clear lack of systematic methods to validate and compare the state-of-the-art techniques, mainly because most proposals base their results on private datasets labeled with techniques of unknown reliability.

The first block of this thesis aims to facilitate the deployment of existing techniques in production networks. To achieve this goal, we study the viability of using NetFlow as input in our classification technique, a monitoring protocol already implemented in most routers and switches. Since the application of packet sampling has become almost mandatory in large networks, we also study its impact on the classification and propose a method to improve the accuracy in this scenario. Our results show that it is possible to achieve high accuracy with both sampled and

unsampled NetFlow data, despite the limited information provided by NetFlow.

Once the classification solution is deployed it is important to maintain its accuracy over time. Current network traffic classification techniques have to be regularly updated to adapt them to traffic changes produced by the continuous evolution of the Internet traffic. The second block of this thesis focuses on this issue with the goal of automatically maintaining the classification solution without human intervention. Using the knowledge of the first block, we propose a classification solution that combines several techniques only using Sampled NetFlow as input for the classification. Then, we show that classification models suffer from temporal and spatial obsolescence and, therefore, we design an autonomic retraining system that is able to automatically update the models and keep the classifier accurate along time.

Going one step further, we introduce next the use of stream-based Machine Learning techniques for network traffic classification. In particular, we propose a classification solution based on Hoeffding Adaptive Trees. Apart from the features of stream-based techniques (i.e., process an instance at a time and inspect it only once, with a predefined amount of memory and a bounded amount of time), our technique is able to automatically adapt to the changes in the traffic by using only NetFlow data as input for the classification. In order to extract sound conclusions we evaluate our technique using a 13 years long dataset from a transatlantic link in Japan.

The third block of this thesis aims to be a first step towards the impartial validation and comparison of state-of-the-art classification techniques. The wide range of techniques, datasets, and ground-truth generators make the comparison and validation of different traffic classifiers a very difficult task. To achieve this goal we evaluate the reliability of different Deep Packet Inspection-based techniques (DPI) commonly used in the literature for ground-truth generation. Although according to conventional wisdom they are, in theory, one of the most accurate techniques, the results we obtain show that some well-known DPI techniques present several limitations that make them not recommendable as a ground-truth generator in their current state.

In addition, we publish some of the datasets used in our evaluations to address the lack of publicly available datasets and make the comparison and validation of existing techniques easier. In particular, the dataset we publish in this last third block is the first reliable labeled dataset with full packet payload available for the research community. To the best of our knowledge, this is the only dataset available that makes possible the comparison and validation of Machine Learning and DPI techniques.

Resumen

Desde sus orígenes hasta la actualidad, Internet ha estado en constante evolución. El análisis y la monitorización de las redes tratan de arrojar luz sobre esta caja negra de ordenadores interconectados que es Internet. En particular, la clasificación de tráfico de red se ha vuelto crucial para la comprensión de Internet. Durante los últimos años, la comunidad investigadora ha propuesto muchas soluciones para identificar y clasificar con precisión el tráfico de red. Sin embargo, la continua evolución de las aplicaciones de Internet y sus técnicas para evitar ser detectadas hace su identificación un tarea muy complicada, que está lejos de estar completamente resuelta.

Esta tesis aborda el problema de la clasificación de tráfico de red desde un punto de vista más práctico, tratando de hacer confluir las necesidades de los entornos reales y la investigación llevada a cabo. En este trabajo identificamos diferentes problemas que entorpecen la introducción de las nuevas propuestas para clasificar el tráfico en redes troncales. En primer lugar, las técnicas propuestas no reúnen normalmente los requisitos necesarios para operar en redes troncales, con enormes volúmenes de tráfico y limitados recursos para procesar, lo que dificulta su despliegue. En segundo lugar, las técnicas de clasificación necesitan ser actualizadas regularmente con el objetivo de mantener su precisión. El coste de este mantenimiento es normalmente inviable en redes con cientos o miles de enrutadores. Por último, pero no por ello menos importante, hay una gran falta de métodos para validar y comparar las técnicas propuestas en la literatura. Esto es debido principalmente a que la mayoría de propuestas basan sus resultados en conjuntos de datos privados etiquetados con técnicas de desconocida fiabilidad.

El primer bloque de esta tesis pretende facilitar el despliegue de las técnicas de clasificación en redes troncales. Para ello estudiamos la viabilidad de usar como entrada de nuestra técnica de clasificación NetFlow, un protocolo de monitorización implementado en la mayoría de enrutadores y conmutadores del mercado. Además, dado que la aplicación de muestreo de paquetes es una práctica muy extendida en

las redes troncales, estudiamos su impacto en la clasificación y proponemos un método para mejorar su precisión en este escenario. Los resultados muestran que es posible conseguir una alta precisión tanto con datos NetFlow muestreados como no muestreados, a pesar de la limitada información que nos proporciona NetFlow.

Una vez desplegado el sistema de clasificación el siguiente objetivo es mantener su precisión a lo largo del tiempo. Las soluciones actuales requieren actualizaciones periódicas para adaptarse a los cambios en el tráfico. El segundo bloque de esta tesis se centra en este problema persiguiendo automatizar el proceso de mantenimiento y hacerlo sin intervención humana. Siguiendo la línea del primer bloque, proponemos un sistema de clasificación que combina varias técnicas que usan únicamente NetFlow como entrada para la clasificación. A partir de este sistema mostramos que los modelos de clasificación sufren de obsolescencia temporal y espacial y, para ello, diseñamos e implementamos un sistema de reentreno automático capaz de actualizar automáticamente los modelos y mantener la clasificación precisa a lo largo del tiempo.

Yendo un paso más allá, introducimos el uso de técnicas de Aprendizaje Máquina (ML, por sus siglas en inglés) basadas en flujos de datos para la clasificación de tráfico de red. En particular, proponemos una solución basada en Hoeffding Adaptive Trees. Además de las características propias de las técnicas basadas en flujos de datos (i.e., inspección única de cada instancia, con una cantidad de memoria predefinida y en un tiempo limitado), nuestra técnica es capaz de adaptarse automáticamente a los cambios en el tráfico usando únicamente datos NetFlow como entrada para la clasificación. Por último, para extraer conclusiones sólidas, evaluamos nuestra técnica usando una traza que comprende 13 años de tráfico de un enlace transatlántico en Japón.

El tercer bloque pretende ser un primer paso hacia la validación y comparación imparcial de las propuestas del estado del arte. El amplio rango de técnicas, conjuntos de datos y generadores de verdad terreno hacen la comparación y validación de los diferentes clasificadores de tráfico una tarea muy complicada. Con ese fin evaluamos la fiabilidad de diferentes técnicas basadas en Inspección Profunda de Paquetes (DPI, por sus siglas en inglés) habitualmente usadas en la literatura para la generación de la verdad terreno. Aunque estás técnicas son, en teoría, unas de las más precisas, los resultados que obtenemos muestran que algunas técnicas DPI presentan graves errores que desaconsejan su uso en su estado actual.

Además, para abordar la falta de conjuntos de datos públicos publicamos algunos de los usados en nuestras evaluaciones para facilitar la comparación y validación de las técnicas existentes. En particular, el conjunto de datos publicado en el tercer bloque es el primer conjunto de datos etiquetado fiablemente y con el

contenido completo que está disponible para la comunidad investigadora. Hasta donde sabemos, este es el primer conjunto de datos disponible públicamente que permite la validación y comparación de técnicas ML y DPI.

Contents

List of Figures xiii

List of Tables xvi

1 Introduction 1
- 1.1 Motivations . . . 2
- 1.2 Contributions and Impact . . . 4
- 1.3 Thesis Organization . . . 7

2 Background 9
- 2.1 Network Traffic Classification Approaches . . . 9
  - 2.1.1 Port-based approach . . . 9
  - 2.1.2 Payload-based approach . . . 10
  - 2.1.3 Flow features-based approach . . . 11
  - 2.1.4 Host-behavior-based approach . . . 13
  - 2.1.5 IP-based approach . . . 14
  - 2.1.6 Comparing Approaches . . . 14
- 2.2 Network Traffic Classification Evaluation . . . 15
  - 2.2.1 Performance Metrics . . . 15
  - 2.2.2 Ground-truth generation . . . 17
- 2.3 Datasets . . . 17
  - 2.3.1 UPC Dataset . . . 18
  - 2.3.2 CESCA Dataset . . . 19
  - 2.3.3 MAWI Dataset . . . 20
  - 2.3.4 PAM Dataset . . . 21

3 Traffic Classification with NetFlow 23
- 3.1 Introduction . . . 23

3.2 Methodology . . . 24
3.2.1 Traffic Classification Method . . . 25
3.2.2 Training Phase and Ground-truth . . . 25
3.2.3 Machine Learning and Validation Process . . . 27
3.2.4 Performance Metrics . . . 28
3.2.5 Evaluation Datasets . . . 28
3.3 Results . . . 29
3.3.1 Performance with Unsampled NetFlow . . . 30
3.3.2 Performance with Sampled NetFlow . . . 32
3.4 Analysis of the Sources of Inaccuracy under Sampling . . . 34
3.4.1 Error in the Traffic Features . . . 35
3.4.2 Changes in the Flow Size Distribution . . . 39
3.4.3 Flow Splitting . . . 42
3.5 Dealing with Sampled NetFlow . . . 44
3.5.1 Improving the Classification Method . . . 45
3.5.2 Evaluation in Other Network Environments . . . 47
3.5.3 Lessons Learned and Limitations . . . 48
3.6 Related Work . . . 48
3.7 Chapter Summary . . . 49

4 Autonomic Traffic Classification System 53
4.1 Introduction . . . 53
4.2 Traffic Classification System . . . 54
4.2.1 The Application Identifier . . . 55
4.2.2 The Autonomic Retraining System . . . 57
4.2.3 Training Dataset Generation . . . 59
4.3 Evaluation . . . 60
4.3.1 Evaluation of labeling DPI-based techniques . . . 60
4.3.2 Training Dataset Evaluation . . . 62
4.3.3 Retraining Evaluation . . . 64
4.3.4 Retraining Evaluation by Institution . . . 67
4.4 Related Work . . . 69
4.5 Chapter Summary . . . 69

5 Streaming-based Traffic Classification 71
5.1 Introduction . . . 71
5.2 Classification of evolving network data streams . . . 73
5.2.1 Hoeffding Tree . . . 73

5.2.2 Hoeffding Adaptive Tree . . . 74
5.2.3 Inputs of our system . . . 74
5.3 Methodology . . . 75
5.3.1 MOA: Massive Analysis Online . . . 75
5.3.2 The MAWI Dataset . . . 76
5.4 Hoeffding Adaptive Tree Parametrization . . . 76
5.4.1 Numeric Estimator . . . 77
5.4.2 Grace Period . . . 78
5.4.3 Tie Threshold . . . 79
5.4.4 Split Criteria . . . 80
5.4.5 Leaf Prediction . . . 80
5.4.6 Other Parameters . . . 81
5.5 Hoeffding Adaptive Tree Evaluation . . . 82
5.5.1 Single Training Evaluation . . . 83
5.5.2 Interleaved Chunk Evaluation . . . 84
5.5.3 Chunk Size Evaluation . . . 84
5.5.4 Periodic Training Evaluation . . . 87
5.5.5 External evaluation . . . 88
5.6 Related Work . . . 88
5.7 Chapter Summary . . . 89

6 Validation of Traffic Classification Methods 91
6.1 Introduction . . . 91
6.2 Methodology . . . 92
6.3 Performance Comparison . . . 96
6.3.1 Sub-classification of HTTP traffic . . . 98
6.4 Lessons Learned and Limitations . . . 99
6.5 Related Work . . . 101
6.6 Chapter Summary . . . 101

7 Other Improvements to Traffic Classifiers 103
7.1 Efficient Profiled Flow Termination Timeouts . . . 104
7.1.1 Introduction . . . 104
7.1.2 Dataset . . . 105
7.1.3 Methodology . . . 105
7.1.4 Results . . . 107
7.1.5 Related work . . . 115
7.1.6 Chapter Summary . . . 117

7.2 Early Classification of Network Traffic . . . 118
7.2.1 Introduction . . . 118
7.2.2 Methodology . . . 119
7.2.3 Results . . . 122
7.2.4 Related Work . . . 130
7.2.5 Chapter Summary . . . 130

8 Conclusions **133**
8.1 Future Work . . . 135

Bibliography **146**

Appendix A **147**
8.2 UPC dataset . . . 148
8.3 PAM dataset . . . 149

Appendix B **151**
8.4 Publications . . . 151
8.4.1 Journals . . . 151
8.4.2 Conferences . . . 151
8.4.3 Technical Reports . . . 152
8.4.4 Supervised Master Students . . . 152
8.4.5 Datasets . . . 152

List of Figures

2.1 Traffic breakdown of the traces in the UPC dataset 19

3.1 Overview of the machine learning and validation process 27
3.2 Precision (mean with 95% CI) by application group (per flow) of our traffic classification method (C4.5) with different sampling rates 30
3.3 Recall (mean with 95% CI) by application group (per flow) of our traffic classification method (C4.5) with different sampling rates . . 31
3.4 Overall accuracy (mean with 95% CI) of our traffic classification method (C4.5) and a port-based technique as a function of the sampling rate 33
3.5 Overall accuracy when removing the error introduced by the inversion of the features (UPC-I trace, using UPC-II for training) 40
3.6 Validation of Eq. 3.11 against the empirical distribution of the *original* flow length detected with $p = 0.1$ (UPC-II trace). 41
3.7 Flow length distribution of the detected flows when using several sampling probabilities. The figure was obtained combining Eq. 3.11 with Eq. 3.12 42
3.8 Amount of split flows as a function of the sampling probability p. The figure shows both the empirical results (UPC-II trace) and the analytical ones (Eq. 3.14) 44
3.9 Overall accuracy (mean with 95% CI) of our traffic classification method with a normal and sampled training set 45
3.10 Precision (mean with 95% CI) by application group (per flow) of our traffic classification method with a sampled training set 46
3.11 Recall (mean with 95% CI) by application group (per flow) of our traffic classification method with a sampled training set 46
3.12 Overall accuracy with a normal and sampled training set using the traces from other environments 51

4.1 Application Identifier and Autonomic Retraining System Architecture 54
4.2 DPI labeling contribution 61
4.3 Impact of the *Autonomic Retraining System* on the *Application Identifier* with the selected configuration (i.e., naive training policy with 500K) 66
4.4 Comparative of the *Autonomic Retraining System* with other solutions 67
4.5 Comparative of the *Autonomic Retraining System* by institution . . 68

5.1 Impact of the *Numeric Estimator* parameter 78
5.2 Impact of the *Grace Period* parameter 79
5.3 Impact of the *Tie Threshold* parameter 79
5.4 Impact of the *Split Criteria* parameter 80
5.5 Impact of the *Leaf Prediction* parameter 81
5.6 Single training configuration 83
5.7 *Interleaved Chunk* evaluation with default configuration 84
5.8 Accuracy by chunk size 85
5.9 Cost by chunk size 86
5.10 Accumulated cost by chunk size 86
5.11 *Interleaved Chunk* comparison with [8] configuration 87
5.12 *Interleaved Chunk* evaluation with CESCA dataset 88

7.1 Blocking scenario behavior. Both sides send data, but since there is no acknowledgment from the other side they try to retransmit (after a time which grows in every attempt) until finally break the connection with a RST 110
7.2 Termination proportions `ISP_Core`, green stands for the standard FIN process, yellow for the unclosed flows, red for the RST and intense red for the RST in a blocking scenario 111
7.3 CDF of inter-packet times with RST termination in a blocking scenario 114
7.4 Timeout PACE evaluation `ISP_Core` 116
7.5 Timeout PACE performance `ISP_Core`. Elements stand for flows . . 117
7.6 Diagram of the Continuous Training Traffic Classification system based on TIE 123
7.7 K-dimensional tree evaluation without the support of the *relevant* ports 126
7.8 K-dimensional tree evaluation with the support of the *relevant* ports 127
7.9 K-dimensional tree evaluation with the support of the *relevant* ports 128

7.10 Accuracy by application group (seven packet sizes and *selected* list of ports as parameters) 128

List of Tables

2.1 IANA assigned port numbers for several well-known applications . . 10
2.2 Strings at the beginning of the payload of P2P protocols defined by Karagiannis . . . 11
2.3 Characteristics of the traffic traces in the UPC dataset . . . 18
2.4 CESCA dataset traffic mix . . . 20
2.5 Top 10 Applications by Flow in the MAWI Dataset . . . 21
2.6 Application classes in the dataset . . . 22

3.1 Set of 10 NetFlow-based features used in this work . . . 26
3.2 Application groups used by L7-filter . . . 28
3.3 Elephant flow distribution in the UPC dataset . . . 29
3.4 Overall accuracy of our traffic classification method (C4.5) and a port-based technique for the traces in our dataset . . . 30
3.5 Average of the relative error of the flow features as a function of p (UPC-II trace) . . . 39
3.6 Characteristics of the traces from other environments . . . 47

4.1 Features used by each classification technique . . . 57
4.2 Application groups and traffic mix . . . 58
4.3 DPI techniques consumption . . . 62
4.4 Long-Term Policy Evaluation . . . 64
4.5 Training Dataset Evaluation . . . 64

5.1 HAT parametrization . . . 82

6.1 DPI-based techniques evaluated . . . 92
6.2 DPI evaluation . . . 98
6.3 HTTP sub-classification by *NDPI* . . . 99
6.4 FLASH evaluation . . . 99

6.5 Summary 100

7.1 Traces properties 105
7.2 Flow usage 106
7.3 Flow proportions 107
7.4 PCK-PCK times TCP `ISP_Core` (ms) 109
7.5 PCK-PCK times TCP `ISP_Mob` (ms) 110
7.6 PCK-PCK times UDP `ISP_Core` (ms) 111
7.7 DAT-DAT times TCP `ISP_Core` (ms) 112
7.8 RST times `ISP_Core` (ms) 113
7.9 Timeout TCP (ms) 115
7.10 Timeout UDP (ms) 116
7.11 Characteristics of the traffic traces in our dataset 124
7.12 Speed Comparison (flows/s) : Nearest Neighbor vs K-Dimensional Tree 125
7.13 Memory Comparison: Nearest Neighbor vs K-Dimensional Tree . . 125
7.14 Building Time Comparative: Nearest Neighbor vs K-Dimensional Tree 126
7.15 Evaluation of the Continuous Training system by training trace and set of *relevant* ports 129

Chapter 1

Introduction

The Internet is becoming central in our life and work. From chatting in Facebook to discovering the cure for Cancer, almost every aspect of our life is somehow related to the Internet. Long gone are the days when the original ARPANET was born. Since then, the network has been in constant evolution transforming the initial ARPANET in what we today know as the Internet, an enormous conglomerate of interconnected computer networks. Despite its important role in our life, the knowledge about its operation is far from being completely understood. The continuous introduction of new network architectures, protocols and applications during the last decades resulted in an ever evolving entity difficult to study and understand. This has spurred the research community to better analyze the network traffic and bring some light about the complex operation of the Internet. Particularly, a new field of study, usually referred to as *traffic classification*, has become crucial for understanding the Internet. The classification of network traffic not only satisfies our curiosity, but it also has many important applications for network operators and IT administrators. BitTorrent, Skype (i.e., P2P), Youtube, Netflix (i.e., streaming) or Megaupload (i.e., direct download) are some examples of network applications that at some point completely changed the paradigms of the Internet. The classification of network traffic helps in many different manners. For instance, studying how new applications impact on the network can help to better plan new infrastructures, architectures or protocols. An accurate classification can also help Internet Service Providers (ISP) to apply reliable techniques to apply Quality of Service policies based on the needs of the applications (e.g., VoIP calls). Finally, this opens a new range of billing possibilities for ISPs to take profit of their infrastructures based on their actual usages.

The desire of network operators would be to be able to accurately classify all

the traffic of their networks online. However, the continuous evolution of Internet applications and their techniques to avoid being detected make their identification a very challenging task. Thus, the research community has thrown itself into the search of techniques to accurately identify and classify the traffic. Nevertheless, a wide range of unaddressed functional problems arise when those techniques are applied in real scenarios dealing with tremendous amount of traffic and limited resources. In this thesis we address the network traffic classification problem from a more practical point of view, emphasizing those problems that arise when this classification is performed online in operational networks.

The rest of this chapter discusses the motivations and challenges behind this dissertation and also provides an overview of the thesis and its main contributions. Finally, it describes the structure of this manuscript.

1.1 Motivations

The network traffic classification problem can be seen as a never ending race between application developers, on the one side, and the network operators and the research community, on the other. Originally, the most common and simplest technique to identify network applications was based on the port numbers (e.g., those registered by the IANA [1]). This solution was very efficient and relatively accurate with traditional applications. The arrival of bandwidth eater P2P-based applications (e.g., Napster, eMule) was the first turning point. Network operators usually reacted to the high consume of those applications directly blocking their default used ports. To avoid being blocked, application developers and users replied by using dynamic ports or even using registered ones from traditional applications, making the solution based on the well-known ports completely unreliable. Nowadays it is widely accepted that this method is no longer valid due to the inaccuracy and incompleteness of its classification results [2–5].

The response from the research community was the introduction of Deep Packet Inspection (DPI) techniques for the classification of those applications [3–10]. DPI methods are based on searching for characteristic signatures (or patterns) in the packet payloads. This solution is potentially very accurate and it is commonly used as ground-truth generator for the validation of other techniques. However, its application online is extremely expensive. Once again, the applications reacted by implementing protocol encryption (i.e., cipher the payload) to evade classification.

Machine learning techniques (ML) were later proposed as a promising solution to the well-known limitations of port- and DPI-based techniques [11–35]. Nguyen

et al. survey and compare the complete literature in the field of ML-based traffic classification in [36]. In general terms, most ML methods study in an offline phase the relation between a pre-defined set of traffic features (e.g., *port numbers*, *flow sizes*, *inter-arrival times*) and each application. This set of features is used to build a model, which is later used to identify the network traffic online. In order to partially decrease the precision of ML techniques, network applications implemented protocol obfuscation methods that randomly modify the characteristics of their traffic.

In parallel with ML and DPI techniques, other less prolific approaches were also proposed. An interesting alternative following a different approach are the host-behavior-based methods [37–39]. BLINC [37] is arguably the most well-known exponent of this alternative branch. These methods base the classification on the behavior of the end-hosts that produced the traffic. Another solution similar to the well-known ports are the IP-based techniques [40, 41]. This approach uses the knowledge extracted from previously seen IP addresses from well-known applications (e.g., Youtube) to classify the traffic related to those applications.

Although most of the existing solutions can achieve high accuracy in theory, there is no universal method suitable for every possible network scenario. In addition, their deployment in production networks, with a large number of users and connections, presents practical constraints that existing methods do not completely address. As a result, currently existing methods have reached limited success among network operators and managers. From our point of view, three main aspects should be addressed in order to introduce new techniques that are feasible and reliable for network traffic classification in production networks, which are the basis of this thesis.

The first issue is related to the deployment of the classification solutions. The limited resources and high throughput requirements of production networks usually do not allow the extra burden imposed by existing techniques. For example, DPI techniques usually require access to the payload of each packet to search for the set of patterns. This operation needs expensive hardware in order to cope with the high data rates of nowadays networks. In contrast, ML methods need to extract for each flow (i.e., connection) the set of features used as input for the classification. Although less demanding, some features are very costly to extract and require specific hardware to compute them. In addition, network operators usually apply packet sampling in their monitoring solutions in order not to compromise the correct operation of the network in highly consuming situations (e.g., attacks). However, the impact of packet sampling on the proposed classification techniques remains unknown.

The second issue is the high maintenance costs involved in current classification solutions. For instance, most ML techniques [35, 36, 42] rely on a costly training phase that requires human intervention. As shown by Li et al. in [43], these techniques usually need regular updates in order to adapt to new traffic or new networks. This not only implies the involvement of the network operator, but also a specific knowledge for carrying out the task. Similarly to ML techniques, DPI and IP-based techniques require regular updates of the signature and IP base used for the classification.

And finally, the third issue is related to the impossibility of validating and comparing the different proposals. Most traffic classification solutions proposed in the literature report very high accuracy. However, most solutions base their results on a private ground-truth (i.e., dataset), usually labeled by techniques of unknown reliability (e.g., ports-based or DPI-based techniques [42, 44–46]). That makes the comparison and validation of different proposals very difficult in order to decide what solution is more suitable for each scenario. The use of private datasets is derived from the lack of publicly available datasets with payload. Mainly because of privacy issues, researchers are not allowed to share their datasets with the research community. Another crucial problem is the reliability of the techniques used to set the ground-truth. Most papers show that researchers usually obtain their ground-truth through port-based or DPI-based techniques [42, 44–46]. The poor reliability of port-based techniques is already well known [4, 47]. According to conventional wisdom they are, in principle, one of the most accurate techniques. However, the reliability of DPI-based techniques is still unknown.

These three aspects hinder the success of currently existing techniques in production networks. Therefore, network operators usually rely on obsolete solutions to classify their network traffic or are just able to classify a small portion of their network traffic due to the expensive hardware, and complex deployment and maintenance costs of state-of-the-art classification solutions.

1.2 Contributions and Impact

This thesis is about filling the gap between the real-world requirements from the network industry, and the research being carried out in the network traffic classification field. The contributions of this thesis follow the same fronts of the motivations. Most of the techniques for network traffic classification proposed in the state of the art reported high accuracy, however those works usually lack to address the practical issues related to the introduction of those systems in real

environments.

Ease of deployment is a crucial element of a realistic network traffic classification solution. We address this problem in Chapter 3. As described, most of the proposed solutions in the literature need additional (usually expensive) hardware in order to perform the classification. To address this issue this thesis focuses on the use of NetFlow as input for the classification. NetFlow is a widely extended protocol developed by Cisco to export IP flow information from routers and switches [48]. The main challenge when using NetFlow is the limited amount of information available, which complicates the classification [14, 42], but significantly reduces the cost of the solution and allows its rapid deployment in production networks given that most network devices already support NetFlow or one of its variants (e.g., J-Flow, IPFIX). However, due to the limited resources in core networks, NetFlow is usually operated under packet sampling. Chapter 3 studies the impact of Sampled NetFlow on the classification. Using the C4.5 decision tree ML technique we study empirically and theoretically how packet sampling affects the classification with NetFlow. In particular, we identify three different sources of error when using packet sampling in NetFlow: (*i*) error in the estimation of the flow features, (*ii*) changes in the flow size distribution, and (*iii*) splitting of sparse flows. Afterwards, we propose a simple but effective methodology to improve the classification accuracy of ML-based techniques in this scenario. As a result, our ML-based classification solution is able to accurately classify the traffic only using as input the information provided by Sampled NetFlow. This considerably facilitates the deployment of our proposal given that the input can be easily provided by the current network infrastructure without compromising its correct operation.

The maintenance is the other practical problem barely addressed by the state-of-the-art proposals. As already pointed out, the continuous evolution of the Internet traffic forces classification solutions to be regularly updated to adapt to traffic changes. Usually, proposed solutions avoid this limitation and only focus on obtaining a high classification accuracy with a static dataset. This forces network operators to intuitively decide how often it is necessary to manually perform the tedious update of each instance of the classification solution. Unlike the rest of the literature, Chapter 4 focuses on this issue with the goal of automatically maintaining the classification solution without human intervention while keeping high classification accuracy. With the objective of proposing a realistic solution we present a technique that combines different classification methods only using NetFlow as input. Furthermore, we design an autonomic retraining system that is able to keep the classifier accurate along time. All these features (i.e., high accuracy, ease of deployment and maintenance) make the proposed system a realistic

solution for network traffic classification in backbone networks.

Chapter 5 introduces the use of stream-based ML techniques for network traffic classification. These techniques, scarcely used in this field, have very appealing features for network traffic classification: (*i*) process an instance at a time and inspects it only once, (*ii*) use a predefined amount of memory, (*iii*) work in a bounded amount of time and (*iv*) are ready to predict at any time. We propose a technique based on Hoeffding Adaptive Trees, a stream-based decision tree that can be as accurate as batch techniques like the already mentioned C4.5. To extract sound conclusions we evaluated this technique with the MAWI dataset [49], a 13 years long dataset from a transatlantic link in Japan. Our technique is able to automatically adapt to the changes of the traffic with just a small sample of labeled data, making our solution very easy to maintain. Also, following previous recommendations, we are able to accurately classify the traffic using only NetFlow data making our solution very easy to deploy.

The literature has mainly focused on proposing techniques to achieve high accuracy while we, in addition, focused on making them feasible in real scenarios. However, an essential element that remained unaddressed also hindered the adoption of new techniques, which is the validation and comparison of proposed techniques. The wide range of techniques, datasets, ground-truth generators make the comparison and validation of new proposed techniques a very difficult task. Chapter 6 sheds light to this problem by evaluating the reliability of six well-known DPI-based techniques usually used for ground-truth generation. We created a labeled dataset and used Volunteer-Based System (VBS) [50] to guarantee the accurate labeling process. The obtained results help researchers to understand the reliability and implications of using the different DPI-based techniques (i.e., PACE, OpenDPI, nDPI, LibProtoIdent, L7-filter and NBAR) for ground-truth generation.

In addition, as part of this research, we published some of the datasets used in our evaluations in an attempt to allow for the fair comparison and validation of state-of-the-art techniques. The dataset used in Chapter 3 anonymized and labeled with L7-filter has been shared with multiple institutions as can be seen in the Appendix A. The dataset created in Chapter 6 had special impact, because it is the first reliable labeled dataset with full packet payloads available to the research community. To the best of our knowledge, this is the only dataset available that makes it possible the comparison and validation of ML and DPI based techniques.

Although these are the most important contributions of this thesis, other improvements related to the network classification field have been also achieved. Almost every proposed solution for network traffic classification needs to keep track

of the connections (i.e., flows) to properly classify the traffic. Usually it is necessary to store information of every packet to extract the features or payload later used in the classification. This process is of capital importance in a scenario with limited resources like core networks. As a result, a proper mechanism for expiration of the flows is a key point in the optimization of this costly process. Section 7.1 presents the study of an efficient profiled termination timeout to optimize this operation. First we carefully study the flow termination by group of applications with current Internet traffic. From that characterization we went further and proposed an expiration technique to achieve optimized timeouts in a profiled way.

Finally, the last contribution of this thesis proposes an early classification technique able to classify the traffic just using the size of the first packets of the flows. We revisited in Section 7.2 the viability of using the Nearest Neighbor algorithm (NN) for online traffic classification, which has been often discarded in previous studies due to its poor classification speed. In order to address this well-known limitation, we presented an efficient implementation of the NN algorithm based on a K-dimensional tree data structure. Our results show that our method can achieve very high accuracy by looking only at the first packet of a flow. When the number of analyzed packets is increased to seven, the accuracy of our method increases beyond 95%. This early classification feature is very important, since it allows network operators to quickly react to the classification results.

1.3 Thesis Organization

This thesis dissertation is organized as follows. Chapter 2 presents important background related to the network traffic classification field and describes in detail the datasets used throughout this manuscript. Chapter 3 first studies the impact of using NetFlow and packet sampling for network traffic classification both theoretically and empirically. Afterwards, it presents a solution to improve the accuracy of ML-based techniques using Sampled NetFlow. This work that aims to facilitate the deployment of state-of-the-art techniques is based on the journal paper published in [42]. Next, Chapter 4 proposes a complete and feasible solution for network traffic classification in backbone networks. Using the previous knowledge and combining different literature techniques, the proposed system is able to sustain a high classification accuracy along time without human intervention and just using Sampled NetFlow for the classification. The solution possesses the main requirements for an appealing network classification system (i.e., easy to deploy and maintain, high classification accuracy and completeness). Its design, opera-

tion and evaluation have been published in [51]. Chapter 5 introduces the use of stream-based ML techniques for network traffic classification, proposing a solution based on Hoeffding Adaptive Trees. The nature of the network scenarios makes this technique the perfect candidate for a continuous traffic classification solution. Chapter 6, based on the work published in [52], presents a first step to address the validation problem of the network traffic classification field. To achieve this goal we created and published a reliable labeled dataset and compared the accuracy of several well-known DPI-based techniques used in the literature for ground-truth generation. Chapter 7 presents two additional contributions in the network traffic classification field. First, we propose a methodology based on profiled termination timeouts to efficiently expire flows in network monitoring solutions. Afterwards, a technique based on K-dimensional trees is proposed for early network traffic classification. These works are based on the publications in [53, 54]. Finally, Chapter 8 concludes this thesis and presents some ideas for future work.

Chapter 2

Background

This chapter presents some important aspects related to network traffic classification, which are necessary to understand the contributions of this thesis. Also, it describes in detail the datasets used throughout this manuscript.

2.1 Network Traffic Classification Approaches

This section describes the state-of-the-art approaches in the field of network traffic classification. The application identification problem has been changing due the efforts of two opponents that are in a continuous competition. On the one hand, the applications, and especially those that do not want to be detected (e.g., P2P applications), in order to use the network resources without control. On the other hand, network operators, researchers and even ISPs who need to know the traffic characteristics of their networks to manage the resources or even charge the users based on their consumption. This rivalry yielded the creation of several techniques for network traffic classification.

2.1.1 Port-based approach

As mentioned in Section 1.1, traditionally, the classification of network applications has been carried out using the well-known ports technique. At the beginning, the initial network applications registered their ports in the Internet Assigned Numbers Authority (IANA) [1]. The solution of the well-known ports identifies the traffic according to the ports registered in the IANA. Table 2.1 shows an example of several ports assigned by the IANA with the corresponding application. For

example, web applications use port 80 and email applications use port 25 (SMTP) to send emails and port 110 (POP3) to receive them.

Assigned Port	Application
20	FTP Data
21	FTP Control
22	SSH
23	Telnet
25	SMTP
53	DNS
80	HTTP
110	POP3
123	NTP
161	SNMP
3724	WoW

Table 2.1: IANA assigned port numbers for several well-known applications

This method is no longer valid because of the inaccuracy and incompleteness of its classification results [2, 3, 5]. It is difficult to set bounds on the current accuracy of this method because it mainly depends on the characteristics of the network being monitored and the system to establish the ground-truth used to validate them. There are some studies where this technique achieves an accuracy of 50%-70% [5, 15], while in others, like in the present work, the accuracy is less than 20%. However, the complete literature agrees that the well-known ports technique is not able to classify the new generation of applications that are the ones that consume more bandwidth. This new type of applications, especially P2P applications, use different strategies to camouflage their traffic in order to evade detection. Additionally, they use dynamic ports in their connections or ports from other well-known applications (e.g., 80 typically used by web applications).

2.1.2 Payload-based approach

The first alternative to the well-known ports method was the inspection of the packet payloads to identify the network traffic [2–7, 9, 10, 46, 55]. These methods, usually called deep packet inspection techniques, examine the content of the packets looking for characteristic signatures of the applications in the traffic. The work done under this approach aims to identify specially P2P applications because they

are the most assiduous to use camouflage strategies and evade the detection of the well-known ports technique. Table 2.2 presents an example of patterns used by Karagiannis et al. in [4].

P2P Protocol	String	Trans. Prot.
eDonkey 2000	0xe319010000	TCP/UDP
	0xe53f010000	
Fasttrack	"Get /.hash"	TCP
	0x2700000002980	UDP
BitTorrent	"0x13Bit"	TCP
Gnutella	"GNUT" "GIV"	TCP
	"GND"	UDP
Ares	"GET hash:"	TCP
	"Get sha1:"	

Table 2.2: Strings at the beginning of the payload of P2P protocols defined by Karagiannis

Also, there are some papers [2–4] that presented hybrid methods using the well-known ports method to identify the traditional network applications and a deep packet inspection to classify the new ones. Although solutions based on pattern matching could achieve high accuracy, they had some problems with three main factors. First, pattern searching in the payload of every packet produces a high consume of resources that needs extremely expensive dedicated hardware to handle nowadays networks. Second, it does not work with encrypted traffic, an increasing trend among the P2P applications. And third, a continuous update of the set of application signatures is necessary to classify new versions and applications. Although there are some proposals that tried to facilitate this process [13], the generation of new signatures is still a tedious and challenging task that requires human supervision. Given the high resource requirements for the pattern searching, the limitations with encrypted traffic and their continuous update requirement made their application impractical in current high-speed networks. However, this family of techniques is still the used solution to establish the ground-truth [7, 9, 10, 46].

2.1.3 Flow features-based approach

In order to overcome the limitations of the payload methods and find an alternative to identify applications without the necessity of packet inspection the machine

learning (ML) techniques were introduced. These methods detect, in an offline phase, characteristic patterns of the different applications based on a set of features. More in detail, machine learning methods use a (usually labeled) dataset from which a set of features is extracted. This information serves as input of the ML technique that extracts and outputs the knowledge in different structures (e.g., decision tree, rules, clusters) depending on the ML technique used. The structure obtained is later used to classify unlabeled instances assuming that the features of the still unknown instances will have the same behavior as the known one.

The wide related work in this field can be structured in two main areas: the supervised and unsupervised learning approaches. Next subsections provide brief descriptions of these two areas. However, Nguyen et al. surveyed the complete literature of the ML techniques in the field of application identification in [36] comparing in detail all the different approaches.

Supervised Learning Approach

Supervised methods, also known as classification methods, extract knowledge structures to classify new instances in pre-defined classes. It is important to note that is called supervised because the output classes are pre-defined. The process of a supervised learning methods start with a training dataset TS defined as, $TS = < x_1, y_1 >, < x_2, y_1 >, ..., < x_N, y_M >$, where x_i is the vector of values of the features corresponding to the i^{th} instance, and y_i is its output class value. It discovers the different relations between the instances and outputs a structure, usually a decision tree or classification rules, that will classify the instances in a discrete set $y_1, y_2, ..., y_M$.

There is a lot of related work that use supervised techniques [11–18] with a promising results.

Unsupervised Learning Approach

Unlike the supervised learning methods, unsupervised methods, also known as clustering methods, do not need a complete labeled dataset. Therefore, the output of the ML training does not classify instances in predefined classes. It is the own method that discovers the natural clusters (groups) in the data. Similar to the supervised methods, the clustering methods have been deeply studied [19–29]

Basically, three main clustering methods have been used in the traffic classification literature: the classic k-means algorithm that forms clusters in numeric domains, partitioning instances into disjoint clusters; the incremental clustering that

generates a hierarchical grouping of instances and; the probability-based method that assigns instances to classes probabilistically, not deterministically. In general, they achieved a slightly lower accuracy than supervised techniques, but with a much more lightweight training phase.

It is important to note that usually the clustering methods have a tradeoff between the number of clusters of the output and the final accuracy. Having more clusters you could classify better but the training and classification time of the method will also increase.

Given its appealing characteristics the ML-based traffic classification techniques are the most prolific solution in the literature. However, similarly to pattern matching techniques, ML-based methods should be periodically updated to adapt the classification model to new applications. This process is difficult and usually requires human supervision. Furthermore, the extraction of the set of features used for the classification can require dedicated hardware to handle nowadays networks.

2.1.4 Host-behavior-based approach

Another branch that appears to solve the limitations of the payload-based methods is the host-behavior-based techniques. Karagiannis et al. with BLINC [37,39] and Xu et al. in [38] developed a classification approach based on the behavior of the hosts. The Karagiannis' approach studies the behavior of the hosts at three levels:

- At social level, they value the popularity of a host depending on the number of communications it has with different hosts.
- At functional level, they value the role of the host in the network finding out if the host is a provider or consumer of a service, or it participates in collaborative communications. For example, a host which has a lot of connections at the same port is likely to be provider of a service in that port.
- At application level, they capture the transport layer interactions between hosts trying to identify the application of origin. For example, a host with a lot of connections from the same source port and IP to a unique destination IP but different ports would be the behavior of a port scan attack.

According with these three metrics and some refining heuristics BLINC classifies the behavior of the applications. BLINC classifies approximately 80%-90% of the total number of flows with 95% accuracy. However, BLINC presents some limitations regarding the necessity to work with traces collected in the edge and

with bidirectional flows. Furthermore, BLINC classifies the applications based on behavior groups but it is not able to identify exactly the applications. It could suppose a problem with applications that are theoretically from different groups but with similar behavior (e.g., VOIP and P2P).

2.1.5 IP-based approach

IP addresses can carry important information related to the traffic they produce. Following the same approach as the port-based techniques, these methods use the information provided by IP addresses to classify the traffic. Two main solutions to use the IP information have been proposed.

The first technique is based on the solely use of well-known IP addresses to classify the traffic [40]. Basically, this technique tracks down in an offline phase the IP addresses belonging to famous web sites (e.g., Google, Twitter). For instance, the IP 31.13.83.16 belongs to Facebook, so the traffic coming or going to that IP is directly classified as Facebook.

The second technique is also known as Service-based method [41]. A service is defined as the triplet <IP, Port, Protocol> assigned to a specific application. The list of services is also created in an offline phase using a dataset of labeled flows. For example, Yoon et al. in [41] aggregate all the available flows by their triplet and then, a service is created when a triplet has a minimum number of flows during a specific time threshold and there is a predominant label for all these flows.

These techniques are very accurate, however its completeness has been significantly degraded given the migration of some applications to Content Delivery Networks. Furthermore, and similarly to other approaches, periodically updates should be applied in order to maintain the techniques accurate.

2.1.6 Comparing Approaches

So far, we presented a spread work in the field of techniques to identify applications. However, it is not clear which one is the best method to identify applications in network traffic. Some researchers addressed this problem in several papers comparing different techniques [30–35].

In 2006, Williams et al. made in [32] perform evaluation between seven supervised algorithms. The used algorithms were C4.5 Decision Tree, Naive Bayes, Nearest Neighbor, Naive Bayes Tree, Multilayer Perception Network, Sequential Minimal Optimization and Bayesian Networks. They concluded that C4.5 is the most accurate supervised technique with and accuracy of 99,4%, followed by Bayes

Network 99,32% and Naive Bayes Tree and Naive Bayes with 98,3%. Although the accuracy difference is minimum, Williams et al. show significantly differences regarding the classification time, where C4.5 outperformed the rest of methods being twice faster than Naive Bayes and more than five times faster than Bayes Network.

Also in 2006, Erman et al. presented the study of the comparison of the supervised method Naive Bayes and the unsupervised method AutoClass. They concluded that the unsupervised method achieved a 91% accuracy, 9% more than the supervised method.

At the end of 2008, Kim et al. presented in [35] a comparison of different supervised methods, BLINC and the well-known port method implemented by CoralReef [56]. Comparing the different approach the supervised machine learning techniques achieved better accuracy than BLINC and CoralReef. Among the different supervised methods (i.e.,Naive Bayes, Naive Bayes Kernel Estimation, Bayesian Network, C4.5, k-NN, Neural Networks and Support Vector Machine (SVM)) the paper concluded that SVM outperformed the rest of supervised methods with more than 98% overall accuracy while C4.5, in the same scenario, achieved an overall accuracy of ≈94%. However, C4.5 classified the instances a hundred times faster than SVM.

We can conclude that according to the literature there is not a clear technique that outperforms the rest of techniques. However, it seems that all the methods could achieve a very good accuracy in specific scenarios.

2.2 Network Traffic Classification Evaluation

This section describes some important aspects related to the evaluation of the results in the network traffic classification field. Two main elements are common in all the evaluations of network traffic classification solutions. The first element are the metrics used in order to measure the quality of the proposals. The second element is the ground-truth used as reference in order to validate and compute the metrics.

2.2.1 Performance Metrics

Depending on the requirements this thesis uses several metrics in order to asses the quality of the experiments. The most extended metric used in the network traffic literature is the *accuracy*. The *accuracy* is computed by calculating the number

of correctly classified flows. The exact definition of the *accuracy* metric in a non binary classification would be:

$$Accuracy = \frac{\sum_{i=1}^{N}(TP)}{\sum_{i=1}^{N}(TP) + \sum_{i=1}^{N}(FP)}$$

where:

- N: number of categories (e.g., groups of applications).
- TP (True Positives): The number of correctly identified flows for a specific category.
- FP (False Positives): The number of falsely identified flows for a specific category.

Other metrics usually computed are the *precision* (i.e., positive predictive value) and the *recall* (i.e., sensitivity) by category. The *precision* is defined as $\frac{TP}{TP+FP}$. This metric indicates the fraction of instances from a particular category that are correctly classified from the total amount of instances classified as that category.

On the other hand, the *recall* of a particular category is defined as $\frac{TP}{TP+FN}$, where FN (False Negatives) stands for those flows of the category under study that are classified as belonging to another category. This metric indicates the fraction of instances from a particular category that are correctly classified from the total amount of instances of that category.

Although these metrics are the most popular metrics used in the network traffic classification literature they have some limitations. Because of this, sometimes it is necessary to compute other metrics to confirm the results obtained. The *Kappa coefficient* metric is considered to be more robust because it takes into account the correct classifications occurring by chance. We computed the *Kappa coefficient* as explained by Cohen in [57]:

$$k = \frac{Po - Pe}{1 - Pe}$$

being:

$$Po = \frac{\sum_{i=1}^{N}(TP)}{\sum_{i=1}^{N}(TP) + \sum_{i=1}^{N}(FP)}$$

$$Pe = \sum_{i=1}^{N}(P_{i1} \times P_{i2})$$

where:

- P_{i1}: proportion of apparition of the category i for the observer 1.
- P_{i2}: proportion of apparition of the category i for the observer 2.

The *Kappa coefficient* takes values close to 0 if the classification is mainly due to chance agreement. On the other hand, if the classification is due the discriminative power of the classification technique then the values are close to 1.

Another important metric is the *completeness*. This metric measures the amount of traffic not classified of the dataset. That is, the percentage of traffic that remains *unknown* after the classification.

2.2.2 Ground-truth generation

The establishment of the ground-truth is one of the most critical phases of any network traffic classification evaluation since the entire classification process relies on the reliability of the technique used to obtain the first labeling used as reference. Three main approaches are used in the literature.

The most accurate technique could be the human manual labeling of the dataset. However, the manual labeling flow by flow is a very tedious task making it impracticable with the current datasets containing millions of flows.

Most papers show that researchers usually obtain their ground-truth through port-based or DPI-based techniques [42, 44–46]. However, the poor reliability of port-based techniques is already well known, given the use of dynamic ports or well-known ports of other applications [4, 47]. Because of this the use of port-based techniques is no longer recommended for the generation of the ground-truth.

The DPI-based techniques are currently the most used solutions for ground-truth generation. Some examples, widely used in the literature, are L7-filter [7], OpenDPI [8], PACE [9], nDPI [10] and Libprotoident [46]. These techniques usually combine pattern matching and heuristics to, according to conventional wisdom, accurately label the traffic. Nevertheless, the reliability of DPI-based techniques is still unknown.

2.3 Datasets

This chapter describes the datasets used in the different works of this thesis. Most of them have been published in order to allow further comparison and validation of our proposals.

Table 2.3: Characteristics of the traffic traces in the UPC dataset

Name	Date	Day	Start Time	Duration	Packets	Bytes	Avg. Util
UPC-I	11-12-08	Thu	10:00	15 min	95 M	53 G	482 Mbps
UPC-II	11-12-08	Thu	12:00	15 min	114 M	63 G	573 Mbps
UPC-III	12-12-08	Fri	16:00	15 min	102 M	55 G	500 Mbps
UPC-IV	12-12-08	Fri	18:30	15 min	90 M	48 G	436 Mbps
UPC-V	21-12-08	Sun	16:00	1 h	167 M	123 G	279 Mbps
UPC-VI	22-12-08	Mon	12:30	1 h	345 M	256 G	582 Mbps
UPC-VII	10-03-09	Tue	03:00	1 h	114 M	78 G	177 Mbps

2.3.1 UPC Dataset

The UPC dataset consists of seven full-payload traces collected at the Gigabit access link of the Universitat Politècnica de Catalunya (UPC), which connects about 25 faculties and 40 departments (geographically distributed in 10 campuses) to the Internet through the Spanish Research and Education network (RedIRIS). This link provides Internet access to about 50000 users. The traces were collected at different days and hours trying to cover as much diverse traffic from different applications as possible.

Table 2.3 presents the details of the traces of this dataset.

To set the ground-truth of the UPC dataset we used L7-filter [7]. This DPI-based technique presents overmatching problems with some of its patterns. In order to reduce the inaccuracy of L7-filter we used 3 rules for the labeling:

- We apply the patterns in a priority order depending on the degree of overmatching of each pattern (e.g., *skype* patterns are in the latest positions of the rule list).

- We do not label those packets that do not agree with the rules given by pattern creators (e.g., packets detected as *NTP* with a size different than 48 bytes are not labeled).

- In the case of multiple matches, we label the flow with the application with more priority, based on the quality of each pattern reported in the L7-filter documentation. If the quality of the patterns is equal, the label with more occurrences is chosen.

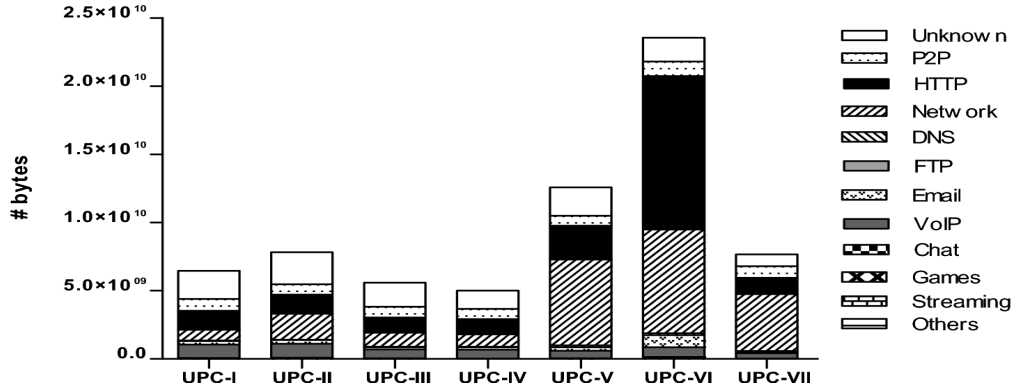


Figure 2.1: Traffic breakdown of the traces in the UPC dataset

As already proposed in previous works (e.g., [19]), we also perform a sanitization process in order to remove incorrect or incomplete flows that may confuse or bias the dataset. The sanitization process removes those TCP flows that are not properly formed (e.g., without TCP establishment or termination, and flows with packet loss or with out-of-order packets) from the training set. However, no sanitization process is applied to UDP traffic.

Figure 2.1 plots the application breakdown according to L7-filter of the traces in the UPC dataset. The breakdown reflects the academic nature of the monitored environment, with a large portion of *HTTP* and *network* traffic. We can also observe that L7-filter is not able to classify about 20% of the traffic (labeled as *unknown*). This was a common problem of the tools available to establish the ground-truth, as already reported in previous works (e.g., [58]).

The set of seven labeled traces presented in Table 2.3 is publicly available in an anonymized form to the research community at [59].

2.3.2 CESCA Dataset

The CESCA trace is a fourteen-days packet trace collected on February 2011 in the 10-Gigabit access link of the Anella Científica, which connects the Catalan Research and Education Network with the Spanish Research and Education Network. This link provides connection to Internet to more than 90 institutions. This trace was collected with 1/400 flow sampling. We applied this ratio because it was the lowest that allowed us to collect the trace without packet loss in our hardware. To obtain a reliable ground-truth, we use a set of DPI techniques, including PACE, a commercial DPI library provided by ipoque [9]. PACE is known to have high

Table 2.4: CESCA dataset traffic mix

Group	Applications	# Flows
web	*HTTP*	17 198 845
dd	*E.g., Megaupload, MediaFire*	40 239
multimedia	*E.g., Flash, Spotify, Sopcast*	1 126 742
p2p	*E.g., Bittorrent, Edonkey*	4 851 103
mail	*E.g., IMAP, POP3*	753 075
bulk	*E.g., FTP, AFTP*	27 265
voip	*E.g., Skype, Viber*	3 385 206
dns	*DNS*	15 863 799
chat	*E.g., Jabber, MSN Messenger*	196 731
games	*E.g., Steam, WoW*	14 437
encryption	*E.g., SSL, OpenVPN*	3 440 667
others	*E.g., Citrix, VNC*	2 437 664

accuracy with low false positive ratio. Moreover, to increase the completeness of the DPI classification we added two extra libraries, OpenDPI [8] and L7-filter [7]. Table 2.4 describes the final traffic distribution of CESCA dataset.

2.3.3 MAWI Dataset

The publicly available MAWI dataset [49] has unique characteristics to study stream oriented techniques for network traffic classification. The MAWI dataset consists of 15-minutes traces daily collected in a transit link since 2001 (i.e., 13 years). Although it is a static dataset, its long duration and amount of data makes it the perfect candidate for stream-based evaluations. Furthermore, its duration allows the study of the ability of classification techniques to adapt to the evolution of the traffic.

To set the ground-truth of the MAWI dataset we used a DPI technique. The packets in this dataset are truncated after 96-bytes, which considerably limits the amount of information available for the DPI techniques. Because of this constraint we rely our ground-truth labeling on Libprotoident [46]. The most important feature of Libprotoident is that its patterns are found just in the first 4 bytes of payload of each direction of the traffic. Unexpectedly, that data is enough to achieve very high accuracy classification as shown in [46, 52]. However, the MAWI dataset is characterized to have asymmetric traffic that can reduce the effectiveness of the Libprotoident. We performed a sanitization process and focused on the TCP and UDP traffic from the MAWI dataset. Table 2.5 presents the top ten

applications by flow along the thirteen years once the sanitization is applied.

After the labeling and the sanitization process, the MAWI dataset consists of almost 4 billions of unidirectional labeled flows. To the best of our knowledge this is the first work in the network traffic classification field that deals with this large amount of data, which is necessary to extract sound conclusions from stream-based evaluations.

Table 2.5: Top 10 Applications by Flow in the MAWI Dataset

Year	Top 1	Top 2	Top 3	Top 4	Top 5
2001	HTTP (49.44%)	DNS (42.11%)	DEMONWARE (3.27%)	SMTP (2.37%)	FTP (0.52%)
2002	HTTP (41.30%)	DNS (37.75%)	OPASERV (11.81%)	DEMONWARE (4.16%)	SMTP (1.79%)
2003	HTTP (30.22%)	DNS (22.55%)	OPASERV (22.46%)	SQL EXPLOIT (19.47%)	SMTP (1.87%)
2004	HTTP (38.77%)	DNS (26.45%)	SQL EXPLOIT (12.11%)	OPASERV (10.46%)	SMTP (3.40%)
2005	HTTP (31.02%)	DNS (30.80%)	SQL EXPLOIT (13.85%)	SKYPE (8.09%)	MSN (3.91%)
2006	DNS (33.34%)	HTTP (31.51%)	SQL EXPLOIT (11.43%)	SKYPE (6.28%)	BITTORRENT (4.39%)
2007	DNS (50.42%)	HTTP (31.61%)	BITTORRENT (3.82%)	SKYPE (3.37%)	SMTP (2.81%)
2008	DNS (50.82%)	HTTP (26.52%)	BITTORRENT (5.27%)	SKYPE (4.13%)	SQL EXPLOIT (3.86%)
2009	DNS (44.31%)	HTTP (22.04%)	BITTORRENT (20.50%)	SKYPE (4.27%)	GNUTELLA (2.74%)
2010	DNS (48.67%)	HTTP (26.75%)	BITTORRENT (9.82%)	TEREDO (4.29%)	SKYPE (3.76%)
2011	DNS (39.91%)	HTTP (29.55%)	BITTORRENT (13.48%)	SKYPE (5.48%)	TEREDO (4.30%)
2012	DNS (44.93%)	HTTP (31.30%)	BITTORRENT (11.11%)	TEREDO (4.17%)	SKYPE (2.12%)
2013	DNS (54.87%)	HTTP (26.78%)	BITTORRENT (6.33%)	NTP (5.16%)	SIP (1.27%)

Year	Top 6	Top 7	Top 8	Top 9	Top 10
2001	NETBIOS (0.43%)	GNUTELLA (0.37%)	CALL OF DUTY (0.28%)	HALF LIFE (0.22%)	IRC (0.19%)
2002	EMULE (0.62%)	FTP (0.48%)	GNUTELLA (0.43%)	MSN (0.23%)	IRC (0.21%)
2003	EMULE (1.22%)	FTP (0.27%)	NORTON (0.23%)	GNUTELLA (0.2%)	MSN (0.18%)
2004	MSN (2.74%)	SKYPE (1.76%)	NETBIOS (1.07%)	GNUTELLA (0.51%)	FTP (0.30%)
2005	OPASERV (3.11%)	SMTP (2.41%)	BITTORRENT (2.10%)	TDS (1.21%)	SMB (0.42%)
2006	SMTP (2.66%)	OPASERV (1.73%)	MSN (1.66%)	PPLIVE (1.60%)	SMB (0.58%)
2007	SQL EXPLOIT (2.74%)	SSH (1.67%)	MSN (0.84%)	FTP (0.37%)	EMULE (0.34%)
2008	SMTP (3.39%)	SSH (2.04%)	MSN (1.61%)	QQ (0.26%)	ORBIT (0.24%)
2009	SQL EXPLOIT (1.40%)	SMTP (1.7%)	SSH (0.83%)	EMULE (0.76%)	PPSTREAM (0.32%)
2010	SSH (1.89%)	SMTP (1.17%)	SQL EXPLOIT (0.68%)	SIP (0.48%)	NTP (0.41%)
2011	NTP (2.30%)	SSH (1.01%)	SMTP (0.59)	EMULE (0.58%)	SIP (0.43%)
2012	SSH (1.50%)	NTP (1.31%)	SIP (0.56%)	SMTP (0.44%)	CANON BJNP (0.36%)
2013	SKYPE (1.18%)	SSH (1.11%)	PANDO (0.93%)	SMTP (0.47%)	CANON BJNP (0.33%)

2.3.4 PAM Dataset

The PAM dataset contains 1 262 022 flows captured during 66 days, between February 25, 2013 and May 1, 2013, which account for 35.69 GB of pure packet data. To collect and accurately label the flows, we adapted Volunteer-Based System (VBS) developed at Aalborg University [50]. The task of VBS is to collect information about Internet traffic flows (i.e., start time of the flow, number of packets contained by the flow, local and remote IP addresses, local and remote ports, transport layer protocol) together with detailed information about each packet (i.e., direction, size, TCP flags, and relative timestamp to the previous packet in the flow). For each flow, the system also collects the process name associated with that flow. The process name is obtained from the system sockets. This way, we can ensure

Table 2.6: Application classes in the dataset

Application	No. of flows	No. of Megabytes
Edonkey	176581	2823.88
BitTorrent	62845	2621.37
FTP	876	3089.06
DNS	6600	1.74
NTP	27786	4.03
RDP	132907	13218.47
NETBIOS	9445	5.17
SSH	26219	91.80
Browser HTTP	46669	5757.32
Browser RTMP	427	3026.57
Unclassified	771667	5907.15

the application associated to a particular traffic. Additionally, the system collects some information about the HTTP content type (e.g., *text/html*, *video/x-flv*). The captured information is transmitted to the VBS server, which stores the data in a MySQL database. The design of VBS was initially described in [50].

Thanks to this methodology, the application name tag was present for 520 993 flows (41.28 % of all the flows), which account for 32.33 GB (90.59 %) of the data volume. Additionally, 14 445 flows (1.14 % of all the flows), accounting for 0.28 GB (0.78 %) of data volume, could be identified based on the HTTP *content-type* field extracted from the packets. Therefore, we were able to successfully establish the ground truth for 535 438 flows (42.43 % of all the flows), accounting for 32.61 GB (91.37 %) of data volume. The remaining flows are unlabeled due to their short lifetime (below ∼1 s), which made VBS incapable to reliably establish the corresponding sockets.

The classes together with the number of flows and the data volume are shown in Table 2.6.

We have published this labeled dataset with full packet payloads in [59]. Therefore, it can be used by the research community as a reference benchmark for the validation and comparison of network traffic classifiers.

Chapter 3

Traffic Classification with NetFlow

3.1 Introduction

This chapter addresses the problem of the deployment of traffic classification solutions. As described in Chapter 1, the limited resources and high throughput requirements of production networks together with the extended application of packet sampling hinder the deployment of state-of-the-art classification techniques. Particularly, concerning the existing ML-based traffic classification solutions, it is worth noting that:

1. Most ML-based techniques only operate with packet-level traces, which requires the deployment of additional (often expensive) monitoring hardware, increasing the burden on network operators.

2. The impact of sampling in traffic classification still remains unknown, although traffic sampling (e.g., Sampled NetFlow) is a common practice among network operators.

3. Most ML-based techniques rely on a costly and time-consuming training phase, which often involves manual inspection of a large number of connections.

In order to counter act the three aforementioned issues, in this chapter we:

- We study the classification problem using NetFlow data instead of packet-level traces (*open issue 1*). NetFlow is a widely extended protocol developed by Cisco to export IP flow information from routers and switches [48]. The main challenge when using NetFlow is the limited amount of information available to be used as features of ML classification methods.

- We use the well-known C4.5 decision tree technique in order to analyze the impact of traffic sampling on the classification accuracy with Sampled NetFlow (*open issue 2*). The main limitation in this case is the low sampling rates typically used by network operators (e.g., 1/1000) to allow routers to handle worst-case traffic scenarios and network attacks. We analyze this problem, both empirically and using sampling theory, and find that packet sampling has a severe impact on the performance of the classification method.

- We propose an automatic ML solution that does not rely on any human intervention and significantly reduces the impact of traffic sampling on the classification accuracy (*open issues 2* and *3*). The main novelty of our solution is that our training set consists of sampled instances that can better capture the properties of the sampled traffic that will be processed in the classification phase. In contrast, previous works used the same training set for both sampled and unsampled scenarios.

- We evaluate the performance of the method using a wide set of traffic traces collected in a large research and education network (4 hours) and validate the results using traces from different environments (CAIDA [60] and WITS [61]).

The remainder of this chapter is organized as follows. Section 3.2 describes the NetFlow-based methodology. Section 3.3 presents an experimental evaluation of the traffic classification method with Sampled NetFlow. Section 3.4 provides a theoretical analysis of the impact of sampling using sampling theory. Section 3.5 proposes a variant of the classification technique to reduce the impact of sampling. Section 3.6 reviews in greater detail the related work. Finally, Section 3.7 concludes the chapter.

3.2 Methodology

This section describes a new ML-based classification method for Sampled NetFlow and its automatic training phase. We also present the methodology and datasets

used for its evaluation and testing.

3.2.1 Traffic Classification Method

We decided to use the C4.5 supervised ML method [62] given its high accuracy and low overhead compared to other ML techniques as will be further discussed in Section 3.6. The C4.5 algorithm builds in an offline phase a decision tree from a set of pre-classified examples (i.e., *training set*). The training set consists of pairs *<flow, application>*, where the *flow* is represented as a vector of features and the *application* is a label that identifies the network application that generated the flow. In our case, the training set contains real traffic flows collected at a large university network (described in Section 3.2.5), and the vector of features includes those features of a flow that are relevant to predict its application.

The main difference between our method and those proposed in previous works is that, when using Sampled NetFlow data, the potential number of features is significantly reduced due to the limited amount of information available in NetFlow version 5 records (i.e., the most extended version). Table 3.1 presents the set of features used in this work. Based on this basic information, we also compute some additional features, such as the average packet size or average inter-arrival time, resulting in 10 features in total. Unlike in some previous works (e.g., [14]), we do not use information about the IP addresses. This decision can negatively impact the final accuracy of the classification method but it allows us to build a classification model less dependent on the particular network scenario used in the training phase. In addition, in the presence of sampling, we correct some features (e.g., number of packets and bytes) by multiplying their sampled values by the inverse of the sampling rate. Table 3.1 shows how each feature is computed according to the sampling rate (p) being applied. In Section 3.4, we provide a theoretical analysis of the error in the estimated features under sampling.

3.2.2 Training Phase and Ground-truth

One of the main limitations of existing ML methods is that they require a very expensive training phase, which usually involves manual inspection of a potentially very large number of connections. In order to automatize the training phase, we developed an automatic training tool based on L7-filter. L7-filter [7] is an open source DPI tool that looks for characteristic patterns in the packet payloads and labels them with the corresponding application. This computationally expensive process is possible during the training phase given that it is executed offline. Finally, the

Table 3.1: Set of 10 NetFlow-based features used in this work

Feature	**Description**	**Value**
sport	Source port of the flow	*16 bits*
dport	Destination port of the flow	*16 bits*
protocol	IP protocol value	*8 bits*
ToS	Type of Service from the first packet	*8 bits*
flags	Cumulative OR of TCP flags	*6 bits*
duration	Duration of the flow in nsec precision	$ts_{end} - ts_{ini}$
packets	Total number of packets in the flow	$\frac{packets}{p}$
bytes	Flow length in bytes	$\frac{bytes}{p}$
pkt_size	Average packet size of the flow	$\frac{bytes}{packets}$
iat	Average packet inter-arrival time	$\frac{duration}{packets/p}$

training phase ends with the generation of a C4.5 decision tree as described in Section 3.2.1. Although the training phase must be performed using packet-level traces, only the NetFlow traffic features listed in Table 3.1 are considered to build the decision tree and to evaluate its performance.

The establishment of the ground-truth is one of the most critical phases of any ML traffic classification method, since the entire classification process relies on the accuracy of the first labeling. Although L7-filter is probably not as accurate as the manual inspection methods used in previous works, it is automatic and does not require human intervention. This is a very important feature given that manual inspection is often not possible in an operational network. For example, each of the traces used in this section contains more than 2 million flows (see Table 2.3). However, our method can also be trained using manually labeled datasets if more accurate results are needed or to avoid the limitations of DPI techniques.

The complete operation to obtain the ground-truth is described in Section 2.3.1. Because of the limitations of L7-filter, we removed the flows labeled as *unknown* from the training set, assuming that they have similar characteristics to other known flows. This assumption is similar to that of ML clustering methods, where unlabeled instances are classified according to their proximity in the feature space to those that are known. More sophisticated labeling methods have recently appeared [8–10, 46, 58], which could be also used to reduce the number of *unknown* flows.

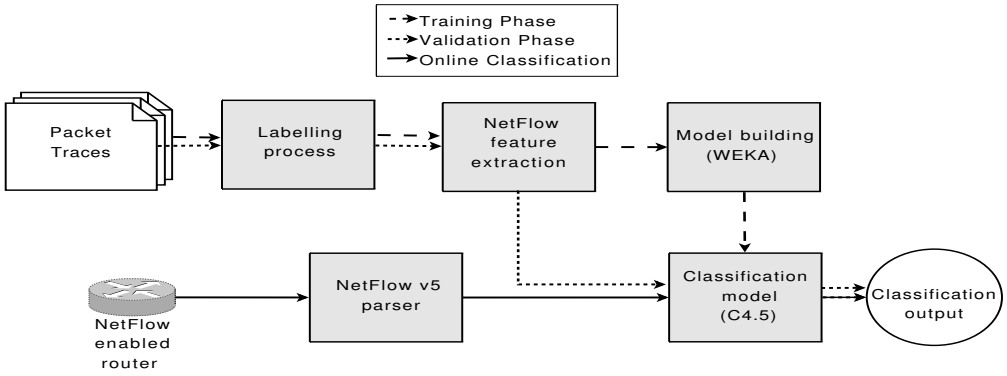


Figure 3.1: Overview of the machine learning and validation process

3.2.3 Machine Learning and Validation Process

Figure 3.1 summarizes the training phase presented in Section 3.2.2 and the validation process used in the evaluation. First, we use a labeled dataset to train our ML method. This dataset establishes the ground-truth of our system and is obtained with L7-filter as described in Section 3.2.2. Next, the *NetFlow feature extraction* module extracts, from each flow in the dataset, the set of NetFlow-based features listed in Table 3.1. After applying the sanitization process described in Section 3.2.2, we obtain the *training set*. This training set is then processed by the *Model building* module to generate the C4.5 decision tree that will be used in the validation phase (*classification model*). We use the WEKA ML software suite [63] to build the J48 decision tree, an open source java extension of the original C4.5.

In the validation phase, we evaluate the performance of the classification model obtained in the training phase. We included the resulting decision tree in the SMARTxAC network monitoring system [64] and processed a large set of traces described later in Section 3.2.5. Although these traces also contain the application label, this information is only relevant to validate the accuracy of our method. After extracting the NetFlow-based features from the evaluation traces similarly to NetFlow, we classify each flow using the C4.5 decision tree generated in the training phase. The quality of the model is measured by comparing the output of the decision tree and the label obtained with L7-filter, according to the performance metrics presented in Section 3.2.4.

Table 3.2: Application groups used by L7-filter

Group	**Applications**
P2P	*File-sharing applications based on peer-to-peer architecture (e.g., Bittorrent, Edonkey)*
HTTP	*HyperText Transfer Protocol applications (e.g, http, httpcachemiss, httpcachehit)*
VoIP	*Voice communications applications (e.g., Skype, TeamSpeak, Ventrilo)*
Network	*Network applications (e.g, BGP, DHCP, Netbios, SSH, Telnet, VNC, X11)*
Streaming	*Stream media applications (e.g., Shoutcast, PPLive, Itunes, QuickTime)*
DNS	*Domain Name System traffic*
Others	*CVS, Hddtemp, IPP, LPD, Subversion, TSP...*
Chat	*Instant messaging applications (e.g, Aim, IRC, Jabber, MSN Messenger, Yahoo Messenger)*
Email	*Email traffic (e.g, IMAP, POP3, SMTP)*
FTP	*File transfer applications (e.g., FTP, Gopher, Tftp, UucP)*
Games	*Massively multiplayer online games (e.g, Battlefield, Counter-Strike Source, Day of Defeat Source, WoW)*

3.2.4 Performance Metrics

In the evaluation, we use three representative performance metrics: *overall accuracy*, *precision* and *recall*. Section 2.2.1 describes in detail how we computed these metrics. In order to simplify the exposition of the results, we show all performance results broken down by application group, according to the groups presented in Table 3.2, which are based on those defined in the L7-filter documentation [7].

A flow is defined as the 5-tuple consisting of the source and destination IP addresses, ports and protocol. In the evaluation we also present the accuracy of the classification method per packets and bytes.

3.2.5 Evaluation Datasets

In this evaluation we used the UPC dataset described in Section 2.3.1. This dataset consists of seven full-payload traces collected at the Gigabit access link of the Universitat Politècnica de Catalunya (UPC). The traces were collected at different days and hours trying to cover as much diverse traffic from different applications as possible.

Table 3.3 shows the estimations of the percentage of elephant flows and the traffic they generate. As we will show later in Section 3.3, this parameter has a

Table 3.3: Elephant flow distribution in the UPC dataset

Name	Flows	Elephant Flows	
		% flows	% bytes
UPC-I	2985 K	0.035818%	52.17%
UPC-II	3369 K	0.048619%	61.45%
UPC-III	3474 K	0.041587%	59.58%
UPC-IV	3020 K	0.048149%	59.79%
UPC-V	7146 K	0.014151%	66.08%
UPC-VI	9718 K	0.042271%	54.51%
UPC-VII	5510 K	0.014075%	72.44%

significant impact on our classification results. In order to identify elephant flows, we use the metric proposed in [65], where a flow is considered as elephant when its size is greater than the mean flow size observed in the trace plus three times the standard deviation.

Among the different traces in the UPC dataset, we selected a single trace (UPC-II) for the training phase, which is the one that contains the highest diversity in terms of instances from different applications. We limit our training set to one trace in order to leave a meaningful number of traces for the evaluation that are not used to build the classification model.

As mentioned in Section 2.3.1, the set of seven labeled traces is publicly available in an anonymized form to the research community at [59].

3.3 Results

We first evaluate the performance of the traffic classification method with unsampled NetFlow. These results are then used as a reference to analyze the impact of traffic sampling on our method. In all experiments, we use the UPC-II trace in the training phase, as described in Section 3.2.5, and evaluate the accuracy of our method with the seven traces presented in Table 2.3, using the performance metrics described in Section 3.2.4.

Table 3.4: Overall accuracy of our traffic classification method (C4.5) and a port-based technique for the traces in our dataset

Name	Overall accuracy			
	C4.5			Port-based
	Flows	Packets	Bytes	Flows
UPC-I	89.17%	66.37%	56.53%	11.05%
UPC-II	93.67%	82.04%	77.97%	11.68%
UPC-III	90.77%	67.78%	61.80%	9.18%
UPC-IV	91.12%	72.58%	63.69%	9.84%
UPC-V	89.72%	70.21%	61.21%	6.49%
UPC-VI	88.89%	68.48%	60.08%	16.98%
UPC-VII	90.75%	61.37%	40.93%	3.55%

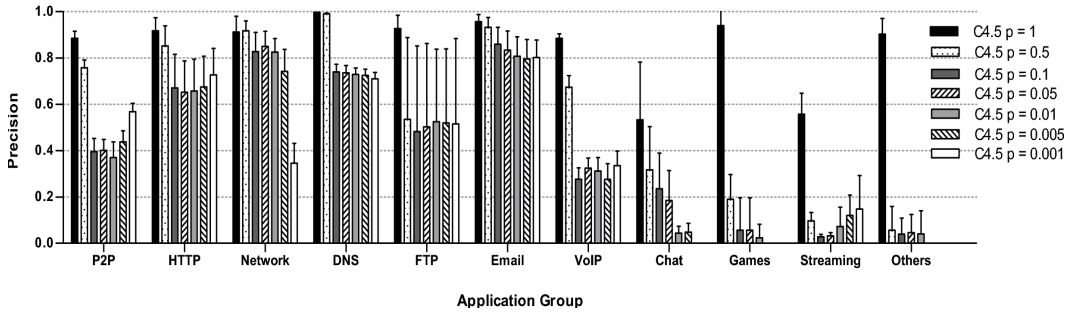


Figure 3.2: Precision (mean with 95% CI) by application group (per flow) of our traffic classification method (C4.5) with different sampling rates

3.3.1 Performance with Unsampled NetFlow

Table 3.4 presents the overall accuracy (in flows, packets and bytes) broken down by trace without applying traffic sampling. Compared to the related work [11–35], our traffic classification method obtains similar accuracy to previous packet-based machine learning techniques, despite the fact that we are only using the features provided by NetFlow.

The C4.5 decision tree obtained in the training phase classifies in average 22057 flows/second with unsampled data using a 3GHz machine with 4GB of RAM. This high classification speed is also reported in other works [31–33, 35].

While the overall flow accuracy is 90.59% in average, the classification accuracy

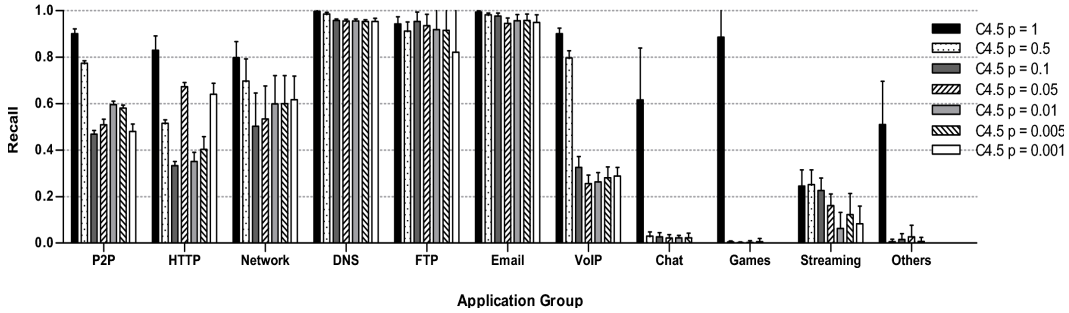


Figure 3.3: Recall (mean with 95% CI) by application group (per flow) of our traffic classification method (C4.5) with different sampling rates

in packets and bytes is, as in the related work, significantly lower (69.83% and 60.32%, respectively). This is a common phenomenon with supervised learning due to the heavy-tailed nature of current Internet traffic. This results in much more instances of mice flows in the training set than of elephant flows, which hinders the identification of a small number of flows that carry a large fraction of the traffic. Table 2.3 confirms that, although the percentage of elephant flows is less than 0.1%, they represent more than 50% of the traffic in our traces. This explains the lower performance achieved in terms of packets and bytes compared to that obtained in terms of flows. However, this effect can be solved by artificially increasing the number of instances of elephant flows in the training set, as we discuss in Section 3.5.3.

Table 3.4 shows that the overall accuracy is similar for all traces. As expected, results for the UPC-II trace are slightly better than for the rest, given that this trace was the one used in the training phase. However, the accuracy per byte with the UPC-VII trace is significantly lower than with the other traces, since this trace was collected at night, when the aggregated traffic mix was more different than that of the training set (e.g., backup and file transfer applications). An example of this different traffic profile can be observed in Table 2.3, which shows that the percentage of elephant flows for those traces collected during weekends (UPC-V) or at night (UPC-VII) is significantly lower (about 0.01%) but, in contrast, they carry a much larger fraction of the traffic (between 60-70%).

Table 3.4 also compares the overall accuracy of our method to a simple technique based on the destination port numbers [1]. As expected, even when using only NetFlow-compatible features, the accuracy of supervised learning methods is significantly higher than that of port-based techniques. It is also important to

note that, while in previous works the accuracy of port-based techniques was in the 50-70% range [14], the accuracy of this method in our traces is only around 10%, which shows the important differences between the traces used in different studies.

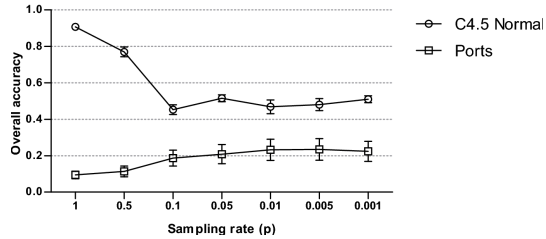
Figure 3.2 presents the precision by application group. When sampling is not applied ($p = 1$), the precision is around 90% for most application groups. Figure 3.3 also shows similar results for the recall by application group. Nevertheless, the performance for some applications (e.g., Streaming, Chat and Others) is significantly lower because L7-filter detected very few flows of these applications in the training set (see Figure 2.1, UPC-II trace). As a result, this inaccuracy has a very limited impact on the overall performance of our method. In order to improve the classification accuracy for these groups, we could always include more instances of these applications in the training set, as we further discuss in Section 3.5.3.

3.3.2 Performance with Sampled NetFlow

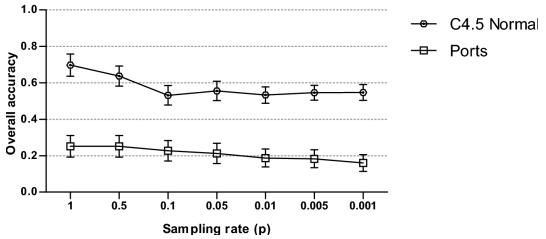
So far, we have shown that our traffic classification method can achieve similar accuracy than previous packet-based machine learning techniques, but using only NetFlow data. Next, we study the impact of packet sampling on our classification method using Sampled NetFlow.

Figure 3.4a shows that the flow classification accuracy degrades drastically with sampling, decreasing from 90.59%, when all packets are analyzed, until 51.02%, when only one packet out of 1.000 is selected. The accuracy decreases much faster for high sampling rates and stabilizes (and even slightly increases) after $p = 10\%$. This effect is inherent to packet sampling, since mice flows tend to rapidly disappear with packet sampling, while the percentage of elephant flows remains fairly stable at low sampling rates. For example, we checked that the percentage of elephant flows in our traces increases by two orders of magnitude (from around 0.05% to 5%) at low sampling rates. Recall that NetFlow only supports static packet sampling and network operators tend to set the sampling rate to very low values in order to allow routers to sustain worst-case traffic mixes and network attacks.

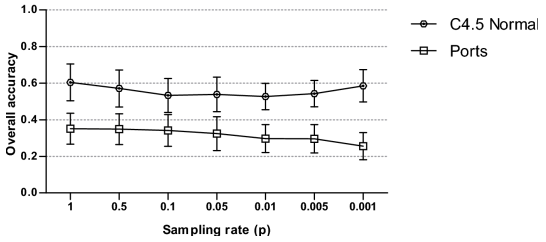
The accuracy of the port-based technique remains almost constant in the presence of sampling (about 10-30%). The accuracy per flow somewhat increases with the sampling rate, which indicates that large flows are easier to identify using the port numbers than small flows. In contrast, the accuracy per packet and byte tends to slightly decay compared to that per flow because, while the percentage of flows stabilizes at low sampling rates, the percentage of packets and bytes decreases in



(a) Accuracy by flows



(b) Accuracy by packets



(c) Accuracy by bytes

Figure 3.4: Overall accuracy (mean with 95% CI) of our traffic classification method (C4.5) and a port-based technique as a function of the sampling rate

the same proportion as the sampling rate.

Figures 3.4b and 3.4c also show that our method is more resilient to sampling for large flows than for small ones, since the accuracy per packet and byte degrades more slowly with the sampling rate than per flow. On the one hand, this is because missing a single packet of a small flow always has a larger impact in the estimation of the flow features than in larger flows. On the other hand, as previously discussed, while our method classifies small flows better than larger ones, small flows tend to rapidly disappear in the presence of sampling.

Figure 3.2 presents the flow precision by application group at different sampling rates. The precision for most applications, except for those with very few instances

in the training set (e.g., Streaming, Chat and Others), decreases with the same proportion depicted in Figure 3.4. Figure 3.3 also shows similar results for the recall by application group.

Both figures when interpreted together give us important information about the quality of our classification method. For example, it can be observed that the precision and recall for HTTP is around 70% and 35% (with $p = 0.01$) respectively, which indicates that while 70% of the flows classified as HTTP are actually HTTP traffic, more than 65% of the HTTP flows in the dataset are incorrectly classified as belonging to another application. On the contrary, while the precision for FTP applications is relatively low in the presence of sampling ($< 50\%$), the recall is very high ($> 90\%$), which means that more than 90% of the FTP flows in the dataset are correctly classified. However, more than 50% of flows identified as FTP belong to other applications.

According to the results presented in this section, we can conclude that the impact of traffic sampling on supervised learning solutions for traffic classification is severe. We consider that an average accuracy of about 50% for sampling rates below 10% is unacceptable to most network operators. Surprisingly, to the best of our knowledge, this is the first work that shows such a drastic impact of sampling in the field of traffic classification. The only previous work that studied the traffic classification problem in the presence of sampling [14] reported instead a minimal decrease of accuracy. We attribute the different results to the more complete datasets used in this study (4 hours of a gigabit link), which are harder to classify using only the port numbers. As we later discuss in Section 3.4, the port numbers are resilient to sampling, which explains the differences in the results. As in this section, a complete, unsampled dataset was employed to train the system in [14]. In contrast, Section 3.5 shows that by simply using a sampled training set to build the classification model, the accuracy and recall figures can be significantly improved.

3.4 Analysis of the Sources of Inaccuracy under Sampling

In Section 3.3.2, we have shown the large impact of packet sampling on the accuracy of a supervised learning technique for traffic classification. However, packet sampling can affect the classification accuracy in many different ways. In particular, we have detected three different sources of error when using packet sampling

in NetFlow: (*i*) error in the estimation of the flow features, (*ii*) changes in the flow size distribution, and (*iii*) splitting of sparse flows. In this section, we analyze these sources of inaccuracy both theoretically and empirically.

3.4.1 Error in the Traffic Features

One of the main sources of inaccuracy under sampling is the estimation of the traffic features. Most of the per-flow features provided by Sampled NetFlow cannot be directly used by the classifier. Instead, they must be inverted prior to the classification process in order to be comparable to the values observed in the training set. We present a theoretical analysis, using sampling theory, of the error in the estimation of the features presented in Table 3.1. We show that, although most of the features are fairly accurate for large flows and moderate sampling rates, some features are clearly biased. For those features that are unbiased, we find that the variance of the error is significant for small flows and low sampling rates. We also present experimental evidence that indicates that this source of error contributes most to the results presented in Section 3.3.2.

- **Source port, destination port and protocol.** These are the only features in Table 3.1 that are not affected by packet sampling, because they have exactly the same value in all the packets of a flow. Thus, their inversion has no impact in the classification accuracy. If one or more packets of the flow are sampled, then the error is 0, while if the flow is not sampled at all it does not contribute to the error. The probability of missing a flow with n packets is $(1-p)^n$, where p is the sampling rate.

- **Flags and ToS.** Under random sampling, the probability p of sampling a packet is independent from the other packets. Let m be the number of packets of a particular flow with the flag f set (i.e., $f = 1$), where $f \in \{0, 1\}$. The probability of incorrectly estimating the value of f under sampling is $(1-p)^m$, independently of how the packets with the flag set are distributed over the flow. The expected value of the absolute error is:

$$\mathrm{E}[f - \hat{f}] = f - \mathrm{E}[\hat{f}] = f - (1 - (1-p)^m) = f - 1 + (1-p)^m \qquad (3.1)$$

  Eq. 3.1 shows that $\hat{f}$ is biased, since the expectation of the error is $(1-p)^m$ when $f = 1$, and it is only 0 when $f = 0$. That is, with packet sampling, $\hat{f}$ tends to underestimate f, especially when $f = 1$ and m or p are small.

For example, if we have a flow with 100 packets with the flag ACK set ($m = 100$) and $p = 1\%$, the expectation of the error in the flag ACK is $(1 - 0.01)^{100} \approx 0.37$. The flag SYN and the ToS are particular cases, where we are only interested in the first packet and, therefore, $m \in \{0, 1\}$.

- **Number of packets.** With sampling probability p, the number of sampled packets x from a flow of n packets follows a binomial distribution $x \sim B(n, p)$. Thus, the expected value of the estimated feature $\hat{n} = x/p$ is:

$$\mathrm{E}[\hat{n}] = \mathrm{E}\left[\frac{x}{p}\right] = \frac{1}{p}\mathrm{E}[x] = \frac{1}{p}np = n \tag{3.2}$$

which shows that $\hat{n}$ is an unbiased estimator of n (i.e., the expected value of the error is 0). The variance of $\hat{n}$ is:

$$\mathrm{Var}[\hat{n}] = \mathrm{Var}\left[\frac{x}{p}\right] = \frac{1}{p^2}\mathrm{Var}[x] = \frac{1}{p^2}np(1-p) = \frac{1}{p}n(1-p) \tag{3.3}$$

Hence, the variance of the relative error can be expressed as:

$$\mathrm{Var}\left[1 - \frac{\hat{n}}{n}\right] = \mathrm{Var}\left[\frac{\hat{n}}{n}\right] = \frac{1}{n^2}\mathrm{Var}[\hat{n}] = \frac{1}{n^2 p}n(1-p) = \frac{1-p}{np} \tag{3.4}$$

Eq. 3.4 indicates that, for a given p, the variance of the error decreases with n. That is, the variance of the error for elephant flows is smaller than for smaller flows. The variance also increases when p is small. For example, with $p = 1\%$, the variance of the error of a flow with 100 packets is $\frac{1-0.01}{100\times 0.01} = 0.99$, which is not negligible.

- **Flow size.** The original size b of a flow is defined as $b = \sum_{i=1}^{n} b_i$, where n is the total number of packets of the flow and b_i is the size of each individual packet. Under random sampling, we can estimate b from a subset of sampled packets by renormalizing their size:

$$\hat{b} = \sum_{i=1}^{n} w_i \frac{b_i}{p} \tag{3.5}$$

where $w_i \in \{0, 1\}$ are Bernoulli distributed random variables with probability p. We can show that $\hat{b}$ is an unbiased estimator of b, since $\mathrm{E}[\hat{b}] = b$:

$$\begin{aligned} \mathrm{E}[\hat{b}] &= \mathrm{E}\left[\sum_{i=1}^{n} w_i \frac{b_i}{p}\right] = \frac{1}{p}\mathrm{E}\left[\sum_{i=1}^{n} w_i b_i\right] = \frac{1}{p}\sum_{i=1}^{n}\mathrm{E}[w_i b_i] = \\ &= \frac{1}{p}\sum_{i=1}^{n} b_i \mathrm{E}[w_i] = \frac{1}{p}\sum_{i=1}^{n} b_i p = \frac{1}{p} p \sum_{i=1}^{n} b_i = b \end{aligned} \tag{3.6}$$

The variance of $\hat{b}$ is obtained as follows:

$$\begin{aligned} \mathrm{Var}[\hat{b}] &= \mathrm{Var}\left[\sum_{i=1}^{n} w_i \frac{b_i}{p}\right] = \frac{1}{p^2}\mathrm{Var}\left[\sum_{i=1}^{n} w_i b_i\right] = \frac{1}{p^2}\sum_{i=1}^{n}\mathrm{Var}[w_i b_i] = \\ &= \frac{1}{p^2}\sum_{i=1}^{n} b_i^2 \mathrm{Var}[w_i] = \frac{1}{p^2}\sum_{i=1}^{n} b_i^2 p(1-p) = \frac{1-p}{p}\sum_{i=1}^{n} bi^2 \end{aligned} \tag{3.7}$$

Thus, the variance of the relative error is:

$$\begin{aligned} \mathrm{Var}\left[1 - \frac{\hat{b}}{b}\right] &= \mathrm{Var}\left[\frac{\hat{b}}{b}\right] = \frac{1}{b^2}\mathrm{Var}[\hat{b}] = \frac{1}{b^2}\frac{1-p}{p}\sum_{i=1}^{n} bi^2 = \\ &= \frac{1-p}{p}\frac{\sum_{i=1}^{n} bi^2}{\left(\sum_{i=1}^{n} bi\right)^2} \end{aligned} \tag{3.8}$$

which decreases with n, since $\sum_i^n bi^2 \leq \left(\sum_i^n bi\right)^2$. This indicates that the variance of the error can be significant for small sampling rates and short flows. For example, if we have a flow with 100 packets of 1500 bytes each, the variance of the error with $p = 1\%$ is $\frac{1-0.01}{0.01} \times \frac{100\times 1500^2}{(100\times 1500)^2} = 0.99$.

- **Duration and interarrival time.** The flow duration is defined as $d = t_n - t_1$, where t_1 and t_n are the timestamps of the first and last packets of the original flow. Under sampling, this duration is estimated as $\hat{d} = t_b - t_a$, where t_a and t_b are the timestamps of the first and last sampled packets

respectively. Thus, the expected value of $\hat{d}$ is:

$$\begin{aligned} \mathrm{E}[\hat{d}] &= \mathrm{E}[t_b - t_a] \\ &= \mathrm{E}[t_b] - \mathrm{E}[t_a] \\ &= \mathrm{E}\left[t_n - \sum_{i=b}^{n} iat_i\right] - \mathrm{E}\left[t_1 + \sum_{i=1}^{a} iat_i\right] \\ &= (t_n - t_1) - \left(\mathrm{E}\left[\sum_{i=b}^{n} iat_i\right] + \mathrm{E}\left[\sum_{i=1}^{a} iat_i\right]\right) \end{aligned} \tag{3.9}$$

where iat_i is the interarrival time between packets i and $i-1$, and a is a random variable that denotes the number of missed packets until the first packet of the flow is sampled (i.e., the number of packets between t_1 and t_a). Therefore, the variable a follows a geometric distribution with probability p, whose expectation is $1/p$. By symmetry, we can consider the number of packets between b and n to follow the same geometric distribution. In this case, we can rewrite Eq. 3.9 as follows:

$$\begin{aligned} \mathrm{E}[\hat{d}] &= (t_n - t_1) - ((\mathrm{E}[n-b]\,\mathrm{E}[iat]) + (\mathrm{E}[a]\,\mathrm{E}[iat])) \\ &= (t_n - t_1) - 2\left(\frac{\overline{iat}}{p}\right) \end{aligned} \tag{3.10}$$

where $\overline{iat}$ is the average interarrival time of the non-sampled packets. Eq. 3.10 shows that the estimated duration is biased (i.e., $\mathrm{E}[d - \hat{d}] > 0$). In other words, $\hat{d}$ always underestimates d. The bias is $(2 \times \overline{iat}/p)$, if we consider the average interarrival time to be equal between packets $1 \ldots a$ and $b \ldots n$. However, we cannot use the feature $\widehat{iat}$ to correct this bias, because this feature is obtained directly from $\hat{d}$. In fact, Eq. 3.10 indicates that the feature $\widehat{iat}$ is also biased, since $\widehat{iat} = \hat{d}/\hat{n}$.

In order to illustrate with empirical data the impact of packet sampling on the estimated features, we present in Table 3.5 the average of the relative error per flow of the features in the UPC-II trace as a function of the sampling rate. We can observe that the error for almost all the features is significant, and it is particularly large for $\hat{n}$ and $\hat{b}$. These large errors are explained by the fact that the average number of packets per flow in the UPC-II trace is only 6.67, given that in NetFlow flows are unidirectional and the trace contains a large portion of DNS traffic (i.e., single-packet flows). According to Eq. 3.4, the variance of the error

Table 3.5: Average of the relative error of the flow features as a function of p (UPC-II trace)

Feature	p = 0.5	p = 0.1	p = 0.05	p = 0.01	p = 0.005	p = 0.001
sport	0.00	0.00	0.00	0.00	0.00	0.00
dport	0.00	0.00	0.00	0.00	0.00	0.00
proto	0.00	0.00	0.00	0.00	0.00	0.00
$\hat{f}$	0.05	0.16	0.18	0.22	0.23	0.24
$\hat{d}$	0.22	0.60	0.66	0.77	0.79	0.81
$\hat{n}$	0.66	3.66	6.90	29.69	55.17	234.61
$\hat{b}$	0.76	3.86	7.05	29.71	55.09	234.24
$\widehat{iat}$	0.29	0.65	0.71	0.78	0.80	0.82

for small flows is very large. For example, with $p = 0.01$ and $n = 1$, the variance of the error in the number of packets is $\frac{1-0.01}{0.01} = 99$. The same observation holds for the flow size.

Given this large error, one would expect the error in the flow features to be the main source of inaccuracy in the classification results presented in Section 3.3.2. In order to confirm this intuition, we performed an experiment to quantify the error introduced by the inverted features in the final classification accuracy. We replaced the estimated features of each flow in the UPC-I trace by their original values, which were directly obtained from the unsampled trace. Note that this cannot be done in the general case, because under sampling the original features are unknown. Figure 3.5 shows the accuracy obtained with the UPC-I trace using the UPC-II trace for training. In particular, the figure presents the classification accuracy removing the error introduced by the inversion of the features. The results of this experiment confirm that this source of error explains the bulk of the classification error, especially for low sampling rates (from $p = 0.1$ to 0.001). The following sections analyze other sources of error that explain the rest of the error observed under sampling.

3.4.2 Changes in the Flow Size Distribution

A second source of inaccuracy is the impact of sampling on the flow size distribution. When using Sampled NetFlow, short flows in terms of packets (i.e., mice) can be easily missed by the sampling process, while long flows (i.e., elephants) are usually sampled, even when using very low sampling rates. As a result, the

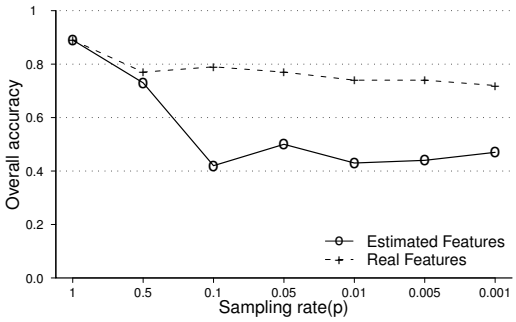


Figure 3.5: Overall accuracy when removing the error introduced by the inversion of the features (UPC-I trace, using UPC-II for training)

distribution of elephant and mice flows in a sampled trace is different from that of an unsampled (complete) trace. As we will see, this impacts the classification accuracy of sampled traffic when an unsampled trace is used for training. Next, we present a theoretical analysis that shows that this distribution changes significantly, even when using high sampling rates.

Let X_p be a discrete random variable that denotes the original size of the sampled flows when using a sampling rate of p. Please note that X_p represents the *original* number of packets of the sampled flows, not the amount of sampled packets. Let Y be a discrete random variable that denotes the original flow length distribution, with probability mass function $P[Y = y]$. Then, the probability mass function of X_p can be expressed as:

$$P[X_p = x] = \frac{\sum_{i=1}^{x} B_p(i,x)P[Y = x]}{\sum_{j=1}^{\infty}\sum_{k=1}^{j} B_p(k,j)P[X = j]} \tag{3.11}$$

where $B_p(k,n)$ is the binomial probability, defined as $B_p(k,n) = \binom{n}{k}p^k(1-p)^{(n-k)}$.

Figure 3.6 validates our model (Eq. 3.11) against the empirical flow length distribution of the detected flows with $p = 0.1$ in the UPC-II trace. In this case, we used the probability mass function $P[Y = y]$ obtained empirically from the UPC-II trace (without sampling). The figure confirms that the model fits very well the empirical distribution under sampling.

Figure 3.6 also confirms our hypothesis for our trace. In order to have a more general result we approximate the original flow length with a Pareto distribu-

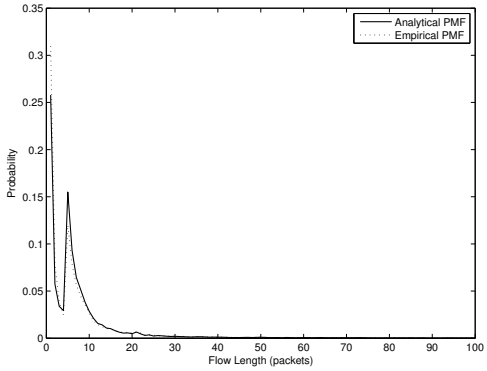


Figure 3.6: Validation of Eq. 3.11 against the empirical distribution of the *original* flow length detected with $p = 0.1$ (UPC-II trace).

tion [66]. Thus, we can express the probability mass function of Y as:

$$P[Y = y] = \frac{\alpha L^{\alpha} x^{-\alpha-1}}{1 - \frac{L}{H}^{\alpha}} \tag{3.12}$$

Figure 3.7 plots the flow length distribution of the detected flows for different sampling probabilities. The figure was obtained by combining Eq. 3.11 with Eq. 3.12. The parameters of the Pareto distribution were approximated using the empirical probability mass function of Y ($\alpha = 3.7003$, $\beta = 0.3650$, $L = 1$ and $H = 1000$).

The figure shows that the distribution of detected mice and elephant flows changes significantly, even with very high sampling probabilities. For high sampling rates (e.g., $p = 0.1$), the amount of mice flows is remarkably larger than that for low sampling rates (e.g., $p = 0.001$). However, the probability of detecting elephant flows remains highly unaffected with respect to the sampling probability. This result confirms that the flow length distribution of the traffic sampled during the classification phase is significantly different from that of the (unsampled) training set. The use of such imbalanced datasets is known to be an important source of inaccuracy of supervised learning methods [67].

Recall the previous experiment presented in Figure 3.5, where we analyzed the impact of the inversion of the traffic features. When $p = 1$, the traffic classification method achieves an overall accuracy above 0.9, while under sampling (even when the features are perfectly accurate), the accuracy drops below 0.8. This decrease

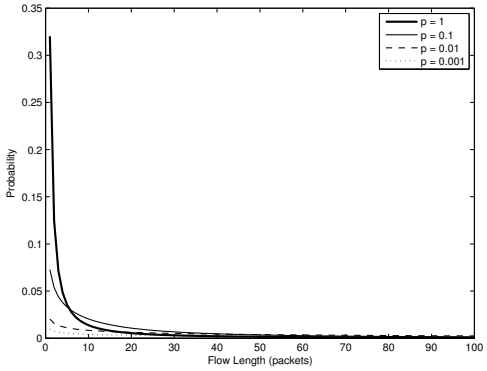


Figure 3.7: Flow length distribution of the detected flows when using several sampling probabilities. The figure was obtained combining Eq. 3.11 with Eq. 3.12

in accuracy can be mostly attributed to the different distribution of the flows observed in the classification and training phases.

3.4.3 Flow Splitting

The last source of inaccuracy that we observed under sampling is flow splitting. NetFlow implements several flow expiration mechanisms [68]. One of them is the *inactivity timeout*, which expires the flows after a certain amount of time of inactivity (TTL seconds). That is, if no packet is received for a given flow during TTL seconds, NetFlow expires the flow and reports it. Under certain circumstances, the packet inter-arrival time of an active flow may be larger than the TTL, and this would result in several instances of the same flow reported, instead of just one. Packet sampling aggravates this problem, since a set of consecutive packets may be missed (not sampled), which may cause NetFlow to incorrectly expire the flow while it is still active.

In this case, the flow is split and reported twice (or more times) to the traffic classification method. This situation affects the classification accuracy, since the features of split flows only reflect partial information of the actual flow (e.g., estimated size and duration of split flows is shorter).

In this section, we aim to model the flow splitting probability under sampling and discuss its impact on the traffic classification method. First, we focus on an analytical model for the probability of splitting a flow, either in one or more

parts, as a function of the sampling rate p. For this purpose, we consider the average inter-arrival time of the packets within a flow ($\overline{iat}$), the inactivity timeout of NetFlow (TTL) and its length in packets (n).

We base our model in the well-known *consecutive-k-out-of-n:F* [69] theory, which accounts for the reliability of a system composed of an ordered sequence of n components, such that the system fails if, and only if, at least k consecutive components fail. In our case, the components are the packets of a flow, and the flow is split if at least k consecutive packets are not sampled, where $k = \lceil TTL/\overline{iat} \rceil$.

From the *consecutive-k-out-of-n:F* theory, we know that the probability P of having at least one run of k missed packets out of n can be expressed with the following recursive formula:

$$P(n,q,k) = P(n-1,q,k) + q^k(1-q)(1-P(n-k-1,q,k)) \tag{3.13}$$

with the following base cases:

- if $n < k$ then $P = 0$
- if $n = k$ then $P = q^k$

where q is the probability of missing a packet ($q = 1-p$). Since a flow cannot be split unless at least 2 packets have been sampled, we can express the split probability $P_{split}(n,p)$ as:

$$P_{split}(n,p) = P(n,1-p,\lceil TTL/\overline{iat} \rceil)[1-(B_p(0,n)+B_p(1,n))] \tag{3.14}$$

where $B_p(k,n)$ is again the binomial probability, defined as $B_p(k,n) = \binom{n}{k}p^k(1-p)^{(n-k)}$.

In order to validate our model, we plot in Figure 3.8 the actual number of split flows in the UPC-II trace as a function of the sampling rate p, along with the values obtained with our model using Eq. 3.14. The figure confirms that our model predicts remarkably well the number of split flows for all the sampling rates. It is also interesting to observe that the amount of split flows is not directly proportional to the sampling rate and shows a peak at $p = 0.1$.

In order to understand this relationship, we must introduce the notion of *sparse* flow [70]. According to our results, split flows typically have very long durations (in the order of hundreds of seconds), but contain very few packets (an average of 16 in our traces). Such *sparse* flows are more prone to be split [70]. However,

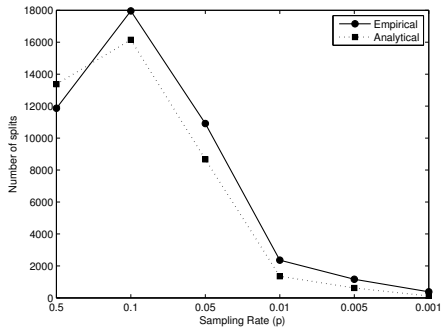


Figure 3.8: Amount of split flows as a function of the sampling probability p. The figure shows both the empirical results (UPC-II trace) and the analytical ones (Eq. 3.14)

the probability of capturing more than one packet of a sparse flow is very small for low sampling rates, which explains the decrease of split flows in Figure 3.8 for sampling rates below 0.1.

Although this source of error decreases the classification accuracy of those flows that have suffered a split (e.g., the accuracy of split flows is 8% below the average for the UPC-II trace with $p = 0.5$), its impact in the overall classification accuracy in our datasets was low. This is explained by the fact that the percentage of split flows in our traces is small (about 1% with $p = 0.5$), since they contain very few sparse flows. However, flow splitting can be an important source of inaccuracy in other traffic profiles with a larger portion of sparse flows.

3.5 Dealing with Sampled NetFlow

Given the severe impact of packet sampling on the classification accuracy observed in previous sections, this section presents a simple improvement in the training process that significantly increases the performance of our traffic classification method in the presence of sampling (e.g., Sampled NetFlow). Finally, we also review some of the limitations detected during the evaluation of our method, which are common to most machine learning-based techniques.

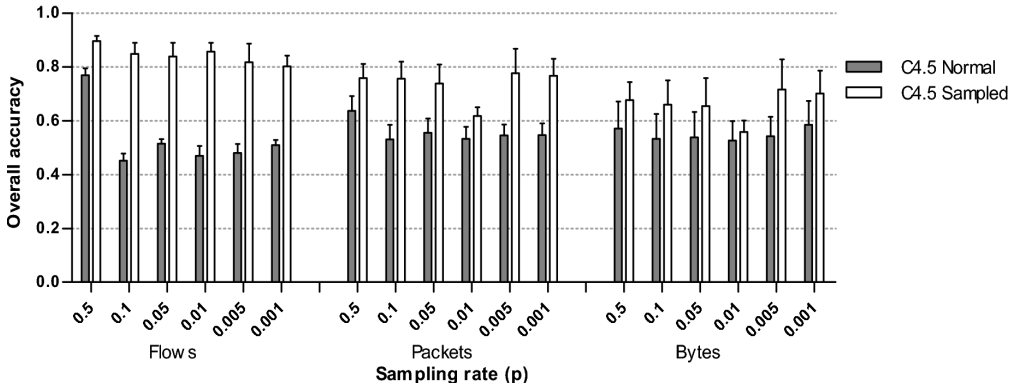


Figure 3.9: Overall accuracy (mean with 95% CI) of our traffic classification method with a normal and sampled training set

3.5.1 Improving the Classification Method

Unlike in previous works, where a set of complete (unsampled) flows are used in the training phase (e.g., [14]), we propose instead to apply packet sampling to the training process. This simple solution is possible because Sampled NetFlow uses a static sampling rate [48], which must be set at configuration time by the network operator. Therefore, we can use exactly the same sampling rate in both the training and classification processes. This solution has the advantage of circumventing the sources of inaccuracy discussed in Section 3.4, since in this case the features do not need to be inverted, while the same flow distribution and split probability is maintained in both the training and classification sets.

In particular, we repeat the training process described in Section 3.2 by applying a set of sampling rates commonly used by network operators, which range from 50% to 0.1%, and then evaluate the accuracy of the classification process under these static sampling rates.

Figure 3.9 shows the substantial improvement obtained in terms of classified flows when applying sampling to the training set. Remarkably, we can observe that the flow accuracy degrades much more gracefully compared to the original technique. For example, with $p = 0.01$, the accuracy per flow increases from 46.96% to 82.38%. Similar improvements are obtained for the other sampling rates. According to the theoretical analysis presented to Section 3.4, this increase of accuracy can be mostly attributed to the error introduced by the inversion of the traffic features when using an unsampled training set. The enhancement

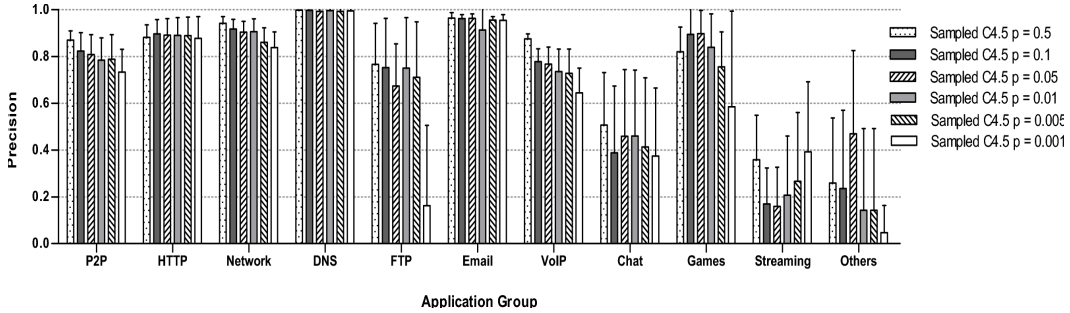


Figure 3.10: Precision (mean with 95% CI) by application group (per flow) of our traffic classification method with a sampled training set

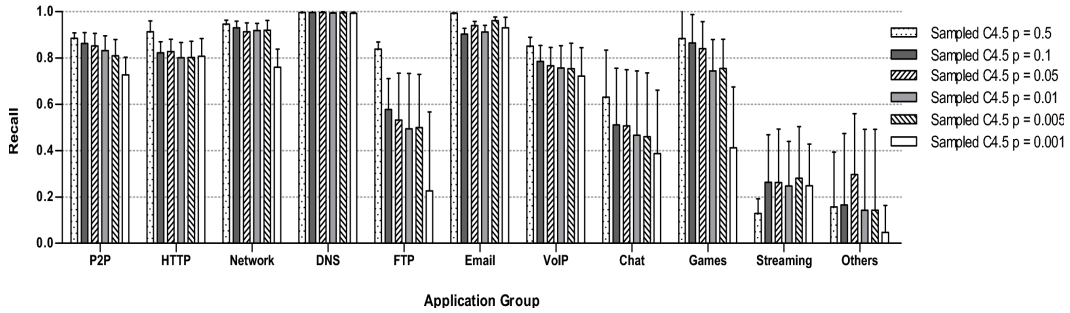


Figure 3.11: Recall (mean with 95% CI) by application group (per flow) of our traffic classification method with a sampled training set

is also partially explained by the already mentioned elephant-mice phenomenon, since now the proportion of elephant and mice flows is kept similar in the training and classification phases. Although the results in terms of packets and bytes are more modest, especially for $p = 0.01$, the improvement is still considerable. This lower gain is expected since the accuracy in terms of packets and bytes under aggressive sampling rates is mainly driven by large flows, which are less affected by our enhancement in the training phase.

The precision (Figure 3.10) and recall (Figure 3.11) are also significantly higher for all application groups and sampling rates. Note also that, while our initial classification method was not able to identify almost any flow of some application groups (e.g., Games, Chat or Others) in the presence of sampling (see Figures 3.2 and 3.3), with the proposed improvement in the training process the pre-

Table 3.6: Characteristics of the traces from other environments

Name	Date	Day	Start Time	Duration	Packets	Bytes	Avg. Util
CAIDA_Chicago_08	18-12-08	Thu	14:00	1 h	3.53 G	2.7 T	6.1 Gbps
CAIDA_Chicago_09	16-07-09	Thu	15:00	1 h	4.28 G	4.2 T	9.3 Gbps
CAIDA_SanJose_08	21-08-08	Thu	15:00	1 h	2.32 G	1.3 T	2.9 Gbps
CAIDA_SanJose_09	17-09-09	Thu	15:00	1 h	3.67 G	2.4 T	5.4 Gbps
Auckland-VIII	10-12-03	Wed	15:00	4 h	35 M	16 G	9.2 Mbps

cision and recall of these applications are notably improved (e.g., precision from 2.78% to 83.97% for Games applications with $p = 0.01$).

3.5.2 Evaluation in Other Network Environments

In previous sections, we showed the performance of our traffic classification method using different traces from a single network viewpoint (UPC). The objective of this section is to evaluate if similar results are also obtained in other network environments.

For this purpose, we used 5 packet-level traces from two public repositories (CAIDA [60] and WITS [61]). The traces from the CAIDA archive were collected in two OC-192 backbone links from a Tier-1 ISP in the US. Conversely, the Auckland-VIII trace from the WITS repository was collected in a 100 Mbps access link of the University of Auckland to the global Internet. Details of the traces are presented in Table 3.6.

A common problem when using public traces is that packet payloads are usually not available for privacy reasons. However, packet payloads are needed to build the ground truth used to measure the classification accuracy. In order to address this limitation, we labeled the traces in Table 3.6 using our own classifier, which we trained using the UPC-II trace as described in Section 3.2.5. Although this approach assumes that the accuracy is 1 when $p = 1$, it is sufficient to analyze the decrease of accuracy due to packet sampling.

Figure 3.12 presents the overall accuracy, in terms of classified flows, when using a normal (unsampled) and a sampled training set with the traces in Table 3.6. The figure shows that similar improvements to those observed in Figure 3.9 with the traces from UPC are also obtained with the traces from other network scenarios. In general, the accuracy when using a complete training set decreases abruptly for moderate sampling rates and starts to recover slowly for low sampling rates.

This effect, which was already observed with the traces from UPC in Figure 3.4, is explained by the fact that, as discussed in Section 3.4, the error in the inversion of the flow features is much smaller for elephants flows, which are the most predominant with low sampling rates (e.g., 1/1000). In contrast, the accuracy when using the improvement presented in Section 3.5.1 is more stable and is kept reasonably high for all sampling rates.

3.5.3 Lessons Learned and Limitations

In the course of this study, we have found that several limitations of most machine learning techniques for traffic classification are directly related to the ground-truth used in the training phase. This explains the accuracy problems observed in Sections 3.3 and 3.5 for some application groups and metrics. As a result of such experience, we present some simple solutions that could help to improve the quality of the training set in future works: (*i*) include a similar number of instances of all application groups, (*ii*) increase the number of instances of elephant flows, for example by using longer traces that can better capture the behavior of large flows, which have a high probability of being ignored in small traces due to the sanitization process, (*iii*) mix several instances of multiple traces from different hours, days and network scenarios in the training set, (*iv*) define alternative groups than those used by L7-filter, according to the actual behavior of the applications, and avoid generic groups (e.g., Others), and (*v*) investigate better, but still automatic, labeling techniques for the training phase (e.g., [8, 58]).[1]

3.6 Related Work

To the best of our knowledge, only a workshop paper [14] has analyzed the traffic classification problem in the presence of sampling. In particular, [14] evaluated the impact of packet sampling on the Naïve Bayes Kernel Estimation supervised machine learning technique. In this work, we analyze instead the impact of sampling on the C4.5 classification method, which is faster in terms of both training and classification time. Despite the similarities between both techniques, we have observed a significantly higher impact of sampling in the classification accuracy.

[1] Note that the accuracy of supervised learning methods is directly restricted by the labeling technique used in the training phase. For example, if DPI techniques are used for training, it is improbable that the resulting classification method can identify encrypted traffic, unless it exhibits similar behavior than the unencrypted traffic of the same application.

While in [14] a complete, unsampled trace is used in the training phase, in this work we show that this approach has a severe impact on the classification accuracy in the presence of sampling. Since the trace used in [14] is not publicly available to further analyze these differences, we attribute them to the more complete datasets used in our study, which are more difficult to classify using only the port numbers (note that port numbers are resilient to sampling) and include a much larger volume of traffic and both TCP and UDP connections. In contrast, [14] only used a single trace that did not contain UDP traffic. In order to allow for further comparison of our results with other classification techniques, we have made our datasets publicly available to the research community. They can be obtained at [59].

The impact of sampling has also been studied in other, closely related research areas. In the field of anomaly detection, several works have analyzed the impact of different sampling techniques on the performance of portscan detection [71–73]. They also conclude that packet sampling has an important impact on the detection accuracy, increasing both false negative and false positive ratios. Although some studies reported lower impacts when using flow sampling (e.g., [71, 72]), Sampled NetFlow only supports packet sampling (systematic and random) [68]. Androulidakis et al. [74] show that systematic sampling is especially problematic when the detection algorithms depend on the observation of a particular packet (e.g., SYN flag). Brauckhoff et al. [75] also find that some anomaly detection metrics are more resilient to sampling than others, especially those based on entropy summarizations, and that detection algorithms based on packet and byte counts are less affected than those based on flow counts.

Finally, in the field of bandwidth estimation, Davy et al. [76] explored the problem of estimating the bandwidth demand from sampled network accounting data for QoS-aware network planning. Their results show estimation errors in the $\pm 10\%$ range. They also find that the estimation error depends on the particular class of traffic and that it is larger for short flows at low sampling rates.

3.7 Chapter Summary

In this chapter, we addressed the traffic classification problem with NetFlow data using a well-known supervised learning technique. Our results allow us to come to the conclusion that: (*i*) supervised methods (e.g., C4.5) can achieve high accuracy ($\approx 90\%$) with unsampled NetFlow data, despite the limited information provided by NetFlow, as compared to the packet-level data used in previous studies, and (*ii*) the impact of packet sampling on the classification accuracy of supervised learning

methods is severe.

Therefore, we proposed a simple improvement in the training process that significantly increased the accuracy of our classification method under packet sampling. For example, for a sampling rate of 1/100, we achieved an overall accuracy of 85% in terms of classified flows, while before we could not reach an accuracy beyond 50%.

Given that Sampled NetFlow is the most widely extended monitoring solution among network operators, we hope that these encouraging results can incentivize the deployment of more accurate techniques in operational networks, where the obsolete port-based method is often the unique real solution.

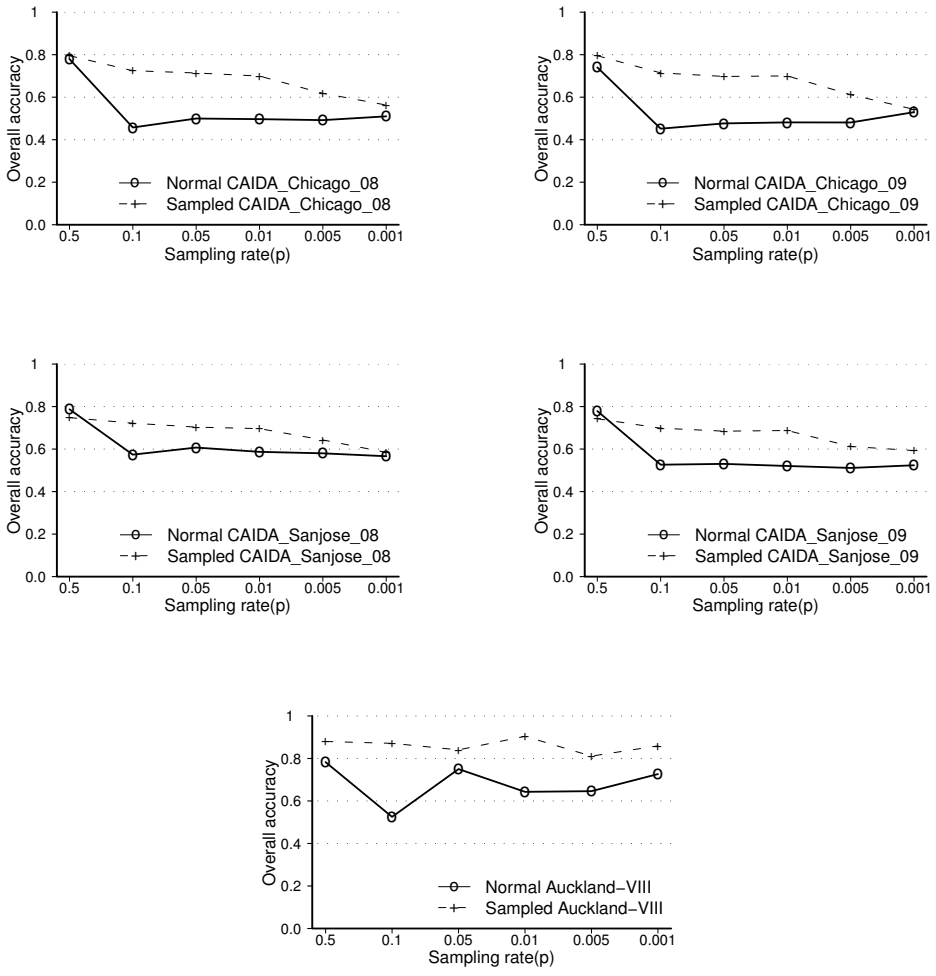


Figure 3.12: Overall accuracy with a normal and sampled training set using the traces from other environments

Chapter 4

Autonomic Traffic Classification System

4.1 Introduction

Once addressed the deployment problem in Chapter 3, in this chapter we address the traffic classification problem with special emphasis on the maintenance of the technique. We propose a complete realistic solution for network operation and management that can be easily deployed and maintained. The classification method combines the advantages of multiple methods, while minimizing their limitations. As described previously in Chapter 3, we use NetFlow as input for the classification to facilitate the deployment. In addition, we propose an autonomic retraining system that can sustain a high classification accuracy during long periods of time. To the best of our knowledge, this is the first solution for traffic classification to provide this feature, which is central for network operation and management, because it removes the need of human intervention of previous solutions, and makes the system easier to maintain. Finally, we evaluate the performance of our method using large traffic traces from a production network.

The rest of this chapter is organized as follows. The proposed classification technique and the autonomic retraining system are described in Section 4.2. Section 4.3 presents a performance evaluation of our method and analyzes the impact of different retraining policies on its accuracy. Section 4.4 reviews in greater detail the related work. Finally, Section 4.5 concludes the chapter.

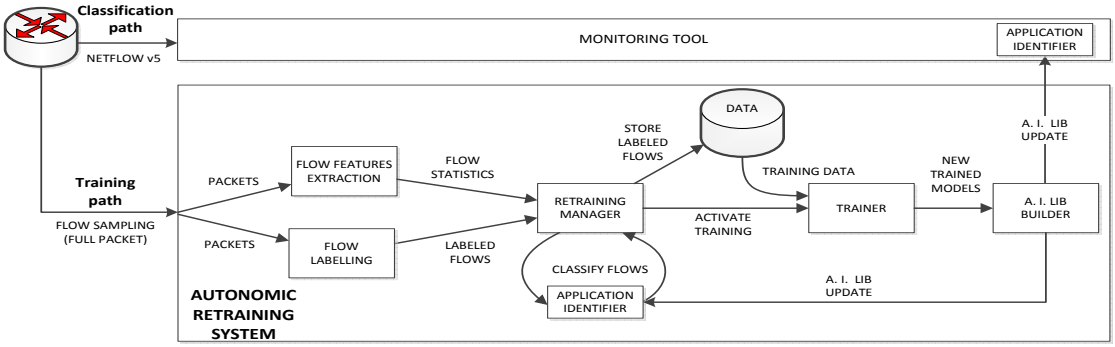


Figure 4.1: Application Identifier and Autonomic Retraining System Architecture

4.2 Traffic Classification System

This section describes our traffic classification method for Sampled NetFlow and its autonomic retraining system. Figure 4.1 illustrates the architecture of the complete traffic classification system, which is divided into two different parts. Above in Fig. 4.1, the *classification path* is in charge of classifying the traffic online. In order to achieve this goal we implement a classifier module, called *Application Identifier*. This module (described later in Section 4.2.1) is loaded as a dynamic library in the monitoring tool and it is only fed with NetFlow v5 data. Given that it only needs the information provided by NetFlow, either sampled or not, and does not have to track the flows, the system is swift and lightweight enough to be deployed in large production networks. As a use-case, we integrated this module in the SMARTxAC system [64] and evaluated it with data from a large production university network, as we describe later in Section 4.3.

Below in Fig. 4.1, the *training path* carries flow-sampled raw traffic to the *Autonomic Retraining System*. This element has two important goals. First, it will provide online information about the current accuracy of the *Application Identifier* that is classifying the traffic online in parallel, as will be described later in Section 4.2.2. Second, it will generate a new classification model when this accuracy falls below a threshold, as we describe in Section 4.2.2. Given that the *Application Identifier* is loaded dynamically it could be reloaded at any time. This capability along with the *Autonomic Retraining System* makes our system resilient to traffic variations as, when the accuracy falls, the system will be automatically retrained and the classification model updated.

4.2.1 The Application Identifier

The *Application Identifier* is the module in charge of the online classification. As aforementioned, it combines different techniques for the sake of exploiting their advantages and reducing their practical limitations. For the reason discussed before, we only consider use methods capable of dealing with Sampled NetFlow data. Furthermore, we need fast classification techniques and methods with a lightweight training phase. Although these constraints complicate the classification, this makes the system easier to deploy and maintain, which are crucial features for network operation and management. The final choice consists of the combination of three techniques of different nature with some improvements especially made to increase their accuracy with Sampled NetFlow data.

Firstly, we use an IP-based technique based on the proposal presented by Mori et al. in [40]. Basically, this technique tracks down in an offline phase the IP addresses belonging to famous web sites (e.g., Google, Megavideo). This technique is very accurate, however its completeness has been significantly degraded given the migration of some applications to Content Delivery Networks. This has relegated its use in our system as a technique to be combined with other more complex and accurate techniques.

The second method used by the *Application Identifier* is an adaptation of the Service-based classifier described by Yoon et al. in [41]. A service is defined as the triplet <IP, Port, Protocol> assigned to a specific application. The list of services is also created in an offline phase using a dataset of labeled flows as follows. In our system, we aggregate all the available flows by their triplet and then, a service is created when a triplet has a minimum number of flows (n) and there is a predominant label ($> m\%$). Unlike in [41], we do not use a time threshold but we require a higher number of flows ($n \geq 5$). We studied offline an efficient configuration of those parameters in order to increase the completeness of the technique while keeping high accuracy. The results have determined that a proper configuration in our setting is selecting $n = 10$ and $m > 95\%$.

The last method used in the *Application Identifier* is a ML technique, namely, the C5.0 decision tree, an optimized successor of the well-known C4.5 presented by Quinlan in [62]. To the best of our knowledge, this is the first work in the field of traffic classification that uses this variant. Nevertheless, several papers have previously highlighted the advantages of its predecessor. Kim et al. in [35] and Williams et al. in [31] compared different classification techniques showing that C4.5 achieves very high accuracy and classification speed due to its inherent feature discretization [77]. Furthermore, the C5.0 is characterized by having shorter

training times compared with its predecessor [78] and with other well-known ML techniques as Support Vectors Machines or Neural Networks [35]. This ability is a key point in our proposal given its importance in the *Autonomic Retraining System*. Because of this we have not applied in our evaluation any improving technique as boosting or bagging. The accuracy obtained by the C5.0 with the default configuration is already very high and the improvement obtained by these techniques was negligible compared with the critical increase of training time. Another important feature of our ML-based technique is its full completeness. As mentioned above, the IP and Service-based techniques have limited completeness given that they rely on IP addresses. However, the ML-based technique allows the *Application Identifier* to classify all the traffic.

It is important to recall that a requirement of our system is that the *Application Identifier* has to work only with Sampled NetFlow traffic. The IP and Service based techniques work properly with Sampled NetFlow because the triplet <IP, Port, Protocol> is not affected by the sampling. However, the ML technique is substantially affected by this constraint, given that NetFlow v5 reduces the amount of features available for the classification and applying sampling considerably impacts on the computation of the features [14, 42]. In order to address this limitation we have implemented the C5.0 ML technique following the recommendations proposed in [42] to improve the classification under Sampled NetFlow, which basically consists of applying sampling to the training phase.

In order to combine the power of the three techniques we combine their classification in a final decision. We give priority to the IP-based technique given the IP addresses have been manually checked. However, given its low completeness, most of the traffic is classified by the Service and ML-based techniques. The distribution of the traffic classified by each technique changes with each retraining, however their contributions are usually around 10% for the IP-based, 60% for the Service-based and 30% for the ML-based technique. Those techniques give their classification decision with a confidence value. The classification decision with highest confidence is selected.

Table 4.1 presents the features used by each technique, all of them obtained from NetFlow v5 data. This is very important because it allows the *Application Identifier* to be very lightweight and easy to deploy given that it works at flow-level and does not have to compute the features for the classification.

Table 4.1: Features used by each classification technique

Technique	Features
IP-based	IP addresses
Service-based	IP, Port and Protocol
ML-based	NetFlow v5 features + average packet size + flow time + flow rate + inter-arrival time

4.2.2 The Autonomic Retraining System

A common limitation of previous proposals presented in the literature, including the ML techniques proposed in [42] in which our *Application Identifier* is based on, is that they usually require an expensive training phase, which involves manual inspection of a potentially very large number of connections. In order to automate this training phase, we developed a retraining system that does not rely on human supervision. This property together with the ability to classify Sampled NetFlow data, makes our proposal a realistic solution for network operators and managers.

Unlike the *Application Identifier*, the *Autonomic Retraining System* presented in Fig. 4.1 uses packet-level data as input. This data is labeled with DPI-based techniques and later used to build the ground-truth for future retrainings. Applying those techniques online is unfeasible given the high resource consumption. However, our system is able to retrain itself and sustain high accuracy rates along time with very few data. This allows us to apply an aggressive flow sampling rate to the *Autonomic Retraining System* input keeping the system very lightweight and economically feasible for the operation and management of large production networks.

The *Autonomic Retraining System* is divided in three phases. The first one corresponds to the *labeling* and *feature extraction*, the second checks the accuracy and stores the ground-truth data and, finally, the last phase retrains and reloads the classifier when it is necessary.

In the *labeling* and *feature extraction* phases, the input data is processed in two different ways. On the one hand, while aggregating the data per flow, a feature extraction is applied to obtain the NetFlow v5 features that would be obtained in the *classification path*. On the other hand, to obtain a reliable ground-truth, we use a set of DPI techniques, including PACE, a commercial DPI library provided by ipoque [9]. PACE is known to have high accuracy with low false positive ratio. Moreover, to increase the completeness of the DPI classification we added two extra

Table 4.2: Application groups and traffic mix

Group	Applications	# Flows	
		UPC-II	CESCA
web	*HTTP*	678 863	17 198 845
dd	*E.g., Megaupload, MediaFire*	2 168	40 239
multimedia	*E.g., Flash, Spotify, Sopcast*	20 228	1 126 742
p2p	*E.g., Bittorrent, Edonkey*	877 383	4 851 103
mail	*E.g., IMAP, POP3*	19 829	753 075
bulk	*E.g., FTP, AFTP*	1 798	27 265
voip	*E.g., Skype, Viber*	411 083	3 385 206
dns	*DNS*	287 437	15 863 799
chat	*E.g., Jabber, MSN Messenger*	12 304	196 731
games	*E.g., Steam, WoW*	2 880	14 437
encryption	*E.g., SSL, OpenVPN*	71 491	3 440 667
others	*E.g., Citrix, VNC*	55 829	2 437 664

libraries, OpenDPI [8] and L7-filter [7]. In addition, our system is extensible and allows the addition of new labeling techniques to increase the completeness and accuracy. Based on their relative accuracy, we have given the highest priority to PACE and the lowest priority to L7-Filter. The final label of each flow is selected from the DPI technique with highest priority. An evaluation of the impact of the different DPI techniques used in the *Autonomic Retraining System* is presented in Section 4.3.1.

In the second phase, the *retraining manager* (see Fig. 4.1) receives the labeled flows together with their NetFlow v5 features. Those that are not labeled as *unknown* are stored for future retrainings. In parallel, the *retraining manager* sends the flows together with their NetFlow v5 features to the *Application Identifier*. This *Application Identifier* is identical to the one that is currently running in the monitoring tool. The *Application Identifier* classifies the flow by obtaining a second label. This label is the same label that the monitoring tool would obtain. By comparing both labels we can compute the actual accuracy of the system. When the accuracy falls under a threshold, we create a new *trainer* in order to build a new classification model. The accuracy in our system is computed from the last flows seen (e.g., 50K in our evaluation). Although the classification is done at the application level, the accuracy is computed aggregating the results at the group level as described in Table 4.2. Table 4.2 also presents the traffic mix of the traces used in the evaluation. These traces that are further described in Section 4.3, although collected in a research/university network, are compounded by

a heterogeneous mixture of applications.

A *trainer* runs as a separate thread and, using the ground-truth dumped in the previous phase, retrains the ML-based and Service-based techniques. The generation of the training dataset is a key point of the retraining system given the important impact it has on the perdurability and accuracy of the models. This process is described in detail in Section 4.2.3. Once the new classification models are built, the new classification library is compiled and dynamically loaded in the *Application Identifiers* that are running on the monitoring tool and the *Autonomic Retraining System* itself. An evaluation of the cost of this process is presented in Section 4.3.2.

4.2.3 Training Dataset Generation

The proper selection of the instances that compose the training dataset will considerably impact on the quality and perdurability of the new classification models created. This way, we have studied two features to build the training dataset: the retraining policy and the training size of the dataset.

The training size is the number of instances (i.e., labeled flows together with their NetFlow v5 features) that compose the training dataset. We refer to the training size as X. The training size substantially impacts on the training times and the quality of the models. On the one hand, selecting a small training size would produce a system highly reactive to accuracy falls given that the retraining time is shorter. However, the classification models built would be less accurate as they have less information (i.e., instances) to build it. On the other hand, a bigger training size would increase the training times but produce more accurate models.

Regarding the retraining policy we implemented two different policies to perform the retraining. The first approach takes into account the last X labeled flows. However, this approach could be biased to the last traffic received. We refer to it as the *naive retraining policy*. The second approach uses random flows from the last Y days, as follows called *long-term retraining policy*. Although it is totally configurable, for the sake of a fair comparison, we also select a total of X flows proportionally distributed in Y days, where X_i is the number of flows for the day i:

$$X_i = X \cdot \frac{2^{(Y-i)}}{2^Y - 1}$$

Thus, it creates a training data set in which recent days have more weight (i.e., more instances) than older ones.

Section 4.3.2 evaluates the impact of those features presenting sound conclusions about the best trade-off between accuracy and performance to obtain proper datasets to maintain an accurate online traffic classifier for a network management tool.

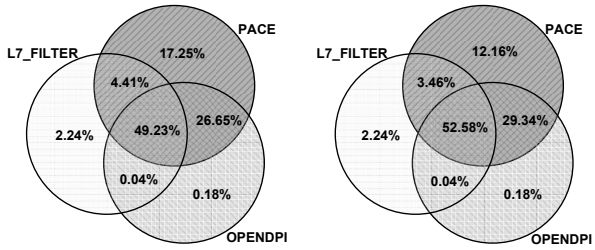
4.3 Evaluation

In this section, we first evaluate the contribution and impact of the different DPI techniques used in the *Autonomic Retraining System*. Then, we evaluate the impact of the policies presented in Section 4.2.3 on the generation of the training dataset. The obtained results are then used to select a proper configuration for the training dataset. Afterwards, we analyze the impact of the *Autonomic Retraining System* on the *Application Identifier*. This evaluation is performed for both sampled and unsampled scenarios. The results show the effectiveness of our system as an autonomous and accurate traffic classifier for large networks.

4.3.1 Evaluation of labeling DPI-based techniques

DPI-based techniques are scarcely used for online classification given their high resources consumption. However, these techniques are commonly used as an automatic ground-truth generator [35,42,54]. In our system, the *Autonomic Retraining System* uses three DPI-based techniques (i.e., PACE, OpenDPI and L7-Filter) to generate the ground-truth online. This is feasible given that the *Autonomic Retraining System* only needs a small sample of the traffic to maintain the *Application Identifier* updated. Samples are selected by applying a high flow sampling rate to the *training path*. This extremely reduces the amount of traffic to be analyzed compared with the whole traffic received by the *classification path*. This type of sampling preserves the entire payload of the flows, allowing DPI-based techniques work properly.

The experiments in this section use the trace *CESCA*. The *CESCA* trace, described in detail in Section 2.3.2, is a fourteen-days packet trace collected on February 2011 in the 10-Gigabit access link of the Anella Científica, which connects the Catalan Research and Education Network with the Spanish Research and Education Network. As described in Section 4.2.2, the *Autonomic Retraining System* only requires a small sample of the traffic to achieve its goal. For this reason, similarly to the flow sampling applied to the *training path*, we applied a 1/400 flow sampling rate. Although the *Autonomic Retraining System* can handle



(a) By application (b) By group of application

Figure 4.2: DPI labeling contribution

higher flow sampling rates, we applied this one because it was the lowest that allowed us to collect the trace without packet loss in our hardware.

Figure 4.2a and 4.2b shows the contribution of the different DPI techniques in the ground-truth generation. The major contributor in the labeling process is PACE. As Fig. 4.2a shows, the contribution of OpenDPI and L7-Filter are very low, 0.22% (0.18% + 0.04%) and 2.24% respectively. This is because most of the labels of these techniques match with the labels of PACE and PACE has higher priority (Sec. 4.2.2). Figure 4.2b shows that OpenDPI and L7-Filter miss some application labels but match them at group level. This can be seen in the decrease of the PACE percentage. These results also help us to understand the completeness our system would achieve in case we have no access to a commercial labeling tool.

In order to guide the network operator in the selection of an appropriate flow sampling rate for their network, Table 4.3 presents the consumption of the DPIs techniques by profiling the *Autonomic Retraining System* running in a 3GHz machine with 4GB of RAM. Table 4.3 shows that the average consumption of the different DPI techniques has the same order of magnitude. However, looking at the *standard deviation* and the *maximum (Max)* by flow, L7-Filter behaves totally different than PACE and OpenDPI. This is because L7-Filter has been limited to the first packets and bytes of each flow in order to reduce the false positive ratio [42]. On the other hand, OpenDPI and PACE perform a more thorough examination in order to find out the application label. In more restrictive scenarios, OpenDPI and L7-Filter could be deactivated to improve the performance of the system. However, given that OpenDPI and L7-Filter detects some applications that PACE does not, we have included both DPI techniques in the system. For instance, the 14-days CESCA trace contains 71 million flows (with 1/400 flow sam-

Table 4.3: DPI techniques consumption

Metric		DPI techniques		
		L7-Filter	**OpenDPI**	**PACE**
Flow	Avg.(μs/flow)	34.54	25.92	32.36
	Std. Dev.(μs/flow)	41.29	1 419.10	1 721.86
	Max (μs)	13 118	1 369 695	1 558 510
Packet	Avg.(μs/packet)	1.74	1.29	1.66
	Std. Dev.(μs/packet)	10.49	4.31	4.87
	Max (μs)	13 118	13 168	12 979

pling applied). Without sampling, 42 μs would be needed in average to process each flow without packet loss (14 days / (71 million flows x 400) = 42 μs/flow). Table 4.3 shows that only the DPI libraries require 92 μs per flow in average. This shows that a traffic classification system based solely on DPI would not be sustainable in our network scenario and it does not scale well to higher link speeds.

4.3.2 Training Dataset Evaluation

In this section, we evaluate the impact of the policies presented in Sec. 4.2.3 in the generation of the training dataset used by the *Autonomic Retraining System*. In all experiments, we use the trace *UPC-II* for the initial offline training and the trace *CESCA* for the evaluation. As described in Section 2.3.1, the trace *UPC-II* is a fifteen-minutes full-payload trace with more than 3 millions of flows. We used different traces for the training and the evaluation in order to show the ability of our system to automatically adapt itself to new scenarios.

In order to asses the quality of the system the *Autonomic Retraining System* uses the accuracy metric. As already mentioned in Sec. 4.2.2, the *Autonomic Retraining System* computes the accuracy by calculating the number of correctly classified flows from the last flows seen (i.e., 50K in our evaluation). The exact definition of the accuracy metric is described in Section 2.2.1.

Although the accuracy is the most popular metric used in the network traffic classification literature it has some limitations. In order to confirm the quality of the *Autonomic Retraining System* we also compute the Kappa coefficient. This metric is considered to be more robust because it takes into account the correct classifications occurring by chance. The computation of the Kappa coefficient is also described in Section 2.2.1.

The Kappa coefficient takes values close to 0 if the classification is mainly due to chance agreement. On the other hand, if the classification is due the discriminative

power of the classification technique then the values are close to 1.

In order to evaluate the impact of the different retraining policies on the *Autonomic Retraining System* we have performed a study of the impact of the parameter Y in the *long-term retraining policy*. The study evaluates the performance of different values of Y (i.e., 5, 7, 9, 11) with a fixed accuracy threshold (i.e., 98%) and a fixed training size (i.e.,X=500K). The results of this evaluation, presented in Table 4.4, show that this parameter has not critic impact on the *Autonomic Retraining System*. However, the values $Y = 7$ and $Y = 11$ achieve the highest accuracies, being $Y = 7$ faster in the training process. As a result, we selected $Y = 7$ for the *long-term retraining policy*. This way, the retraining is performed with flows processed during the last seven days, allowing the system to cover the traffic of an entire week.

Table 4.5 presents the results of the evaluation using three different training sizes (i.e., X={100K, 500K, 1M}) and two retraining policies (i.e., *naive retraining policy* and *long-term retraining policy*). The evaluation has been performed using a high retraining threshold (i.e., the *Application Identifier* is retrained if the accuracy goes below 98%) in order to stress the system to perform multiple retrainings by highlighting the differences between the different configurations. Unlike we initially expected, Table 4.5 shows that the *long-term retraining policy* performs slightly worst than the *naive retraining policy* in terms of accuracy. Moreover, the average training time is shorter for the *naive retraining policy*. This is mainly due the creation of the dataset that, although it could be optimized for the *long-term retraining policy*, it will be always longer than the *naive retraining policy*. Regarding the training sizes, the option *X=100K* achieves lower average accuracy than the other training sizes. However, the *X=100K* training size obtains the highest minimum accuracy and the lowest average training time. This could be interesting if the network demands a fast-recovery system to an accuracy fall. The results comparing *X=500K, 1M* and the *naive retraining policy* show that these configurations obtain similar average accuracy. However, we have decided to choose *X=500K* and *naive retraining policy* as the optimum configuration given that it offers slightly better results. Regarding the impact of this policy on the system, the training with a 98% accuracy threshold only requires 3.93h compared to the 336h (14 days) of duration of the whole experiment, which represents only 13% of the total trace time. If the threshold is lowered up to 96%, the training time is reduced to 0.54h (1.8% of the total trace time).

Although the *Autonomic Retraining System* bases its decisions on the accuracy metric, we have also computed the Kappa coefficient. Table 4.5 shows that the values of the Kappa coefficient are very close to 1. This result confirms the

Table 4.4: Long-Term Policy Evaluation

Metric	**Training Policy**			
	5 days	**7 days**	**9 days**	**11 days**
Avg. Accuracy	98.04%	98.12%	98.07%	98.12%
Min. Accuracy	95.64%	95.44%	95.44%	95.42%
# Retrainings	126	125	125	125
Avg. Training Time	229 s	232 s	234 s	242 s
Cohen's Kappa (k)	0.9635	0.9634	0.9634	0.9633

Table 4.5: Training Dataset Evaluation

Training Size	**Metric**	**Training Policy**	
		Long-Term policy	**Naive policy**
100K	Avg. Accuracy	97.57%	98.00%
	Min. Accuracy	95.95%	97.01%
	# Retrainings	688	525
	Avg. Training Time	88 s	25 s
	Cohen's Kappa (k)	0.9622	0.9567
500K	Avg. Accuracy	98.12%	98.26%
	Min. Accuracy	95.44%	95.70%
	# Retrainings	125	108
	Avg. Training Time	232 s	131 s
	Cohen's Kappa (k)	0.9634	0.9652
1M	Avg. Accuracy	98.18%	98.26%
	Min. Accuracy	94.78%	94.89%
	# Retrainings	61	67
	Avg. Training Time	485 s	262 s
	Cohen's Kappa (k)	0.9640	0.9650

actual classification power of the *Autonomic Retraining System* showing that its classification is not just due to chance agreement.

4.3.3 Retraining Evaluation

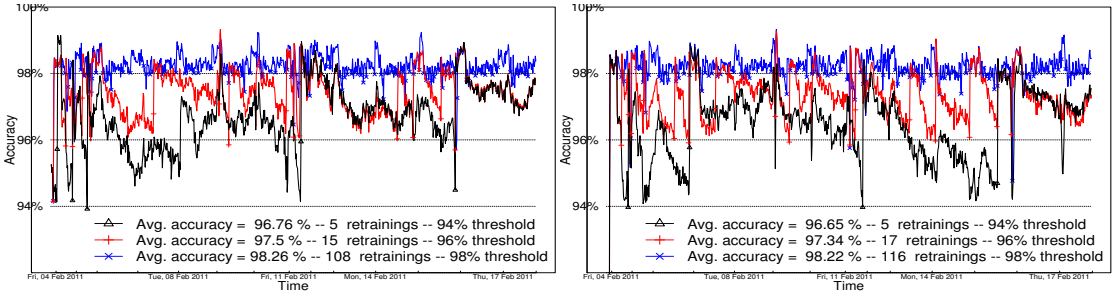
So far, we have separately studied the performance of the labeling techniques and the impact of the different policies on the training dataset generation. Based on these results, we selected a final configuration: the *Autonomic Retraining System* uses the three DPI-based techniques (i.e., PACE, OpenDPI and L7-Filter) for the labeling process (Sec. 4.3.1), 500K flows as training size (i.e., $X = 500\text{K}$) and the *naive retraining policy* (Sec. 4.3.2). In this section, we evaluate the *Application*

Identifier and the impact of the *Autonomic Retraining System* on its accuracy with both sampled and unsampled traffic.

As discussed in Sec. 4.3.2, we use in all experiments the trace *UPC-II* for the initial training and the trace *CESCA* for the evaluation. It is important to note that the trace *UPC-II* was collected in December 2008 while the trace *CESCA* was collected in February 2011. As a result, the system usually performs an initial retraining to update the initial outdated model. This decision has been taken in order to show the impact of the spatial obsolescence, showing that in order to obtain the most accurate classification model it is crucial to train the system with the traffic of the scenario that it is going to be monitored.

Figure 4.3a presents the evaluation of the *Application Identifier* when no packet sampling is applied to the traffic. We tested different accuracy thresholds in order to show the behavior of the system depending on the preferences of the network operator. The system maintains the accuracy of 94% by performing five retrainings during the 14 days. With the 96% threshold it is able to sustain the accuracy during long periods of time with only 15 retrainings. Using the highest threshold, our method achieves better average accuracy than previous thresholds. However, it is not capable to continuously maintain the 98% accuracy. Because of this, the *Autonomic Retraining System* is almost continuously updating the classifier. Nevertheless, these continuous retrainings have not any impact on the *Application Identifier* giving that this procedure is done completely apart. Figure 4.3a also shows the effectiveness of the retrainings, pointed out with cross symbols, that usually produce a substantial increment of accuracy. An interesting result seen by the 94% threshold is the ability of the system to automatically find a proper classification model. As can be seen at the left part of the Fig. 4.3a, the system performs three retrainings but finally builds a model that remains stable for about a week.

We have also evaluated the performance of our system when packet sampling is applied in the *classification path*. We perform the experiments with a common sampling rate of 1/1000 using the configuration before described. Figure 4.3b shows the impact of the retraining in the presence of sampling. The initial low performance showed in Fig. 4.3b is derived from the fact that we build the initial classification library with the unsampled *UPC-II* trace. As a consequence, the system needs to perform an initial retraining to build a representative model of the current sampled scenario. As aforementioned, this also shows the importance of the spatial obsolescence and justifies the importance of the *Autonomic Retraining System*. Surprisingly, after the initial retraining, the system is able to sustain the same accuracy as the unsampled scenario. The greater decrease of information



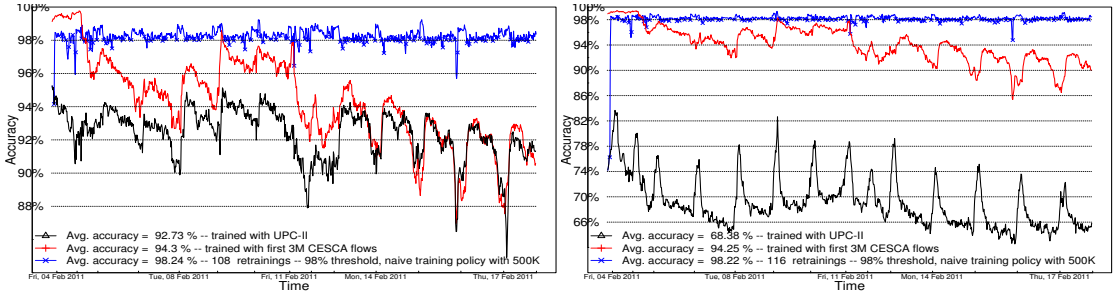
(a) Retraining without sampling

(b) Retraining with 1/1000 sampling rate

Figure 4.3: Impact of the *Autonomic Retraining System* on the *Application Identifier* with the selected configuration (i.e., naive training policy with 500K)

produced by packet sampling [42] is only reflected in a slight increment in the number of retrainings given that, as described in Sec. 4.2.1, our techniques has been adapted to deal with it.

Finally, in order to completely understand the influence of the *Autonomic Retraining System*, we have performed two additional experiments that confirm its necessity. The first experiment creates a model with data from one network to classify traffic from another network (i.e., use the trace *UPC-II* to classify *CESCA*). The second experiment creates the model with the traffic of the own network but does not retrain it (i.e., use the *CESCA* trace to train and classify). Giving both trainings can be performed offline we used 3M of flows for both experiments, instead of the 500K of our solution, trying to build the models as accurate and representative as possible. Even so, Figs. 4.4 show that our solution outperforms these experiments. Fig. 4.4a, that presents the results when no sampling is applied, shows two main outcomes. First, the origin from the data used in the training impacts on the accuracy of the classification (i.e., spatial obsolescence). Even both traces carry traffic from a similar scenario (i.e., educational/research network) there is a substantial difference of accuracy as can be seen in the left part of the figure. The second outcome arises in the right part of the figure where both experiments obtain similar accuracy and this accuracy is gradually decreasing as long as times goes by (i.e., temporal obsolescence). On the other hand, our solution keeps stable during the whole evaluation. Figure 4.4b, that presents the results when 1/1000 sampling rate is applied, emphasizes the outcomes previously mentioned. Here, the application of sampling totally deprecates the classification



(a) Comparative without sampling

(b) Comparative with 1/1000 sampling rate

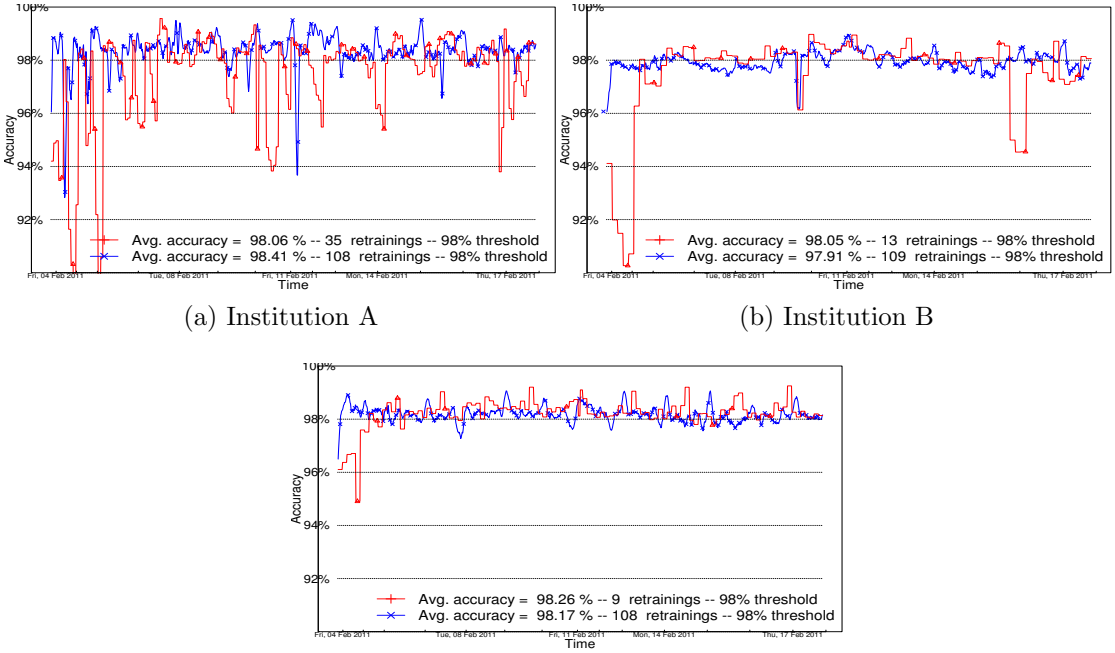
Figure 4.4: Comparative of the *Autonomic Retraining System* with other solutions

model created with the unsampled traffic of *UPC-II* producing a very inaccurate classification. In both scenarios, the experiments that use *CESCA* for training and classification start with a very high accuracy given that they are classifying the same flows used for the training. Because of that, after the first 3M of flows the accuracy decreases even below than 86%. However, our solution with its continuous retraining is able to deal with both temporal and spatial obsolescence achieving a stable accuracy beyond 98%.

4.3.4 Retraining Evaluation by Institution

As described in Sec. 4.3.1, the *CESCA* trace was collected in the 10-Gigabit access link of the Anella Científica, which connects the Catalan Research and Education Network with the Spanish Research and Education Network. This link provides connection to Internet to more than 90 institutions. So far, the evaluation has been performed using the complete traffic of the link. This section presents the results of the performance of the *Autonomic Retraining System* with the disaggregated traffic by institution.

Similarly to the previous evaluation we have used 500K flows as training size (i.e., $X = 500\text{K}$), the *naive retraining policy* (Sec. 4.3.2) and the highest accuracy threshold (i.e., 98%). Two different approaches are used in order to study the performance by institution. First, the *Autonomic Retraining System* uses its normal operation (i.e., using all the traffic and performing the retrainings based on the total accuracy). However, only the accuracy related to the specific institution is presented. Second, the operation of the *Autonomic Retraining System*



(a) Institution A

(b) Institution B

(c) Institution C

Figure 4.5: Comparative of the *Autonomic Retraining System* by institution

is changed and, instead of using all the traffic, it only uses the traffic related to the specific institution. Also, the decision of retraining is carried out based on the particular accuracy of the institution and not with the total one. Figure 4.5 presents the results of this evaluation for three different institutions. The results show the reliability of the *Autonomic Retraining System* for achieving high accuracies with different institutions and scenarios. Although the accuracy is very similar between the three institutions, three different behaviors can be observed. *Institution A* plotted in Fig. 4.5a has a very volatile accuracy. Even when the model is trained with its own traffic the accuracy is sharply changing, although almost always keeping an accuracy higher than 92%. On the other side, *Institution C* plotted in Fig. 4.5c has a more stable accuracy. This is translated into a smaller number of trainings compared with *Institution A*. Finally, the behavior of *Institution B* would be among the other two. These three behaviors plotted in Figure 4.5 are the result of different grades of heterogeneity (i.e., *Institution A*) and homogeneity (i.e., *Institution C*) in the the traffic of the institutions.

Another interesting output from this evaluation is the impact of the origin of the training data on the accuracy. Figure 4.5a shows that *Institution A* achieves higher accuracy performing the retrainings with data from the complete link. However, Fig. 4.5c shows that *Institution C* achieves higher accuracy when the classification model is created with its own data. It is important to note that the amount of retrainings is not comparable between the two approaches. Although the configuration is the same between them, the amount of data available for each approach is different. These two different results regarding the two approaches could be also related to the grade of heterogeneity of the traffic. Training the classification model with traffic from others institutions can help to classify unexpected traffic (e.g., new applications) in networks with heterogeneous traffic.

All these results confirm that the combination of the three techniques and the ability to automatically update the classification model outperform the solutions proposed in the literature for Sampled NetFlow traffic classification [14, 42]. The proposed system has been deployed in production in the Catalan Research and Education network and it is currently being used by network managers of more than 90 institutions connected to this network.

4.4 Related Work

Although some works have also proposed the combination of different classification techniques [79–81], previous solutions do not support sampling, require packet-level data and cannot automatically adapt the classification model to the changing conditions of the network traffic and applications. To the best of our knowledge, only one previous paper has addressed the problem of automatic retraining [54]. However, it only presents a superficial evaluation with the unique goal of showing the feasibility of retraining with a classifier based on K-Dimensional Trees. In this work, we implement and test a complete automatic retraining system able to update multiple classification techniques without relying on human intervention.

4.5 Chapter Summary

In this chapter, we presented a realistic solution for traffic classification for network operation and management. Our classification system combines the advantages of three different techniques (i.e., IP-based, Service-based and ML-based) along with an autonomic retraining system that allows it to sustain a high classification

accuracy during long periods of time. The retraining system combines multiple DPI techniques and only requires a small sample of the whole traffic to keep the classification system updated.

Our experimental results using a long traffic trace from a large operational network shows that our system can sustain a high accuracy (>96%) and completeness during weeks even with Sampled NetFlow data. We also evaluated different training policies and studied their impact on the traffic classification technique. From these results we can draw several conclusions:

- The classification models obtained suffer from temporal and spatial obsolescence. Our results in Sec. 4.3.3 confirm this problem which was already pointed out by Li et al. in [43]. Our system addressed this problem by implementing the *Autonomic Retraining System* that is able to automatically update the classification models without human supervision.

- The life of the classification models is not fixed. As indicated by the results in Sec. 4.3.4, we show that the frequency of retrainings partially depends on the grade of heterogeneity and volatility of the traffic in the network.

- Although several classification techniques have been proposed by the research community, there is no one suitable for all the types of traffic and scenarios. We truly believe that the combination of different techniques is the best approach for properly classifying all the different types of traffic. Our approach, based on three different techniques, is able to achieve very high accuracy and completeness, something that would not be possible if they were not combined.

In summary, we presented a traffic classification system with several features that are particularly appealing for network management: (*i*) high classification accuracy and completeness, (*ii*) support for NetFlow data, (*iii*) automatic model retraining, and (*iv*) resilience to sampling. These features altogether result in a significant reduction in the cost of deployment, operation and maintenance compared to previous methods based on packet traces and manually-made classification models. The proposed system has been deployed in production in the Catalan Research and Education network and it is currently being used by network managers of more than 90 institutions connected to this network.

Chapter 5

Streaming-based Traffic Classification System

5.1 Introduction

The main problem of previous proposals, like the one presented in Chapter 3, is that most of these techniques are based on a static view of the network traffic (i.e. they build a model or a set of patterns from a static, invariable dataset). However, very few works have addressed the practical limitations that arise when facing a more realistic scenario with an infinite, continuously evolving stream of network traffic flows.

On the contrary, this chapter proposes a flow-based network traffic classification solution that can automatically adapt to the continuous changes in the network traffic. We introduce for the first time the use of Hoeffding Adaptive Trees (HAT) for traffic classification. In contrast to previous solutions that rely on static datasets, this technique addresses the classification problem from a more realistic point of view, by considering the network traffic as an evolving, infinite data stream. This technique has very appealing features for network traffic classification, including the following:

- It processes a flow at a time and inspects it only once (in a single pass), so it is not necessary to store any traffic data.

- It uses a limited amount of memory, independent of the length of the data stream, which is considered infinite.

- It works in a limited and small amount of time, so it can be used for online classification.

- It is ready to predict at any time, so the model is continuously updated and ready to classify.

Our solution also has some interesting features that simplify its deployment in operational networks compared to other alternatives based on DPI or ML techniques. The main problem with DPI-based techniques is that they rely on very powerful and expensive hardware to deal with nowadays traffic loads, which must be installed in every single link to obtain a full coverage of a network. Similarly, traditional ML-based techniques for traffic classification require access to individual packets, which involves the use of optical splitters or the configuration of span ports in switches. In contrast, our solution works at the flow level and is compatible with NetFlow v5, a widely extended protocol developed by Cisco to export IP flow information from network devices [68], which has already been deployed in most routers and switches. Although our solution uses NetFlow v5 as input, it can easily work with other similar exporting protocols (e.g., J-Flow, sFlow, IPFIX).

In order to present sound conclusions about the quality, simplicity and accuracy of our proposal we evaluate our traffic classification solution with the entire MAWI dataset [49] described in Section 2.3.3, a unique publicly available dataset that covers a period of 13 years. To the best of our knowledge, this is the first work that deals with such amount of real traffic data for traffic classification. Our results show that our solution for traffic classification is able to automatically adapt to the changes in the traffic over the years, while sustaining very high accuracies. We show that our technique is not only more accurate than other state-of-the-art techniques when dealing with evolving traffic, but it is also less complex and easy to maintain and deploy in operational networks.

The rest of this chapter is organized as follows. The proposed classification technique based on Hoeffding Adaptive Trees is described in Section 5.2. The methodology used for the evaluation of our technique is presented in Section 5.3. Section 5.4 analyzes the impact of different configuration parameters of HAT when used for network traffic classification. Section 5.5 evaluates our solution based on HAT with the MAWI dataset and compares it with the decision tree C4.5 [62], a widely used supervised learning technique. The related work is briefly presented in Section 5.6. Finally, Section 5.7 concludes the work.

5.2 Classification of evolving network data streams

We propose a flow based traffic classification technique for evolving data streams based on Hoeffding Adaptive Trees. This technique has very interesting features for network traffic classification, and addresses the classification problem from a more realistic point of view, because it considers the network traffic as a stream of data instead of as a static dataset. This way, we better represent the actual streaming-nature of the network traffic and address some practical problems that arise when these techniques are deployed in operational networks. We describe our proposal to classify network traffic streams in this section. We first present the original Hoeffding Tree (HT) technique oriented to data streams and then we briefly describe the adaptation to deal with evolving data streams, called Hoeffding Adaptive Tree (HAT). Finally, we present the traffic attributes selected to perform the classification of the network traffic.

5.2.1 Hoeffding Tree

Hoeffding Tree (HT) is a decision tree-based technique oriented to data streams originally introduced by Hulten et al. in [82]. As already mentioned, stream-oriented techniques have many appealing features for network traffic classification: (*i*) they process an example at a time and inspect it only once (i.e., they process the input data in a single pass), (*ii*) they use a limited amount of memory independent of the length of the data stream, which is considered infinite, (*iii*) they work in a limited amount of time, and (*iv*) they are ready to predict at any time. However, these features considerably complicate the induction of the classification model. ML batch techniques (e.g., C4.5, Naive Bayes) are usually performed over static datasets, and therefore, they have access to the whole training data to build the model as many times as needed. On the contrary, models resulting from stream-oriented techniques should be inducted incrementally from the data they process just once and on-the-fly. Therefore, the technique cannot store any data related to the training, which makes the decision-making a critical task.

A key operation in the induction of a decision tree is to decide when to split a node. Batch techniques have access to all the data in order to perform this operation and decide the most discriminating attribute in a node. HT uses the Hoeffding bound [83] in order to incrementally induce the decision tree. Briefly, this bound guarantees that the difference of discriminating power between the best attribute and the second best attribute in a node can be well estimated if enough instances are processed. The more instances it processes the smaller is the error.

The method to compute this discriminating power, which depends on the split criteria (e.g., Information Gain), as well as other HT parameters are later studied in Sec. 5.4.

5.2.2 Hoeffding Adaptive Tree

Hoeffding Tree allows the induction of a classification model according to the requirements of a data stream scenario. However, an important characteristic of the Internet is that the stream of data continually changes over time (i.e., it evolves). Batch models should be regularly retrained in order to adapt the classification model to the variations of the network traffic, which is a complex and very costly task [51]. Hoeffding Adaptive Tree (HAT), proposed in [84], solves this problem by implementing the Adaptive Sliding Window (ADWIN). This sliding window technique is able to detect changes in the stream (i.e., concept drift) and provide estimators of some important parameters of the input distribution using data saved in a limited and fixed amount of memory, which is independent of the total size of the data stream. The interested reader is referred to [85] for more details on how ADWIN is implemented.

5.2.3 Inputs of our system

The implementation of our system can indistinctly receive two different types of instances: labeled and unlabeled flows. Depending on the type of instance, our solution will perform a classification (if the flow is not labeled) or a training operation (if it is labeled). The classification process labels a new unknown flow using the HAT model. The input of the classification process consists of a set of 16 flow features that can be directly obtained from NetFlow v5 data: source and destination port, protocol, ToS, # packets, # bytes, TCP flags, average packet size, flow time, flow rate and flow inter-arrival time. The choice of features is based on our previous work in [51]. The use of standard NetFlow v5 data considerably decreases the cost of deployment and computation requirements of the solution, given that the input is already provided directly by the routers.

The other type of instances our solution can receive are the retraining flows. These flows will be labeled by an external tool, as will be described later. In order to automatically update the model, our technique should receive training flows with the same set of 16 features used by the classification process together with the label associated to them. Unlike batch techniques, the retraining process is performed incrementally, which allows the model to be ready to classify at any

time. Therefore, our solution can indistinctly deal with a mix of instances and operate with them according to their type (i.e., classification or retraining flows). The best ratio between classification and retraining instances depends on the scenario to be monitored. However, as shown in [51], a very small ratio of retraining instances (e.g., less than 1/4000) is sufficient to keep a high accuracy along time. This labeling process can be performed with several techniques, including DPI, given that only a small sample of the traffic needs to be labeled, and therefore it is computationally lightweight. For instance, a common example would be the deployment of our solution in a network with several routers exporting NetFlow v5 data. The labeling of the training flows could be done with NBAR2 [86], using a small sample of the traffic from only one the routers. NBAR2 is a DPI-based technique implemented in the last versions of the CISCO IOS. Otherwise, activating NBAR2 in all the routers and with all the traffic is usually not possible, given the high computational cost and impact it would have on their performance. Another alternative is the use of the methodology presented in [51]. This consists of a small sample of data with full payload, which is labeled using an external DPI tool. This is the solution used in the evaluation presented in Sec. 5.5.

5.3 Methodology

This section describes the methodology used to evaluate the performance of our proposal. First, the tool used for the evaluation is presented and then, the dataset used as ground-truth for the evaluation is described.

5.3.1 MOA: Massive Analysis Online

Massive Online Analysis (MOA) [87] is a Java open source software for data stream mining. Unlike its well-known predecessor WEKA [63], MOA is oriented to the evaluation and implementation of machine learning techniques for data streams. It is specially designed to compare the performance of stream oriented techniques in streaming scenarios. MOA implements the HAT technique with a set of configuration parameters. In addition, it allows the use of batch techniques implemented in WEKA, which simplifies the comparison of traditional batch ML techniques like the decision tree C4.5.

MOA implements different benchmark settings to evaluate stream techniques. For our evaluation, we chose *Evaluate Interleaved Chunks* among the different options available in MOA. *Interleaved Chunks* uses all the instances dividing the

stream in chunks (i.e., set of instances). Every chunk is used first for testing and then for training.

We believe that this approach is the most representative because it uses the complete dataset (i.e., stream) for both testing and training. Similar conclusions are drawn with other evaluations methods. In our evaluation we first use the default configuration of their parameters to simplify its comparison. We then study the impact of the chunk size on its performance.

5.3.2 The MAWI Dataset

In order to obtain representative results for the evaluation of stream oriented techniques we need datasets that are long enough to capture the evolution of Internet traffic over time. We use the publicly available MAWI dataset [49] to perform the evaluation because it has unique characteristics to study stream oriented techniques for network traffic classification. Although it is a static dataset, its long duration (i.e., 13 years) and amount of data makes it the perfect candidate for the evaluation of our technique. Furthermore, its duration allows us to study the ability of HAT to automatically adapt to the evolution of the traffic. Section 2.3.3 describes in detail the MAWI dataset, the methodology used to obtain the ground-truth and its traffic mix. In our evaluation we performed a sanitization process and focused on the TCP and UDP traffic from the MAWI dataset. After the labeling and the sanitization process, the MAWI dataset consists of almost 4 billions of unidirectional labeled flows. To the best of our knowledge this is the first work in the network traffic classification field that deals with this large amount of data, which is necessary to extract sound conclusions from our evaluation.

5.4 Hoeffding Adaptive Tree Parametrization

In this section we study the parametrization of the Hoeffding Adaptive Tree for network traffic classification. As described in Section 5.3 we use MOA and the MAWI dataset to perform the evaluation. Since this is the first work to use the Hoeffding Adaptive Tree for network traffic classification the configuration of the different parameters of HAT and their impact on network traffic classification remain unknown. Because of this, we next present a complete study of the impact of the different parameters of HAT when applied to network traffic. We have studied a total of ten parameters implemented in MOA for HAT: *numeric estimator*, *grace period*, *tie threshold*, *split criteria*, *leaf prediction*, *stop memory management*,

binary splits, *remove poor attributes*, *no preprune*, and *split confidence*. In this section we chose 40 million of instances to perform the evaluation. We split them in four different dates to ensure the representativeness of the results, more exactly we have selected the first 10 million of instances from October 2001, January 2004, July 2008 and March 2011. We perform a specific experiment for each date and then compute the average of them to present the results. After the parametrization Section 5.5 presents an evaluation with the complete MAWI dataset. We briefly describe each parameter, however, we refer the interested reader to [84] for a detailed explanation.

5.4.1 Numeric Estimator

An important issue of ML techniques oriented to data streams is how they deal with numeric attributes. Unlike most batch ML techniques (e.g., C4.5, Naive Bayes), the techniques for data streams can only pass one time over the data. Because of that, the discretization of the features (i.e., numeric attributes are transformed into discrete attributes) is a more difficult task. MOA implements 4 different numeric estimators for classification using HAT: Exhaustive Binary Tree, Very Fast Machine Learning (VFML), Gauss Approximation (i.e., default one) and Quantile Summaries (i.e., Greenwald-Khanna). Figure 5.1 (left) presents the performance results of this criteria. We tested different values for each numeric estimator, however, we studied more values of the VFML numeric estimator given its better results. These values correspond to the number of bins used for discretization of the numeric attributes. Gauss Approximation as much as Greenwald-Khanna obtain very poor results. The best numeric estimators in our scenario are VFML and the Exhaustive Binary Tree (BT). More specifically, VFML 1 000 and the Exhaustive Binary Tree are the most accurate.

Apart from the accuracy, another important feature to take into account is the overhead every option implies. Note that this technique should work online and deal with a huge amount of data in a limited amount of time. Because of this, it is important to keep the solution as lightweight as possible while keeping a high accuracy. Figure 5.1 (right) presents the model cost of each numeric estimator in our evaluation. Greenwald-Khanna, Gauss Approximation, and VFML 10 and 100 are hidden behind VFML 1 000. The huge difference of load between the three most accurate techniques makes the VFML 1 000 the best numeric estimator for our scenario.

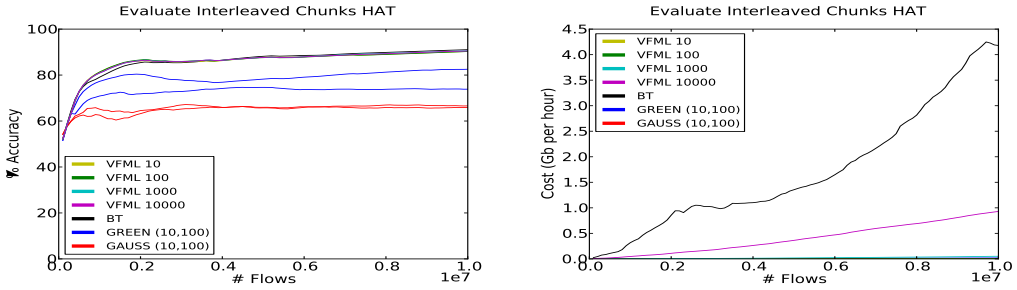


Figure 5.1: Impact of the *Numeric Estimator* parameter

5.4.2 Grace Period

The next parameter studied is the *grace period*. This parameter configures how often (i.e., how many instances between computations) the values in the leafs of HAT are computed. This computation is performed in order to decide if a further split is necessary. This computation is considerably costly and the impact of each instance in the result of this computation is small. Therefore, it is reasonable to perform this computation periodically instead of repeating it for each instance. High values would reduce the cost of the technique but slow down the growth of the tree, thus decreasing its accuracy in theory. Figure 5.2 (left) presents the impact of different grace values on the accuracy of the technique. At first glance there are no huge differences between the different values. As expected the lowest value is initially getting the best results since it is extracting the knowledge by quickly splitting the leaves. However, we are dealing with a data stream and making a decision with few instances can sometimes produce inaccuracies in the future. In Fig. 5.2 (left) the most accurate grace periods are 1 000 and 200 (i.e., default one). Both values are able to keep a stable high accuracy and avoid down peaks presented for the other values.

However, the importance of this parameter is its ability to decrease the overhead of the technique without decreasing significantly its accuracy. Figure 5.2 (right) presents how the different values of the grace period affects to the cost of the technique. We decided to use 1 000 as grace period giving it is the best trade-off between accuracy and load.

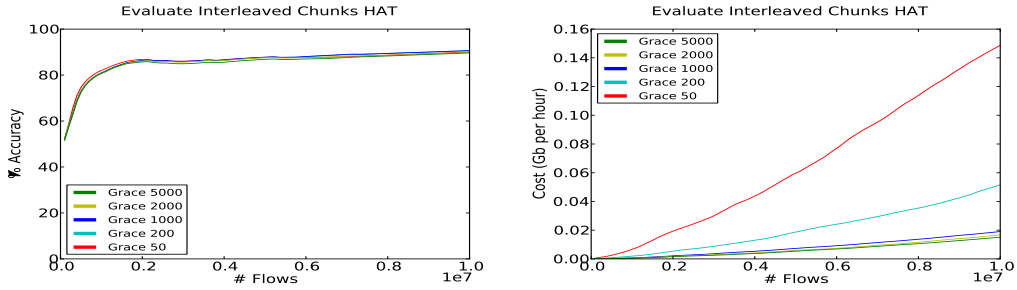


Figure 5.2: Impact of the *Grace Period* parameter

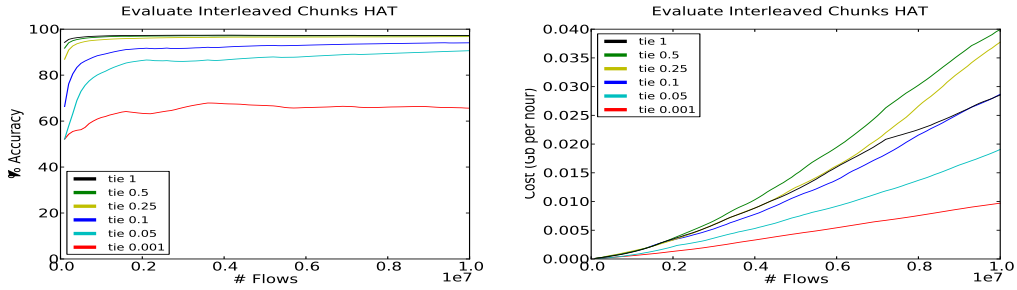


Figure 5.3: Impact of the *Tie Threshold* parameter

5.4.3 Tie Threshold

A well-known parameter from decision tree techniques is the *tie threshold*. Sometimes two or more attributes in a leaf cannot be separated because they have identical values. If those attributes are the best option for splitting the node the decision would be postponed until they differ and this can decrease the accuracy. Figure 5.3 (left) presents the accuracy obtained with different values of the *tie threshold* parameter. The most accurate value is 1, closely followed by 0.5 and 0.25.

In order to decide between the most accurate tie thresholds we rely on the cost of the model they produce. Figure 5.3 (right) shows that 0.25 and 1 are the best options depending on the evaluation approach among the three more accurate values. We decided to use 1 as tie threshold because it is the most accurate.

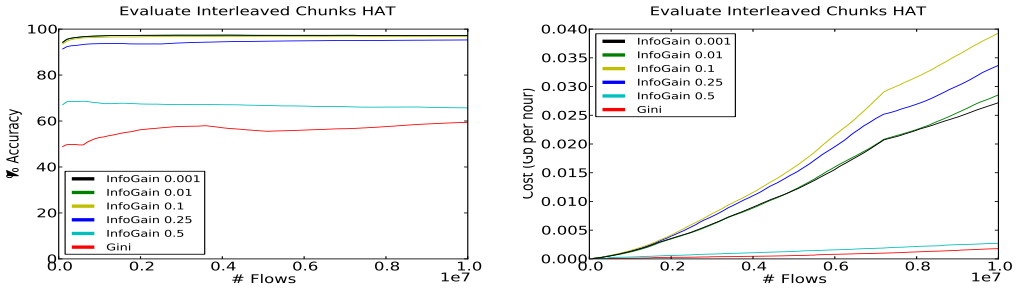


Figure 5.4: Impact of the *Split Criteria* parameter

5.4.4 Split Criteria

As mentioned before, the *grace period* indicates when to compute the necessary values to decide if a node should be split. This computation refers to the *split criteria*. This parameter decides when an attribute is enough discriminative to split a node. There are two approaches implemented in MOA: Information Gain and Gini. Figure 5.4 (left) presents the accuracy obtained with the Gini split criteria and different values of the Information Gain. These values correspond to the minimum fraction of weight required to down at least two branches. The performance of the Gini option is considerably poor in our scenario. Regarding the different values of the Information Gain, the values 0.001, 0.01 and 0.1 achieve the highest accuracies.

Figure 5.4 (right) shows how the cost of technique is impacted by the different split criteria. We decided to use the Information Gain value 0.001 because it is the lightest among the most accurate.

5.4.5 Leaf Prediction

An important feature of HAT is that, since the model is continuously being updated, it is always ready to classify. The next parameter is related to this classification and describe how HAT performs the classification decision at leaf nodes. MOA implements three different approaches: Majority Class, Naive Bayes and Naive Bayes Adaptive. The Majority Class approach consists of assigning the most frequent label in that leaf. Apart from the most frequent label in a leaf, we have much information related to the instance (i.e., attributes). The Naive Bayes approach tries to use this extra information to make a more accurate prediction. This approach computes the probability an instance belongs to the different pos-

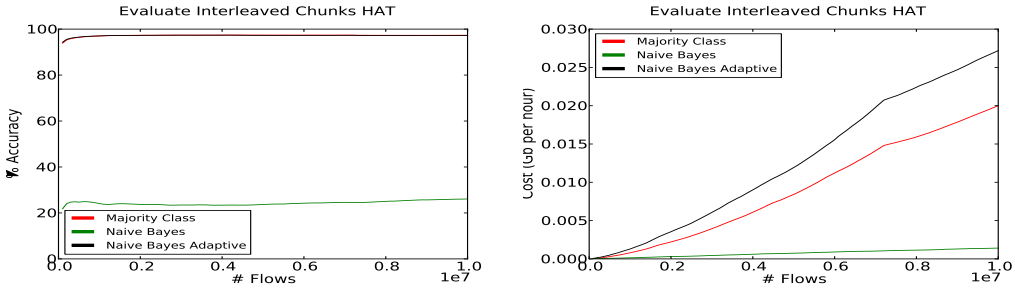


Figure 5.5: Impact of the *Leaf Prediction* parameter

sible labels from a leaf based on its attributes. The most probable label is the one assigned. However, this technique can reduce the accuracy depending on the scenario. The Naive Bayes Adaptive approach tries to take advantage of both approaches by combining them. It computes the error rate of the Majority Class and Naive Bayes in every leaf, and use for future predictions the approach that has been more accurate so far. Figure 5.5 (left) presents the accuracy obtained with the different approaches. Unexpectedly, the Naive Bayes approach obtains very poor results. On the other hand, the Majority Class and the Naive Bayes Adaptive approaches obtain similar high accuracies.

Figure 5.5 (right) shows how the different approaches impact on the solution in terms of model cost. Taking into account these results we decided to use Majority Class as the leaf prediction technique. Apart from having a lower cost, while achieving similar high accuracy, the Majority Class approach is not affected by other parameters. Approaches based on Naive Bayes can decrease its accuracy if parameters like *removing poor attributes* or *stopping memory management* are activated.

5.4.6 Other Parameters

So far, the parameters studied have substantially impacted the accuracy or cost of HAT. However, we have also evaluated some parameters with marginal impact. This is the case of the *Stop Memory Management* parameter. When this parameter is activated HAT stops growing as soon as the memory limit is reached. However, it seems that the default value of the memory limit in MOA is never reached or this parameter is not implemented for the HAT technique. The *Binary Split* parameter, describing if the splits of a node have to be binaries or not, has

Table 5.1: HAT parametrization

Parameter	Value
Numeric Estimator	VFML with 1 000 bins
Grace Period	1 000 instances (i.e., flows)
Tie Threshold	1
Split Criteria	Information Gain with 0.001 as minimum fraction of weight
Leaf Prediction	Majority Class
Stop Memory Management	Activated
Binary Splits	Activated
Remove Poor Attributes	Activated

also a marginal impact. We truly believe that this result is directly related to our scenario characteristics. All our attributes are numerically and hence all the splits performed are almost always binary splits. The last parameter studied with marginal impact is the *Remove Poor Attributes* parameter. This feature removes attributes in the leafs whose initial values indicate their uselessness for the splitting decision. In our scenario, these parameters have not impacted on the accuracy of HAT. However, a marginal improvement has been observed in terms of cost. Thus, we also activated them in the final configuration.

We have also studied the parameters *No PrePrune* and *Split Confidence* and no differences have been observed. As a result, none of them are activated in our final configuration.

Finally, similarly to other ML-based techniques, HAT can be used in ensembles techniques. MOA implements several ensembles methods (e.g., bagging, boosting) that basically combine several models to improve the final accuracy. However, this improvement comes with a higher computational cost. Given that we already achieve a very high accuracy with the current configuration we dismissed the use of ensembles techniques in our scenario.

Table 5.1 presents the final configuration of the parameters obtained in this section. We use this configuration for the evaluation of the HAT technique for network traffic classification.

5.5 Hoeffding Adaptive Tree Evaluation

Once the best configuration is selected we compare the HAT technique with a well-known technique from the literature. The goal of this comparison is to show that our solution can be as accurate as batch-oriented techniques, but with the appealing features of those oriented to streams. As mentioned in Sec. 5.1, batch techniques are usually built from a static dataset and do not address the ever-changing nature of the Internet traffic [43] or rely on complex custom-made solutions [51].

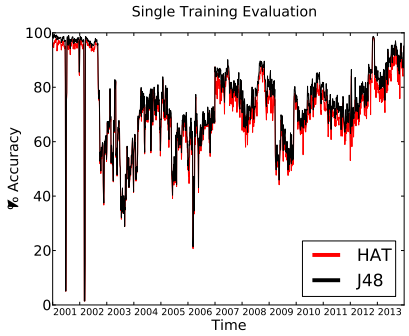


Figure 5.6: Single training configuration

However, our solution can automatically adapt to changing traffic conditions without storing any data and being always ready to classify. For this comparison, we chose the J48 technique as a representative example of batch-oriented techniques, which is an open source version of the C4.5 decision tree implemented in WEKA. We selected this technique because it has been widely used for network traffic classification [36, 42, 43, 51], achieving very good results when compared with other techniques [31, 35].

5.5.1 Single Training Evaluation

Usually ML-based network traffic classification solutions presented in the literature are evaluated from a static point of view using limited datasets. The first evaluation performed pretends to show the temporal obsolescence of the models produced with static datasets [43, 51]. To achieve this goal we performed an evaluation applying just an initial training with 3 million of flows in 2001 for the complete classification of the 13 years of traffic of the MAWI dataset. The accuracy of both techniques is substantially degraded in this evaluation showing that the models should be regularly updated to adapt to the changes in the traffic. The deep drops in the accuracy are related to new applications that are not present in the initial training dataset. The increment of accuracy during the last years of the evaluation is due to the change of the traffic mix in the MAWI dataset. As showed in Table 2.5, there is an increment of traditional applications (i.e., DNS, HTTP and NTP) and a decrease of novel applications (i.e., BitTorrent and Skype) during those years. Giving that this evaluation is performed from a static point of view, HAT is not able to make use of its interesting features for streams.

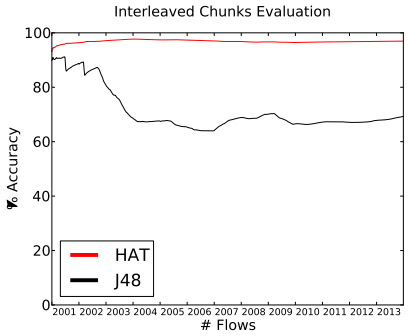


Figure 5.7: *Interleaved Chunk* evaluation with default configuration

5.5.2 Interleaved Chunk Evaluation

The second experiment consists of an *Interleaved Chunk* evaluation with the default evaluation method of MOA. That is, a stream-based evaluation where the 4 000 million of flows from the 13 years of the MAWI dataset are segmented in chunks of 1 000 instances that are first used to classify and later to train. Figure 5.7 presents the results regarding this evaluation. Our solution achieves considerably better results than the J48 batch technique. This can be easily explained by the fact that this evaluation methodology is oriented to evaluate incrementally inducted techniques. The J48 batch technique creates a new decision tree from scratch with every chunk of 1 000 instances forgetting all the previous knowledge extracted. In contrast, our solution updates the classification model with the new information but considering also all the information extracted so far, which results in a more robust classification model.

5.5.3 Chunk Size Evaluation

As shown in the previous experiment, J48 is significantly less accurate than HAT with the default stream-based evaluation. However, that difference seems mainly because of the small chunk size that produces very poor J48 trees. In order to address this problem, we next study the impact of the chunk size on both techniques. We evaluate six different chunk sizes (i.e., 1, 100, 1 000, 10 000, 100 000, 1 000 000 flows) in the *Interleaved Chunk* evaluation. Given the large number of executions involved, we decided to use a sample of more than 4 million of flows of the MAWI dataset in this experiment. Figure 5.8 shows the accuracy of both techniques for each chunk size. Given that HAT builds its tree incrementally, it is

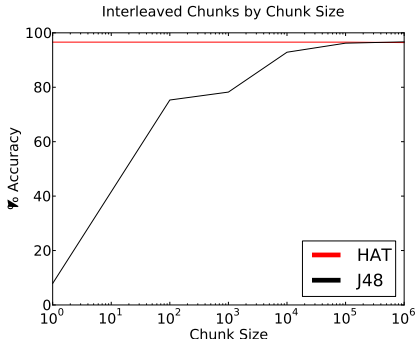


Figure 5.8: Accuracy by chunk size

barely affected by the chunk size, achieving always a very high accuracy. Unlike HAT, J48 is substantially impacted by the chunk size. As expected, the small values of the chunk size (i.e., 1, 100, 1 000) produce inaccurate J48 trees. Only the highest chunk sizes (i.e., 100 000 and 1 000 000) are able to achieve similar accuracies to the HAT technique. Moreover, large chunk sizes imply the storage of large amounts of traffic as we will discuss next.

As important as the accuracy is the cost of the techniques. The J48 decision tree, as a batch technique, needs to store first the data of each chunk to continuously build the model from scratch, which results in huge memory requirements. Figure 5.9 presents the cost (i.e., bytes per second) by flow in log scale directly obtained from MOA. For clarity, only the extremes values (i.e., 1, 1 000 000) and the default value (1 000) are plot. The rest of values follow a similar behavior as the 1 000 chunk size. Initially, all the sizes have a high cost per flow, especially the smallest and the highest chunk sizes (i.e., 1 and 1 000 000). The cost quickly decreases after the initial peak. However, it decreases differently for both techniques. After the initial peak, the cost of J48 remains more or less constant along time. The cost for J48 among the different chunk sizes is similar but the highest chunk size (i.e., 1 000 000), being more than five times higher. In contrast, the cost of HAT rapidly decreases to very low values. Even with the highest chunk size it is able to decrease the cost similarly to the lowest values of the J48 technique. The constant cost of J48 is related to the cost of the training of each model for each chunk. Unlike J48, the model of HAT is incrementally built. Once it is consistent (i.e., around 2 million in our evaluation) only small modifications are applied in the model for every chunk.

To better show the differences in the cost of both techniques, Figure 5.10 presents the accumulated cost of both techniques by chunk size. The growth of

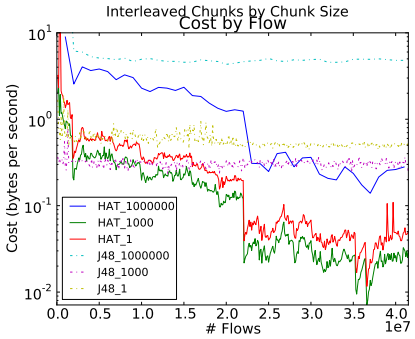


Figure 5.9: Cost by chunk size

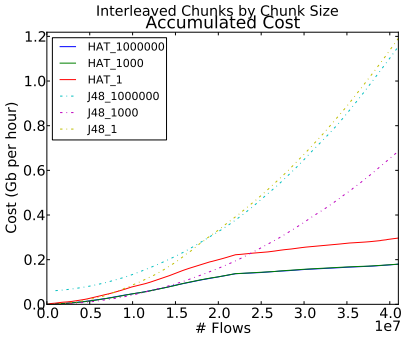


Figure 5.10: Accumulated cost by chunk size

the cost by the HAT technique is almost plain after 2 million of flows. On the other hand, J48 has a continuous growth along time. It is important to note that this evaluation is done with a static dataset of 4 million. However, the difference of cost between both techniques would considerably increase in an infinite stream-based scenario (e.g., network traffic classification).

In summary, in a stream-based scenario the HAT technique is usually more accurate than J48. Only when high chunk sizes are used J48 is able to be as accurate as the HAT technique. Furthermore, HAT consumes less resources than the J48 decision tree, especially when those high chunk sizes (i.e., 100 000 and 1 000 000) are used to increase the accuracy of J48.

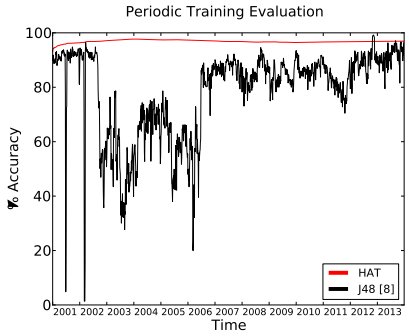


Figure 5.11: *Interleaved Chunk* comparison with [8] configuration

5.5.4 Periodic Training Evaluation

In order to compare our results with other retraining proposals from the literature, we modified the original idea of the *Interleaved Chunk* evaluation by following the configuration proposed in [51]. The new evaluation consists of the use of chunks of 500 000 instances for training and 500 000 000 for testing, using the last seen chunk to train the next group. This evaluation represents the scenario presented in Section 5.2.3, where a sample of the traffic is labeled by a DPI-based technique to retrain the model, while it is used to classify all the traffic. Therefore, with the exception of the first chunk, the complete MAWI dataset is classified. We selected 500 000 as chunk size derived from the results obtained in [51]. However, in [51] the retrained decision is based on a threshold accuracy while, in our evaluation, due to software constraints, it is based on the amount of instances processed (i.e., 500 000 000). Although the evaluation has been changed, the operation to compute the accuracy is maintained to make the comparison possible. Figure 5.11 presents the results of this evaluation. The accuracy of the J48 technique has been improved significantly. However, the stable accuracy seen in the previous evaluation has changed to a more volatile one. This is because the initial configuration is continuously retrained and quickly adapting itself to the changes in the traffic. The results suggest that in this particular dataset, the retraining should be performed more often in order to adapt faster to the changes in the traffic with the related cost it would produce. Note however that the choice of the chunk size for HAT is quite irrelevant as shown in Section 5.5.3

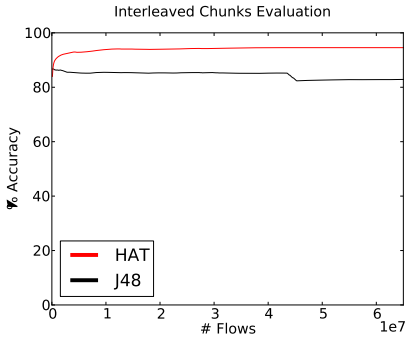


Figure 5.12: *Interleaved Chunk* evaluation with CESCA dataset

5.5.5 External evaluation

So far, we presented the parametrization and evaluation of the HAT technique with the MAWI traffic. The results show that the HAT technique is, at least, as accurate as a state-of-the-art technique, such as C4.5 (i.e., J48 in MOA) but with considerably less costs. In order to show these results are not only related to the MAWI dataset, we next evaluate the performance of the HAT technique with a different dataset. We used the CESCA dataset used in [51] to compare the performance of HAT and J48 and make easier the comparison between both works. The CESCA dataset is a fourteen-days packet trace collected on February 2011 in the 10-Gigabit access link of the Anella Científica, which connects the Catalan Research and Education Network with the Spanish Research and Education Network. A 1/400 flow sampling rate was applied accounting for a total of 65 million of labeled flows. We use the default configuration of the *Interleaved Chunk* evaluation and the parametrization obtained in Section 5.4 for the HAT configuration. Although possible tuning could be applied to this specific scenario, we show that the configuration obtained in Section 5.4 seems suitable for other scenarios. Figure 5.12 shows that, similar to Fig. 5.7, the HAT technique is more accurate than J48 in a stream-based scenario. The smaller differences in terms of accuracy with the CESCA dataset can be related to a less heterogeneous traffic mix and a shorter dataset (i.e., 14 days vs 13 years).

5.6 Related Work

Machine learning techniques for evolving data streams have been widely used in many fields during the last years [88, 89]. However, their application in the field of

network traffic classification has been minimal despite of their appealing features. To the best of our knowledge just two authors have applied similar techniques in the network traffic classification field. Tian et al. in [90,91] presented an evaluation of a tailor-made technique oriented to data streams. They also compare it with different ML batch techniques from the literature (i.e., C4.5, BayesNet, Naive Bayes and Multilayer Perceptron). The results obtained are aligned with our results, however the dataset used was very limited for the evaluation of a stream data technique (i.e., 2 000 instances per application). Also, Raahemi et al. introduced in [92] the use of Concept-adapting Very Fast Decision Tree [82] for network traffic classification. This technique, closely related to HATs, achieves high accuracy. However, the study focus on the differentiation of P2P and non-P2P traffic. The dataset was labeled using a port-based technique with the problems of reliability it implies [5, 47]. Unlike them, our solution is based on a more reliable labeling technique [46,93] and is evaluated with a realistic dataset for evolving data streams (i.e., 13 years of traffic, 9 000 millions of flows). We also perform a complete study of HATs in order to understand the impact of its different parameters on the classification of network traffic.

The problems that arise when a technique is deployed in an actual scenario have been scarcely studied in the literature. To the best of our knowledge only our previous work [51] have addressed the problem of automatically updating the classification models without human intervention. However, the features of this new proposal considerably reduces the requirements of [51]. Although it also needs a small sample of labeled traffic to keep the model updated no data is stored nor periodically retraining is performed. These appealing features makes our proposal a solution very easy to maintain and deploy.

5.7 Chapter Summary

In this chapter we proposed a new stream-based classification solution based on Hoeffding Adaptive Trees. This technique has very appealing features for network traffic classification: (*i*) processes an instance at a time and inspects it only once, (*ii*) uses a predefined amount of memory, (*iii*) works in a bounded amount of time and (*iv*) is ready to predict at any time. Furthermore, our technique is able to automatically adapt to the changes of the traffic with just a small sample of labeled data, making our solution very easy to maintain. As a result, we are able to accurately classify the traffic using only NetFlow v5 data, which is already provided by most routers at no cost, making our solution very easy to deploy.

We evaluate our technique using the publicly available MAWI dataset, 4 000 millions of flows from 15-minutes traces daily collected in a transit link in Japan since 2001 (13 years). We first evaluate the impact of the different parameters on the HAT technique when used for traffic classification and then compare it with one of the state-of-the-art techniques most commonly used in the literature (i.e., C4.5).

The results show that our technique is a perfect solution for network traffic classification. It is not only more accurate than traditional batch-based techniques, but it also sustains this very high accuracy over the years with less cost. Furthermore, our technique does not require complex, ad-hoc retraining systems to keep the system updated, which facilitates its deployment and maintenance in operational networks.

Chapter 6

Validation of Traffic Classification Methods

6.1 Introduction

As described in Chapter 1, most traffic classification solutions proposed in the literature report very high accuracy. However, these solutions mostly base their results on a private ground-truth (i.e., dataset), usually labeled by techniques of unknown reliability (e.g., ports-based or DPI-based techniques [42, 44–46]). That makes it very difficult to compare and validate the different proposals. The use of private datasets is derived from the lack of publicly available datasets with payload. Mainly because of privacy issues, researchers and practitioners are not allowed to share their datasets with the research community.

Another crucial problem is the reliability of the techniques used to set the ground-truth. Most papers show that researchers usually obtain their ground-truth through port-based or DPI-based techniques [42, 44–46]. The poor reliability of port-based techniques is already well known, given the use of dynamic ports or well-known ports of other applications [4, 47]. Although the reliability of DPI-based techniques is still unknown, according to conventional wisdom they are, in principle, one of the most accurate techniques.

This chapter presents two main contributions. First, we publish a reliable labeled dataset with full packet payloads [59]. The dataset has been artificially built in order to allow us its publication. However, we have manually simulated different behaviors to make it as representative as possible. We used VBS [50] to guarantee the reliability of the labeling process. This tool can label the flows

Table 6.1: DPI-based techniques evaluated

Name	Version	Applications
PACE	1.41 (June 2012)	1000
OpenDPI	1.3.0 (June 2011)	100
NDPI	rev. 6391 (March 2013)	170
L7-filter	2009.05.28 (May 2009)	110
Libprotoident	2.0.6 (Nov 2012)	250
NBAR	15.2(4)M2 (Nov 2012)	85

with the name of the process that created them. This allowed us to carefully create a reliable ground-truth that can be used as a reference benchmark for the research community. Second, using this dataset, we evaluated the performance and compared the results of 6 well-known DPI-based techniques, presented in Table 6.1, which are widely used for the ground-truth generation in the traffic classification literature.

These contributions pretend to be a first step towards the impartial validation of network traffic classifiers. They also provide to the research community some insights about the reliability of different DPI-based techniques commonly used in the literature for ground-truth generation.

The remainder of this work is organized as follows. Section 6.2 describes the methodology used to create and label the dataset. Section 6.3 compares the performance of the selected DPI-based techniques. Section 6.4 extracts the outcomes from the results previously obtained. Section 6.5 briefly reviews the related work. Finally, Section 6.6 concludes the work and outlines our future work.

6.2 Methodology

The Testbed

Our testbed is based on VMWare virtual machines (VM). We installed three VM for our data generating stations and we equipped them with Windows 7 (W7), Windows XP (XP), and Ubuntu 12.04 (LX). Additionally, we installed a server VM for data storage.

As described in Section 2.3.4, to collect and accurately label the flows, we adapted Volunteer-Based System (VBS) developed at Aalborg University [50]. The task of VBS is to collect information about Internet traffic flows (i.e., start time of the flow, number of packets contained by the flow, local and remote IP addresses, local and remote ports, transport layer protocol) together with detailed information about each packet (i.e., direction, size, TCP flags, and relative timestamp to

the previous packet in the flow). For each flow, the system also collects the process name associated with that flow. The process name is obtained from the system sockets. This way, we can ensure the application associated to a particular traffic. Additionally, the system collects some information about the HTTP content type (e.g., *text/html*, *video/x-flv*). The captured information is transmitted to the VBS server, which stores the data in a MySQL database. The design of VBS was initially described in [50]. On every data generating VM, we installed a modified version of VBS. The source code of the modified version was published in [94] under a *GPL license*. The modified version of the VBS client captures full Ethernet frames for each packet, extracts HTTP *URL* and *Referer* fields. We added a module called *pcapBuilder*, which is responsible for dumping the packets from the database to PCAP files. At the same time, INFO files are generated to provide detailed information about each flow, which allows us to assign each packet from the PCAP file to an individual flow. We also added a module called *logAnalyzer*, which is responsible for analyzing the logs generated by the different DPI tools, and assigning the results of the classification to the flows stored in the database.

Selection of the Data

The process of building a representative dataset, which characterizes a typical user behavior, is a challenging task, crucial on testing and comparing different traffic classifiers. Therefore, to ensure the proper diversity and amount of the included data, we decided to combine the data on a multidimensional level. Based on w3schools statistics, we selected Windows 7 (55.3 % of all users), Windows XP (19.9 %), and Linux (4.8 %) - state for January 2013. Apple computers (9.3 % of overall traffic) and mobile devices (2.2 %) were left as future work. The selected applications are shown below.

- Web browsers: based on w3schools statistics: Chrome and Firefox (W7, XP, LX), Internet Explorer (W7, XP).
- BitTorrent clients: based on CNET ranking: uTorrent and Bittorrent (W7, XP), Frostwire and Vuze (W7, XP, LX)
- eDonkey clients: based on CNET ranking: eMule (W7, XP), aMule (LX)
- FTP clients: based on CNET ranking: FileZilla (W7, XP, LX), SmartFTP Client (W7, XP), CuteFTP (W7, XP), WinSCP (W7, XP)
- Remote Desktop servers: built-in (W7, XP), xrdp (LX)

- SSH servers: sshd (LX)
- Background traffic: DNS and NTP (W7, XP, LX), NETBIOS (W7, XP)

The list of visited websites was based on the top 500 websites according to Alexa statistics. We chose several of them taking into account their rank and the nature of the website (e.g., search engines, social medias, national portals, video websites) to assure the variety of produced traffic. These websites include: Google, Facebook, YouTube, Yahoo!, Wikipedia, Java, and Justin.tv. For most websites we performed several random clicks to linked external websites, which should better characterize the real behavior of the real users and include also other websites not included in the top 500 ranking. This also concerns search engines, from which we manually generated random clicks to the destination web sites. Each of the chosen websites was processed by each browser. In case it was required to log into the website, we created fake accounts. In order to make the dataset as representative as possible we have simulated different human behaviors when using these websites. For instance, on Facebook, we log in, interact with friends (e.g., chat, send messages, write in their walls), upload pictures, create events or play games. On YouTube, we watched the 10 most popular videos, which we randomly paused, resumed, and rewound backward and forward. Also, we randomly made some comments and clicked *Like* or *Not like* buttons. The detailed description of actions performed with the services is listed in our technical report [93]. We tested the P2P (BitTorrent and eDonkey) clients by downloading files of different sizes and then leaving the files to be seeded for some time, in order to obtain enough of traffic in both directions. We tried to test every FTP client using both the active transfer mode (PORT) and passive transfer mode (PASV), if the client supports such mode.

Extracting the Data for Processing

Each DPI tool can have different requirements and features, so the extracting tool must handle all these issues. The PCAP files provided to PACE, OpenDPI, L7-filter, NDPI, and Libprotoident are accompanied by INFO files, which contain the information about the start and end of each flow, together with the flow identifier. Because of that, the software, which uses the DPI libraries, can create and terminate the flows appropriately, as well as to provide the classification results together with the flow identifier. Preparing the data for NBAR classification is more complicated. There are no separate INFO files describing the flows, since the classification is made directly on the router. We needed to extract the packets in

a way that allows the router to process and correctly group them into flows. We achieved that by changing both the source and destination MAC addresses during the extraction process. The destination MAC address of every packet must match up with the MAC address of the interface of the router, because the router cannot process any packet which is not directed to its interface on the MAC layer. The source MAC address was set up to contain the identifier of the flow to which it belongs, so the flows were recognized by the router according to our demands. To the best of our knowledge, this is the first work to present a scientific performance evaluation of NBAR.

The Classification Process

We designed a tool, called *dpi_benchmark*, which can read the PCAP files and provide the packets one-by-one to *PACE, OpenDPI, L7-filter, NDPI* and *Libprotoident*. All the flows are started and terminated based on the information from the INFO files. After the last packet of the flow is sent to the classifier, the tool obtains the classification label associated with that flow. The labels are written to the log files together with the flow identifier, which makes us later able to relate the classification results to the original flows in the database. A brief description of the DPI-tools used in this study is presented in Table 6.1. Although some of the evaluated tools have multiple configuration parameters, we have used in our evaluation the default configuration for most of them. A detailed description of the evaluated DPI-tools and their configurations can be found in [93].

Classification by *NBAR* required us to set up a full working environment. We used GNS3 - a graphical framework, which uses Dynamips to emulate our Cisco hardware. We emulated the 7200 platform, since only for this platform supported by GNS3 was available the newest version of Cisco IOS (version 15), which contains Flexible NetFlow. The router was configured by us to use Flexible NetFlow with *NBAR* on the created interface. Flexible NetFlow was set up to create the flows taking into account the same parameters as are used to create the flow by VBS. On the computer, we used *tcpreplay* to replay the PCAP files to the router with the maximal speed, which did not cause packet loss. At the same time, we used *nfacctd*, which is a part of PMACCT tools, to capture the Flexible NetFlow records sent by the router to the computer. The records, which contain the flow identifier (encoded as source MAC address) and the name of the application recognized by *NBAR*, were saved into text log files. This process is broadly elaborated in our technical report [93].

The Dataset

Our dataset contains 1 262 022 flows captured during 66 days, between February 25, 2013 and May 1, 2013, which account for 35.69 GB of pure packet data. Section 2.3.4 describes in detail the properties of this dataset. The classes together with the number of flows and the data volume are shown in Table 2.6.

We have published this labeled dataset with full packet payloads in [59]. Therefore, it can be used by the research community as a reference benchmark for the validation and comparison of network traffic classifiers.

6.3 Performance Comparison

This section provides a detailed insight into the classification results of different types of traffic by each of the classifiers. All these results are summarized in Table 6.2, where the ratio of correctly classified flows (i.e., precision or true positives), incorrectly classified flows (i.e., errors or false positives) and unclassified flows (i.e., unknowns) are respectively presented. The complete confusion matrix can be found in our technical report [93].

Regarding the classification of P2P traffic, *Edonkey* is the first application studied. Only *PACE*, and especially *Libprotoident*, can properly classify it (precision over 94 %). *NDPI* and *OpenDPI* (that use the same pattern), as well as *NBAR*, can classify almost no *Edonkey* traffic (precision below 1 %). *L7-filter* classifies 1/3 of the flows, but it also produces many false positives by classifying more than 13 % of the flows as *Skype*, *NTP*, and *finger*. The wrongly classified flows in *NDPI* were labeled as *Skype*, *RTP* and *RTCP*, and in *NBAR* as *Skype*. The classification of *BitTorrent* traffic, the second P2P application studied, is not completely achieved by any of the classifiers. *PACE* and *Libprotoident* achieve again the highest precision (over 77 %). The rest of the classifiers present severe problems to identify this type of traffic. When misclassified, the *BitTorrent* traffic is usually classified as *Skype*.

The performance of most DPI tools with more traditional applications is significantly higher. FTP traffic is usually correctly classified. Only *L7-filter* and *NBAR* present problems to label it. The false positives produced by *L7-filter* are because the traffic is classified as *SOCKS*. Table 6.2 also shows that all the classifiers can properly classify *DNS* traffic. Similar results are obtained for *NTP*, which almost all the classifiers can correctly classify it. However, *NBAR* completely miss the classification of this traffic. *SSH* was evaluated in its *Linux* version. Table 6.2

shows that *NBAR* almost classified all the flows while the rest of classifiers labeled more than 95 % of them.

Similar performance is also obtained with *RDP*, usually employed by VoIP applications, as shown in Table 6.2. Again, *L7-filter* and *NBAR* can not classify this application at all. The false positives for *L7-filter*, *Libprotoident*, and *NBAR* are mainly due to *Skype*, *RTMP*, and *H323*, respectively.

Unlike previous applications, the results for *NETBIOS* are quite different. Surprisingly, *NBAR* and *NDPI* are the only classifiers that correctly label *NETBIOS* traffic. *PACE* can classify 2/3 of this traffic and *OpenDPI* only 1/4. On the other hand, the patterns from *L7-filter* and *Libprotoident* do not properly detect this traffic. The wrongly classified flows in *Libprotoident* are labeled as *RTP* and *Skype*, and in *L7-filter* as *Edonkey*, *NTP*, and *RTP*.

We also evaluated *RTMP* traffic, a common protocol used by browsers and plugins for playing *FLASH* content. It is important to note that only *Libprotoident* has a specific pattern for *RTMP*. Because of that, we have also counted as correct the *RTMP* traffic classified as *FLASH* although that classification is not as precise as the one obtained by *Libprotoident*. *L7-filter* and *NBAR* can not classify this type of traffic. The rest of the classifiers achieve a similar precision, around 80 %. The surprising amount of false positives by *NDPI* is because some traffic is classified as *H323*. *L7-filter* errors are due to wrongly classified traffic as *Skype* and *TSP*.

Table 6.2 also presents the results regarding the *HTTP* protocol. All of them but *L7-filter* can properly classify most of the *HTTP* traffic. *L7-filter* labels all the traffic as *finger* or *Skype*. *NDPI* classifies some *HTTP* traffic as *iMessage_Facetime*. The amount of errors from PACE is surprising, as this tool is usually characterized by very low false positive ratio. All the wrong classifications are labeled as *Meebo* traffic. The older *Meebo* pattern available in *OpenDPI* and the newer from *NDPI* seems not to have this problem.

Most incorrect classifications for all the tools are due to patterns that easily match random traffic. This problem especially affects L7-filter and, in particular, with the patterns used to match *Skype*, *finger* and *ntp* traffic. The deactivation of those patterns would considerably decrease the false positive ratio but it would disable the classification of those applications. In [42], the authors use a tailor-made configuration and post-processing of the L7-filter output in order to minimize this overmatching problem.

Table 6.2: DPI evaluation

Application	Classifier	% correct	% wrong	% uncl.
Edonkey	PACE	94.80	0.02	5.18
	OpenDPI	0.45	0.00	99.55
	L7-filter	34.21	13.70	52.09
	NDPI	0.45	6.72	92.83
	Libprotoident	98.39	0.00	1.60
	NBAR	0.38	10.81	88.81
BitTorrent	PACE	81.44	0.01	18.54
	OpenDPI	27.23	0.00	72.77
	L7-filter	42.17	8.78	49.05
	NDPI	56.00	0.43	43.58
	Libprotoident	77.24	0.06	22.71
	NBAR	27.44	1.49	71.07
FTP	PACE	95.92	0.00	4.08
	OpenDPI	96.15	0.00	3.85
	L7-filter	6.11	93.31	0.57
	NDPI	95.69	0.45	3.85
	Libprotoident	95.58	0.00	4.42
	NBAR	40.59	0.00	59.41
DNS	PACE	99.97	0.00	0.03
	OpenDPI	99.97	0.00	0.03
	L7-filter	98.95	0.13	0.92
	NDPI	99.88	0.09	0.03
	Libprotoident	99.97	0.00	0.04
	NBAR	99.97	0.02	0.02
NTP	PACE	100.00	0.00	0.00
	OpenDPI	100.00	0.00	0.00
	L7-filter	99.83	0.15	0.02
	NDPI	100.00	0.00	0.00
	Libprotoident	100.00	0.00	0.00
	NBAR	0.40	0.00	99.60
SSH	PACE	95.57	0.00	4.43
	OpenDPI	95.59	0.00	4.41
	L7-filter	95.71	0.00	4.29
	NDPI	95.59	0.00	4.41
	Libprotoident	95.71	0.00	4.30
	NBAR	99.24	0.05	0.70
RDP	PACE	99.04	0.02	0.94
	OpenDPI	99.07	0.02	0.91
	L7-filter	0.00	91.21	8.79
	NDPI	99.05	0.08	0.87
	Libprotoident	98.83	0.16	1.01
	NBAR	0.00	0.66	99.34
NETBIOS	PACE	66.66	0.08	33.26
	OpenDPI	24.63	0.00	75.37
	L7-filter	0.00	8.45	91.55
	NDPI	100.00	0.00	0.00
	Libprotoident	0.00	5.03	94.97
	NBAR	100.00	0.00	0.00
RTMP	PACE	80.56	0.00	19.44
	OpenDPI	82.44	0.00	17.56
	L7-filter	0.00	24.12	75.88
	NDPI	78.92	8.90	12.18
	Libprotoident	77.28	0.47	22.25
	NBAR	0.23	0.23	99.53
HTTP	PACE	96.16	1.85	1.99
	OpenDPI	98.01	0.00	1.99
	L7-filter	4.31	95.67	0.02
	NDPI	99.18	0.76	0.06
	Libprotoident	98.66	0.00	1.34
	NBAR	99.58	0.00	0.42

6.3.1 Sub-classification of HTTP traffic

Our dataset also allows the study of *HTTP* traffic at different granularity (e.g., identify different services running over *HTTP*). However, only *NDPI* can sub-classify some applications at this granularity (e.g., *Youtube*, *Facebook*). Newer

versions of *PACE* also provide this feature but we had no access to it for this study. Table 6.3 presents the results for four applications running over *HTTP* identified by *NDPI*. Unlike the rest of tools that basically classify this traffic as *HTTP*, *NDPI* can correctly give the specific label with precision higher than 97 %. Furthermore, the classification errors are caused by traffic that *NDPI* classifies as *HTTP* without providing the lower level label.

Table 6.3: HTTP sub-classification by *NDPI*

Application	% correct	% wrong	% unclassified
Google	97.28	2.72	0.00
Facebook	100.00	0.00	0.00
Youtube	98.65	0.45	0.90
Twitter	99.75	0.00	0.25

Another sub-classification that can be studied with our dataset is the *FLASH* traffic over *HTTP*. However, the classification of this application is different for each tool making its comparison very difficult. *PACE*, *OpenDPI* and *NDPI* have a specific pattern for this application. At the same time, these tools (as well as *L7-filter*) have specific patterns for video traffic, which may or may not run over *HTTP*. In addition, *NDPI* has specific labels for *Google*, *Youtube* and *Facebook* that can also carry *FLASH* traffic. *Libprotoident* and *NBAR* do not provide any pattern to classify *FLASH* traffic over *HTTP*. Table 6.4 shows that *NDPI* can correctly classify 99.48 % of this traffic, 25.48 % of which is classified as *Google*, *Youtube* or *Facebook*. *PACE* and *OpenDPI* can properly classify around 86 % of the traffic. The errors produced in the classification are almost always related to traffic classified as *HTTP* with the exception of *L7-filter* that classifies 86.49 % of the traffic as *finger*.

Table 6.4: FLASH evaluation

Classifier	% correct	% wrong	% unclassified
PACE	86.27	13.18	0.55
OpenDPI	86.34	13.15	0.51
L7-filter	0.07	99.67	0.26
NDPI	99.48	0.26	0.26
Libprotoident	0.00	98.07	1.93
NBAR	0.00	100.00	0.00

6.4 Lessons Learned and Limitations

This section extracts the outcomes from the results obtained during the performance comparison. Also, we discuss the limitations of our study. Table 6.5 presents

the summary of the results from Section 6.3. The *Precision* (i.e., first column) is computed similarly to Section 6.3, but we take into account all the applications together (i.e., 100 * # correctly classified flows / # total flows). However, this metric is dependent on the distribution of the dataset. Because of that, we also compute a second metric, the *Average Precision*. This statistic is independent from the distribution and is calculated as follow:

$$Avg.\ Precision = \frac{\sum_{i=1}^{N} \frac{correctly\ classified\ i\ flows}{total\ i\ flows}}{N} \tag{6.1}$$

where N is the number of applications studied (i.e., N = 10).

As it can be seen in Table 6.5, *PACE* is the best classifier. Even while we were not using the last version of the software, PACE was able to properly classify 94 % of our dataset. Surprisingly for us, *Libprotoident* achieves similar results, although this tool only inspect the first four bytes of payload for each direction. On the other hand, *L7-filter* and *NBAR* perform poorly in classifying the traffic from our dataset. The more fair metric, *Avg. Precision*, presents similar results. *PACE* is still the best classifier, however, it has increased the difference by several points to the second best classifier, *Libprotoident*. Unlike before, *NDPI* is almost as precise as *Libprotoident* with this metric. *L7-filter* and *NBAR* are still the tools that present the worst performance.

Table 6.5: Summary

Classifier	% Precision	% Avg. Precision
PACE	94.22	91.01
OpenDPI	52.67	72.35
L7-filter	30.26	38.13
NDPI	57.91	82.48
Libprotoident	93.86	84.16
NBAR	21.79	46.72

Nonetheless, the previous conclusions are obviously tied to our dataset. Although we have tried our best to emulate the real behavior of the users, many applications, behaviors and configurations are not represented on it. Because of that, it has some limitations. In our study we have evaluated 10 well-known applications, however adding more applications as *Skype* or *Spotify* is part of our ongoing future work. The results obtained from the different classifiers are directly related to those applications. Thus, the introduction of different applications could arise different outcomes. The traffic generated for building the dataset, although has been manually and realistically created, is artificial. The backbone traffic

would carry different behaviors of the applications that are not fully represented in our dataset (e.g., P2P clients running on port 80). Therefore, the performance of the tools studied could not be directly extrapolated from the current results, but it gives an idea of their precision in the evaluated set of applications. At the same time, the artificially created traffic allowed us to publish the dataset with full packet payloads.

6.5 Related Work

Some previous works evaluated the accuracy of DPI-based techniques [45,46,95,96]. These studies rely on a ground-truth generated by another DPI-based tool [46], port-based technique [45] or a methodology of unknown reliability [95,96], making their comparison very difficult. Recently, a concomitant study to ours [96] compared the performance of four DPI-based techniques (i.e., *L7-filter*, *Tstat*, *NDPI* and *Libprotoident*). This parallel study confirms some of the findings of our work presenting *NDPI* and *Libprotoident* as the most accurate open-source DPI-based techniques. In [97] the reliability of *L7-filter* and a port-based technique was compared using a dataset obtained by GT [98] showing that both techniques present severe problems to accurately classify the traffic.

To the best of our knowledge, just one work has tackled the problem of the lack of publicly available labeled datasets. Gringoli et al. in [98] published anonymized traces without payload, but accurately labeled using GT. This dataset is very interesting to evaluate Machine Learning-based classifiers, but the lack of payload makes it unsuitable for DPI-based evaluation.

6.6 Chapter Summary

This work presents the first step towards validating the reliability of the accuracy of the network traffic classifiers. We have compared the performance of six tools (i.e., *PACE*, *OpenDPI*, *L7-filter*, *NDPI*, *Libprotoident*, and *NBAR*), which are usually used for the traffic classification. The results obtained in Section 6.3 and further discussed in Section 6.4 show that *PACE* is, on our dataset, the most reliable solution for traffic classification. Among the open-source tools, *NDPI* and especially *Libprotoident* present the best results. On the other hand, *NBAR* and *L7-filter* present several inaccuracies that make them not recommendable as a ground-truth generator.

In order to make the study trustworthy, we have created a dataset using VBS [50]. This tool associates the name of the process to each flow making its labeling totally reliable. The dataset of more than 500 K flows contains traffic from popular applications like *HTTP*, *Edonkey*, *BitTorrent*, *FTP*, *DNS*, *NTP*, *RDP*, *NETBIOS*, *SSH*, and *RDP*. The total amount of data properly labeled is 32.61 GB. Furthermore, and more important, we release to the research community this dataset with full payload, so it can be used as a common reference for the comparison and validation of network traffic classifiers.

Chapter 7

Other Improvements to Traffic Classifiers

This chapter collects other contributions achieved in parallel with the main theme of this thesis. Section 7.1 presents the study of an efficient profiled flow termination timeout. Although this study is directly related to network traffic classification field, its application is not limited to this topic. Most monitoring tools continuously keep the state of the connections (i.e., flows) of the network monitored. An efficient implementation of this process is crucial in scenarios with limited resources and huge amount of connections. As a result, a proper mechanism for expiration of the flows is a key point in the optimization of this costly process. Section 7.1 studies a mechanism for flow termination by group of applications with current Internet traffic. From that characterization we propose an expiration technique to achieve optimized timeouts in a profiled way.

Section 7.2 address the classification problem from a different point of view. The last contribution of this thesis proposes an early classification technique able to classify the traffic just using the size of the first packets of the flows. This early classification feature is very important, since it allows network operators to quickly react to the classification results (e.g., traffic shaping). In addition, we present the prototype of the system for continuous network traffic classification described in Chapter 4.

7.1 Efficient Profiled Flow Termination Timeouts

7.1.1 Introduction

The increase of bandwidth and diversity of protocols on the Internet has made the online traffic classification a highly difficult and exigent task. In current networks (e.g. 10/40/100 Gbps) packets are received every few ns, and that is the time available for parsing them without requiring data buffering. However, many proposed traffic classifiers require to store statistics or even the actual content of the packets for the classification at flow level. Furthermore, it is usually necessary to keep the state of each flow in order to properly classify it. Because of this, the operations involved in this process have to be carried out very efficiently in terms of memory and CPU.

In this section we focus our attention in the optimization of the memory consumption. A crucial issue in this process is the expiration of the structure that keeps the state of each flow. On the one hand, expiring the flows too late incredibly increase the amount of memory necessary. On the other hand, expiring the flows very quickly would create segmented flows with less information, making the accuracy of classification more difficult or even impossible. As a result, a proper expiration of the flows is a key point in the optimization of this costly process. So far, this process has been usually carried out using timeouts or specific protocol mechanisms (e.g., TCP termination handshake).

The first contribution of this work is a comprehensive study of the flow termination by group of applications with current Internet traffic. The study shows that the non-desired flow terminations (e.g., RST, undefined termination) are considerably common in the current traffic. Furthermore, we have detected different behaviors related to the flow termination among the different groups of applications. From that characterization we have gone further and proposed an expiration technique to achieve optimized timeouts in a profiled way. The proposed method has been evaluated using the PACE engine [9] (a well-known commercial DPI tool) achieving a substantial reduction of memory while maintaining the accuracy and the CPU consumption.

The rest of the work is organized as follows. Section 7.1.2 describes the traces used. Section 7.1.3 briefly presents the methodology used to extract the results. Section 7.1.4 presents the statistical results, and based on them, the estimation and evaluation of the timeouts is done. Section 7.1.5 reviews the related work.

Finally, Section 7.1.6 concludes the study.

7.1.2 Dataset

The traces used for the evaluation were recorded in two different network scenarios and they provide a good comparative basis for the results. These traces were taken under confidential agreements, which means that specific details can not be revealed. In Table 7.1 are synthesized some of their properties.

ISP_Core This trace was recorded on 2009 in an internal link of a Tier-1 ISP, after the access and aggregation sections and before the output router to the backbone. This type of networks are often designed with multiple links due to balancing and failure issues. As a consequence of that, many of the traffic recorded is asymmetric.

ISP_Mob This trace is a good complement for the `ISP_Core` one since it was recorded in a mobile operator network. It was taken somewhere between the SGSN and the GGSN, elements of a standard GPRS network. As it is near to the edge, this trace contains a high proportion of symmetric traffic. The transmission of the data is done over GTP, however, PACE includes features for decapsulation. It was recorded on 2010.

Table 7.1: Traces properties

	Duration	Size (MB)	Packets	Flows
ISP_Core	38,160 s	2,591,636	7,074,618,384	295,729,886
ISP_Mob	2,700 s	133,046	233,359,695	6,093,604

7.1.3 Methodology

The main tool employed in this study is based on the Ipoque's PACE engine [9], a commercial DPI library. Thus, the classification in several protocol groups has the reliability of this product (i.e., nearly 100% detection rate of protocols and applications with no false positive). The protocol groups considered are the following: `generic`, `p2p`, `gaming`, `tunnel`, `voip`, `im`, `streaming`, `mail` `network_management`,

`filetransfer` and `web`. The `generic` group is mainly composed by unknown traffic. A detailed classification can be found in [99].

In order to derive the set of timeouts for the optimization of the expiration process we have developed a module over PACE that calculates different statistics by group of applications. Although this module computes different statistics for each flow, the most important for our purpose are the times between packets, named PCK-PCK, and the times between packets with data, named DAT-DAT. The difference among them is that the PCK-PCK times include the TCP's acknowledgment packets (ACK) while the DAT-DAT times include only the packets with application data. All the evaluations are done both globally and considering each of the Client-Server and Server-Client direction. In addition, we also study the termination processes of the different group of applications for the TCP traffic. The terminations detected are the standard FIN process, the RST process, the RST process in a blocking scenario and the undefined. These termination modes are further described in Section 7.1.4.

To avoid, as far as possible, the inclusion of worthless data from our traces the following considerations are taken. For TCP traffic, only those flows showing the first step of the 3 way-handshake (i.e., the SYN-SYNACK process) are parsed. Doing so, we avoid considering ongoing flows initiated before the beginning of the traces and we only consider bidirectional flows. On the other hand, due to its intrinsic characteristics, for UDP traffic the only condition is to see at least two packets (otherwise any PCK-PCK time could not be calculated). In Table 7.2 can be seen the impact of applying these considerations on each trace.

Table 7.2: Flow usage

	TCP	UDP	TCP used	UDP used
ISP_Core	159,444,165	127,930,249	42,521,933	55,182,658
ISP_Mob	3,904,103	2,063,391	3,454,210	1,850,579

The establishment of Client-Server or Server-Client direction is easy for TCP traffic, the side that starts the connection (i.e., SYN message) is considered as the client. On the other hand, for UDP traffic the consideration is not so straightforward. First it is checked if PACE can determine the direction based on the protocol information. If it is not able, or the direction is either Client-Client or Server-Server, we assume that the first packet seen comes from the client.

7.1.4 Results

The results obtained in this study are presented in two parts. First, the results observed for the PCK-PCK and DAT-DAT times and the proportion of the different types of flow termination are discussed. Second, using these previous results, a set of timeouts by application group are derived for an efficient flow expiration.

Additionally, we show in Table 7.3 the traffic distribution of the traces used in the evaluation. The percentages of each trace can be seen for both the TCP and UDP traffic. The heterogeneous composition of these two traces from completely different scenarios supports the representativeness of the results of our study. Since we have studied the already cited protocol groups (see Section 7.1.3), the excess traffic is grouped as `other`.

Table 7.3: Flow proportions

protocol	TCP Core	TCP Mob	UDP Core	UDP Mob
generic	12.57%	10.02%	14.16%	13.22%
p2p	5.98%	2.57%	13.81%	13.21%
gaming	0.02%	0.00%	0.03%	0.08%
tunnel	10.66%	9.29%	0.02%	0.30%
voip	0.84%	0.07%	1.94%	1.05%
im	10.60%	0.46%	0.35%	0.05%
streaming	1.07%	0.71%	58.33%	0.42%
mail	14.17%	1.50%	0.00%	0.00%
management	0.01%	0.01%	11.35%	69.93%
filetransfer	0.69%	0.30%	0.00%	0.00%
web	42.98%	74.69%	0.00%	0.00%
other	0.41%	0.38%	0.01%	1.74%

Time profiled study

Inter-packet times evaluation This section presents the results related to the PCK-PCK and DAT-DAT times (see Section 7.1.3) measured for both the TCP and UDP flows. They will be fundamental to establish the flow termination timeouts, especially for UDP traffic, as further explained in Section 7.1.4. Additionally,

some interesting outcomes are extracted from the statistics obtained.

- **Similar PCK-PCK times over TCP regardless of the scenario:** Excluding the `network_management` group, there is a strong correlation comparing the PCK-PCK values between the `ISP_Core` (see Table 7.4) and `ISP_Mob` (see Table 7.5). This is also accomplished for the DAT-DAT values, but not for the UDP traffic (which is always DAT-DAT). This result suggests that the average packet cadence over TCP is independent on the network.

- **PCK-PCK similarities in Client-Server and Server-Client:** In both scenarios and for the TCP (see Tables 7.4 and 7.5) and the UDP (see Table 7.6) traffic there are similar PCK-PCK values comparing the Client-Server and Server-Client directions. In the UDP case, it could be expected to see an asymmetric behavior in some protocol groups like the `streaming` as it happens in the DAT-DAT values over TCP (see Table 7.7). This highlights the different role that it takes over UDP.

- **TCP and UDP usage differences:** UDP flows for some protocol groups in the `ISP_Core` show (see Table 7.6) a lower or higher packet transmission rate than the DAT-DAT seen over TCP (see Table 7.7), or as just explained for the `streaming` group it has an unexpected behavior. That is justified by the usage of control flows over TCP or UDP by some protocols. For example, this behavior is seen in the `p2p`, where the DAT-DAT times are around 4 times lower in the TCP case (probably real data transfer) than in the UDP (probably control flows) one. This behavior also occurs but in the opposite way for the `gaming`, `tunnel`, `voip` and `im` groups, which would imply that in this case the real stream of data is sent over UDP while the TCP flows are the control ones. This behavior is similar in the `ISP_Mob` trace, with the main exception seen for the `streaming` group since in this case the UDP traffic seems to carry as well real traffic (low value for the PCK-PCK times).

Flow termination evaluation In order to obtain the set of profiled timeouts we calculate not only the timing parameters but also the proportion of the different flow termination types. This study is carried out for the TCP traffic given its different termination mechanism. Four possible terminations are observed: TCP handshake, RST, undefined termination and RST in a blocking scenario. Fig. 7.1 shows the behavior of the termination of a RST in a blocking scenario. This is

Table 7.4: PCK-PCK times TCP `ISP_Core` (ms)

group	avg	std	avg c-s	std c-s	avg s-c	std s-c
generic	1,510	12,860	2,760	17,840	2,720	16,740
p2p	1,196	7,084	2,216	9,201	2,217	9,593
gaming	209	1,319	448	1,777	384	1,732
tunnel	1,029	11,080	2,004	15,412	1,737	14,706
voip	1,688	9,981	3,161	13,677	3,450	14,137
im	2,715	13,356	5,240	17,857	5,302	18,390
stream	120	3,200	286	4,842	181	3,871
mail	518	8,259	802	10,435	1,040	11,959
manage	1,494	12,774	2,430	17,382	3,313	19,552
filetx	390	36,037	855	55,946	647	46,807
web	1,127	13,383	2,294	18,757	1,827	17,586

when one or both extremes of the communication do not receive the corresponding packets from the other side.

Unlike we expected, as we can see in Fig. 7.2, the number of flows not finished by the standard TCP handshake is very high. Additionally, there is a big amount of flows terminating with a RST in a blocking scenario. In these situations the retransmission times grow exponentially [100, 101]. Thus, we have particularly studied the times for the flows finished with a RST in a blocking scenario (i.e., time between a RST and the last packet of the flow, and PCK-PCK time). They are shown in Table 7.8. Notice that these PCK-PCK times are generally bigger than the standard ones (see Table 7.4). The results for the `ISP_Mob` are very similar and are omitted for the sake of space. These results can find a small comparison basis in [102].

Estimation and evaluation of profiled timeouts

Once studied the times between packets and flow terminations by group of applications we can proceed with the study of the profiled timeouts. A flow timeout has to be big enough to not consider new flows with a late packet of an existing one, but as low as possible to release the memory as soon as possible. An old previous study already defined the use of a global timeout of 64 seconds. [103].

Table 7.5: PCK-PCK times TCP `ISP_Mob` (ms)

group	avg	std	avg c-s	std c-s	avg s-c	std s-c
generic	1,796	14,293	3,043	18,868	3,526	19,840
p2p	1,907	8,660	3,475	11,648	3,579	13,058
gaming	109	824	139	946	473	2,520
tunnel	1,396	20,782	2,778	29,860	2,414	27,576
voip	1,976	11,315	3,744	15,264	3,892	15,843
im	1,897	15,667	3,587	21,663	3,894	22,373
stream	87	2,269	216	3,564	117	2,870
mail	1,057	13,678	2,135	20,005	1,957	18,696
manage	316	4,374	604	6,055	682	6,527
filetx	427	9,819	700	13,992	808	14,469
web	704	7,349	1,364	9,793	1,117	8,861

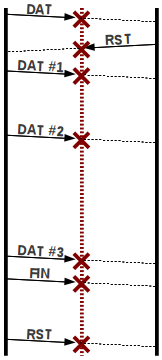


Figure 7.1: Blocking scenario behavior. Both sides send data, but since there is no acknowledgment from the other side they try to retransmit (after a time which grows in every attempt) until finally break the connection with a RST

However, this value has lost its reliability because of the evolution of the different applications on the Internet. In order to address this problem we have found a way to establish it according to different protocol groups by using the statistics obtained, basically the PCK-PCK times. For TCP traffic, also are considered the times regarding the PCK-PCK after a RST in a blocking scenario (see Table 7.8) and the proportion of the different terminations. It is important since in these

Table 7.6: PCK-PCK times UDP `ISP_Core` (ms)

group	avg	std	avg c-s	std c-s	avg s-c	std s-c
generic	2,878	23,551	3,427	25,744	4,719	27,850
p2p	17,310	64,100	26,099	77,177	18,017	70,828
gaming	199	2,997	295	3,816	374	3,948
tunnel	454	7,215	572	8,284	1,061	12,091
voip	306	7,888	509	10,211	404	10,430
im	151	1,471	209	1,829	198	3,171
stream	4,375	37,134	7,068	47,118	5,846	43,562
manage	11,161	55,806	20,342	73,220	21,947	79,760

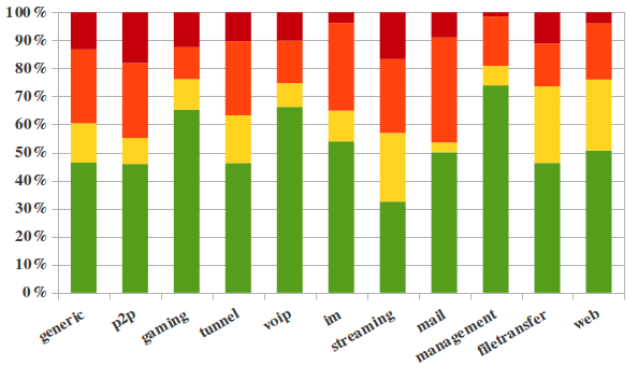


Figure 7.2: Termination proportions `ISP_Core`, green stands for the standard FIN process, yellow for the unclosed flows, red for the RST and intense red for the RST in a blocking scenario

situations the time of inactivity grows exponentially and forces a high timeout.

The selection of a threshold for the timeouts is done considering the corresponding statistical results, and it is set up as $\{average + 2.5 * STD\}$[1]. By using this method, unusual big measures are excluded, while still representing around 99% of the complete time values. For each trace and separately for the TCP and UDP flows we choose the biggest threshold from either the Client-Server or the Server-Client direction. Thereby, we consider always the worst case, making it valid as well when dealing with unidirectional flows. Thus, we define these thresh-

[1]STD = Standard deviation

Table 7.7: DAT-DAT times TCP `ISP_Core` (ms)

group	avg	std	avg c-s	std c-s	avg s-c	std s-c
generic	2,137	14,585	8,673	41,852	2,566	16,870
p2p	1,772	8,077	3,763	14,747	2,741	11,162
gaming	372	1,521	1,020	4,274	574	2,271
tunnel	1,334	13,057	3,465	24,198	1,584	23,618
voip	2,656	12,609	5,571	19,021	4,825	18,193
im	4,321	16,135	12,360	37,903	6,167	38,115
stream	136	2,960	3,230	18,987	137	3,055
mail	746	10,423	826	11,717	1,945	19,036
manage	1,278	10,442	7,825	33,821	2,634	13,823
filetx	533	46,120	2,944	130,181	589	49,281
web	880	9,269	4,714	24,456	911	9,843

olds as $\widehat{t_{pck\_pck}}$ for the PCK-PCK standard times and $\widehat{t_{pck\_blk}}$ and the PCK-PCK times after a RST in a blocking scenario.

We have seen that $\widehat{t_{pck\_blk}}$ times are forcing a higher value when considering the optimal timeout. For that reason, we have used these values to round off the final results, making them closer to the $\widehat{t_{pck\_pck}}$ times. The criteria used is based in two aspects. On the one hand, we have considered which proportion of the traffic is behaving as in a blocking scenario. On the other, we have also studied the CDF (Cumulative Distribution Function) of the number of packets in relation to the timeout for each group to see what is its variability. That is, what is the effect of reducing the timeout in the proportions of values inside the $\widehat{t_{pck\_blk}}$ interval. Fig. 7.3 shows the three groups we have detected. A strong variation is seen for the `p2p`, `voip`, `streaming` and `network_management` groups. A medium variation for the `generic`, `tunnel`, `im`, `mail` and `web` groups. Finally, a slow variation is seen for the `gaming` and `filetransfer` groups.

It is difficult to include these behaviors in the calculation of the final timeout. Notwithstanding, we have done it applying a factor, named F, which modifies the final timeout taking into account the behavior above described. This factor is set up as $\{0.33, 0.66, 1\}$ for the slow, medium and strong variation respectively. Thus, we have considered that for TCP traffic a good theoretical timeout could be approximated as follows:

Table 7.8: RST times `ISP_Core` (ms)

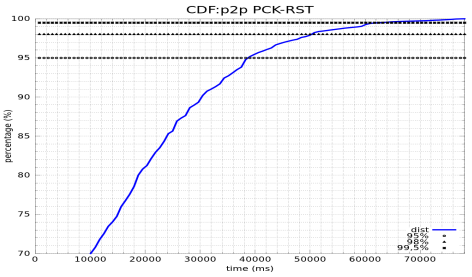
group	avg rst-end	std rst-end	avg pck	std pck
generic	34,276	138,999	6,722	33,147
p2p	15,095	53,451	6,354	18,664
gaming	57,834	128,352	765	5,673
tunnel	13,618	96,933	5,618	33,423
voip	20,311	91,428	4,753	29,569
im	8,819	68,851	1,964	17,671
stream	7,449	50,738	344	4,060
mail	29,384	126,616	9,357	38,504
manage	182	259	157	239
filetx	11,967	86,210	3,768	45,231
web	11,534	59,228	1,996	13,051

$$t_{timeout} = (1 - fin_{blk}) * \widehat{t_{pck\_pck}} + fin_{blk} * \widehat{t_{pck\_blk}} * F \tag{7.1}$$

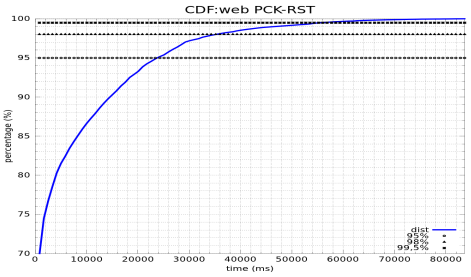
$$F = \{0.33, 0.66, 1\} \tag{7.2}$$

Remember the $\widehat{t_{pck\_blk}}$ regards the RST times in a blocking scenario. Therefore, the proportion fin_{blk} is as well related to that field, but concerning the proportion of termination modes. Both parameters are considered getting the worst case value among the `ISP_Core` and the `ISP_Mob` (i.e., the case where the timeout is bigger), so we intend to find out a global timeout independent of the traces evaluated. By applying this formula we consider the $\widehat{t_{pck\_blk}}$ according to its proportion of appearance in terms of flows termination and also the behavior seen from the CDF's. The parameters used for this calculation and the results are shown in Table 7.9. For UDP traffic no rounding off has been done since the only parameter to consider is the $\widehat{t_{pck\_pck}}$.

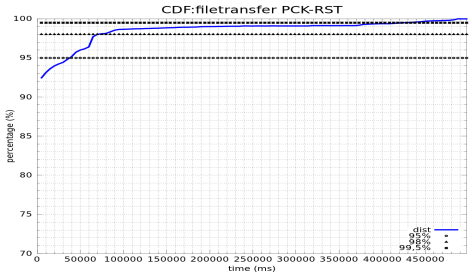
In addition, we have gone further and we have evaluated the impact of these outcomes by means of the Ipoque's PACE engine [9]. Thanks to that, we have been able to measure not only the memory saving but also the detection rate impact. However, due to technical limitations in the PACE engine we have not been able to set up each timeout in a profiled way for each protocol group. Notwithstanding,



(a) Strong variation (P2P)



(b) Medium variation (Web)



(c) Slow variation (Filetransfer)

Figure 7.3: CDF of inter-packet times with RST termination in a blocking scenario

we have elaborated a method to see what is the effect of varying the timeout on each group. To that purpose, we have measured for a set of intervals from 20 to 300 seconds, in steps of 20 seconds, what are the number of flows detected by PACE according to each protocol group. We have also added the values for 600 seconds to have a wide timeout to compare with. Then we have plotted (see Fig. 7.4) the proportion of flows per timeout compared with the maximum number of flows seen per protocol group (when the timeout is 20 seconds). Therefore, we can get a curve to see when the number of flows per protocol group tends to be stable.

It is only evaluated the `ISP_Core` trace since it is more representative. However, the results regarding the `ISP_Mob` trace show similarities. It has to be considered that for the FTP (File Transfer Protocol) inside the `filetransfer` group the timeout is directly established by PACE and its operation is independent from the one we vary. That is why in Fig. 7.4 it is not seen a wide variation for this group. Also, it is important to comment the effect on the `voip` and lesser extent in the `p2p` group, since at some point the variation increases, hindering the evaluation

Table 7.9: Timeout TCP (ms)

group	$\widehat{t_{pck\_pck}}$	$\widehat{t_{pck\_blk}}$	fin_{blk}	F	$t_{timeout}$
generic	53,126	117,269	0.1326	0.66	56,344
p2p	36,255	78,086	0.181	1	43,826
gaming	6,772	26,440	0.1246	0.33	7,015
tunnel	77,426	119,298	0.1244	0.66	77,589
voip	43,500	90,671	0.1008	1	48,255
im	59,826	163,225	0.0787	0.66	63,596
stream	12,392	35,098	0.3653	1	20,687
mail	52,147	127,211	0.1011	0.66	55,363
manage	52,192	76,480	0.1018	1	54,665
filetx	140,720	497,736	0.291	0.33	147,568
web	47,804	83,657	0.0428	0.66	48,121

of our theoretical timeouts. This happens due to internal detection methods of PACE. With the exception of the `generic` group (remember that it contains the unknown traffic) and the already commented situations, and considering that the evaluation is done over all the traffic tighter (i.e., TCP and UDP), the theoretical results are roughly accomplished.

Similarly, we have also checked a global timeout by measuring what is the direct impact in the memory requirements, the detection rate and the performance of PACE. We can see in Fig. 7.5 that for a certain timeout both the detection rate and the total number of flows (which is directly related to the performance) tend to be stable. Comparing a timeout in which the detection rate and number of flows stabilizes (approximately 200 seconds) with 600 seconds (this was the reference timeout implemented by PACE), we achieve a reduction of around 60% of memory necessities.

7.1.5 Related work

To the best of our knowledge, this is the first work that presents a comprehensive study about the flow termination by group of applications. Some papers in the literature are related to this work but in a simplified way and considering just few parameters. In [102] it is intended to find out what is the influence of different

Table 7.10: Timeout UDP (ms)

group	$t_{timeout}$
generic	74,343
p2p	219,043
gaming	45,584
tunnel	40,814
voip	71,636
im	14,030
streaming	124,862
management	221,346

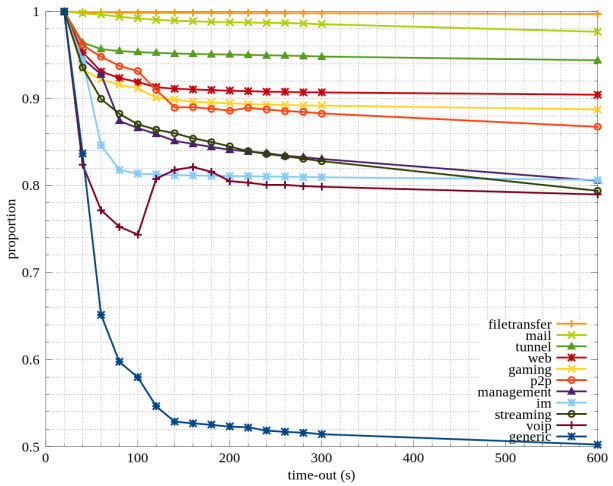


Figure 7.4: Timeout PACE evaluation `ISP_Core`

types of traffic on the Internet in terms of volume of data and establishment and termination behavior. Notwithstanding, the classification only covers P2P and HTTP traffic. Concerning the statistical study of different protocol groups, the somehow related papers with the findings and objectives of this work are either old or sharing just a few similarities, so they are not considered. The only exception is [103]. Although this paper is quite old (1995) it establishes a single timeout which can be compared to this work findings. The use of a single timeout for the

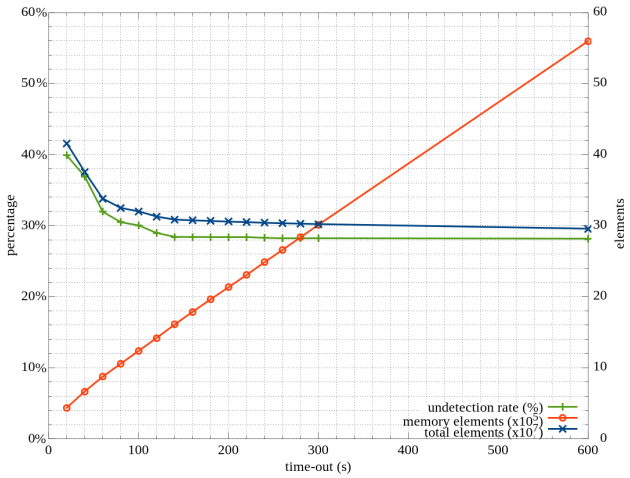


Figure 7.5: Timeout PACE performance `ISP_Core`. Elements stand for flows

expiration of the flows is a mechanism commonly used by the traffic classification techniques proposed in the literature [35, 42].

7.1.6 Chapter Summary

Nowadays Internet bandwidth has forced online traffic classifiers to be implemented very efficiently. Among the different constraints that online traffic classification present, this section focuses on the optimization of the memory consumption. A key issue in this aspect is the expiration of the structures that keep the state of each flow. This work proposes a new methodology for an efficient expiration of the flows based on profiled timeouts. In order to obtain these timeouts it has been studied the behavior of the traffic by group of applications. Unexpected results have been observed regarding the proportion and types of flow termination in the current traffic.

The profiled timeouts obtained have been evaluated in a well-known commercial DPI tool (the Ipoque's PACE engine), achieving a drastic reduction of memory while keeping the computation cost and classification accuracy. The results suggest that the implementation of the profiled timeouts in the traffic classification techniques proposed in the literature [36] could considerably improve their memory consumption, alleviating by this, their feasibility for online classification.

7.2 Early Classification of Network Traffic

7.2.1 Introduction

Gaining information about the applications that generate traffic in an operational network is much more than mere curiosity for network operators. Traffic engineering, capacity planning, traffic management or even usage-based pricing are some examples of network management tasks for which this knowledge is extremely important. Although this problem is still far from a definitive solution, the networking research community has proposed several machine learning (ML) techniques for traffic classification that can achieve very promising results in terms of accuracy. However, in practice, most network operators still use either obsolete (e.g., port-based) or unpractical (e.g., pattern matching) methods for traffic identification and classification. One of the reasons that explains this slow adoption by network operators is the time-consuming training phase involving most ML-based methods, which often requires human supervision and manual inspection of network traffic flows.

In this work, we revisit the viability of using the well-known Nearest Neighbor (NN) machine learning technique for traffic classification. As we will discuss throughout the sections, this method has a large number of features that make it very appealing for traffic classification. However, it is often discarded given its poor classification speed [33,35]. In order to address this practical problem, we present an efficient implementation of the NN search algorithm based on a K-dimensional tree structure that allows us not only to classify network traffic online with high accuracy, but also to retrain the classifier on-the-fly with minimum overhead, thus lowering the barriers that hinder the general adoption of ML-based methods by network operators.

Our K-dimensional tree implementation only requires information about the length of the very first packets of a flow. This solution provides network operators with the interesting feature of *early classification* [19,20]. That is, it allows them to rapidly classify a flow without having to wait until its end, which is a requirement of most previous traffic classification methods [15,25,29]. In order to further increase the accuracy of our method along with its classification speed, we combine the information about the packet sizes with the relevant data still provided by the port numbers [35].

We present an actual implementation of our method based on the Traffic Identification Engine (TIE) [104]. TIE is a community-oriented tool for traffic classification that allows multiple classifiers (implemented as plugins) to run concurrently

and produce a combined classification result.

Given the low overhead imposed by the training phase of our method and the plugins already provided by TIE to set the ground truth (e.g., L7 plugin), our implementation has the unique feature of continuous training. This feature allows the system to automatically retrain itself as the training data becomes obsolete. We hope that the large advantages of our method (i.e., accuracy ($>$ 95%), classification speed, early classification and continuous training) can give an incentive to network operators to progressively adopt new and more accurate ML-based methods for traffic classification.

The remainder of this work is organized as follows. Section 7.2.2 describes our ML-based method based on TIE. Section 7.2.3 analyzes the performance of our method and presents preliminary results of its continuous training feature. Section 7.2.4 reviews the related work. Finally, Section 7.2.5 concludes the work and outlines our future work.

7.2.2 Methodology

This section describes our ML-based classification method based on multiple K-dimensional trees, together with its continuous training system. We also introduce TIE, the traffic classification system we use to implement our technique, and the modifications made to it in order to allow our method to continuously retrain itself.

Traffic Identification Engine

TIE [104] is a modular traffic classification engine developed by Universitá di Napoli Federico II. This tool is designed to allow multiple classifiers (implemented as plugins) to run concurrently and produce a combined classification result. In this work, we implement our traffic classification method as a TIE plugin.

TIE is divided in independent modules that are in charge of the different classification tasks. The first module, *Packet Filter*, uses the Libpcap library to collect the network traffic. This module can also filter the packets according to BPF or user-level filters (e.g., skip the first n packets, check header integrity or discard packets in a time range). The second module, *Session Builder*, aggregates packets in flows (i.e., unidirectional flows identified by the classic 5-tuple), biflows (i.e., both directions of the traffic) or host sessions (aggregation of all the traffic of a host). The *Feature Extractor* module calculates the features needed by the classification plugins. There is a single module for feature extraction in order to avoid redundant calculations for different plugins. TIE provides a multi-classifier engine

divided in a *Decision Combiner* module and a set of classification plugins. On the one hand, the *Decision Combiner* is in charge of calling several classification plugins when their features are available. On the other hand, this module merges the results obtained from the different classification plugins in a definitive classification result. In order to allow comparisons between different methods, the *Output* module provides the classification results from the $Classification\ Combiner$ based on a set of applications and groups of applications defined by the user.

TIE supports three different operating modes. The $offline\ mode$ generates the classification results at the end of the TIE execution. The *real-time mode* outputs the classification results as soon as possible, while the *cycling mode* is an hybrid mode that generates the information every n minutes.

KD-Tree plugin

In order to evaluate our traffic classification method, while providing a ready-to-use tool for network operators, we implement the K-dimensional tree technique as a TIE plugin. Before describing the details of this new plugin, we introduce the K-dimensional tree technique. In particular, we focus on the major differences with the original NN search algorithm.

The K-dimensional tree is a data structure to efficiently implement the Nearest Neighbor search algorithm. It represents a set of N points in K-dimensional spaces as described by Friedman et al. [105] and Bentley [106]. These spaces are maintained in a binary tree in which each non-leaf node generates a hyperplane that divides the spaces into two subspaces.

The original NN algorithm searches iteratively the nearest point i, from a set of points E, to a point p. In order to find the i point, it computes, for each point in E, the distance (e.g., Euclidean or Manhattan distance) to the point p. Likewise, if we are performing a K-NN search, the algorithm looks for the K i points nearest to the point p. This search has O(N) time complexity and becomes unpractical with the amount of traffic found in current networks.

On the contrary, the search in a K-dimensional tree allows to find in average the nearest point in O(log N), with the additional cost of spending once O(N log N) building the binary tree. Besides this notable improvement, the structure also supports approximate searches, which can substantially improve the classification time at the cost of producing a very small error.

The K-dimensional tree plugin that we implement in TIE is a combination of the K-dimensional tree implemented in the C++ ANN library and a structure to represent the relevant information still provided by the port numbers. In partic-

ular, we create an independent K-dimensional tree for each *relevant* port. We refer as *relevant* ports as those that generate more traffic. Although the list of *relevant* ports can be computed automatically, we also provide the user with the option of manually configuring this list. Another configuration parameter is the approximation value, which allows the method to improve its classification speed by performing an approximate NN search. In the evaluation, we set this parameter to 0, which means that this approximation feature is not used. However, higher values of this parameter could substantially improve the classification time in critical scenarios, while still obtaining a reasonable accuracy.

Unlike in the original NN algorithm, the proposed method requires a lightweight training phase to build the K-dimensional tree structure. Before building the data structure, a sanitization process is performed on the training data. This procedure removes the instances labeled as unknown from the training dataset assuming that they have similar characteristics to other known flows. This assumption is similar to that of ML clustering methods, where unlabeled instances are classified according to their proximity in the feature space to those that are known. The sanitization process also removes repeated or indistinguishable instances.

The traffic features used by our plugin are the destination port number and the length of first n packets of a flow (without considering the TCP handshake). By using only the first n packets, the plugin can classify the flows very quickly, providing the network operator with the possibility of quickly reacting to the classification results. In order to accurately classify short flows, the training phase also accepts flows with less than n packets by filling the empty positions with null coordinates.

Continuous training system

In this section, we show the interaction of our *KD-Tree* plugin with the rest of the TIE architecture, and describe the modifications done in TIE to allow our plugin to continuously retrain itself.

Figure 7.6 shows the data flow of our continuous training system based on TIE. The first three modules are used without any modification as found in the original version of TIE. Besides the implementation of the new *KD-Tree* plugin, we significantly modified the *Decision Combiner* module and the *L7* plugin.

Our continuous training system follows the original TIE operation mode most part of the time. Every packet is aggregated in bidirectional flows while its features are calculated. When the traffic features needed by our plugin are available (i.e., n first packet sizes or the flow expires), the flow is classified by the *KD-Tree*

plugin. Although the method was tested with bidirectional flows, the current implementation also supports the classification of unidirectional flows.

In order to automatically retrain our plugin, as the training data becomes obsolete, we need a technique to set the ground-truth. TIE already provides the *L7* plugin, which implements a DPI technique originally used by TIE for validation purposes. We modified the implementation of this plugin to continuously produce training data (which includes flow labels - that is, the ground-truth - obtained by L7) for future trainings. While every flow is sent to the *KD-Tree* plugin through the main path, the *Decision Combiner* module applies flow sampling to the traffic, which is sent through a secondary path to the *L7* plugin. This secondary path is used to (*i*) set the ground-truth for the continuous training system, (*ii*) continuously check the accuracy of the *KD-Tree* plugin by comparing its output with that of *L7*, and (*iii*) keep the required computational power low by using flow sampling (performing DPI on every single flow will significantly decrease the performance of TIE).

The *Decision Combiner* module is also in charge of automatically triggering the training of the *KD-Tree* plugin according to three different events that can be configured by the user: after p packets, after s seconds, or if the accuracy of the plugin compared to the *L7* output is below a certain threshold t. The flows classified by the *L7* plugin, together with their features (i.e., destination port, n packet sizes, L7 label), are placed in a queue. This queue keeps the last f classified flows or the flows classified during the last s seconds.

The training module of the *KD-Tree* plugin is executed in a separate thread. This way, the *KD-Tree* plugin can continuously classify the incoming flows without interruption, while it is periodically updated. In addition, it is possible to automatically update the list of *relevant* ports by using the training data as a reference.

7.2.3 Results

This section presents the performance evaluation of the proposed technique. First, Subsection 7.2.3 describes the dataset used for the experiment. Subsection 7.2.3 performs a comparison between the original Nearest Neighbor algorithm and the K-dimensional tree implementation. Subsection 7.2.3 presents a performance evaluation of the proposed plugin described in Subsection 7.2.2 evaluating different aspects of the technique as the *relevant* ports or the number of packet sizes used for the classification. Finally, Subsection 7.2.3 presents a preliminary study of the impact of the continuous training system in the traffic classification.

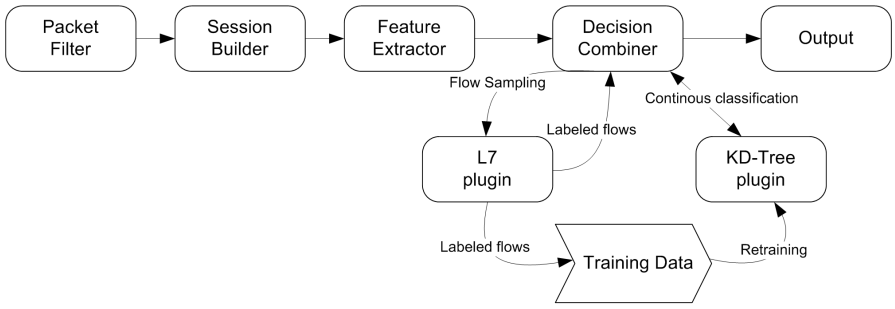


Figure 7.6: Diagram of the Continuous Training Traffic Classification system based on TIE

Evaluation Datasets

The evaluation dataset used in our performance evaluation consists of 8 full-payload traces collected at the Gigabit access link of the Universitat Politècnica de Catalunya (UPC). This dataset is derived from the UPC dataset described in Section 2.3.1. However, new traces are added and different ground-truth labeling is applied.

Table 7.11 presents the details of the traces used in the evaluation. In order to evaluate the method proposed we used the first seven traces. Among those traces we selected a single trace (UPC-II) as training dataset, which is the 15 min. long trace that contains the highest diversity in terms of instances from different applications. We limit our training set to one trace in order to leave a meaningful number of traces for the evaluation that are not used to build the classification model. Therefore, the remaining first seven traces have been used as the validation dataset. The last trace, UPC-VIII, was recorded with a difference in time of four months with the trace UPC-II. Given this time difference we perform a preliminary experiment with both traces in order to evaluate the gain provided by our continuous training solution.

Nearest Neighbor vs K-dimensional Tree

In Section 7.2.2 we already discussed the main advantages of the K-dimensional tree technique versus the original Nearest Neighbor algorithm. In order to present numerical results showing this gain we perform a comparison between both methods. We evaluate our method with the original NN search implemented for validation purposes by the ANN library. Given that the ANN library implements

Table 7.11: Characteristics of the traffic traces in our dataset

Name	Date	Day	Start Time	Duration	Packets	Bytes	Valid Flows	Avg. Util
UPC-I	11-12-08	Thu	10:00	15 min	95 M	53 G	1936 K	482 Mbps
UPC-II	11-12-08	Thu	12:00	15 min	114 M	63 G	2047 K	573 Mbps
UP-III	12-12-08	Fri	01:00	15 min	69 M	38 G	1419 K	345 Mbps
UPC-IV	12-12-08	Fri	16:00	15 min	102 M	55 G	2176 K	500 Mbps
UPC-V	14-12-08	Sun	00:00	15 min	53 M	29 G	1346 K	263 Mbps
UPC-VI	21-12-08	Sun	12:00	1 h	175 M	133 G	3793 K	302 Mbps
UPC-VII	22-12-08	Mon	12:30	1 h	345 M	256 G	6684 K	582 Mbps
UPC-VIII	10-03-09	Tue	03:00	1 h	114 M	78 G	3711 K	177 Mbps

both methods in the same structure we could not be able to show original memory resources for the naive NN technique. We tested both methods with the trace $UPC-II$ (i.e. 500.000 flows after the sanitization process) using a 3GHz machine with 4GB of RAM. It is important to note that given we are performing an offline evaluation we do not approximate the NN search neither in the NN original algorithm or in the K-dimensional tree technique. For this reason the accuracy in both methods is the same.

Table 7.12 summarizes the improvements obtained with the combination of the K-dimensional tree technique with the port information. It presents the results in terms of classifications per second depending on the number of packets sizes needed for the classification and the list of *relevant* ports. There are three possible lists of *relevant* ports. The unique list, where there is no *relevant* port and all the instances belong to the same k-dimensional tree or NN structure. The selected list, which is composed by the set of ports that contains most of the traffic from the trace UPC-II. And the list where all the ports from the trace $UPC-II$ are *relevant*. The first column represents the original NN presented in previous works [16, 33, 35] where all the information is represented in a single structure. When only one packet is required the proposed method is ten times faster than the original NN, however the speed of the original method dramatically decreases when the number of packets required increases becoming even a hundred times slower than the K-dimensional tree technique. In almost all the situations the introduction of the *relevant* ports substantially increases the classification speed in both methods.

Tables 7.13 and 7.14 show the extremely low price that the K-dimensional tree technique pays for a notable improvement in classification speed. The results show that the memory resources required for our method are few. The memory used in

Table 7.12: Speed Comparison (flows/s) : Nearest Neighbor vs K-Dimensional Tree

Packet Size	Naive Nearest Neighbor			K-Dimensional Tree		
	Unique	Selected Ports	All Ports	Unique	Selected Ports	All Ports
1	45578	104167	185874	423729	328947	276243
5	540	2392	4333	58617	77280	159744
7	194	1007	1450	22095	34674	122249
10	111	538	796	1928	4698	48828

Table 7.13: Memory Comparison: Nearest Neighbor vs K-Dimensional Tree

Packet Size	Naive Nearest Neighbor	K-Dimensional Tree		
		Unique	Selected Ports	All Ports
1	Unknown	40.65 MB	40.69 MB	40.72 MB
5	Unknown	52.44 MB	52.63 MB	53.04 MB
7	Unknown	56.00 MB	56.22 MB	57.39 MB
10	Unknown	68.29 MB	68.56 MB	70.50 MB

the K-dimensional tree is almost independent from the *relevant* ports parameter and barely affected by the number of packet sizes. Regarding time, we believe that the trade-off of the training phase is well compensated by the ability to use our method as an online classifier. In the worst case our method only takes about 20 seconds for the building phase.

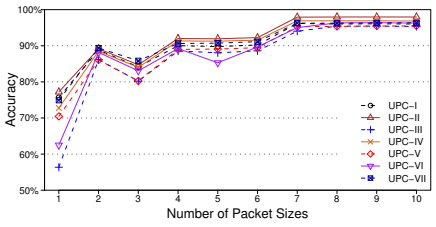
Considering that both methods output the same classification results, the data presented in this subsection show that the combination of the *relevant* ports and the K-dimensional tree technique significantly improves the original NN search with the only drawback of a (very fast) training phase but allowing us to use our method as an efficient online traffic classifier.

K-dimensional tree plugin evaluation

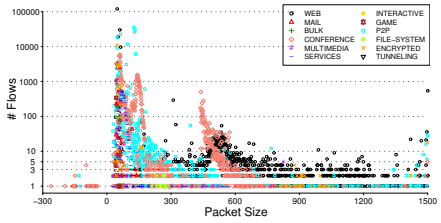
In this section we study the accuracy of our method depending on the different parameters of the $KD-Tree$ plugin. Figure 7.7a presents the accuracy of our method by the number of packet sizes for the different traces of the dataset. In

Table 7.14: Building Time Comparative: Nearest Neighbor vs K-Dimensional Tree

Packet Size	Naive Nearest Neighbor	K-Dimensional Tree		
		Unique	Selected Ports	All Ports
1	0 s	13.01 s	12.72 s	12.52 s
5	0 s	16.45 s	16.73 s	15.62 s
7	0 s	17.34 s	16.74 s	16.07 s
10	0 s	19.81 s	19.59 s	18.82 s



(a) K-dimensional tree accuracy (by flow) without *relevant* ports support, by number of packet sizes



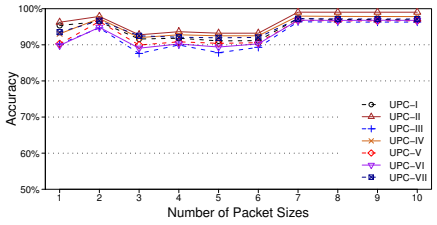
(b) First packet size distribution in the training trace $UPC - II$

Figure 7.7: K-dimensional tree evaluation without the support of the *relevant* ports

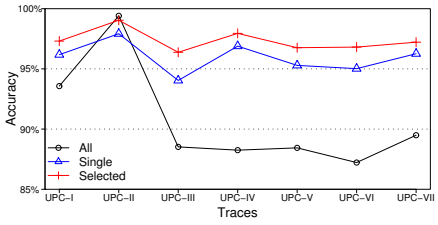
this case no information from the *relevant* ports is taken into account producing a single K-dimensional tree. With this variation using only the first two packets we achieve an accuracy of almost 90%. The accuracy increases with the number of packet sizes until a stable accuracy $> 95\%$ is reached with seven packet sizes.

In order to show the impact of using the list of *relevant* ports in the classification, in Figure 7.7b we show the distribution of the first packet size for the training trace $UPC - II$. Although there are some portions of the distribution dominated by a group of applications, most of the applications have their first packet size between the 0 and the 300 bytes ticks. This collision explains the poor accuracy presented in the previous figure with only one packet.

The second parameter of our method, the *relevant* ports, furthermore of improve the classification speed appears to alleviate that situation. Figure 7.8a present the accuracy of our method by number of packets using the set of *relevant* ports that contains most of the traffic in $UPC - II$. With the help of the *relevant*



(a) K-dimensional tree accuracy (by flow) with *relevant* ports support, by number of packet sizes



(b) K-dimensional tree accuracy (by flow) by set of *relevant* ports with a fixed number of packet sizes(i,e,. 7)

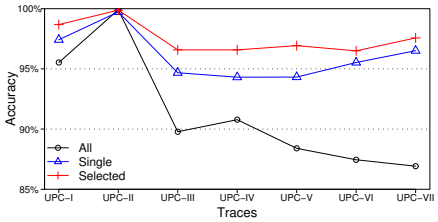
Figure 7.8: K-dimensional tree evaluation with the support of the *relevant* ports

ports our method achieves an accuracy $> 90\%$ using only the first packet size and achieving a stable accuracy of 97% with seven packets.

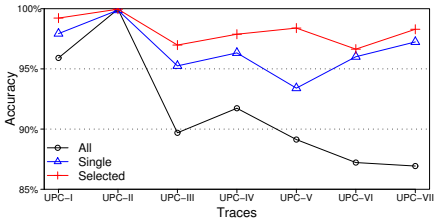
Figure 7.8b presents the accuracy of our method depending on the set of *relevant* ports with seven packet sizes. We choose seven because as can be seen in the Figures 7.7a and 7.8a increasing the number of packet sizes for values greater than seven do not improve its accuracy but decrease its classification speed. Using all the ports of the training trace $UPC - II$ the method achieve the highest accuracy with the own trace however classifying the rest of traces the accuracy substantially decrease but being always higher than 85%. This decrease is given because using all the ports as *relevant* ports is a very dependent option of the scenario and could present classification inaccuracies with new instances belonging to ports not represented in the training data. However, the figure shows that using a set of *relevant* ports - in our case the ports that receive more than 1% of the traffic - besides increasing the classification speed also improves accuracy.

Erman et al. in [26] pointed out a common situation found among the ML techniques: the accuracy when measured by flows is much higher than when measured by bytes or packets. This usually happens because some elephant-flows are not correctly classified. Figures 7.9a and 7.9b present the classification results of our method considering also the accuracy by bytes and packets, showing that, unlike other ML solutions, our method is able to keep high accuracy values even with such metrics. This is because our method is very accurate with the group of applications $P2P$ and WEB, the groups that represent in terms of bytes most of the traffic in our traces.

Finally we also study the accuracy of our method regarding the groups of applications. In our evaluation we use the original definition of the applications and



(a) K-dimensional tree accuracy (by packet) by set of *relevant* ports with a fixed number of packet sizes(i,e,. 7)



(b) K-dimensional tree accuracy (by byte) by set of *relevant* ports with a fixed number of packet sizes(i,e,. 7)

Figure 7.9: K-dimensional tree evaluation with the support of the *relevant* ports

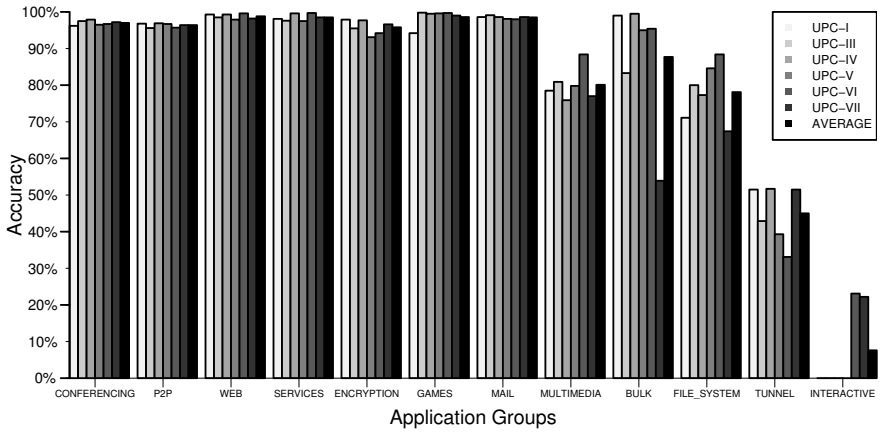


Figure 7.10: Accuracy by application group (seven packet sizes and *selected* list of ports as parameters)

group of applications given in TIE. Figure 7.10 shows that our method is able to classify with excellent accuracy the most popular groups of applications. However, the accuracy of the applications groups that are not very common substantially decreases. These accuracies have a very low impact on the final accuracy of the method given that the representation of these groups in the traces used is almost negligible. A possible solution to improve the accuracy for these groups of applications could be the addition of artificial instances of these groups in the training data.

Definitely, we present a set of results showing how the k-dimensional tree technique using the still useful information provided by the ports improves almost all

Table 7.15: Evaluation of the Continuous Training system by training trace and set of *relevant* ports

Training Trace	UPC-II		First 15 min. UPC-VIII	
Relevant Port List	UPC-II	UPC-VIII	UPC-II	UPC-VIII
Accuracy	84.20 %	76.10 %	98.17 %	98.33 %

the aspects of previous methods based in the NN search. With the unique drawback of a short training phase our method is able to perform online classification with a very high accuracy, $>$ 90% with only one packet or $>$ 97% with seven packets.

Continuous training system evaluation

This section presents a preliminary study of the impact of our continuous training traffic classifier. Due to lack of traces comprising a very long period of time and because of the intrinsic difficulties in processing such large traces, we simulate a scenario in which the features of the traffic evolve by concatenating the $UPC-II$ and $UPC-VIII$ traces. The trace $UPC-VIII$, besides belonging to a difference day-time, was recorded four months later than $UPC-II$, this suggests a different traffic mix with different properties. Using seven as the fixed number of packets sizes, the results in Table 7.15 confirm our intuition. On one hand, using the trace $UPC-II$ as training data to classify the trace $UPC-VIII$ we obtain an accuracy of almost 85%. On the other hand, after detecting such decrease of accuracy and thus performing retraining using the first fifteen minutes of the trace $UPC-VIII$, we obtain and impressive accuracy of 98,17% showing the important impact of the continuous training.

The results of a second experiment are also presented in Table 7.15. Instead of retraining the system with a new training data we study if the modification of the list of *relevant* ports is enough to obtain the original accuracy. The results show that this solution does not bring any improvement when applied alone. However the optimum solution is obtained when both the training data and the list of *relevant* ports are updated and the system is then retrained.

7.2.4 Related Work

The NN method for traffic classification was firstly proposed in [16], where a comparison of the NN technique with the Linear Discriminant Analysis method was presented. They showed that NN was able to classify, among 7 different classes of traffic, with an error rate below 10%.

However, the most interesting conclusions about the NN algorithm are found in the works from Williams et al. [33] and Kim et al. [35]. Both works compared different ML methods and showed the pros and cons of the NN algorithm for traffic classification. In summary, NN was shown to be one of the most accurate ML methods, with the additional feature of requiring zero time to build the classification model. However, NN was the ML-based algorithm with the worst results in terms of classification speed. This is the reason why NN is often discarded for online classification.

The efficient implementation of the NN algorithm presented in this paper is based instead on the K-dimensional tree, which solves its problems in terms of classification speed, while keeping very high accuracy. Another important feature of our method is its ability to early classify the network traffic. This idea is exported from the work from Bernaille et al. [19, 20]. This early classification feature allows the method to classify the network traffic by just using the first packets of each flow. Bernaille et al. compared three different unsupervised ML methods (K-Means, GMM and HMM), while in this work we apply this idea to a supervised ML method (NN).

7.2.5 Chapter Summary

In this work, we revisited the viability of using the Nearest Neighbor algorithm (NN) for online traffic classification, which has been often discarded in previous studies due to its poor classification speed. In order to address this well-known limitation, we presented an efficient implementation of the NN algorithm based on a K-dimensional tree data structure, which can be used for online traffic classification with high accuracy and low overhead. In addition, we combined this technique with the relevant information still provided by the port numbers, which further increases its classification speed and accuracy.

Our results show that our method can achieve very high accuracy ($> 90\%$) by looking only at the first packet of a flow. When the number of analyzed packets is increased to seven, the accuracy of our method increases beyond 95%. This early classification feature is very important, since it allows network operators to

quickly react to the classification results.

We presented an actual implementation of our traffic classification method based on the TIE classification engine. The main novelty of our implementation is its continuous training feature, which allows the system to be automatically re-trained by itself as the training data becomes obsolete. Our preliminary evaluation of this unique feature presents very encouraging results.

Chapter 8

Conclusions

This thesis has addressed some of the most important challenges in the deployment and maintenance of traffic classification solutions in production networks. We focused on three main aspects of network traffic classifiers that hinder the introduction of existing techniques in production networks: (*i*) the ease of deployment, (*ii*) the ease of maintenance and (*iii*) the validation and comparison of existing techniques. Furthermore, to reduce the lack of adequate data to the research community, we published several datasets used in our evaluation to allow the fair validation and comparison of current and future classification techniques.

First, to facilitate the deployment of existing techniques in production networks we:

1. addressed the traffic classification problem with NetFlow data using a well-known supervised learning technique. Our results show that supervised methods (e.g., C4.5) can achieve high accuracy (≈90%) with unsampled NetFlow data, despite the limited information provided by NetFlow, as compared to the packet-level data used in previous studies.

2. empirically and analytically analyzed the severe impact of packet sampling on the classification accuracy of supervised learning methods.

3. proposed a simple improvement in the training process that significantly increased the accuracy of our classification method under packet sampling. For example, for a sampling rate of 1/100, we achieved an overall accuracy of 85% in terms of classified flows, while before we could not reach an accuracy beyond 50%.

Given that Sampled NetFlow is the most widely extended monitoring solution among network operators, we sincerely think that based on our results, using Sampled NetFlow as input considerably facilitates the deployment of existing techniques in operational networks.

Second, aiming to the address the maintenance problem of existing techniques in operational networks we:

1. showed that classification models suffer from temporal and spatial obsolescence.

2. addressed this problem by first proposing a complete traffic classification solution with a novel automatic retraining system. The classification system combines the advantages of three different techniques (i.e., IP-based, Service-based and ML-based) along with the autonomic retraining system to sustain a high classification accuracy during long periods of time. The retraining system combines multiple DPI techniques and only requires a small sample of the whole traffic to keep the classification system updated.

3. introduced the use of stream-based ML techniques for network traffic classification by proposing a new stream-based classification solution based on Hoeffding Adaptive Trees. The main novelty of our technique is its ability to automatically adapt to the changes of the traffic with just a small sample of labeled data, making our solution very easy to maintain. This solution is not only more accurate than traditional batch-based techniques, but it also sustains high accuracy over the years with less cost.

Our proposals have several features that are particularly appealing for network management: (*i*) high classification accuracy and completeness, (*ii*) support for NetFlow data, (*iii*) automatic model retraining, and (*iv*) resilience to sampling. These features altogether result in a significant reduction in the cost of deployment, operation and maintenance compared to previous methods based on packet traces and manually-made classification models.

Third, to present a first step towards the fair validation and comparison of existing network traffic classifiers we:

1. created a reliable labeled dataset with full packet payload. The dataset of more than 500 K flows contains traffic from popular applications like *HTTP*, *Edonkey*, *BitTorrent*, *FTP*, *DNS*, *NTP*, *RDP*, *NETBIOS*, *SSH*, and *RDP*. The total amount of data properly labeled is 32.61 GB. We released this dataset to the research community with full payload, so it can be used as

a common reference for the comparison and validation of network traffic classifiers.

2. using the previous dataset, we compared the performance of six tools (i.e., *PACE*, *OpenDPI*, *L7-filter*, *NDPI*, *Libprotoident*, and *NBAR*), which are usually used for ground-truth generation. The results obtained show that *PACE* is, on our dataset, the most reliable solution for ground-truth generation. Among the open-source tools, *NDPI* and especially *Libprotoident* present the best results. On the other hand, *NBAR* and *L7-filter* present several inaccuracies that make them not recommendable as a ground-truth generator.

All in all, we truly believe the different contributions obtained in this thesis will help to reduce the gap between the real-world requirements from the network industry, and the research being carried out in the field of network traffic classification. The results presented in this thesis give useful indications to researchers and developers to design future accurate and realistic network traffic classification solutions for production networks.

8.1 Future Work

In this thesis we focused on addressing practical challenges of network traffic classification solutions. Moreover, due the continuous evolution of the Internet traffic and its applications the problems we tackled allow further research and improvements.

To maintain the classification solution updated we relied on the labeling of a small sample of the traffic by DPI-based techniques. However, DPI-based techniques should be also periodically updated. Although, this is usually provided by DPI tools developers, some techniques can help us to detect when an update of the ground-truth generator is necessary. For instance, the detection of new applications not classified by the ground-truth generator could be obtained using unsupervised machine learning techniques.

Another future work derived from the continuous evolution of the Internet traffic is the extension of the publicly available reliable labeled datasets. In order to allow for future comparison and validation of new proposals new datasets containing new applications traffic should be provided to the research community.

There is also a wide range of applications following the early classification approach. For example, we can use this early classification to apply traffic shaping

based on the identified application. However, the application of traffic shaping is a very controversial topic given its implications in network neutrality.

Finally, an interesting approach barely studied is the multilabel classification. Most of the existing solutions for traffic classification focus on the classification of the traffic at a single level. However, network traffic can be classified at different levels. For example, the traffic produced by a person watching a Youtube video in his cell phone can be considered correctly classified as: Youtube traffic, RTSP traffic, Streaming traffic, Flash traffic, UDP traffic or Mobile traffic. It would be interesting to classify the traffic at different levels trying to be as complete as possible.

Bibliography

[1] Internet Assigned Numbers Authority (IANA), http://www.iana.org/assignments/port-numbers, as of August 12, 2008.

[2] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "File-sharing in the internet: a characterization of p2p traffic in the backbone," *Univ. of California, Riverside, Tech. Rep*, 2003.

[3] ——, "Is p2p dying or just hiding?" in *Proc. of IEEE GLOBECOM, November*, 2004.

[4] T. Karagiannis, A. Broido, and M. Faloutsos, "Transport layer identification of P2P traffic," in *Proc. of ACM SIGCOMM IMC, August*, 2004.

[5] A. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Proc. of PAM Conf., March*, 2005.

[6] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," in *Proc. of WWW Conf., May*, 2004.

[7] L7-filter. Application layer packet classifier., http://l7-filter.sourceforge.net/.

[8] OpenDPI, the Open Source version of ipoque's DPI software, http://www.opendpi.org/.

[9] PACE, ipoque's Protocol and Application Classification Engine, http://www.ipoque.com/en/products/pace.

[10] nDPI, Open and Extensible GPLv3 Deep Packet Inspection Library, http://www.ntop.org/products/ndpi/.

[11] T. Auld, A. Moore, and S. Gull, "Bayesian neural networks for Internet traffic classification," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, 2007.

[12] M. Crotti and F. Gringoli, "Traffic classification through simple statistical fingerprinting," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, 2007.

[13] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: automated construction of application signatures," in *Proc. of ACM SIGCOMM MineNet, August*, 2005.

[14] H. Jiang, A. Moore, Z. Ge, S. Jin, and J. Wang, "Lightweight application classification for network management," in *Proceedings of the ACM SIGCOMM Workshop on Internet Network Management (INM)*, 2007, pp. 299–304.

[15] A. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proc. of ACM SIGMETRICS, June*, 2005.

[16] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification," in *Proc. of ACM SIGCOMM IMC, October*, 2004.

[17] D. Zuev and A. Moore, "Traffic classification using a statistical approach," in *Proc. of PAM Conf., March*, 2005.

[18] G. Szabo, I. Szabo, and D. Orincsay, "Accurate traffic classification," in *Proc. of IEEE WoWMoM, June*, 2007.

[19] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proc. of ACM CoNEXT, December*, 2006.

[20] L. Bernaille, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, 2006.

[21] L. Bernaille and R. Teixeira, "Early recognition of encrypted applications," in *Proc. of PAM Conf., April*, 2007.

[22] A. Dainotti, W. de Donato, A. Pescape, and P. Rossi, "Classification of network traffic via packet-level hidden Markov models," in *Proc. of IEEE GLOBECOM, November*, 2008.

[23] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *Proc. of ACM SIGCOMM MineNet, September*, 2006.

[24] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/realtime traffic classification using semi-supervised learning," *Performance Evaluation*, vol. 64, no. 9-12, 2007.

[25] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and peer-to-peer traffic in the network core," in *Proc. of WWW Conf.*, 2007.

[26] J. Erman, A. Mahanti, and M. Arlitt, "Byte me: a case for byte accuracy in traffic classification," in *Proc. of ACM SIGMETRICS MineNet, June*, 2007.

[27] A. Soule, K. Salamatian, N. Taft, R. Emilion, and K. Papagiannaki, "Flow classification by histograms: or how to go on safari in the internet," in *Proc. of ACM SIGMETRICS, June*, 2004.

[28] S. Zander, T. Nguyen, and G. Armitage, "Self-learning IP traffic classification based on statistical flow characteristics," in *Proc. of PAM Conf., March*, 2005.

[29] ——, "Automated traffic classification and application identification using machine learning," in *Proc. of IEEE LCN Conf., November*, 2005.

[30] J. Erman, A. Mahanti, and M. Arlitt, "Internet traffic identification using machine learning," in *Proc. of IEEE GLOBECOM, October*, 2006.

[31] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 5, 2006.

[32] ——, "Evaluating machine learning methods for online game traffic identification," *CAIA Technical Report, April*, 2006.

[33] ——, "Evaluating machine learning algorithms for automated network application identification," *CAIA Technical Report, April*, 2006.

[34] G. Szabo, D. Orincsay, S. Malomsoky, and I. Szabo, "On the validation of traffic classification algorithms," in *Proc. of PAM Conf, April*, 2008.

[35] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *Proc. of ACM CoNEXT, December*, 2008.

[36] T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 4, 2008.

[37] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: multilevel traffic classification in the dark," in *Proc. of ACM SIGCOMM, August*, 2005.

[38] K. Xu, Z. Zhang, and S. Bhattacharyya, "Profiling Internet backbone traffic: behavior models and applications," in *Proc. of ACM SIGCOMM, August*, 2005.

[39] T. Karagiannis, K. Papagiannaki, N. Taft, and M. Faloutsos, "Profiling the end host," in *Proc. of PAM Conf., April*, 2007.

[40] T. Mori, R. Kawahara, H. Hasegawa, and S. Shimogawa, "Characterizing traffic flows originating from large-scale video sharing services," *Proceedings of Traffic Monitoring and Analysis (TMA)*, pp. 17–31, 2010.

[41] S. Yoon, J. Park, J. Park, Y. Oh, and M. Kim, "Internet application traffic classification using fixed IP-port," *Manag. Enabling the Future Internet for Changing Bus. and New Comput. Serv.*, vol. 5787, pp. 21–30, 2009.

[42] V. Carela-Espanol, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Sole-Pareta, "Analysis of the impact of sampling on NetFlow traffic classification," *Comput. Netw.*, vol. 55, no. 5, pp. 1083–1099, 2011.

[43] W. Li, M. Canini, A. Moore, and R. Bolla, "Efficient application identification and the temporal and spatial stability of classification schema," *Comput. Netw.*, vol. 53, no. 6, pp. 790–809, 2009.

[44] Valenti, S. et al., "Reviewing Traffic Classification," in *Data Traffic Monitoring and Analysis*. Springer, 2013, pp. 123–147.

[45] Fukuda, K., "Difficulties of identifying application type in backbone traffic," in *Int. Conf. on Network and Service Management (CNSM)*. IEEE, 2010, pp. 358–361.

[46] Alcock, S. and Nelson, R., "Libprotoident: Traffic Classification Using Lightweight Packet Inspection," University of Waikato, Tech. Rep., 2012, http://www.wand.net.nz/publications/lpireport.

[47] A. Dainotti, F. Gargiulo, L. Kuncheva, A. Pescape, and C. Sansone, "Identification of traffic flows hiding behind tcp port 80," in *IEEE International Conference on Communications (ICC)*, 2009, pp. 1–6.

[48] Cisco Systems: Sampled NetFlow., http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/12s_sanf.html.

[49] MAWI Working Group Traffic Archive, [Online]. Available: http://mawi.wide.ad.jp/mawi/.

[50] Bujlow, T. et al., "Volunteer-Based System for classification of traffic in computer networks," in *19th Telecommunications Forum TELFOR*. IEEE, 2011, pp. 210–213.

[51] V. Carela-Espanol, P. Barlet-Ros, O. Mula-Valls, and J. Sole-Pareta, "An automatic traffic classification system for network operation and management," *Journal of Network and Systems Management*, 2013.

[52] V. Carela-Español, T. Bujlow, and P. Barlet-Ros, "Is Our Ground-Truth for Traffic Classification Reliable?" in *Proc. of PAM Conf., March*, 2014.

[53] J. Molina Rodriguez, V. Carela Espanol, P. Barlet Ros, R. Hoffmann, and K. Degner, "Empirical analysis of traffic to establish a profiled flow termination timeout," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*. IEEE, 2013, pp. 1156–1161.

[54] V. Carela-Espanol, P. Barlet-Ros, M. Sole-Simo, A. Dainotti, W. de Donato, and A. Pescape, "K-Dimensional Trees for Continuous Traffic Classification," *Proceedings of Traffic Monitoring and Analysis (TMA)*, pp. 141–155, 2010.

[55] D. Antoniades, M. Polychronakis, S. Antonatos, E. Markatos, S. Ubik, and A. Øslebø, "Appmon: an application for accurate per application network traffic characterization," 2006.

[56] CoralReef, http://www.caida.org/tools/measurement/coralreef/.

[57] J. Cohen, "A coefficient of agreement for nominal scales," *Educ. and Psychol. Meas.*, vol. 20, no. 1, pp. 37–46, 1960.

[58] M. Canini, W. Li, A. Moore, and R. Bolla, "GTVS: boosting the collection of application traffic ground truth," in *Proc. 1st Intl. Workshop on Traffic Monitoring and Analysis, Aachen, Germany*. Springer, 2009.

[59] Traffic classification at the Universitat Politècnica de Catalunya (UPC)., http://www.cba.upc.edu/monitoring/traffic-classification.

[60] C. Shannon, E. Aben, kc claffy, D. Andersen. The CAIDA Anonymized 2008/2009 Internet Traces- Equinix-Chicago/Equinix-SanJose, http://www.caida.org/data/passive/.

[61] Waikato Internet Traffic Storage (WITS): Auckland VIII trace, http://www.wand.net.nz/wits/auck/8/auckland_viii.php.

[62] J. Quinlan, *C4. 5: programs for machine learning*. Morgan Kaufmann, 1993.

[63] WEKA: data mining software in Java., http://www.cs.waikato.ac.nz/ml/weka/.

[64] P. Barlet-Ros, J. Sole-Pareta, J. Barrantes, E. Codina, and J. Domingo-Pascual, "SMARTxAC: a passive monitoring and analysis system for high-speed networks," *Campus-Wide Information Systems*, vol. 23, no. 4, 2006.

[65] K. Lan and J. Heidemann, "A measurement study of correlations of internet flow characteristics," *Computer Networks*, vol. 50, no. 1, pp. 46–62, 2006.

[66] W. Liu, "Estimating flow length distributions using least square method and maximum likelihood estimation," in *ICCS '07: Proc. of the 7th international conference on Computational Science, Part IV*. Springer-Verlag, 2007, pp. 793–796.

[67] N. V. Chawla, "C4.5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure," in *Proc. of the ICML'03 Workshop on Class Imbalances*, 2003.

[68] Cisco IOS. NetFlow white papers, http://www.cisco.com/en/US/products /ps6601/prod_white_papers_list.html.

[69] H. Pham, *Handbook of Reliability Engineering.* Springer London, 2006.

[70] N. Duffield, C. Lund, and M. Thorup, "Properties and prediction of flow statistics from sampled packet streams," in *In Proc. ACM SIGCOMM Internet Measurement Workshop*, 2002, pp. 159–171.

[71] J. Mai, A. Sridharan, C. Chuah, H. Zang, and T. Ye, "Impact of packet sampling on portscan detection," *IEEE Journal on Selected Areas in Communications*, 2006.

[72] J. Mai, C. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?" in *Proc. of the 6th ACM SIGCOMM conference on Internet measurement*, 2006.

[73] I. Paredes-Oliva, P. Barlet-Ros, and J. Sole-Pareta, "Portscan detection with Sampled NetFlow," in *Proc. of Intl. Workshop on Traffic Monitoring and Analysis*, May 2009.

[74] G. Androulidakis, V. Chatzigiannakis, S. Papavassiliou, M. Grammatikou, and V. Maglaris, "Understanding and evaluating the impact of sampling on anomaly detection techniques," in *Military Communications Conference*, 2006.

[75] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, "Impact of packet sampling on anomaly detection metrics," in *Proc. of the 6th ACM SIGCOMM conference on Internet measurement*, 2006.

[76] A. Davy, D. Botvich, and B. Jennings, "On the use of accounting data for QoS-aware IP network planning," in *Proc. of the 20th international teletraffic conference on Managing traffic performance in converged networks*, 2007, pp. 348–360.

[77] Y. Lim, H. Kim, J. Jeong, C. Kim, T. Kwon, and Y. Choi, "Internet traffic classification demystified: on the sources of the discriminative power," in *Proceedings of ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2010, p. 9.

[78] Is See5/C5.0 Better Than C4.5?, http://rulequest.com/see5-comparison.html.

[79] J. Li, S. Zhang, C. Li, and J. Yan, "Composite lightweight traffic classification system for network management," *Int. J. of Netw. Manag.*, vol. 20, no. 2, pp. 85–105, 2010.

[80] A. Dainotti, A. Pescapé, and C. Sansone, "Early classification of network traffic through multi-classification," *Proceedings of Traffic Monitoring and Analysis (TMA)*, pp. 122–135, 2011.

[81] S. Lee, H. Kim, D. Barman, S. Lee, C. Kim, T. Kwon, and Y. Choi, "NeTraMark: a network traffic classification benchmark," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, pp. 22–30, 2011.

[82] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001, pp. 97–106.

[83] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.

[84] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Advances in Intelligence Data Analysis*, 2009, pp. 249–260.

[85] ——, "Learning from time-changing data with adaptive windowing," in *Siam International Data Mining Conference*, 2007, pp. 443–448.

[86] NBAR2 or Next Generation NBAR - Cisco Systems, 2013, [Online]. Available: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6558/ps6616/qa_c67-697963.html.

[87] MOA: Massive Online Analysis (Data Stream Analytics in Real Time)., [Online]. Available: http://moa.cms.waikato.ac.nz/.

[88] J. Gama, "A survey on learning from data streams: current and future trends," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 45–55, 2012.

[89] J. Gama, R. Sebastião, and P. P. Rodrigues, "Issues in evaluation of stream learning algorithms," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009, pp. 329–338.

[90] X. Tian, Q. Sun, X. Huang, and Y. Ma, "Dynamic online traffic classification using data stream mining," in *IEEE MMIT'08, MultiMedia and Information Technology*, 2008, pp. 104–107.

[91] ——, "A dynamic online traffic classification methodology based on data stream mining," in *IEEE Computer Science and Information Engineering*, vol. 1, 2009, pp. 298–302.

[92] B. Raahemi, W. Zhong, and J. Liu, "Peer-to-peer traffic identification by mining ip layer data streams using concept-adapting very fast decision tree," in *IEEE ICTAI'08, Tools with Artificial Intelligence*, vol. 1, 2008, pp. 525–532.

[93] Bujlow, T. et al., "Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification," UPC BarcelonaTech, Tech. Rep., 2013, accessible: https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2013,en.html.

[94] [Online], "Volunteer-Based System for Research on the Internet," 2012, uRL: http://vbsi.sourceforge.net/.

[95] Shen, C. et al., "On detection accuracy of L7-filter and OpenDPI," in *3th Int. Conf. on Networking and Distributed Computing (ICNDC)*. IEEE, 2012, pp. 119–123.

[96] Alcock, Shane and Nelson, Richard, "Measuring the Accuracy of Open-Source Payload-Based Traffic Classifiers Using Popular Internet Applications," in *IEEE Workshop on Network Measurements*, 2013.

[97] Dusi, M. et al., "Quantifying the accuracy of the ground truth associated with Internet traffic traces," *Computer Networks*, vol. 55, no. 5, pp. 1158–1167, 2011.

[98] Gringoli, F. et al., "Gt: picking up the truth from the ground for internet traffic," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 5, pp. 12–18, 2009.

[99] Ipoque, "Supported Protocols and Applications," Ipoque GmbH, 2012, website: http://www.ipoque.com/sites/default/files/mediafiles/documents/data-sheet-protocol-support.pdf.

[100] R. Braden, "RFC 1122 - Requirements for Internet Hosts – Communication Layers," Internet Engineering Task Force, October 1989, website: http://tools.ietf.org/html/rfc1122.

[101] V. Paxson, M. Allman, J. Chu, M. Sargent, "RFC 6298 - Computing TCP's Retransmission Timer," Internet Engineering Task Force, June 2011, website: http://tools.ietf.org/html/rfc6298.

[102] W. John, S. Tafvelin, and T. Olovsson, "Trends and differences in connection-behavior within classes of internet backbone traffic," in *Passive and Active Network Measurement*. Springer, 2008, pp. 192–201.

[103] K. Claffy, H. Braun, G. Polyzos, S. Center, G. Atomics, and C. San Diego, "A parameterizable methodology for Internet traffic flow profiling," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, 1995.

[104] A. Dainotti, W. de Donato, A. Pescapé, and G. Ventre, "TIE: a community-oriented traffic classification platform," in *Proceedings of the First International Workshop on Traffic Monitoring and Analysis*, 2009, p. 74.

[105] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[106] J. L. Bentley, "K-d trees for semidynamic point sets," pp. 187–197, 1990.

Appendix A

8.2 UPC dataset

Next, we present the relation of collaborations related to the UPC dataset.

Name	Supervisor	Institution	Date	Research Purpose
Giantonio Chiarelli	Domenico Vitali	Universita degli Studi di Roma La Sapienza (Rome, Italy)	Jan 2011	Evaluation of traffic classification techniques.
Sanping Li	-	University of Massachusetts Lowell (Lowell, USA)	Feb 2011	My current research is mainly about online network traffic classification based on multi-core architecture.
Qian Yaguan	Wu Chunming	CS College of Zhejiang University (Hangzhou, China)	Apr 2011	Our purpose is to analyze the details of mixed Internet traffic, and decide proper policy for network recources distribution.
Yulios Zabala	Lee Luan Ling	State University of Campinas - Unicamp (So Paulo, Brazil)	Aug 2011	The purpose of our research is to develop a new technique for classification of network traffic based on multifractal theory and machine learning algorithms, along with techniques to make the system usable in real time.
Massimiliano Natale	Domenico Vitali	Universita degli Studi di Roma La Sapienza (Rome, Italy)	Jan 2012	Traffic classification based on Netflow data.
Elie Bursztein	-	Stanford University (Stanford, USA)	Feb 2012	Using the dataset to evaluate the effectiveness of our algorithm that reconstruct the statistical distribution of one classifier into another.
Jesus Diaz Verdejo	-	Universidad de Granada (Granada, Spain)	Feb 2013	Validation of traffic classification techniques.
Ning Gao	Quin Lv	University of Colorado Boulder (Boulder, USA)	Feb 2013	I'm now doing a project to classify network flows into different applications.
Wesley Melo	Stenio Fernandes	GPRT - Networking and Telecommunications Research Group (Recife, Brazil)	Jul 2013	On my previous work, I used synthetic traces and I would like to evaluate it using your real traces.
Adriel Cheng	-	Department of Defence (Edinburgh, Australia)	Sep 2013	Investigation of machine-learning/data-mining techniques for classification of flows data.
Corey Hart	-	Lockheed Martin (King of Prussia, PA, USA)	Oct 2013	Spiking neural nework based intrusion detection.
Rajesh NP	-	Cisco (Bangalore, India)	Dec 2013	Evaluating multiple machine learning algorithms currently available for Network analysis.
Raja Rajendran	Andrew Ng	Stanford University (Stanford, USA)	Dec 2013	I would like to experiment with some of the machine learning algorithms from my course using the netflow dataset.
Indranil Adak	Raja Rajendran	Cisco (Bangalore, India)	Dec 2013	We are currently researching on applying machine learning to predict the traffic flow and recommending network/traffic growth for service providers.

8.3 PAM dataset

Next, we present the relation of collaborations related to the PAM dataset.

Name	Supervisor	Institution	Date	Research Purpose
Said Sedikki	Ye-Qiong Song	University of Lorraine (Villers-les-Nancy, France)	Mar 2014	Testing flow classification program that will be used in the controller for an OpenFlow Network.
Oliver Gasser	Georg Carle	Technische Universitat Munchen (Munchen, Germany)	Mar 2014	We are doing research in DOS detection. Your dataset would be used for background traffic purposes.
Viktor Minorov	Pavel Celada	Masaryk University (Brno, Czech Republic)	Mar 2014	Data will be used in work about measuring the accuracy of traffic classifiers.
Yiyang Shao	Jun Li	Tsinghua University (Beijing, China)	Apr 2014	The ground-truth of traffic data is a fundamentally important issue in the research of this area. So I hope I could have a copy of PAM dataset under CBA privacy policies.
Yinsen Miao	Farinaz Koushanfar	Rice University (Houston, USA)	Apr 2014	Apply this dataset to test our traffic classification method's accuracy.
Le Quoc Do	Christof Fetzer	Technische Universitat Dresden (Dresden, Germany)	May 2014	Optimizing Support Vector Machine (SVM) to improve classification accuracy in analyzing network data.
Zuleika Nascimento	Djamel Salok	Federal University of Pernambuco (Recife, Brazil)	May 2014	I'm currently working on network traffic classification using soft computing techniques and I'd like to test my model on different datasets.
Garrett Cullity	Adriel Cheng	University of Adelaide (Adelaide, Australia)	May 2014	We aim to apply machine learning techniques to network flow data (netflow/IPFIX) for traffic classification, device characterisation, and hopefully, identification of malicious traffic.
Zeynab Sabahi	Ahmad Nickabadi	University of Tehran (Tehran, Iran)	Jun 2014	I have implemented a new hybrid DPI tool based on data fusion techniques. I need to compare the results of my tool with the existing DPI tools.
Joseph Kampeas	Omer Gurewitz	Ben Gurion University of the Negev (Beer Sheva, Israel)	Jun 2014	Traffic classification and identification using machine learning technique.
Hossein Doroud	Andres Marin	Univesidad Carlos III (Madrid, Spain)	Jul 2014	Evaluate performance of a new network traffic classifier that uses Machine Learning approach.
Alioune BA	Cedric Baudoin	Thales Alenia Space (Toulouse, France)	Jul 2014	Use of DPI for dynamic QoS architecture. The aim is to validate the feasibility of DPI approach for crosslayer QoS optimization in satcom network.
Jan-Erik Stange	-	University of Applied Science Postdam (Postdam, Germany)	Jul 2014	The dataset will be used for visualization research in the EU-project SaSER. More specifically our research team will use this dataset as a basis for developing NetFlow visualization that assist in the process of detecting brute-force and amplification attacks.

Appendix B

8.4 Publications

8.4.1 Journals

- **V. Carela-Español**, P. Barlet-Ros, A. Bifet and K. Fukuda. "*A streaming flow-based technique for traffic classification applied to 12+1 years of Internet traffic*". Journal of Telecommunications Systems, 2014.(Under Review)

- T. Bujlow, **V. Carela-Español** and P. Barlet-Ros. "*Independent Comparison of Popular DPI Tools for Trac Classification*". Computer Networks, 2014 (Under Review)

- **V. Carela-Español**, P. Barlet-Ros, O. Mula-Valls and J. Solé-Pareta. "*An Autonomic Traffic Classification System for Network Operation and Management*". Journal of Network and Systems Management, October 2013.

- **V. Carela-Español**, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Solé-Pareta. "*Analysis of the impact of sampling on NetFlow traffic classification*". Computer Networks 55 (2011), pp. 1083-1099.

8.4.2 Conferences

- **V. Carela-Español**, T. Bujlow, and P. Barlet-Ros. "*Is Our Ground-Truth for Traffic Classification Reliable?*". In Proc. of the Passive and Active Measurements Conference (PAM'14), Los Angeles, CA, USA, March 2014.

- J. Molina, **V. Carela-Español**, R. Hoffmann, K. Degner and P. Barlet-Ros. "*Empirical analysis of traffic to establish a profiled flow termination timeout*". In Proc. of Intl. Workshop on Traffic Analysis and Classification (TRAC), Cagliari, Italy, July 2013.

- **V. Carela-Español**, P. Barlet-Ros, M. Solé-Simó, A. Dainotti, W. de Donato and A. Pesacapé. "*K-dimensional trees for continuous traffic classification*". In Proc. of Second International Workshop on Traffic Monitoring and Analysis. Zurich, Switzerland, April 2010.

- P. Barlet-Ros, **V. Carela-Español**, E. Codina and J. Solé-Pareta. "*Identification of Network Applications based on Machine Learning Techniques*". In Proc. of TERENA Networking Conference. Brugge, Belgium, May 2008.

8.4.3 Technical Reports

- T. Bujlow, **V. Carela-Español** and P. Barlet-Ros. "*Extended Independent Comparison of Popular Deep Packet Inspection (DPI) Tools for Traffic Classification*". Technical Report, UPC-DAC-RR-CBA-2014-1, Jan. 2014.

- T. Bujlow, **V. Carela-Español** and P. Barlet-Ros. "*Comparison of Deep Packet Inspection (DPI) tools for traffic classification*", Technical Report, UPC-DAC-RR-CBA-2013-3, June 2013.

- **V. Carela-Español**, P. Barlet-Ros and J. Solé-Pareta. "*Traffic classification with Sampled NetFlow*". Technical Report UPC-DAC-RR-CBA-2009-6. February 2009.

- **V. Carela-Español**, P. Barlet-Ros and J. Solé-Pareta. "*Identification of Network Applications based on Machine Learning Techniques*". Technical Report UPC-DAC-RR-CBA-2009-2. February 2009.

8.4.4 Supervised Master Students

- Juan Molina Rodriguez : "*Empirical analysis of traffic to establish a profiled flow termination timeout*", 2013 in collaboration with ipoque.

8.4.5 Datasets

- **UPC Dataset**: NetFlow v5 dataset labeled by L7-filter

- **PAM Dataset**: full packet payload dataset labeled by VBS