Machine Learning based Classification of Emulated Internet Communications on the Cloud

Maestría en Ciencias de la Computación

Facultad de Matemáticas

Universidad Autónoma de Yucatán

M2 Informatique

Parcours Industry 4.0

Université de Pau et Pays de l'Adour

By

Daniel Sánchez Ferriz

Tutors:

Jorge Ricardo Gómez Montalvo

Ernesto Exposito García

September 2019

Dedicatoria

A mi padres, cuya calidez me abriga en tiempos de frío, me cubre del granizo desconocido e ilumina las tinieblas de mi senda.

A mis hermanos, quienes brindan el soporte para seguir adelante y me recuerdan el camino a casa.

A mis amigos y compañeros, sin los cuales, esta formidable experiencia habría sido incompleta.

Acknowledgments

Firstly, I would like to thank Jorge Ricardo Gómez Montalvo for his patience, support, wisdom, and trust he gave me throughout both masters programs. Without him, I wouldn't have had the opportunity of working alongside brilliant people from around the globe.

I need to thank Ernesto Exposito García for providing me so much knowledge in so little time, and Fannia Pacheco Montilla for sharing her unfathomable talent.

This research was in part possible for the support of the Nouvelle-Aquitaine region and the Mexican Science and Technology National Council (CONACYT) through their scholarship program to the holder 630946 with CVU (unique curriculum vitae) 853054.

Resumen

La clasificación del tráfico de Internet consiste en etiquetar el tráfico en categorías predefinidas. Uno de los intereses en la realización de esta tarea es priorizar cierto tipo de tráfico para ofrecer Calidad de Servicio (QoS). Uno de los métodos más comunes encontrados en la literatura es el aprendizaje automático, sin embargo, hay una falta de datos de tráfico de Internet etiquetados públicamente actualizados. En la mayoría de los trabajos, los investigadores usan bases de datos públicas que pueden estar desactualizadas o usan los flujos de tráfico de usuarios privados, el problema con estos últimos es que se requiere el permiso de cada usuario para evitar la violación de leyes de privacidad. Otro problema es que los datos capturados de usuarios privados carecen de un proceso de etiquetado de tráfico correcto. Un método utilizado para etiquetar datos es el análisis de carga útil que tiene como objetivo inspeccionar el contenido del tráfico; sin embargo, si los datos están cifrados, dicho método no reconoce la categoría de tráfico de Internet. En esta tesis, empleamos una arquitectura de nube para emular comportamientos similares a los humanos y generar flujos de tráfico para etiquetarlos y almacenar su información estadística en una base de datos. La información dentro de la base de datos se utilizó para entrenar diferentes algoritmos para detectar comportamientos y filtrar ruido. Con el conjunto de datos reducido, se determinó el clasificador de aprendizaje automático más adecuado, esta selección se sustenta en varios experimentos.

Abstract

Internet traffic classification is to label traffic into predefined categories. One of the interests in performing this task is prioritizing some sensitive traffic in order to offer Quality of Service (QoS). One of the most common methods found in the literature is Machine Learning, however, there is a lack of public labeled Internet traffic data up to date. In most works, researchers either use public databases which might be outdated or use the traffic flows from private users, the issue with the latter is that the permission of every user is required to avoid violating privacy laws. Even with their permission, the captured data would lack a correct traffic labeling process. A method used to label data is payload analysis that aims at inspecting the traffic content; however, if the data is encrypted, such method fails to recognize the Internet traffic category. In this thesis, we employed a cloud architecture to emulate human-like behavior and generate traffic flows to label them to store their statistical information in a database. The information within the database was used to train different algorithms to detect behaviors and filter out the noise. With the reduced dataset, we created the most suitable Machine Learning classifier, this selection is sustained by several experiments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preliminaries

According to the ITU (International Telecommunication Union), the Internet international bandwidth increased up to 32% between 2015 and 2016 [5]. This constant increase generates a greater demand for information and communication transport services. Such demands can be approached by meeting the requirements of the network. According to [4], a network has three different types of requirements: user's, application's, and device's requirements. Table 1.1 describes those requirements.

Table 1.1: Network Requirements [4]

Type	Requirement	Definition
User	Timeliness	User access, transfer, or modification of information within a tolerable time frame. Tolerance is dependant on the user's perception of delay
	Interactivity	User access, transfer, or modification of information within a tolerable time frame in terms of response time
	Reliability	Consistently available service from the user's perspective
	Presentation Quality	Quality of the presentation to the user

		Adaptability	Adaptation to users' changing needs
		Security	Confidentiality, integrity, and authenticity of a user's information and physical resources
		Affordability	Purchases must fit within a budget
		Functionality	Any functional requirement that the user has for the system
		Supportability	Set of characteristics that describe how well the customer can keep the network operating at designed performance, through the full range of mission scenarios described by the customer during the requirements analysis process.
		Future Growth	Determining if and when users are planning to deploy and use new applications and devices on the network.
Application	Types	Mission-critical	Predictable, guaranteed, and/or high-performance RMA requirements.
		Rate-critical	Predictable, guaranteed, and/or high-performance capacity requirements
		Real-time and interactive	Predictable, guaranteed, and/or high- performance delay requirements
	Groups	Telemetry/Command-and-Control Applications	Data and command information is transmitted between remote devices and one or more control stations for command, control, tracking, and determining status of the remote devices
		Visualization Applications	Applications that range from two-dimensional viewing of objects to three-dimensional and virtual reality viewing, immersion, and manipulation of objects
		Distributed-Computing Applications	Applications that may range from having the computing devices sharing the same local bus, to being co-located at the same LAN, to being distributed across LAN, MAN, and WAN boundaries
		Web Development, Access, and Use Applications	Applications that are the current interactive equivalents of the traditional remote device and information access utilities telnet and FTP

		Bulk Data Transport Applications	Applications that can optimize the data transfer rate at the expense of interactivity when the amounts of information desired are relatively large and the sessions are less interactive (or asynchronous)
		Tele Service Applications	Applications that provide a subset of voice, video, and data together to be delivered concurrently to groups of people at various locations
		Operations, Administration, Maintenance, and Provisioning (OAM&P) Applications	Applications that are required for the proper functioning and operation of the network
		Client–Server Applications	Applications whose traffic flows behave in a client–server fashion
	Locations	Logical	User, User Groups
		Physical	Servers, Floors within a building, Building
Device Types		Generic Computing Devices	End-to-end perspective
		Servers	Per-device basis perspective
		Specialized devices	Location dependent

In addition, the device requirements also consist of the performance characteristics (which relate to the hardware, firmware, and software used in the devices) and the device locations. Each of these requirements aims to a functional networking system, and as the network grows, so does its performance requirements.

As defined in [4], performance can be defined as the set of levels for capacity, delay, and RMA in a network. This definition reflects the network requirements described in Table 1.1 illustrating that there is a link between the requirements of a network and its performance. Therefore, focusing on the performance of a network reduces error frequency and impact.

[4] also establishes that an aspect to consider for performance is the capacity, delay, and RMA levels through their respective traffic flows. The management of traffic flows is determined through their classification in terms of previously determined rules, such as SLAs or QoS. Through these rules, it can be determined whether to deny or grant access to an identified

traffic flow, this process is known as admission control. Similarly, traffic management rules can use the process of traffic conditioning to identify the traffic flow type and mark its packets with a label in order to assign them a priority. Both traffic conditioning and admission control are only achievable by classification of traffic flows.

In terms of Internet services, traffic conditioning and admission control can be implemented to provide better services to the users. To do so, it is required a set of rules to adhere to, and since the experience of the user is related to the metrics of Quality of Service [6], then it can be said that meeting such metrics concludes in a better experience for the user.

Furthermore, traffic flow classification is comprised of several methods, and given the rapid growth of the Internet, the accuracy of some of the classifiers is reduced while other classifiers become more computationally expensive. An increasingly-in-popularity method that does not meet such problems is traffic classification through Machine Learning. The key issue of this classification method is that there is a lack of labeled data to train the models. In this research, we aim to solve this by developing and implementing a platform that generates the required traffic flow data.

1.2 Context

In the literature [7–9], Internet traffic classification is commonly categorized into one of five types: by port matching, by Deep Packet Inspection, statistical classification, behavioral techniques, and Machine Learning.

Machine Learning (ML) classification has shown to be viable both in computing cost and accuracy [10–12], and is comparable to the results obtained by the most precise classifiers [11]. In addition, ML classification does not require to analyze the payload of the packages [12], and in some cases, it only requires statistical information about the flow [7]. By avoiding a payload inspection, it can be said that Machine Learning classification respects privacy laws [13].

ML algorithms (MLAs) are trained with a dataset. In broad terms, if the dataset correlates

its data tuplets with a class (label), each tuplet with its respective class can be used as input so that the resulting algorithm assigns a label to new input; this is known as supervised learning. On the other hand, unsupervised learning does not associate the data tuplets to a class, rather it finds similarities among the training data and sorts it into nameless groups (clusters).

The key problem facing traffic classification by Machine Learning is the lack of *ground truth* [8]. Ground truth is a dataset where a relationship between independent attributes and a dependent attribute exists [14] [15]. While there are some public traces that can be used as training dataset for the MLAs, there is no commonly agreed-upon dataset for most traffic-related classification problems [16].

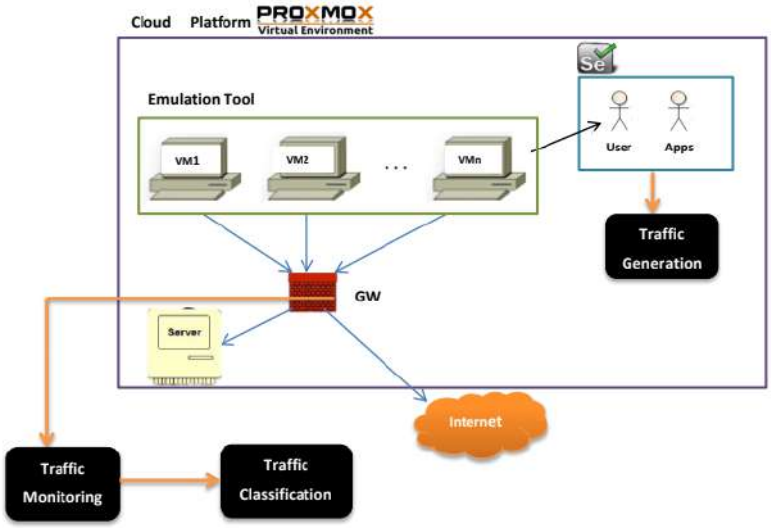
Most datasets used for Internet traffic classification research are generated by collecting traffic traces at different levels of a test network, most commonly, this harvest is done at a university router [12, 17–22]. The problem of a dataset acquired through this method is that it will always lack the required labels for classification. This issue leads to the implementation of either unsupervised classification or label generation for supervised classification.

Unsupervised MLAs can be used to group data in clusters, however, there still is a need for a post-validation process to assign a label to each cluster. Alternatively, many researchers employ payload inspection techniques to label traffic traces [11, 17–21, 23–25]. Unfortunately, it has been demonstrated that payload inspection techniques as labeling methods add incertitude and error to further analysis, in particular to ML [7]. In [7] is mentioned that the label assignment process can be established by an emulation or generation system. Such a system could be used to emulate human-like behavior and generate real traffic flows in a controlled manner so that each time a traffic flow is generated, the system labels it accordingly.

In “Emulating Application and User Behavior on a Cloud Platform for Later Traffic Analysis” [1], a cloud architecture is proposed composed by virtual machines that are able to perform an internet activity that a human user might do (such as web browsing, online video streaming, and so on) automatically. Each virtual machine connects to the Internet through a router. The router is the only virtual machine that is connected to the cloud platform router.

Before performing the activity, the virtual machines that perform the activity send a UDP packet to the router informing which activity they are going to perform, this allows the router to sniff the packets (inbound and outbound) and store them with the label it received. By having a cloud architecture, the virtual machines, as well as the network, are elastic and may adapt according to the requirements. Figure 1.1 illustrates the solution of Aguilar and Sadok [1]

Figure 1.1: Model of Aguilar and Sadok



In the present research, we furthered the study of the architecture of [1] to set a service-oriented approach applied to the cloud platform using the ARCADIA method. In addition, we developed a clustering method to filter out the noise from the emulated traffic flows and classifying the most representative data. Finally, we tested several MLAs and compared their quality metrics. This work was reproduced in an emulated satellite architecture for data collection [26].

1.3 Problem Statement

As presented in [8], one of the biggest obstacles in the study of traffic classification is the lack of a dataset that properly represents common Internet activities. This poses an issue for any ML solution where a dataset is required. To solve this, researches normally use traces from universities, ISPs, or public traces, such as [27]. The main issue with these approaches is that their data is not labeled. A common method to assign labels to traffic flows is through DPI

analysis to, as employed in [28]. Machine Learning algorithms trained with DPI-obtained labels could inherit the deficiencies of that sort of classification, especially the issues on correctly labeling encrypted data.

In this research, we aim to solve the lack of labeled data issue by implementing a cloud platform that enables the systematic creation of labeled datasets. We utilize a cloud platform so that the virtual machines in the solution be always available. By constructing a platform with virtual machines, specifically-task clients can be programmed, and their traffic flows can be captured and labeled. Our proposal is the implementation of such architecture and a later Machine Learning analysis.

In order to achieve the main research objective that is presented in the previous section, the following central research question is defined: *can a ML classifier perform Internet traffic classification with high accuracy if its ground truth was obtained through emulated traffic flows?*. To answer this question we must first decompose it into several questions to study the implications. In turn, each question requires the answers of a sub-set of questions.

1. **What is the background of the elements for the proposed solution?**
    (a) How can Machine Learning solve classification problems?
    (b) Which MLAs exist for classification?
    (c) What is Internet traffic classification?
    (d) Why is Cloud Computing a viable solution for the lack of ground truth?
    (e) What is the background of the current solutions?
    (f) What MLAs are mostly used and how accurate are them?
    (g) How was the ground truth obtained in other researches?
2. **How can traffic flows be emulated?**
    (a) What is the workflow for a traffic emulation system?
    (b) What does the system require?

3. **What is the proposed Machine Learning solution?**

   (a) How can noise be detected?

   (b) Which algorithms are considered?

4. **How is the implementation for this architecture?**

   (a) Which are the physical requirements for the implementation?

   (b) How do the physical components interact with each other?

To answer these questions, we structured the thesis to provide the answers in each chapter.In Chapter 2 the general background is studied. Section 2.2 provides the basic definitions of Machine Learning and types of classification, therefore, answering questions 1a and 1b. In Section 2.3 traffic classification is discussed and in that summary, the question 1c is answered. Section 2.3.2 reviews the state of the art of ML as traffic classifier so we answer questions 1e, 1f, and 1g.

Chapter 3 answers questions 2a and 2b. In Chapter 2.2 the noise filter is described, as well as the offline training and the online metering, answering question 3a. Chapter 5 answers the questions 4a, and 4b. In Chapter 6 the experiments and their results are shown. The results are made to test the viability of the cloud platform, therefore we may answer question 1d, additionally, the selected MLAs are deployed so a general assessment may be performed, this answers question 3b.

1.4 Objectives

The aim of this research is to propose and implement a cloud architecture for traffic flow generation, labeling, and storage, with the purpose of a later classification testing different MLAs. Mainly, the objective of the project is to generate labeled traffic data, separate the resulting data into a training dataset and a testing dataset, train MLAs with the training dataset, identify rep-

resentative data in the testing dataset, perform a sublabeling process, filter out the noise, and identify an application type of each flow through a supervised Machine Learning classifier.

1.4.1 Main Objectives

There are three main aspects of this research, each one has a main objective:

- **Formally model the solution.** Employ a modeling methodology to identify the functional and non-functional requirements of the solution in order to satisfy the needs of Internet traffic classification.
- **Cloud Architecture.** To implement the cloud computing property of availability to constantly generate and store data flows in a database using an autonomic framework.
- **Machine Learning classification.** To design, develop, and implement a filtering and sublabeling system that identifies key flows based on their statistical data for a later supervised ML classification. Test and compare different MLAs' metrics.

1.4.2 Specific Objectives

From the main objectives, we consider the specific requirements.

- **Develop a platform that generates and monitors traffic flows**. The platform must have an architecture that uses the elements of the network to generate the flows required for training.
- **Emulate user and application behavior**. Virtual machines from the cloud platform must be configured so that they can emulate user activities (scenarios) to generate real traffic.
- **Monitor all flows and all observable characteristics**. The captured data is stored in a database for constant monitoring and analysis.

- **Filter and Sublabel data**. From the generated database, the characteristics of the flows for MLA training are grouped by clusters in order to identify key behaviors. Based on such behaviors, establish rules within the clusters so that noise is filtered out and flows that belong to a well-defined cluster become sublabeled.
- **Use supervised MLAs for traffic analysis**. Unfiltered data which has not been sublabeled is classified by different supervised MLAs to compare different metrics.

1.5 Methodology

Given the specific objectives, the following points are proposed.

- Develop a platform that generates and monitors traffic flows.
  - Propose an architecture to emulate real network traffic scenarios.
- Emulate user and application behavior.
  - Implement scenarios in a monitoring platform. Design scenarios comparable to those of the users. When each scenario is implemented, its respective flow is observed in the router of the network.
  - Acquire and store the observable characteristics proposed in the experimental scenarios in a database. Based on the traffic files captured in the *router* a database is managed to take into account the observable characteristics that can be used in the Machine Learning algorithms.
- Monitor all flows and all observable characteristics.
  - Manage and analyze the database. The information in the database must be consolidated to manage the data obtained from the different traffic flows to improve the training of the Machine Learning algorithms.
- Use Machine Learning algorithms for flow traffic analysis.

- Filter data. From the database, run a filter algorithm based on data clusterization to discover noise in the data.
- Train and test Machine Learning models for traffic classification such as Decision Trees, Random Forest, and SVM.

1.6 Contributions

The current research is relevant in two ways:

Practical Relevance. In this research, we addressed the issue of the lack of ground truth for traffic classification MLAs. Specifically, we provided an architecture capable of generating datasets with human-like behavior and used it to test different MLAs.

Scientific Relevance. We provided an implementation of the ARCADIA method for a cloud architecture solution for a Machine Learning process.

1.7 Thesis structure and reading guide

The current thesis has been structured as follows. In Chapter 2, we review different types of internet traffic classification, we give a brief summary of Machine Learning and its most used methods for traffic classification, and we study the state of art of the current ML solutions for traffic classification with emphasis on the method used to gather ground truth. In Chapter 3, we present the system analysis requirements for the cloud architecture. In Chapter 4, we describe the proposed algortihm to improve the accuracy of ML. In Chapter 5, we review the specific structure of the cloud architecture. In Chapter 6, we describe the employed proposal and the data we generated. Finally, in Chapter 7, we summarize the findings and propose future challenges.

Chapter 2

Background

In this chapter, we contextualize the problem and the basis for our proposed solution. Firstly we provide the background on cloud computing as a model which enables the development of our labeled traces generation solution. Through the cloud platform, the output data will then be studied by different MLAs. Finally, we review the state of the art of Internet traffic classification solutions and compare the related works to ours in terms of employed MLA, data preprocessing, labeling method, and source of dataset (i.e., public traces, university router, and so on). Therefore, the remaining of the chapter is organized as follows: Section 2.1 provides the core definitions of cloud computing and studies which requirements the proposed solution needs to meet, Section 2.2 studies the background on Machine Learning solutions, focusing on the means of acquiring a dataset for their training. Section 2.3 reviews the common methods of traffic classification.

2.1 Cloud Computing

Cloud Computing is defined by the NIST referential architecture [29] as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly

provisioned and released with minimal management effort or service provider interaction."

According to [29], cloud computing can be divided into three levels of services, depending on the required configuration and the consumer needs. The first level is Software as a Service (SaaS) which includes the software applications and is targeted to end-users. This level only provides access to the final software, consumers are not involved in the infrastructure or deployment. The second level, also known as the middleware level, is Platform as a Service (PaaS), it provides software building blocks, such as libraries, databases, and Java virtual machines, for software applications development. The third layer, also known as the OS level, is the Infrastructure as a Service (IaaS), it provides operating systems and drivers. An IaaS cloud allows one or multiple guest OS to run virtualized on a single physical host.

In Cloud Computing context, there are two actors that can be used to describe the relation between the different service models: the cloud provider and the cloud consumer. In [2], a cloud provider is defined as a person, organization, or entity responsible for making a service available to cloud consumers. A cloud provider builds the requested services, manages the technical infrastructure, provisions the services at agreed-upon service levels, and protects the security and privacy of the services.

In [2], a cloud consumer is the actor that represents a person or organization that maintains a relationship with, and uses services from, a cloud provider. A cloud consumer browses the service catalog from a cloud provider, requests the appropriate service, sets up service contracts with the cloud provider, and uses the service.

The consumer activities and the provider activities depend on the type of service model that is deployed. Table 2.1 represents the consumer and provider activities based on each service model.

The cloud services levels transfer different amounts of responsibility between the customers and the cloud provider. This shift in responsibility also means that the cloud provider undertakes greater responsibility, ranging from physical and personnel security to the secure development and maintenance of applications and the management of identities for access control. Figure 2.1

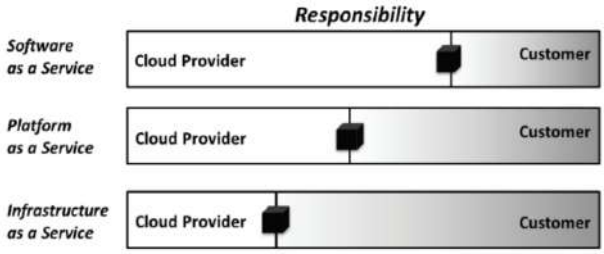
Table 2.1: Cloud Consumer and Cloud Provider Activities

Service Model	**Consumer Activities**	**Provider Activities**
SaaS	Uses application/service for business process operations.	Installs, manages, maintains, and supports the software application on a cloud infrastructure.
PaaS	Develops, tests, deploys, and manages applications hosted in a cloud system.	Provisions and manages cloud infrastructure and middleware for the platform consumers; provides development, deployment, and administration tools to platform consumers.
IaaS	Creates/installs, manages, and monitors services for IT infrastructure operations.	Provisions and manages the physical processing, storage, networking, and the hosting environment and cloud infrastructure for IaaS consumers.

From "NIST Cloud Computing Standards Roadmap" [2]

shows the shifts of responsibilities in the cloud services levels.

Figure 2.1: Cloud Service Levels [2]

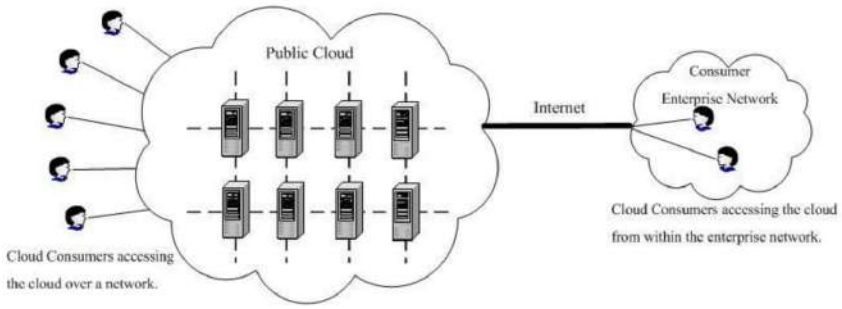


In the recommendation"The NIST Definition of Cloud Computing" [29], the cloud infrastructure operation is divided into four different deployment models: public cloud, private cloud, community cloud, or hybrid cloud. The differences are based on how exclusive the computing resources are made to a Cloud Consumer.

In the public cloud deployment model, the service provider opens up the cloud infrastructure to open use. The infrastructure is in the premises of the service provider, but is operated by whoever uses it. Figure 2.2 illustrates the use of a public cloud in which a consumer, such as an enterprise, make use of it, however other clients are able to use the same cloud.

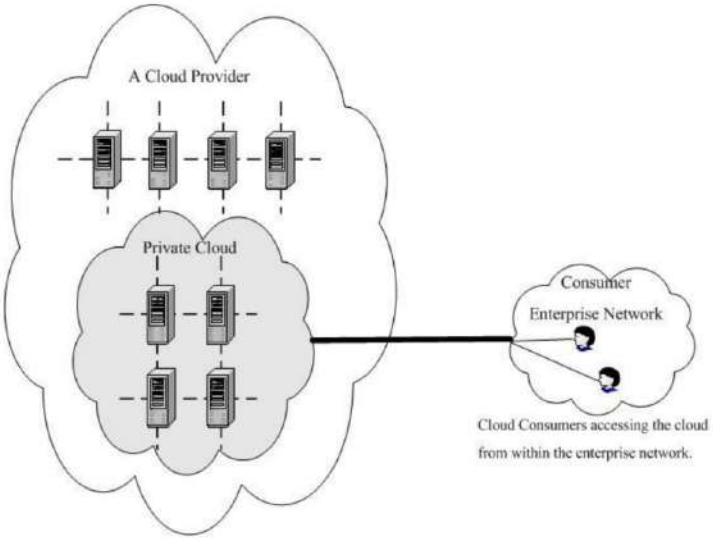
Private cloud is solely owned by a particular institution, organization or enterprise. It gives the exclusive access and usage of the infrastructure and computational resources to an organi-

Figure 2.2: Public Cloud [2]



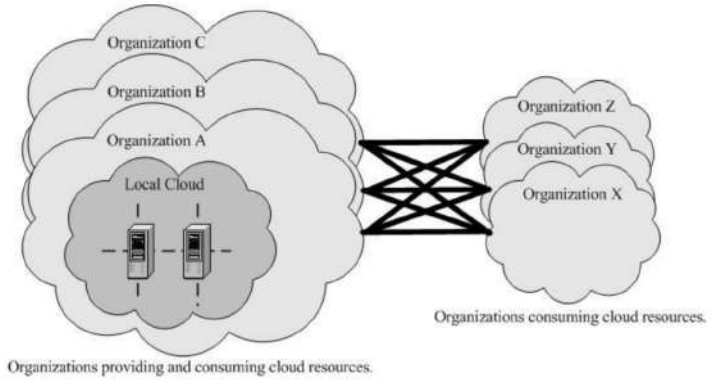
zation. A private cloud can be hosted internally with on-site private clouds, or externally by outsourced private clouds. Private clouds allow management of host applications and other applications used by their customers. Figure 2.3 illustrates an outsourced private cloud in which consumers use a private cloud provided by a third party.

Figure 2.3: Private Cloud [2]



A community cloud is a multi-tenant platform which allows several companies work on the same platform, given that they have similar needs and concerns. Similar to private clouds, a community cloud may be managed by the organizations or by a third party, and may be implemented on customer premise (i.e. on-site community cloud) or outsourced to a hosting company (i.e. outsourced community cloud). Figure 2.4 illustrates an on premise community cloud in which organizations sharing cloud resources.

Figure 2.4: Community Cloud [2]



A hybrid cloud is a composition of two or more clouds (on-site private, on-site community, off-site private, off-site community or public) that remain as distinct entities but are bound together by standardized or proprietary technology that enables data and application portability. Using a hybrid cloud not only allows companies to scale computing resources, it also eliminates the need to make massive capital expenditures to handle short-term spikes in demand as well as when the business needs to free up local resources for more sensitive data or applications.

In the presented solution, the clients are the users of the platform. They provide the virtual machine with a scenario, configure the connection to the Internet through a second virtual machine which will be an instance of a router template. To isolate the virtual machines depending on the user who implemented it, the cloud architecture is defined then as a private deployment model. In addition, the proposed cloud architecture is a PaaS solution, since it will focus on the generation of labeled traffic flows. The generated labeled traffic flows will be then used for the training of classification Machine Learning Algorithms.

2.2 Machine Learning

In this section, we will give a brief summary of the Machine Learning solutions for classification problems. This summary is given as follows: Section 2.2.1 provides a general overview on Machine Learning. In Section 2.2.2, the classification paradigm is summarized and some MLAs for classification are shown in the context of their algorithms.

2.2.1 Introduction to Machine Learning

Machine Learning is the field of study which allows computers to learn without being coded explicitly [30], to achieve this, it is required a learning algorithm. [31] defined learning algorithm as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

As explained in [32], learning is not the task itself rather the means of attaining the ability to perform a task. The improvement over MLAs' performance for task execution is given by the experience. In Machine Learning context, experience denotes a dataset, where every instance is represented with the same set of features [33]. If the dataset has a relation between independent and dependent attributes, then we say that the dataset is labeled [14]. This type of dataset is often referred to as ground truth [15]. Performance in classification MLAs can be defined by their accuracy [32].

2.2.2 Classification by Machine Learning

Classification by Machine Learning is defined by [32] as the task of specifying which of k categories some input belongs to, usually through the implementation of a function $y = f(x)$ so that $f : \mathbb{R}^n \rightarrow \{1, ..., k\}$. Classification problems may be approached by different types of MLAs. Machine Learning may be categorized into supervised, unsupervised, reinforcement, and neural networks [14].

Supervised MLAs require labeled data [33]. As explained in [7], "most of the supervised learning algorithms adjust their model parameters minimizing the error between the model output and the real expected output of an input. This means that historical data has to be labeled."

In contrast, unsupervised MLAs do not require labeled data for they are used to find similarities among data [7]. Unsupervised MLAs, also known as clustering, might be used to discover new classes of items [33]. Clustering algorithms are commonly utilized to form subsets of data

to find abnormalities and similarities in the data [7].

In this section, we will review some of the MLAs used in classification and illustrate their usage by showing their algorithms. Having a grasp on the general implementation of such algorithms will be beneficial for this study since different MLAs will be used to classify different traffic flows. In this section, we will study three kinds of MLAs: supervised, unsupervised, and neural networks. We will not study all classification MLAs given the abundance of MLAs and MLAs variants that could be applied. Because of this, we narrowed the studied algorithms to the following set: for unsupervised algorithms, we will study the K-means algorithm, for supervised we will review Naive Bayes, Linear Discriminant Analysis (LDA)/ Quadratic Discriminant Analysis (QDA), AdaBoost, Decision Trees, and Random Forest, and for neural networks we will study Support Vector Machines with the Radial Basis Function Kernel and the linear kernel and we will review the Multilayer Perceptron feedforward artificial neural network.

K-means

K-means is a popular clustering algorithm, used in works like [34] where it performed traffic classification using only the first five packets of the flow, and in works like [35], where K-means was used alongside Support Vector Machine to classify traffic with an accuracy of 95%.

K-means implements the usage of clusters defined by k centroids (points which are the center of a cluster) of the dataset. A point is considered to be in a particular cluster if it is closer to that cluster's centroid than any other centroid [36]. Algorithm 1 summarizes the steps of K-means as explained by [36], where $x^{(1)}, ..., x^{(m)}$ is the dataset denoted by m instances, and each instance is grouped in cluster $c^{(i)}$ with $i = 1, ..., k$ where k represents the number of clusters.

Naive Bayes

Naive Bayes is a supervised learning algorithm based on the application of the Bayes' theorem, which studies the probability of an event given prior knowledge. In Naive Bayes the prior

Algorithm 1 K-means

Require: $\{x^{(1)},...,x^{(m)}\}, k$
  Initialize cluster centroids $\mu_1,...,\mu_k \in \mathbb{R}^n$ randomly.
  repeat
    for all i **do**
      $c^{(i)} \leftarrow \underset{j}{\text{argmin}} ||x^{(i)} - \mu_j||^2$
    end for
    for all j **do**
      $\mu_j \leftarrow \frac{\sum_{i=1}^{m} \{c^{(i)}=j\} x^{(i)}}{\sum_{i=1}^{m} \{c^{(i)}=j\}}$
    end for
  until convergence

knowledge is the ground truth. Its name comes from the "naive assumption" that stipulates that there is conditional independence between every pair of features given the value of the class variable [37]. Algorithm 2 shows the Naive Bayes algorithm as explained in [38].

Algorithm 2 Naive Bayes

$P(y|x_1,...,x_n) = \frac{P(y)P(x_1,...,x_n|y)}{P(x_1,...,x_n)}$ ▷ Bayes' Theorem

$P(x_i|y,x_1,...,x_{i-1},x_{i+1},...,x_n) = P(x_i|y)$ ▷ Naive Bayes assumption

$P(x_1,...,x_n|y) = P(x_1|y)P(x_2|y,x_1)...P(x_n|y,x_1,...,x_{n-1})$

$P(y|x_1,...,x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1,...,x_n)}$ ▷ We can substitute for all i

$P(y|x_1,...,x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$

Since $P(x_1,...,x_n)$ is constant, we might say the probabilities are proportional. In addition, by analyzing $P(y|x_1,...,x_n)$ we can determine which label is more likely to provide the correct class.

For traffic classification, Naive Bayes is usually modified or used alongside other algorithms for improved accuracy, for instance, in [39] refinements for Naive Bayes traffic classification were made. Without those refinements, Naive Bayes had an accuracy of 65%, however, with the refined variants it had an accuracy of 95%.

LDA/QDA

Linear Discriminant Analysis (LDA) is a supervised classifier that models the class likelihood as a Gaussian distribution in a linear space [40]. LDA is regarded as a simple method that can be derived via a number of approaches [41]. As stated in [42], "(LDA) can be used to perform supervised dimensionality reduction, by projecting the input data to a linear subspace consisting of the directions which maximize the separation between classes".

In [40] it is assured that "(LDA) arises in the special case when we assume that the classes have a common covariance matrix", however, when their covariance matrices are different, the space becomes quadratic. In this sense, LDA is a special case of Quadratic Discriminant Analysis (QDA).

Classification by either LDA or QDA is similar to Naive Bayes in that the process requires the calculation of probabilities. Equation 2.1 show the steps required to calculate the LDA/QDA outcome for classification problems as presented in [42].

$$P(y=k|X) = \frac{P(X|y=k)P(y=k)}{P(X)} \tag{2.1a}$$

$$P(y=k|X) = \frac{P(X|y=k)P(y=k)}{\sum_i P(X|y=i)P(y=i)} \tag{2.1b}$$

$$P(X|y=k) = \frac{1}{(2\pi)^{d/2}|\sum_k|^{1/2}} exp(-\frac{1}{2}(X-\mu_k)^t \sum_k^{-1}(X-\mu_k)) \tag{2.1c}$$

Where d is the number of features.

Adaboost

Introduced in [43], Adaboost is a supervised MLA based on the usage of boosting, "Boosting refers to this general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb." [43] The rules-of-thumb are found by an algorithm

refered to as *weak learner*. The Adaboost algorithm adjusts adaptively to the errors of the weak learners. To achieve this, Adaboost combines the weak hypotheses by summing their probabilistic predictions. Algorithm 3 shows the Adaboost algorithm presented by [43]

Algorithm 3 Adaboost

Require: $< (x_1, y_1), ..., (x_N, y_N) >, y \in \{-1, +1\}$, Iterator T

Initialize weight vector $w_i^1 \leftarrow \frac{1}{N}$ for $i = 1, ..., N$

for $t \leftarrow 1, 10$ **do**

$p^t \leftarrow \frac{w^t}{\sum_{i=1}^{N} w_i^t}$ ▷ Set

$h_t \leftarrow WeakLearner(p^t)$ ▷ *WeakLearner* returns X so that $X \rightarrow [0, 1]$

$\varepsilon_t \leftarrow \sum_{i=1}^{N} p_i^t |h_t(x_i) - y_i|$ ▷ Calculate the error of h_t

Set $\beta_t \leftarrow \varepsilon_t / (1 - \varepsilon_t)$

$w_i^{t+1} \leftarrow w_i^t \beta_t^{1-|h_t(x_i)-y_i|}$ ▷ New weights vector

end for

Decision Tree

Decision Trees are a family of solutions which base their methods in the partition of the feature space into a set of rectangles, and then fit a simple model in each one [40]. A popular tree-based classification method is CART (Classification and Regression Tree). Decision Trees are based on recursive partitioning according to the mean of Y in each region. In CART, the difference between regression and classification is criteria for splitting nodes and pruning the tree.

Random Forest

A random forest is defined by [44] as: "a classifier consisting of a collection of tree-structured classifiers $\{h(x, \Theta_k), k = 1, ...\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x." In that definition, Θ_k represents a random vector of the *kth* tree to create an ensemble of decision tree predictors. Algorithm 4 shows the Random Forest algorithm as presented in [45].

Algorithm 4 Random Forest

Require: A training set: $< (x_1, y_1), ..., (x_N, y_N) >$, features F, and number of trees in forest B

```
function RandomForest(S, F)
    H ← ∅
    for i ∈ 1, ..., B do
        S^(i) ← A bootstrap sample from S
        h_i ← RandomizedTreeLearn(S^(i), F)
        H ← H ∪ {h_i}
    end for
    return H
end function
function RandomizedTreeLearn(S, F)
    for each node do
        f ← very small subset of F
        Split on best feature in f
    end for
    return The learned tree
end function
```

SVM

According to [32], Support Vector Machines (SVM) is a classification supervised learning algorithm that is driven by the linear function $w^\intercal x + b$. SVM outputs class identity rather than probabilities. A defining feature of SVM is the kernel trick, which consists of rewriting MLAs exclusively in terms of dot products between examples, this allows the linear function used by the support vector machine to be re-written as:

$$w^\intercal x + b = b + \sum_{i=1}^{m} \alpha_i x^\intercal x^{(i)} \tag{2.2}$$

A feature function $\phi(x)$ can then be used to produce an output that replaces x so that the kernel function is defined as:

$$k(x, x^{(i)}) = \phi(x) \cdot \phi(x^{(i)})) \tag{2.3}$$

It is also stated in [32] that, in many cases, $\phi(x)$ might be infinite dimensional. To solve

this issue, the kernel function can be rewritten as $k(x,x^{(i)}) = min(x,x^{(i)})$ and would be "exactly equivalent to the corresponding infinite-dimensional dot product". The most commonly used kernel is the Radial Basis Function (RBF) kernel, also known as Gaussian kernel. RBF decreases its value along lines in v space radiating outward from u. RBG can be expressed as:

$$k(u,v) = \mathcal{N}(u-v;0,\sigma^2 I) \tag{2.4}$$

Where $\mathcal{N}(x;\mu,\sum)$ is the standard Normal density.

MLP

Multilayer Perceptrons (MLPs), also known as feedforwarding neural networks, are deep learning models that employ intermediate computations to approximate some function f^*, so that, for instance, a classifier $y = f^*(x)$ maps an input x to a category y. Therefore, MLPs define a mapping $y = f(x;\theta)$ and learns the value of the parameters θ that result in the best function approximation. There are no feedback connections in which outputs of the model are fed back into itself [32]. Algorithm 5 shows the MLP classification algorithm as described in [46].

2.3 Internet Traffic Classification

In this section concepts of traffic classification are reviewed to provide context on the different methods. Section 2.3.1 presents a brief summary of some techniques used in different layers of the OSI model. In section 2.3.2 the traffic classification methods commonly found in the literature are summarized, to support this, section 2.3.2 defines some aspects of the classification by port matching, section 2.3.2 provides context of classification by DPI, section 2.3.2 descibes the statistical based approach. Section 2.3.2 presents the perspective on traffic classification by behavior analysis. In section 2.3.2 the general context of ML as traffic classifier is given.

Algorithm 5 MLP

```
Initialize total error to 0
Apply first pattern as the input and train the neural network
Get error pattern for each output neuron in network and calculate the total error
if last pattern has trained then
    Start again
else
    Load next pattern
    Train
end if
if last has trained then
    if Total error is less than target error then
        STOP Training
    else
        Repeat steps 1-4
    end if
else
    if Last pattern has not trained then
        Repeat 4
    end if
end if
Apply the test patterns as the input to neural network to get the classified results
```

2.3.1 Introduction to Traffic Classification

Traffic classification can be defined as the use of policies to identify a subset of traffic based on the content of some portion of the packet header [47]. In the recommendation [48], it is defined as a tool for specifying a subset of data packets for subsequent treatment as indicated in the action part rule. As stated in [4], traffic classification is a requirement for the traffic flows prioritization, which enhances the performance of a network.

Traffic classification can also be used to provide network security since it can detect new forms of malware that may threaten network links. Furthermore, traffic classification can be implemented in ISPs to optimize their networking services to increase return on capital investments, including application-based service differentiation and content-sensitive pricing [8].

Identification of flows may use the *Type of Class* field for IPv4 packets or the *Traffic Class* field for IPv6 packets, as *Differentiated Services* (Diffserv) does [49]. For instance, in [50]

diffserv (*Differentiated Services* abbreviation) is used alongside the MAC procedures proposed in the standard IEEE 802.11e to provide QoS in wireless LAN networks.

Another tool employed in traffic classification is the MPLS (*Multiprotocol Label Switching*) architecture. MPLS assigns a label to each packet so routers read the label instead of performing a packet header analysis [51].

However, according to [52], the lack of QoS signaling and of an effective service pricing mechanism have obstructed the deployment of QoS solutions such as DiffServ and IntServ, and [41] affirms that "service differentiation mechanisms like diffserv only allow a relatively small number of application classes". In addition, most of the network traffic classifiers at transport layer, are based on port matching [53]. Finally, QoS must minimize the vulnerabilities of the resources and data [54]. To secure services, applications such as Secure Shell (SSH) and Skype encrypt their traffic.

2.3.2 Methods for traffic flow classification

In this section we will review the most commonly used methods for traffic flow classification. There are 5 main methods used: port matching, DPI, Statiscal Analysis, Behavior based, and Machine Learning.

Port matching

As the name implies, port matching classification method aims to identify a packet based on its port number. This sort of classification is considered to be the simplest and quickest form of classification [34] yet it has a low accuracy ($<$70% 55). In [8] three reasons are given to explain this situation: applications without IANA registered ports, users and application designers using alternate ports to avoid firewalls, and shortage of IPv4 addresses. Additionally, some applications, such as P2P services, does not utilize predefined ports [56], and some server ports are dynamically allocated as needed [41]. Finally, classification by port matching might become a

security hassle since, as defined in [41], “trojans and other security (e.g. DoS) attacks generate a large volume of bogus traffic which should not be associated with the applications of the port numbers those attacks use”.

DPI

DPI is defined in the recommendation *Requirements for deep packet inspection in next generation networks* ITU-T Y.2770 [57] as “Analysis, according to the layered protocol architecture OSI-BRM, of payload and/or packet properties”. DPI uses policies conditions, referred as DPI signatures, that identify applications and determine whether an action (policy) should be taken.

In payload-based traffic classification, the payload is compared to a string of bytes to correlate it with some application [8]. Normally, the payload classifiers compare the payload with a set of previously stored signatures (pattern matching) [8].

[11] compared different DPI classification tools. They acquired a dataset of traffic flows, the packages were grouped by flows and each flow was collected together with the name of the process taken from the Windows and Linux sockets. The DPI tools that were tested were PACE, OpenDPI, NDPI, Libprotoident, NBAR, and four L7 Filter variants. Then, they divided the dataset into two collections: in one collection the number of flows per application class was uniform, while in the other collection the number of flows per application class was diverse, imitating a more realistic scenario. When the DPI tools were trained in the dataset with a variant number of flows per application class, the classifier with the highest accuracy was PACE, with a 93.5% rate of perfectly classified packets (the rest were misclassified, not classified at all, or classified correctly although with a lesser granularity). When trained with the uniform number of flows per application class dataset, the classifier with the highest accuracy was NDPI, with a 82.73% rate of perfectly classified packets.

DPI classification is highly accurate but costly in terms of processing [58]. Moreoever, if the payload is encrypted, DPI classification become less accurate [59] [60]. This poses a major challenge since the traffic of encrypted data has become a standard nowadays [61], because of

this, in *Traffic classification on the fly*, Bernaille et al. [34] qualify this sort of classification as "unrealistic". Finally, since DPI studies the payload, it may be considered intrusive in terms of legal privacy [13].

Statistical Analysis

Classification using statistical approaches analyze traffic flows to correlate information among them and/or the network configurations [7]. For example, in [56] a protocol fingerprint is proposed, where the fingerprint represents IP flows produced by network applications exchanging data through TCP connections. In [41] a classification method is presented based on the statistics of applications in order to form signatures designed by the way they are used, e.g. interactively or for bulk data. Some of the statistical-based approaches are adapted and improved by the ML techniques [7].

Behavioral Analysis

Behavioral techniques try to identify the application that the host is using by analyzing the patterns (number of connected hosts, transport layer protocol) of generated traffic 62. Classification based on host behavior has proven to provide better results when combined with payload classification [63]. An example of this is [59], where the heuristic proposed in [8] was used. [64] consists in the creation of IP addresses and ports pairs. This kind of analysis requires access to the TCP/UDP packets in both the receiver and the transmitter.

[7] asserts that behavioral patterns can also be used in graph modeling to connect hosts where "graph theory is used to find highly connected nodes (hosts), number of connections, and opened ports, among others". In [65] is said that the drawback of behavioral solutions is the specificity of their requirements.

Machine Learning

Machine Learning classification has shown to be viable both in cost and accuracy [10] [11] [12]. Previously we mentioned [11], where several DPI classification tools were compared. In that work, the Machine Learning algorithm *C5.0* was tested. To test it, the algorithm was trained with two sets of data: one with the same number of flows per application class, and another with different number of flows per application class. The purpose of training the MLA with two different datasets was to determine whether emulation of traffic flows could be used as a method to generate training datasets for Machine Learning. The dataset with the same number of flows per application class represents emulated data, while the dataset with different number of flows per application class represents the opposite: data generated by actual traffic consumption.

When trained with the same number of flows per application class, the MLA had an accuracy of 60.91% while when trained with a different number of flows per application class, the MLA had an accuracy of 98.45%. This showed that datasets used for MLA training require a certain degree of human-like behavior, such as an environment capable of generating traffic flows as close in diversity as the ones employed in traffic classification.

The challenge of using Machine Learning classifier is the absence of ground truth [8]. In most of the literature, researchers mainly use an implementation of local package capture, as in [23]. However, this lack of dataset standard represents an issue that may imply the usage of privacy laws violations [13] as well as the employment of old data. Researches in the area encourage the development of tools that label training data [66].

This issue grows severe given the rapid increase of new applications. In [67] is explained that traffic clustering has the potential of identifying unknown applications, and propose a two-phase clustering algorithm using statistical and packet payload features.

Works of MLAs for Internet Traffic Classification

Given that there are several categories and implementations of MLAs, many have been studied as network traffic classifiers. These studies cover different aspects of this topic. Firstly, it has to be defined whether the tested method is to make a coarse-grained or a fine-grained classification. According to [9], coarse-grained classification classifies data according to the application layer protocols, such as HTTP and SMTP, whereas a fine-grained classification classifies data according to specific applications. In the QoS context, is more common to work with coarse-grained solutions because the requirements within each group are very similar [10], for this reason, the presented solutions have a coarse-grained focus.

Another aspect that a study has to define is how the data is treated, whether it used a sort of filter, data preprocessing, or data cleaning previous to the classification. In addition, it should also state which MLA was utilized during the research. However, most importantly, it needs to state how the training dataset was obtained, and if it was labeled, how it was so. Table 2.2 summarizes the work of MLAs as classifiers.

Table 2.2: Summary of works about MLAs as traffic classifiers

Work	Summary	Data Preprocessing	Trainig Data	Labeling
Erman et al. [17]	K-means, DBSCAN, and Auto-Class	Same number of samples per flow type selected randomly	Flows generated in the Internet link of the University of Calgary	DPI complemented with port-based techniques
Bujlow et al. [10]	C5.0	Noise Filter	Volunteer-Based System	Volunteer-Based System
Bujlow et al. [11]	C5.0	-	Volunteer-Based System	nDPI complemented with knowledge from the HTTP headers
Li and Moore [12]	C4.5	Filter based on TCP flow characteristics	University router	Hand classified using a content-based approach
Li et al. [18]	SVM	-	University router	L7 Filter
Este et al. [19]	SVM	-	Faculty router	nDPI
Wang et al. [20]	Random forest	-	Traces of the University of Beihang	nDPI
Erman et al. [21]	K-Means, DBSCAN, and EM Clustering	-	Traces of university	DPI for non-encrypted data and port-based heuristics for encrypted data
Erman et al. [22]	Semi-supervised algorithms	-	Traces of university	Cluster-based probabilistic assignment

Singh and Agrawal [23]	MLP, RBF, C4.5, Bayes Net, Naive Bayes	Experiments performed with two datasets, one full-feature and the other one with reduced features	Package capture	Wireshark
Singh [24]	K-means and EM	Correlation-Based Feature Selection	Package capture	Packet Capturing Software
Aouini et al. [25]	C5.0 and KNN	Extension of nProbe Tool called *Learning Feature Extraction*	From french ISP	nDPI complemented with knowledge from HTTP headers
Bakhshi and Ghita [68]	K-means and C5.0	Conversion to Netflow format and Feature Selection expansion with nfdump utility	Netflow records collected from 2 PCs in two environments: typical residential premises and an academic setting	nDPI
Carela-Español et al. [28]	C4.5	-	Netflow records collected from the Polytechnical University of Catalunya (UPC)	L7 Filter
Present Work	Adaboost, MLP, NaiveBayes, QDA, KNN, Decision Tree, Random Forest	Noise Filter & Sublabeling	Cloud Platform	Cloud Platform

[17] implements the management of clustering MLAs for traffic classification with data from the transport layer; the traffic used as training consisted of flows generated in the Internet link of the University of Calgary for one hour (60 GB) and a public trace of the University of Auckland. In [12] a C4.5 algorithm was used that classified with a 99.8% accuracy, the traces collected for the training were gotten during two consecutive weekly days of Internet traffic with an interval of 8 months at the University of Cambridge. In [10] the training data was obtained through the association of application name and flows, which was gotten through a system of volunteers which consisted in installing clients on the computers of the volunteers and on a server responsible of the collected data storage; a C5.0 algorithm was implemented and had an average accuracy between 99.3% and 99.9%.

In [18] a *Support Vector Machine* (SVM) was used to classify 7 different classes of applications with an accuracy of 96.9% when the data is not biased (that is, there is the same number of flows per application class) and 99.4% when there is bias; the data used was obtained from a router of the university for a total of 8 hours in a period of one week. In[19] the application

protocol responsible for sending packets through a monitoring node is recognized, its analysis is by SVM obtaining an accuracy of 92.37% in the data collected from the faculty router.

[20] proposes an improvement to the network traffic classification by random forest that consists in granting a probability of selection to each variable according to its classification priority. The model was trained with traces from Beihang University, which were collected on campus. The network traffic classification by standard random forest was compared with the proposed version and there was an average accuracy improvement from 96.846% to 96.999%.

[21] uses non-supervised Machine Learning algorithms to form clusters of unlabeled flows in real time (classified according to the first TCP packets of a communication) with 95% accuracy, their training data was obtained from an external university. [22] classifies in real-time using semi-supervised algorithms with an average accuracy of 94%. The proposed semi-supervised algorithm consists of the clusterization of labeled and unlabeled flows. This approach allows the mapping of unlabeled flows to known labeled data.

In [23] compared five classification MLAs: MLP, RBF, C4.5, Bayes Network, and Naive Bayes, and tested them with a dataset obtained by package capture. From the original dataset, a second one was created through the implementation of Feature Reduction named *Best First*. Both datasets were used in the experimentations. It was concluded that Bayes Network and C4.5 are effective ML techniques for IP traffic classification with accuracy in the range of 94%. In [24] K-means was compared with Expectation Maximization (EM) concluding that K-Means is better than EM. Their data was obtained by package capture.

In [68] individual flow classes were derived per application through K-means and were further used to train a C5.0 decision tree classifier, in order to improve the results of records from the monitoring tool Netflow. The obtained accuracy in that research was 92.37%. In [25] an approach is proposed that consists of early identification of packets using the C5.0 algorithm achieving an accuracy of 98.8%

2.4 Conclusions

In Section 2.1, we defined the types of service that are offered in a Cloud Computing context, as well as deployment models which can be used to determine a certain behavior within the cloud platform.

In this chapter we divided the background in four sections. In Section 2.3.2, we described the state of the current situation on traffic classification. In Section 2.2 we summarized some aspects of Machine Learning in the context of classification problems. We established definitions and concepts that help us contextualize the issue with the lack of ground truth and the advantages of having an architecture that generates labeled data. We reviewed the implementation requirements for unsupervised MLAs and supervised MLAs. In addition, we studied different MLAs. We answered the questions: *How can Machine Learning solve classification problems?*, and *Which MLAs exist for classification?*.

In Section 2.3, we gave a brief summary of the Internet classification methods. We mentioned the built-in mechanisms such as diffserv, and MPLS to provide QoS. In addition, we listed the most common methods for traffic flow classification with some of their characteristics. We answered the question: *What methods are most commonly used?*.

And in Section 2.3.2, we reviewed the most common methods for internet traffic classification found in the literature. We compared their accuracies and observed how was the ground truth gotten. By doing this analysis we have answered the questions *What is the background of current solutions*, *What MLAs are mostly used and how accurate are them?*, and *How was the ground truth obtained in other researches?*.

Chapter 3

Requirements Analysis

In this chapter, we present our proposal: a cloud architecture capable of creating labeled traffic flows for later ML analysis. In Section 3.1, we provide the definition of the ARCADIA methodology. In Section 3.2, we define the principles used for this architecture. In section 3.3, the operational capabilities are studied and in section 3.4 the system requirements for a specific scenario analyzed.

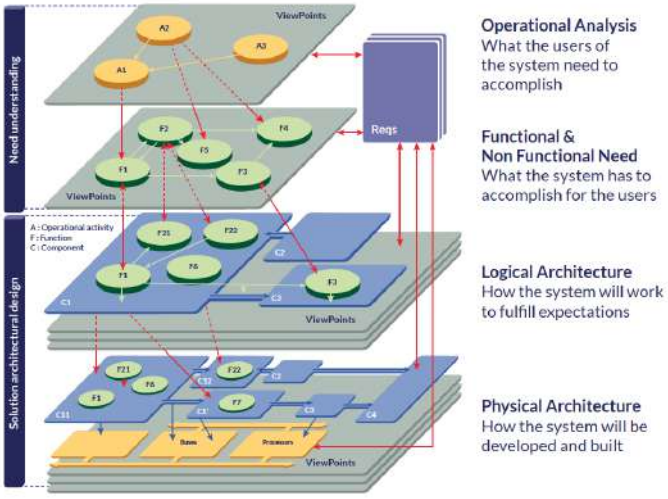
3.1 ARCADIA

As part of the methodology, we implemented the ARCADIA method to analyze the system requirements for the cloud architecture, and then used the Capella software to implement the design. ARCADIA (ARChitecture Analysis and Design Integrated Approached) is a Model-Based engineering method for systems, hardware, and software architectural design. It enforces an approach structured on successive phases which establishes a separation between needs and solutions. The needs are represented in the operational analysis and in the system analysis, while the solutions are reviewed in the logical and physical architectures [69].

The first layer is the *Operational Analysis* which is focused on the user's activities. It aims to establish what system users must achieve by the identification of actors that have to interact with

the system, their goals, and the interaction between them. The second layer is the *Functional and Non Functional Need* and represents the requirements for the system, i.e., this layer provides the system analysis. The third layer is the *Logical Architecture*, which represents how will the system work to meet expectations, it describes the functions to be performed and assembled, and continues with the identification of the operational components. Lastly, there is the *Physical Architecture*, explains how will the system be built with a focus on the functions required by the implementation and technical choices, and reveals the behavioral components that perform these functions [3]. Figure 3.1 summarizes the phases of the ARCADIA method.

Figure 3.1: ARCADIA Phases [3]



By using the ARCADIA method, we adhere our solution to a standard, so it may be replicated in other research projects. To implement ARCADIA, we used the Capella software.

Capella

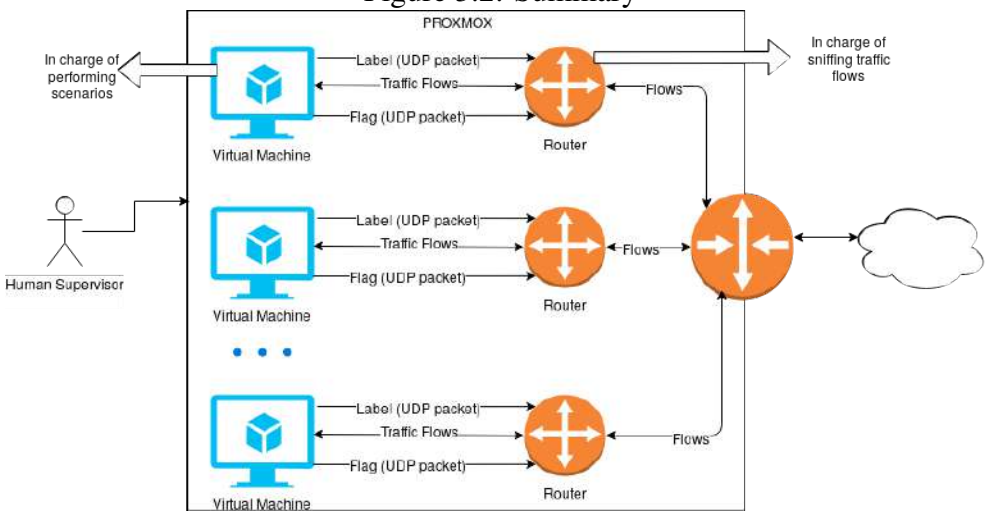
Capella is the Model-Driven Engineering (MDE) open-source solution provided by Thales. It is a Tooled-Up-Process solution that enacts Systems Engineering and Software Architecture Engineering recommendations defined by ARCADIA [70]. This solution provides a process and tools for graphical modeling of systems, hardware or software architectures.

3.2 Proposed Cloud Architecture

This thesis is based in "Emulating Application and User Behavior on a Cloud Platform for Later Traffic Analysis"[1]. We used an Infrastructure as a Service model (IaaS) because in this service model the consumer is able to implement and execute any software [29]. The IaaS consumers have access to virtual machines, accessible storage through the network and network infrastructure components [71]. Once the model is deployed, we may offer a Platform as a Service (PaaS) solution for the implementation of distinct models for traffic flow generation.

In our architecture, as illustrated in Figure 3.2, we propose the employment of two types of virtual machines: clients, and routers. Each client machine connects to the Internet through a router, where the latter performs a sniffing process and stores the generated traffic flows in local *pcap* files. The client machine performs certain internet activity that resembles human-like behavior. Each client machine will generate traffic flows based on the activity of the employed application; we call these activities *scenarios*. Before the deployment of a scenario, the client sends a UDP packet to the router in order to inform it about the type of flow is going to send and/or request, this allows the router to store the pcap files by flow type, and thus, providing a labeling system. In addition, once the scenario is over, the client virtual machine (VM) sends another UDP packet to inform the router that the performance of the scenario has finished.

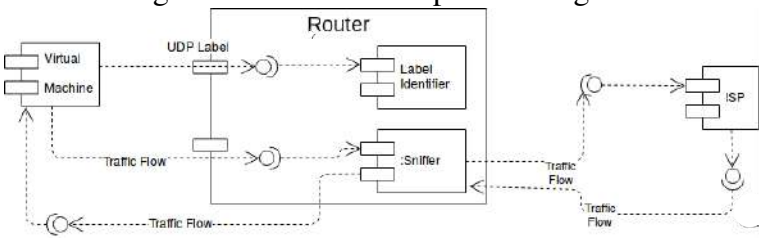
Figure 3.2: Summary



Each client has two components: the agent of organization and the performer. The *per-*

former is the component in charge of the scenarios implementation. Given that we only use one scenario per client machine, the labeling process at the router is simple. In Figure 3.3, the relationship between the router component, the sniffer subcomponent, and the client virtual machine is shown.

Figure 3.3: Router Component Diagram



With the definition of the general concept of this solution, we implement the ARCADIA method using the Capella software to model the operational requirements and the functional and non-functional need, this process constitutes the requirements analysis. From the requirement analysis, we are able to find the key elements and define them. Once every element is defined, we can model the logical and physical architecture.

3.3 Operational Analysis

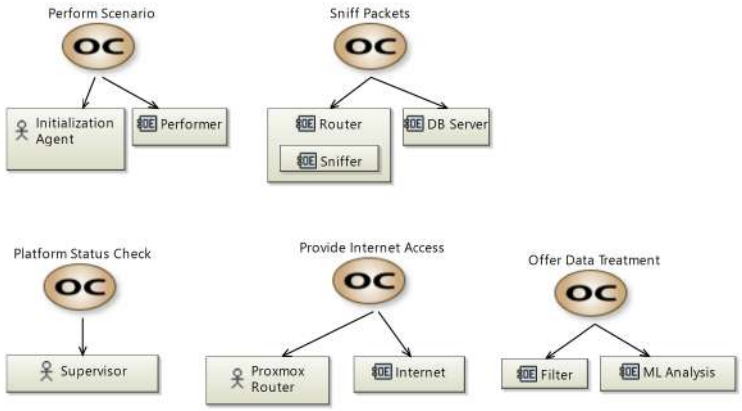
In this research, we propose the implementation of a cloud architecture for labeled traffic generation and ML classification. The first step towards a proper design for the solution is to determine the basic needs that have to be satisfied by the proposed solution. To achieve this, we use the ARCADIA method with Capella.

In the ARCADIA method, as defined in [3], the operational analysis captures what system users must achieve as part of their work or mission. In the operational analysis, the model is based without any assumptions about how the system will contribute; in other words, the operational analysis studies the problem and defines which steps are required in a general view. In this sense, the actual system is irrelevant in the operational analysis.

Given that the goal of the proposed architecture is to provide an environment of data gener-

ation and data labeling, the operational capabilities can be defined as follows: scenario implementation, packet sniffing, platform status check, provide Internet access, and data treatment. Figure 3.4 represents the operational capabilities as well as the actors and components they require. The aim of this proposal is the systematic generation of labeled data for ML analysis, to provide this, five operational capabilities must be satisfied.

Figure 3.4: Operational Capabilties



Provide Internet Access

The first capability the system ought to have is accesing to the Internet. Since in this level of the ARCADIA method is imperative to define actors and their interactions with the system, we may define the router of the cloud platform as the first actor in the model, having the Internet itself as a component.

Perform Scenario

The second operational capability is scenario performance. We define scenario performance as the deployment of a set of steps required to mimic human-like behavior of certain application activity. The two components required for the implementation are the initialization agent and the performer.

Sniff Packets

One of the most relevant capabilities in the system is regarding packet sniffing. This capability enables the data generation and its labelization. This capability requires the sniffer component, however, since our proposed solution is to capture data at the router, we define the sniffer component as a subcomponent for the router component. For each scenario performance, the captured data ought to be stored in a database for later analysis.

Offer Data Treatment

If the ground truth generation requires the scenario implementation capability and the packet sniffing capability, then the storage of ground data requires the data treatment operational capability. In this level, and for any given scenario, data treatment will be defined as a filtering process which enables the identification of behaviors in the data and an ML analysis. The behaviors found through the filter component facilitate pattern recognition among the data, which enables a sublabeling and filtering process. The ML analysis component is responsible for the MLA selection process and their implementation.

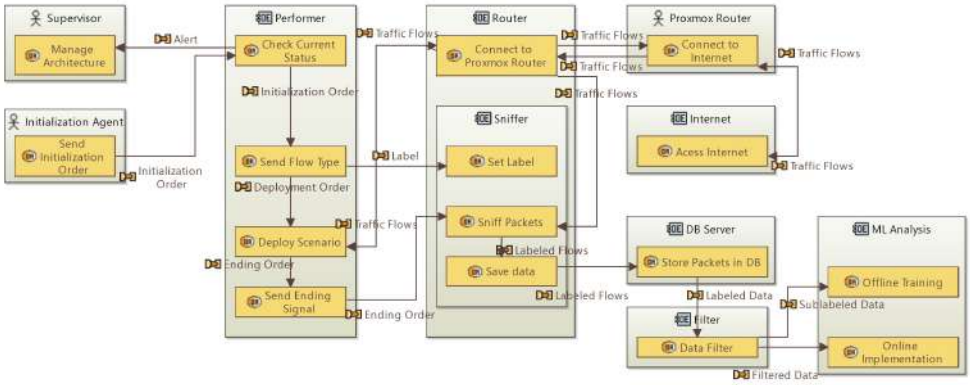
Manage Platform

Lastly, the architecture should inform a human supervisor if it detects an error during the implementation of the scenarios. For this reason, a human actor is included in the model. Since the platform is autonomous, the responsibility of the actor is to check the status of the platform and if out-of-scope errors are met, the human supervisor has to manage them. In this regard, an out-of-scope error is an error which the platform cannot resolve.

3.3.1 Operational Contex Model

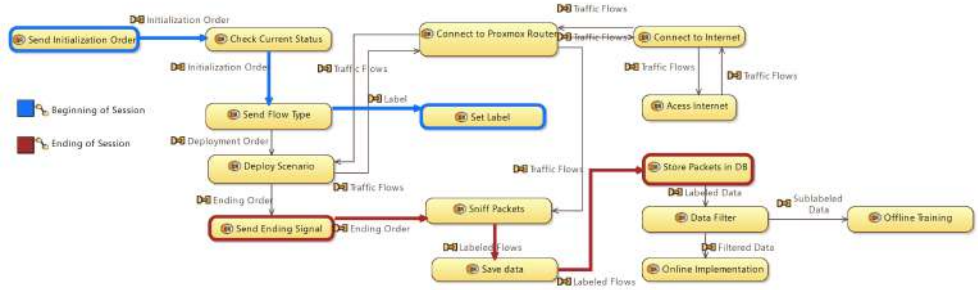
Figure 3.5 shows the operational context for this solution, where it can be observed that the process begin when the initialization agent sends the activation order to the performer, the performer then reviews its own status. Upon reviewal, the performer indicates to the sniffer the type of flow it will generate, this sets the label. The performer deploys the scenario through the connection given by the router, the router connects to the Internet through the router and utilizes the sniffer to capture the packet data. Once the scenario finishes its deployment, an ending acknowledgment is sent to the sniffer so it the label and store the data in a database server. From the database server, the ground truth is obtained and it is separated into two sets. The first set is sublabeled for the offline training and the second filters noise from the data and use it to test the online implementation.

Figure 3.5: Operational Context



The flow of the process can be seen in Figure 3.6, which represents the operational activity. The operational activity diagram illustrates how the solution will interact with its elements. In our proposed solution, the operational activity shows the processes of the beginning of any given scenario and its proper ending.

Figure 3.6: Operational Activities



3.4 System Analysis

According to [3], the system analysis delimits the functions required of the system, distinguishing them from those assumed by the users or external systems. It is also said that, in system analysis, it is essential to limit the functional analysis to capture only the need, excluding any implementation choice or details.

Through the operational analysis, we can perform a mission analysis to model the functional needs of our model. Each mission is composed of a set of capabilities and actors. From the operational capabilities analysis, four key missions can be defined: scenario implementation, packet sniffing, platform supervision, and data treatment.

Implementation

For the scenario implementation mission, the system requires to generate flows according to the scenario, i.e., the scenario must work properly. Therefore, the first capability can be summarized as the generation of flows. The second capability for this mission is the performance of self status to determine whether the current configuration is providing the expected results. A third and fourth capability the mission requires is the communication with the sniffer component, to send the flow type first (send label) and, once the implementation is over, it should send an acknowledgment of the scenario ending its performance. The scenario implementation mission requires the initialization agent and the performer component as actors.

Packet Sniffing

The packet sniffing mission is based on two capabilities and two actors. The first capability is to monitor packets and use their statistical features to form flows and label them with the information sent by the performer. Once the sniffer receives the acknowledgment of the conclusion of the scenario, it is required to store the labeled data in a database. The involved actors are the sniffer component and the database server.

Platform Supervision

The platform supervision mission depends on the activities performed by the human supervisor, which in this context, is an actor. The capabilities of this mission are to configure the platform and to receive notifications from the system. The system must be capable of detecting an error and inform the supervisor of it. In turn, the supervisor should be capable of configuring the elements of the architecture.

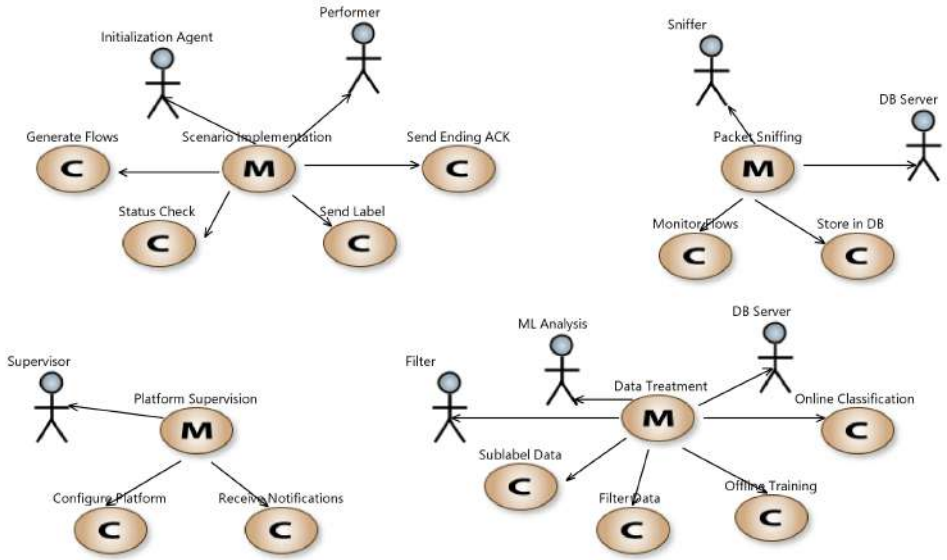
Data Treatment

Lastly, the data treatment mission requires four capabilities. In this proposal is presented a sub-labeling and filtering solution for a later classification MLA. The solution requires an offline training for an online classification. These four components comprise the capabilities for this mission. The actors required for the management of this mission are the filter, the ML analysis, and the database server. Figure 3.7 displays the four missions, their capabilities, and their involved actors.

3.4.1 Functional & Non Functional Need

[3] also states that from the operational analysis it is required to define the actors which the system will interact with and to assign each of the operational capabilities to a function. The

Figure 3.7: System Missions



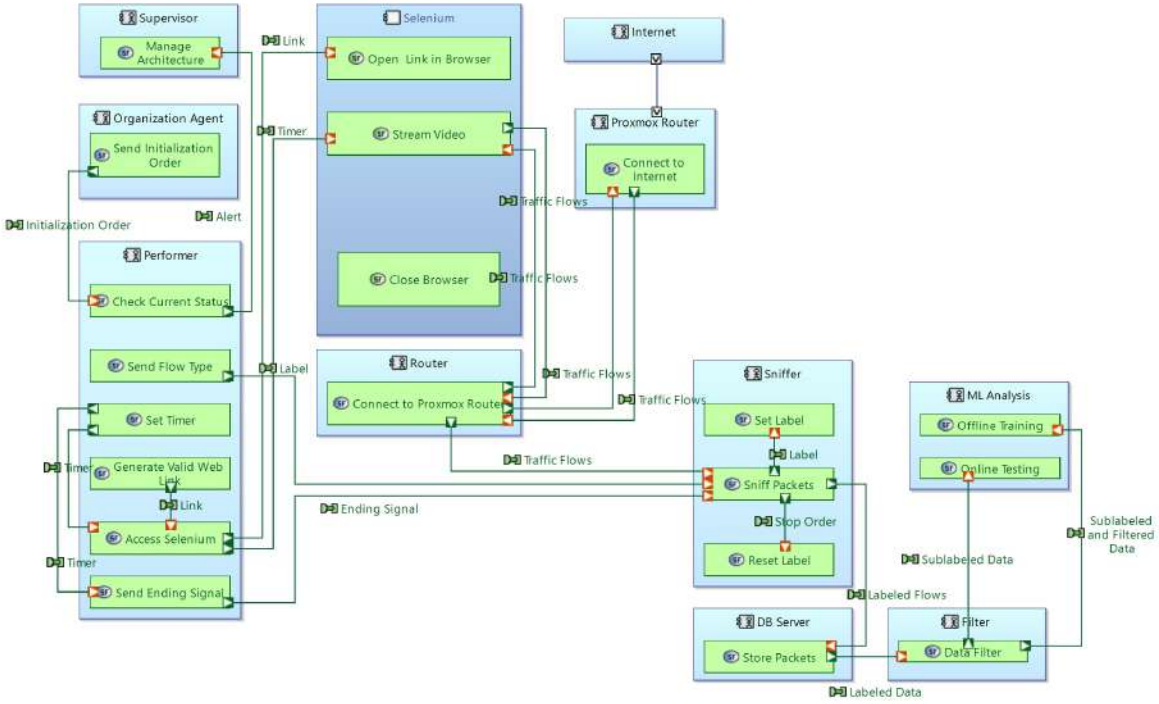
creation of these function correlate with the implementation of the scenario. Certain activities require additional systems that enrich the scenario with features that allow better performance. To illustrate the system requirements, Figure 3.8 shows the system analysis for the streaming service performer.

The scenario requires the implementation of a system that allows the streaming service on a browser, therefore, browser control is needed. The implementation of the scenario shares the same general structure of the scenario, i.e., the initialization agent indicates the activation of the scenario, the performer, if able, deploys the scenario, the router connects the performer to the Internet while the sniffer captures data and later stores them in a database server. However, to implement browser control, it is required an additional system. To achieve this, we propose the usage of the Selenium API.

Selenium can be defined as a system that operates based on the input of the performer. Once integrated to the rest of the system, the performer is contextualized with Selenium so that it generates a web link to a streaming service and send it to the Selenium system to generate the corresponding traffic flows.

Through the implementation shown in Figure 3.8 the system analysis for the streaming video

Figure 3.8: System Analysis



scenario is represented, hence it can be said that in a diverse collection of scenarios, different systems will be required. In the scenarios used for this research, it was found the requirement for different systems in three scenarios.

In the scenario of email activity, an additional system is required to provide the mailing service. For the scenarios of chat and file transfer, the Skype API might be implemented. As a system, the Skype API should be able to connect to a valid account and send text and/or files to another valid Skype account. Similarly, the P2P scenario would require an additional system to function as a seed and/or leech.

3.5 Conclusions

In this chapter we presented our system analysis for a cloud architecture for the creation and labeling of internet traffic flows. We defined the system requirements based on the operational

capabilities obtained through the usage of the ARCADIA method. We answered the questions *What is the workflow for a traffic emulation system?*, and *What does the system requires?*.

From the system architecture, we can observe that the requirements of the system will be determined by a sort of activity, for instance, the streaming service performer will need a different set of components than the FTP performer, however, this platform is designed to generate data which shall go through a sublabeling and filtering process based on ML, therefore, it is required to define the ML components that will be part of the proposed solution.

As part of an integrated solution, this research aims to design a service-oriented model which implementation uses ML as a process of filtering, sublabeling, and classification, hence the ML components are prerequisites of the implementation. In short, the system requirement analysis demonstrated that to implement the solution, the ML components have to be defined before structuring the implementation of the solution. In the following chapter, the ML solution will be defined so that during the implementation design, we understand the components before allocating them in the final solution.

Chapter 4

Machine Learning solution

In this chapter we define our Machine Learning solution for traffic flow classification based on the requirements analysis. Specifically, this chapter analyses the behavior of the data filter component and its relation to the classification process, to achieve this, Section 4.2 reviews the data requirements for the application of MLAs in this context. In Section 4.3, a two-step method for data preprocessing is presented consisting in offline sublabeling process and an online filter.

4.1 Proposal

A traffic flow is defined by the recommendation [72] as "a sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination that a node desires to label as a flow". We may define as flow all the packets related to a given application use. There are different types of packets in the same flow for instance, the packets used in a TLS handshake may be distinct from the rest of the packets. Moreover, every packet generated by the application is part of the flow disregarding in which node it was instantiated.

In this research, the ML components need to classify traffic flows based on their statistical data. Such data will be retrieved from the DB servers od the platform, which store the data of the sniffed packets. Therefore, the ML components will use the retrieved data during its training

for later implementation. We approach this requirement by studying the training phase of the algorithm and its implementation. The training phase of the algorithm will be referred to as the offline training while the implementation will be referred to as the online testing.

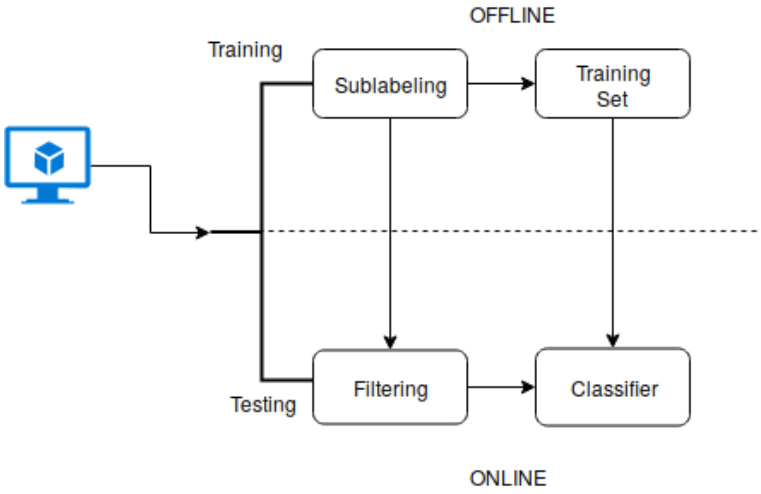
During the offline training, an ML classifier is trained with traffic flows data, however, instead of adhering to one MLA, we propose the usage of a previous clusterization of data to form groups of data that share features. If a cluster has a strong bias towards a class, then there is a reasonable argument to declare that flows within that cluster belong to the biased class. Likewise, if a cluster has non-representative data, i.e. flows within the cluster have similar features despite being from different classes and, in relation to the other clusters, the number of flows within the cluster is small, then flows that belong to the cluster can be defined as noise and therefore, they can be filtered out. In short, we can use clusterization to identify flows based on the cluster they are grouped in and make a decision based on the cluster. Additionally, the same filter can be inherited from the offline training to the online testing so that, during implementation, the workload for the ML classifier is reduced.

The online training consists in labeling the data, and omit those entries which are labeled as noise. Additionally, the online training uses the clusters formed in the offline training to group subflows and, if possible, label them. To label a flow during online training, the cluster that it is assigned to must have a defined label obtained during the offline training. By doing this, the number of flows labeled by the ML classifier is reduced. A method similar to this was used in [68]. Figure 4.1 shows the implementation of the online and offline approach of our solution, where data is separated in a dataset for offline training and a dataset for online testing.

4.2 Data collection and labeling

In [73] a 5-tuple is proposed for identifying packets by subflow. The 5-tuple consist of the source IP address, the destination IP adress, the source port, the destination port, and the used protocol. In order to have one subflow per direction as opposed to two, in the mentioned work it's suggested to reverse the source and destination elements of the packets in the opposite direction

Figure 4.1: Offline and Online Architecture



of the first observed packet. Therefore, we can build a database containing the statistical information regarding each subflow and their respective flow. While in [73] the data set is classified by 249 features, we selected 22 as Machine Learning features after applying a feature selection process.

The first features we selected were the number of packets of any given flow and divided them in two features: number of packets from source to destination, and number of packets from destination to source. Additionally, we selected 20 features shown in Table 4.1, which utilizes the Inter-Arrival Time of each packet (IAT) in the Source to Destination direction (SD) as well as the Destination to Source direction (DS). Likewise, it uses the packet length in both directions.

Table 4.1: Additional selected features for statistical classification

Property\Stat	**Median**	**Mean**	**Max**	**Min**	**Variance**
IAT SD	*	*	*	*	*
IAT DS	*	*	*	*	*
Packet Length SD	*	*	*	*	*
Packet Length DS	*	*	*	*	*

IAT stands for Inter-Arrival Time
The *SD* banner marks the packets going from the Source of the initial node to the Destination of the intial node
The *DS* banner marks the packets going from the Destination of the initial node to the Source of the intial node

By using these statistical features, we can analyze the subflow behavior, including which

flow generated it.

4.3 Sublabeling and Filter

Machine Learning can be used to obtain relevant information in a dataset, for instance [74] presents an algorithm to discover knowledge in databases implementing data mining methods which, in their majority, are based on ML techniques [7]. [66] employs the K-means clustering algorithm to group a dataset by similarities.

In our solution we implemented the K-means algorithm to sort the subflows into clusters. Each cluster might consist in an arrange of heterogeneous flow types since several flows generate similar subflows. Since the clusterization algorithm does not take into account the labels of the flows, the similarities it finds will be based solely on the subflow characteristics.

To define the number of clusters, we used the elbow analysis. We obtained the inertia and compared it with an incline. During the experimentation, we noted that the maximum value for the best number of clusters is lesser than the number e raised to the number of flow types (i.e., e^{NFT} where NFT is the number of flow types), however, it usually is greater than the number of applications, and the square of the number of applications, but lesser than the number of flow types raised to itself (i.e., NFT^{NFT}) . However, to reduce computing time, we set the maximum number of clusters to be ten times the number of flow types (i.e., $10 * NFT$), since it works during a linear time and in most cases, the best number of clusters is lesser than ten times the number of flow types.

Based on the clusters, it is viable to label cluster with its most representative flow type, if there is such label. Firstly we must determine whether a label exist within a cluster so that the cluster might be labeled based on the subflows it contains. To determine this we propose, in Algorithm 6, a method. This method consists in obtaining the relativity of flows per cluster, that is to say, if cluster C_i can be represented as a group that might contains $C_{i,j}$ where j is a flow

type, then the relative cluster might be defined as:

$$\hat{C_{i,j}} = \frac{C_{i,j}}{\sum_j C_{i,j}} \tag{4.1}$$

If we then define the relevance of a cluster in terms of flows, we might express it as $R = \sum_j C_{i,j}$. We can apply a similar method to discover noise in the cluster.

Algorithm 6 Sublabeling

Require: $C, N, M, CERTAINTY, NOISE$
  $NewLabels \in \emptyset^N$
  for $i \leftarrow 1, N$ **do**
    $R_i = \sum_{\forall j} C_{i,j}$
    for $j \leftarrow 1, M$ **do**
      $\hat{C_{i,j}} = \frac{C_{i,j}}{R_i}$
    end for
  end for
  for $i \leftarrow 1, N$ **do**
    $\hat{R} = \frac{R_i}{\sum_{\forall j} R_j}$
    $MAX = 0$
    $L_{MAX} \leftarrow \emptyset$
    for $j \leftarrow 1, M$ **do**
      if $\hat{C_{i,j}} > MAX$ **then**
        $MAX = \hat{C_{i,j}}$
        $L_{MAX} \leftarrow j$
      end if
    end for
    if $1 - MAX * \hat{R} > CERTAINTY$ **then**
      $NewLabels_i = L_{MAX}$
    end if
    if $\hat{R} < NOISE$ **then**
      $NewLabels_i =$"noise"
    end if
  end for
  return $NewLabels$

Where $CERTAINTY$ defines the accuracy that would have been obtained if the cluster was relabeled with the application with most flows in it. $NOISE$ is the minimum relevance that a cluster should have to not be classified as noise. For $NewLabels$ there are three possible outcomes: a previously defined label, the label *noise*, which implies that the cluster is noisy, or

an empty string, which reveals that there is not sufficient information within a cluster to relabel through this method. After grouping the subflows by clusters, we proceeded to apply the filtering process.

4.4 Conclusions

In this chapter we reviewed an approach to infer statistical information about the packets and use it to train the MLAs. We also proposed a two-step filter for noise reduction that implements an offline training and an online filter. Through this filter we may answer the question *How can noise be detected?*.

Furthermore, we defined the ML components and established that one component is required to perform a clusterization process and a second component to perform traffic flow classification through the usage of an MLA. The filter component provides a data filter and a sublabeling process. During the offline training, the filter component is used to correlate clusters with types of flow or classify them as noise. Traffic flows that belong to clusters that neither can be labeled as a flow type nor as noise, will be used to train the MLA. On the other hand, the filter component can be used similarly during the online testing, where it will inherit the correlation between cluster and labels from the offline training in order to filter the noise out and, if possible, label some traffic flows.

The ML analysis component can be defined then as the component where the MLA is trained and then implemented. Both offline and online, the ML analysis component workload is reduced by the filter component. Similar to the filter component, the ML analysis component inherits its model from the offline training to the online testing.

From these definitions, we are able to define their implementation as a part of the cloud platform. Therefore, we can now define the implementation of the entire cloud solution. In the following chapter, we determine the implementation of the solution using the ARCADIA method through the Capella software.

Chapter 5

Implementation

In the previous chapter, we defined the elements for the ML components, and in the chapter before that, we established that the requirements of any given scenario is defined in terms of the scenario itself. In this chapter, we review the system analysis to provide the logical and physical implementation of the platform.

The aim of the implementation is to generate a rich environment capable of generating a large labeled dataset. To achieve this, a cloud platform was developed conformed of virtual machines that generated traffic flows by connecting to Internet and deploying a preconfigured scenario, and by virtual routers that sniff and label incoming traffic flows. This data capture is performed by session, i.e., each time a scenario is performed, a sniffing session will open, then it will capture the data of the incoming packets, and once the scenario is over, the sniffer will store the information in a unique local file. Therefore, the number of local files is equal to the number of correctly executed scenarios.

The implementation also requires a defined functionality that enables the ML analysis and the filtered required for the offline and online training. Such functionalities, as well as the ones used for traffic generation are studied in Section 5.1. Section 5.2 analyzes the physical architecture for the implementation.

5.1 Logical Architecture

From the system analysis, we know the missions that the platform must accomplish. We have defined the key components for the solution and established their requirements. However, to produce the expected logical results there must be an internal function that guarantees proper results. In other words, it is necessary to ensure that the data generated by the platform has been produced properly. From this perspective, properly generated data refers to labeled data which performance was successful. The system should be able to discard the data produced with errors, such as lack of connectivity, or errors related to the specific scenario, for instance, a streaming performance where the browser could not open the video link.

The logical system should be also composed of a communication component that creates a connection between the two kinds of virtual machines within the platform. Furthermore, the logical system should define the needed functionality to interpret packets in order to detect the flags sent through UDP packets from client to router.

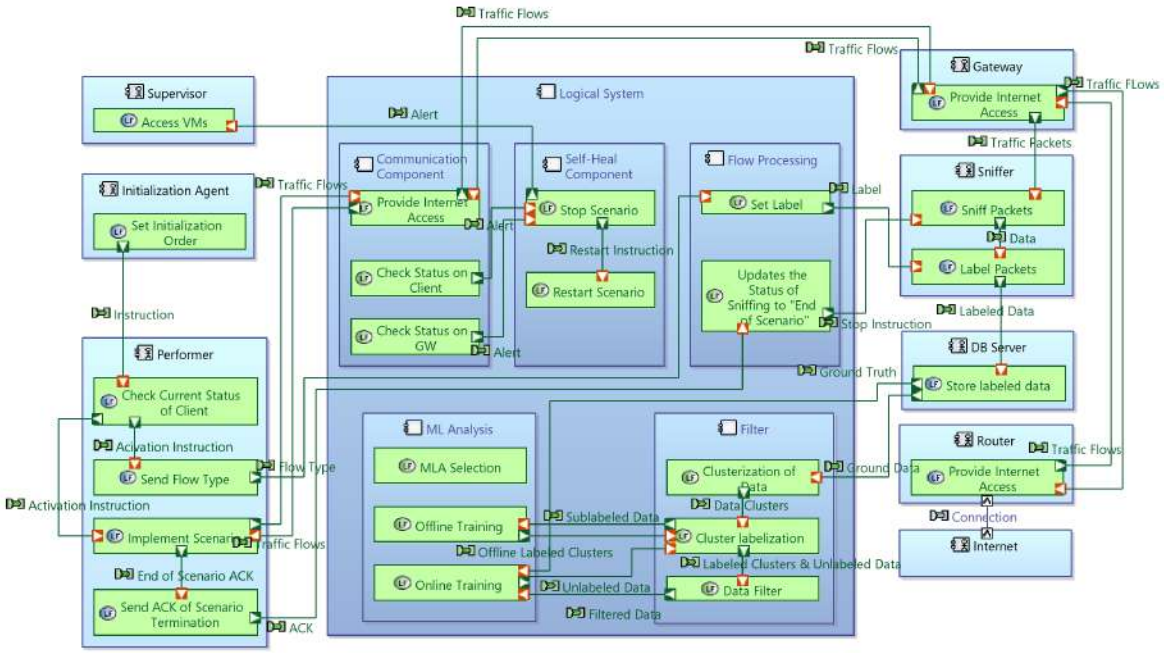
5.1.1 Logical Components

The logical system in the ARCADIA method is used to represent the logical requirements for the architecture in development. The aim is to describe the process and the order for the implementation with the identification of the operational components. The logical system employs the usage of logical components, which communicate among themselves and with external actors. In the presented research, five logical components are used. Figure 5.1 summarizes the logical architecture.

Communication

The first implemented logical component is the communication component. The functionality of this component is to assure the connection between the performer and the router. The communication between both actors is studied by this component, analyzing the status on both the router

Figure 5.1: Logical Architecture



and the performer. From this component, errors in performance may be discovered, however, it does not act upon failure, rather it works as a mean for detection.

Self-Healing

The self-healing component is based on one of the autonomic computing characteristics. The self-healing property is able to discover, diagnose and react to disruptions without disrupting the IT environment [75].

In the proposed solution, the most common error during implementation was the over-performance, i.e., scenarios continued their performance despite reaching their ending point, this problem affected the sniffing process. By not correctly ending the scenario, the sniffing lapse observed overlapped sessions, therefore, created files that did not accurately represent one session per run scenario. To solve this issue, once the communication component detects an error on either side of the communication, the self-healing component receives an alert to which it stops the performance and sends an alert to the human supervisor so that a former study can

be executed.

Flow Processing

The flow processing component receives a UDP packet and translates it to a label, i.e., it interprets the packet content as an initialize sniffing instruction and as a label to recognize the type of incoming traffic flows. Through this label, the sniffer is able to categorize traffic flows by session. The second function of this component is to close the sniffing session by translating a second UDP packet from the performer as an acknowledgment of the conclusion of a session.

ML Analysis

The ML analysis component uses three functions. The first function is the MLA selection process, which is required for testing different MLAs. Given that in this implementation the goal is to test several MLAs, the aim for the MLA selection component is to execute different MLAs and store their performance metrics for later analysis. The ML analysis component uses as well the offline training and the online testing functions. The offline training function has a two-step process, the first process is to partition the data and use the cluster labelization function from the filter component to sublabel the data, the second step is to use the sublabeled data to train the selected MLA. Similarly, the online training function utilizes the filter for its implementation. Firstly, it also utilizes the cluster labelization to assign labels to well-defined clusters, reducing the amount of data the MLA has to classify. Then, it uses the filter function to eliminate the noise from the dataset. With a clean and reduced dataset, the online testing performs the selected MLA and provide its respective metrics.

Filter

To correctly implement the filter, its functions must be defined. The filter must complete three tasks: data clusterization, cluster labelization, and data filtering. The component of data clus-

terization groups in n clusters the ground truth it is given. The resulting clusters represent data with similar statistical behavior.

The cluster labelization component requires the input from the data clusterization. This component has two different roles depending on whether it is used by the offline training or the online testing functions. If used by the offline function, the cluster labelization function receives labeled data and returns sublabeled data, i.e., after clustering the dataset, the filter component uses the labels of the ground data to assess the formed clusters and detect whether they represent a specific application or application class, they are classless, or they are noise. The difference between a classless cluster and a noise cluster is that a classless cluster is representative of the data, however, it does not have a direct relation with any specific application class. On the other hand, the noise clusters are classless clusters which do not represent data. If used by the online testing, receives the new unlabeled clusters and the clusters from the offline training. With basis on the offline clusters, the cluster labelization assigns the new data to the clusters and, if the cluster they were assigned to has a well-defined label, then that data is labeled as such. In addition, during online training, once the cluster labelization ends its process, the data filter is used. This function inherits the labels provided by the previous function and uses them to detect noise, and if it does, it eliminates it from the dataset.

5.2 Physical Architecture

In the ARCADIA method, the aim of the physical architecture is to define the requirements for the system's development. The objective of this architecture is to allocate the functionalities of the system to physical nodes, such as sensors. To represent the requirements at this level, the physical architecture utilizes three types of components: node components, behavior components, and functional components. Node components represent the physical aspect of the architecture, to do so, it uses physical nodes, physical actors, and physical links and ports. The behavior components describe the performances a physical component has and uses component exchanges to describe relations between behavior components and nodes. The functional com-

ponents represent the physical functions that a physical node or actor has and links them with functional exchanges.

In the presented research a six-node physical architecture with two physical actors was implemented. Five of the proposed nodes are virtual components in the cloud platform, one of which has a subcomponent. The physical actors for this architecture are the *human supervisor* and the *Internet*. The *human supervisor* has the behavior of manage architecture, which uses functions for analyzing the performer, the router, the router, the database server, and the initialization agent.

The client component's operations are categorized into five behaviors. The *send initialization* behavior contains the operations of the initialization agent, which requires to fulfill two needs: set the time for each performance as well as the total time of repetitions between performances, and to activate the performer. Each time the initialization agent activates the performer, the *self-analysis* behavior determines whether the last performance was closed properly if not, it sends an error acknowledgment to the detect failure behavior for it to stop the scenario and alert the human supervisor. If the last performance was closed properly, then the *self-analysis* behavior sends the activation order to the *user emulation behavior*. This behavior starts the performance by sending the flow type to the router through a UDP packet, then it deploys the scenario and locally stores the PID for later revision by the *self-analysis* behavior. Once the performance reached its time limit, the *user emulation* behavior activates the finish scenario behavior to properly close the session. A session is properly closed when the scenario is no longer performed and a UDP packet is sent to the router acknowledging that the performance has finished.

The router node is composed of the *connect to router* behavior and the sniffer subnode. The *connect to router* behavior provides the basic functionality of a gateway, however, every packet it receives is sniffed by the sniffer node. The *packet sniffer* behavior of the sniffer node sniffs every packet incoming and outcoming packet from the router. If the special UDP packet containing the label of a new session arrives, the *packet sniffer* behavior sends it to the *open session* behavior to set the path for the packets storage. As long as the session is active, the *packet sniffer* behavior

stores the packets on the set path. When the acknowledgment for the session's ending arrives in a UDP packet, then the *packet sniffer* behavior sends the ending order to the *close session* behavior. The *close session* behavior cleans the memory on the router, and sends the labeled packets to the database server node.

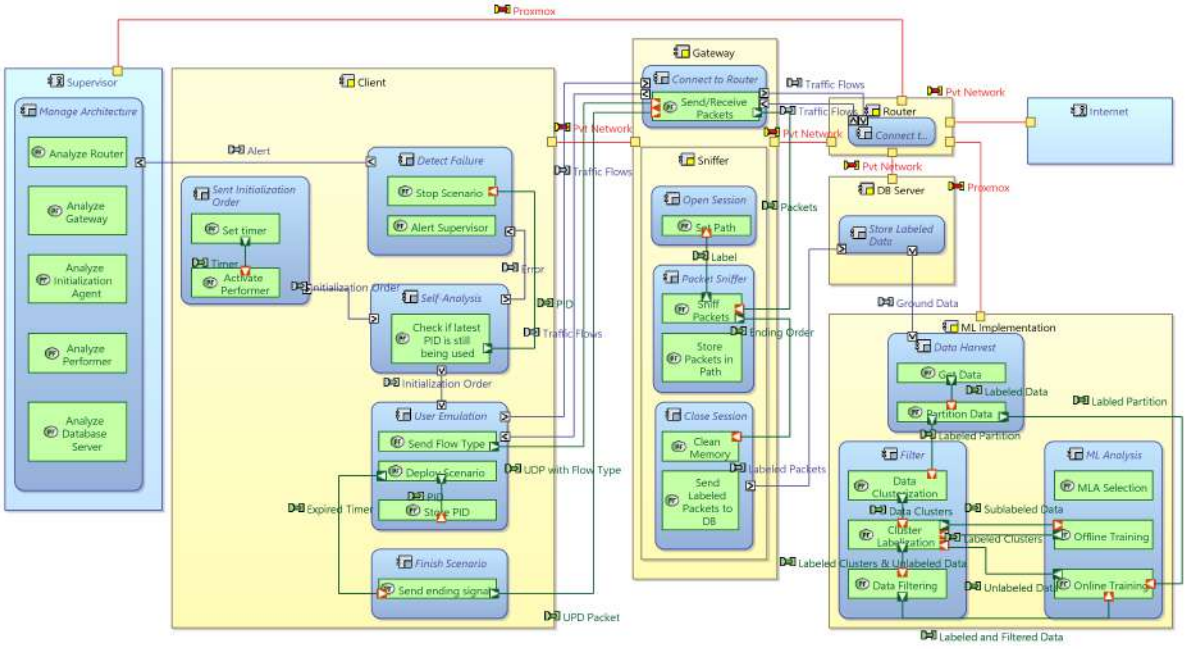
The remaining two nodes of the cloud architecture are the router node, which represents the router of the cloud platform and is the accessing point for the platform itself with only one behavior, *connect to Internet*. The other node is the database server node which only has the *store labeled data* behavior.

A physical node which is not integrated into the cloud platform is the node for the Machine Learning implementation. The ML implementation node has three behaviors. Firstly, it requires the *data harvest* behavior to collect the data from the database server node. Upon collection, the *data harvest* behavior partitions the data in two sets, one for the offline training and another for the online testing. The filter component is first used when it receives the data for the offline training, it uses it to form data clusters, and then sublabels them. The *ML analysis behavior* selects an MLA and initiates the offline training with the sublabeled data from the filter node. The *online testing* function then sends the unlabeled data to the filter node to filter data by removing noise and labeling data that belong to clusters with well-defined flow types. The online testing then makes predictions with the remaining data and checks the accuracy of both the data it classified with the MLA and the data it was labeled by the filter component. Figure 5.2 the physical architecture can be observed.

5.3 Conclusions

In this chapter we presented the proposed architecture based on the system requirements analysis defined in chapter 3. We studied the logical functions and modeled the physical requirements. We answered the questions *Which are the physical requirements for the implementation?*, and *How does the physical components interact with each other?*.

Figure 5.2: Physical Architecture



Chapter 6

Results

In this chapter we present the settings and outcome of the experiments done from the creation of the dataset using a cloud architecture to the filtering and training processes of ML.

6.1 Platform

In our proposed solution we analyze the problem of automatic traffic flow labeling by assuming two perspectives in the cloud environment; a PaaS model, and an IaaS model. During the first phase of design, we worked with an IaaS virtual environment named *Proxmox VE*. *Proxmox VE* is an open-source platform for all-inclusive enterprise virtualization that integrates KVM (kernel-based virtual machines) hypervisor and LXC (Linux containers) [76]. In this phase of the design, it is important to establish the computing requirements and the system capabilities. Each virtual machine in the platform has 4GB of memory, a 1 socket and 1 core processor, and a hard disk of 32 GB. In addition, they are running in Ubuntu 18.04.1 LTS.

The second phase of design was to structure the architecture so that it satisfied the system requirements and became an instance of the physical architecture. By building a system through an IaaS environment, this proposal provides a PaaS model to allow the design and implementation of diverse scenarios of human-like behavior. Therefore, the client of the platform should be

able to develop more scenarios to enrich the quality of the data training set.

6.2 Scenarios

As defined in section 3.2, a scenario is a set of steps a client machine does to replicate human-like behavior. In general terms, all scenarios work under the same premise: first send the label to the router through a UDP packet, perform the scenario for a certain amount of time, and once the timer reached its limit, stop the scenario and inform it to the router through another UDP packet. The scenarios implemented in our solutions generate the traces for the following activities: Web browsing, audio, and video streaming, P2P file download is done through eMule, file transfer, VoIP and video calls, branded as "Interactive Communication", and chat.

The web browsing scenario performs an online search through the google chrome browser and google search engine. The streaming scenario is composed of two variants: audio and video. For audio streaming, the scenario access either Spotify or the site EcouterRadioEnLigne.com, whereas the video streaming is performed by accessing a Youtube or Twitch video. The traces generated for file transferring were obtained through the flows generated in an SCP request. Interactive Communication flows were sniffed by the usage of their respective activities on Facebook. For the chat scenario, the Skype API *Skpy* [1] was used.

6.3 Experiments

We created a database containing the statistical information of flow records generated by Web browsing, audio and video streaming, VoIP and video calls (labeled as Interactive Communication), and eMule P2P download. We then set four experiments: accuracy and confusion matrix for several MLAs with both the raw dataset to determine the best MLAs for traffic classification using the data we generated, the second experiment is the study of accuracies and confusion

[1] https://skpy.t.allofti.me/index.html

matrix using the filtered and sublabeled dataset, the third experiment was to test the accuracy, filtering and sublabeling of sessions different from the ones during the MLA training, and the fourth experiment was meant to test the realtime classification. During the experiments, we used the scikit-learn library [77] with the imbalance-learn package [78].

6.3.1 Dataset

With the parameters proposed in Section 4.2, we created a database containing the flow records obtained through sniffing the communication of several virtual machines. Our dataset was comprised of a total of 4.7 GB. Table 6.1 shows the number of sessions per application as well as their file size. As seen in said table, the data has a bias towards the streaming and browsing services. If the imbalance was kept, the application with higher flow number would not allow the propper classification of other applications, hence we balanced the data through the algorithm proposed in [79].

Table 6.1: Captured Flows Summary

Fine-grain	**Coarse-grain**	**Number of Sessions**	**Total Size**
Google browing	Browsing	3,207	1.2 GB
eMule	P2P	6	412 MB
Facebook Videocall	Interactive Communication	38	12 MB
Facebook VoIP	Interactive Communication	343	103 MB
SCP	FT	111,759	522 MB
Twitch	Streaming	3,233	856 MB
Youtube	Streaming	99,024	1,001 MB
EcouterRadioEnLigne	Streaming	2,522	415 MB
Spotify	Streaming	1,036	148 MB

6.3.2 Test of accuracy for the balancing methods

To prevent misclassification due to strong bias, the data during training requires to be balanced. Balancing data can be done through a manually (or randomly) selection of data, alternatively,

it can also be obtained by a balancing method. During testing, we considered two methods: SMOTEENN and TOMEK from the imblearn library.

Results

We trained the database with both methods and tested for accuracy using the following MLAs: AdaBoost, Decision Tree, MLP, Naive Bayes, KNN, QDA, Random Forest, and SVM with a RBF kernel and with a linear kernel. During the experimentation, the SVM models were restrained to a maximum of 1000 iterations per analysis.

In terms of accuracy, the best three ML classifiers for the testing data balanced by SMOTEENN are Random Forest, KNN, and Decision Tree, getting 98.15%, 99.35%, and 100% of accuracy respectively. The next best classifier in terms of accuracy is AdaBoost with an accuracy of 59.87%. When balanced with the TOMEK method, the best three classifiers are KNN, Random Forest, and Decision Tree with 99.03%, 99.25%, and 100% respectively. Similarly, with SMOTEENN, the next best classifier is AdaBoost with 59.97%. As shown in Table 6.2, the SMOTEENN produces slightly better results than TOMEK does.

Table 6.2: Accuracy of Balancing Methods

MLA	**SMOTEENN**	**TOMEK**
AdaBoost	50.87%	59.97%
Decision Tree	100%	100%
MLP	19.81%	20.02%
Naive Bayes	35.72%	30.07%
KNN	99.35%	99.03%
QDA	57.76%	57.30%
Random Forest	98.15%	99.25%
SVM Linear	14.80%	20.83%
SVM RBF	49.28%	49.32%
Average	**59.41%**	**60.08%**

6.3.3 Sublabeling and Filtering Analysis

In this experiment, we wanted to measure the impact of the filtering and sublabeling processes on the traffic data. We performed this experiment with the following alterations to the dataset. The first two datasets used for these tests are dataset balanced by SMOTEENN, and the dataset balanced by TOMEK. We then created a third and a fourth dataset using the same methods with a reduced dataset that omitted the features of the median values of the inter-arrival time and the packet size, as well as the number of packets, this was done so to share similarities with the realtime implementation. Finally, we also studied a dataset with full features but containing only three flow types (internet browsing, video streaming, and P2P downloading) to represent graphically the behavior of the filter and the sublabeling processes on observable clusters.

Results

To represent how data is filtered and after is sublabeled, we used a dataset containing only three flow types. The clustering algorithm was able to group the data into four clusters. The first cluster failed to obtain a label given its complex composition, the second and third clusters showed a predominance of the Youtube flows, which allowed to qualify any flows within these clusters as Youtube flows, and lastly, the fourth cluster did not contain any significant data, so it was defined as noise and any flow that was set in that cluster would have been filtered out during the filtering process. Figure 6.1 shows the data clusters before the sublabeling process and Figure 6.2 shows the data clusters after the sublabeling process.

Table 6.3 shows the results of the filtering and sublabeling processes. The number of filtered and sublabeled flows is small relative to the number of total flows during testing because that the clusterization process generates groups that contain few flows. This can be seen in Figure 6.3 where most data is contained in the first two clusters.

Additionally, we confirm that, in the full feature dataset, the SMOTEENN balancing method produces slightly more recognizable data. However, in the reduced dataset, compared to the total number of flows, the sublabeling process trained with the SMOTEEN balanced data subla-

Figure 6.1: Before Sublabeling

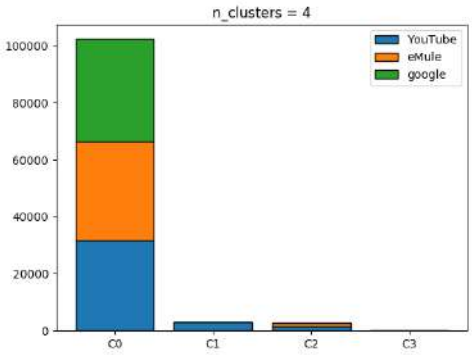


Figure 6.2: After Sublabeling

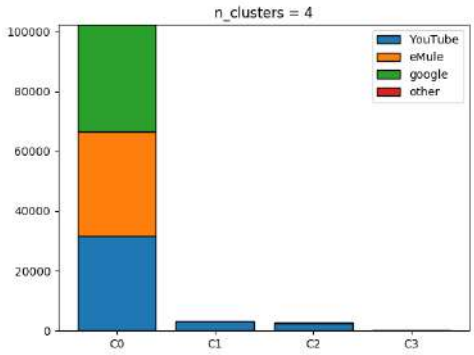


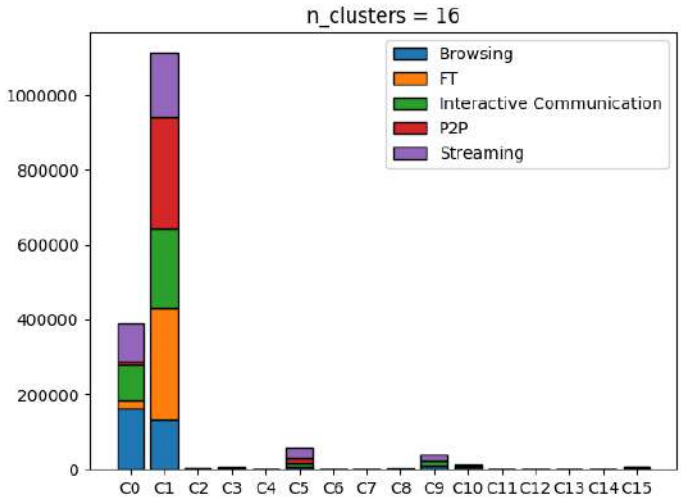
Table 6.3: Filtered and Sublabeled Flows

Dataset	**Filtered**	**Sublabeled Correctly**	**Mistakenly Sublabeled**	**Total Flows**
Full SMOTEENN	120	997	3	1,307,393
Full TOMEK	0	810	132	1,316,379
Reduced SMOTEENN	0	775	133	1,097,046
Reduced TOMEK	1,376	997	220	1,090,652

bels correctly the 0.07% of data while when trained with the TOMEK balanced data sublabels correctly the 0.09%. The error rate for the sublabeling process shows that, with the TOMEK data, there is an error rate of 0.02%, while with the SMOTEEN data the error rate is of 0.01%. In short, these results show that either method produces similarly recognizable data, with SMOTEENN being slightly better than TOMEK. Aside from the balancing methods, these experiments showed that the sublabeling process across the four datasets is able to recognize successfully the web browsing flows, while the major misconception it may have is to label other flows, usually the streaming flows, like browsing.

Aside from the balancing methods, these experiments showed that the sublabeling process across the four datasets is able to recognize successfully the web browsing flows, while the major misconception it may have is to label other flows, usually the streaming flows, as browsing.

Figure 6.3: Clustered data



6.3.4 Test of filter and sublabeling accuracy and confusion matrix

In addition, we also calculated the confusion matrix to complement the information of the misclassification. Through the confusion matrix we can visualize the performance by flow type of the sublabeling process. The purpose of this test is to consider the accuracy of each MLA in terms of both the general accuracy and the specific misclassifications. The need to identify these behaviors is to determine which MLA generates the least costly errors. For instance, classifying a file transfer flow as web browsing is not as inadequate as classifying a streaming flow as a VoIP flow, while both are data misrepresentation, regarding QoS, the latter might represent a heavier impact than the former.

Results

Table 6.4 shows the general accuracy of the same tasted MLAs as in the past experiments with both the data balanced by SMOTEENN and the data balanced by TOMEK. With the confusion matrices, the results show that the filter and sublabeling affect the overall classification of the flows. In the SMOTEENN balanced data, flows are not filtered and the sublabeling process can either give the label of browsing or the label of streaming, despite this, after being subjected to

this process, some of the confusion matrices alter their information on file transfer and VoIP and Videocalls.

Table 6.4: Accuracy of Balancing Methods

MLA	SMOTEENN	TOMEK
AdaBoost	95.60%	59.99%
Decision Tree	99.97%	99.98%
MLP	20.15%	20.09%
Naive Bayes	35.63%	30.06%
KNN	99.34%	99.03%
QDA	54.84%	57.21%
Random Forest	99.21%	96.81%
SVM Linear	14.87%	13.45%
SVM RBF	49.35%	49.38%
Average	**63.22%**	**59.01%**

This edit is dependant on the MLA that is being used. For instance, when using QDA, the filter enhances the classification by reducing the confusion by 1463 flows of Videocalls and VoIP that were previously labeled as Streaming, likewise, 929 Streaming flows were misclassified as Videocalls and VoIP before the implementation of the filter. While the results of QDA show an improvement over the Interactive Communication classification, the results of Naive Bayes show the opposite, misclassifying ten Interactive Communication as Streaming.

To study the impact of the filter, and to select the optimal tool for traffic classification, we need to analyze the result of the filter in the application of the three best classifiers, i.e., KNN, Decision Tree, and Random Forest. In KNN, the filter improves the classification by decreasing the number of web browsing flows misclassified as streaming, however, the filter increases the confusion of streaming flows classifying some of them as web browsing. In Decision Tree, the accuracy without the filter is 100%, so everything the filter sublabels will cause a decrement of accuracy with the test data. When testing the filter with Random Forest, the filter improved the accuracy by greatly diminishing the confusion caused by file transfer flows classified as Interactive Communication, Interactive Communication classified as P2P, and streaming classified as P2P.

6.3.5 Online Testing

The methodology of the previous tests was based on partitioning the dataset so that 60% of the data was used for training and 40% of the data was used for testing, however, a test is required that determines whether the MLAs are able to classify internet flows generated by applications which were not used in their training yet belong to the same coarse-grain definition.

To make this test, we generated new traces from an external machine and capture its flows with a packet sniffing software, then we fed that data to the MLAs to compare their results, every MLA was firstly trained with the SMOTEENN balanced data, and after with the TOMEK balanced data. The traces we generated were Netflix streaming, and web browsing using Firefox instead of Google Chrome. The MLAs chosen for this test were KNN, Decision Tree, and Random Forest, since they have the highest accuracy among the tested MLAs.

Results

Netflix streaming generated 77 flows, which all were classified correctly by Decision Tree and Random Forest with both SMOTEENN and TOMEK balanced data. On the other hand, when trained with the SMOTEENN balanced data, KNN classifies two flows as file transfer, and one as P2P, the rest is classified correctly, when trained with the TOMEK balanced data, KNN classifies one flow as web browsing and the rest are classified correctly.

Web browsing generates a large misconception among the MLAs, which classify most of their flows as streaming flows. This session of web browsing was comprised of 397 flows. When trained with SMOTEENN balanced data, KNN classified three as file transfer, two as P2P, and the rest as streaming, Decision Tree classified every flow as streaming, and Random Forest classified 174 as Interactive Communication, and 223 as streaming.

6.4 Conclusions

In this section we presented three different experiments to test the viability of Machine Learning for traffic classification. We tested our method for flow sublabeling and filter to clean data. We implemented the MLAs provided by the python library scikit-learn [77] and used them to train a database which was balanced by the imbalance-learn package [78]. Therfore, we answered the question *Which algorithms are considered?*. In addition, by implementing a PaaS model, based on a IaaS environment, to generate ground data, we answered the question: *Why is Cloud Computing a viable solution for the lack of ground truth issue?*

Chapter 7

Conclusions

This research approached the issue in Internet traffic classification of a lack of a standardized dataset for MLA's training. To solve this, we proposed the question *can a MLA perform Internet traffic classification with high accuracy if its ground truth was obtained through emulated traffic flows?* Results showed that this proposal is viable for that end and can be further improved by a sublabeling and filtering process.

In the feature clustering process, it was observed that the overall majority of the flows were grouped in the same cluster, this could reveal that the selected features we used to train the clustering model were not sufficient for K-means to detect classes. The other clusters contained information about anomalous behavior which is easily identifiable by our proposed approach. For instance, Figure 6.1 shows the initial clustering for the training flows. Cluster $C0$ contains the majority of the flows, but the other three are nearly empty. Figure 6.2 shows how the anomalous clusters were used to relabel the data in the third cluster identifying it as Youtube.

Future challenges include the incorporation of other scenarios, such as Voice over IP, email, and chat. It is of great interest to implement a VPN and analyze the classification of encrypted and tunneled data, and later for multiplexed data. Another challenge is to improve the sublabeling and filtering proposal.

Chapter 8

Appendix

8.1 Abreviations

- **API**. Application Programming Interface.
- **ARCADIA**. Architecture Analysis and Design Integrated Approach.
- **CART**. Classification and Regression Trees.
- **DS / DiffServ**. Differentiated Services
- **EM**. Expected Maximization.
- **ENN**. Edited Nearest Neighbours.
- **GW**. Gateway.
- **HTTP**. Hypertext Transfer Protocol.
- **IaaS**. Infrastructure as a Service.
- **IEEE**. Institute of Electrical and Electronics Engineers.
- **Intserv**. Integrated Services.

- **IP**. Internet Protocol.
- **IPv4**. Internet Protocol version 4.
- **IPv6**. Internet Protocol version 6.
- **ISP**. Internet Service Provider.
- **IT**. Information Technology.
- **ITU**. International Telecommunication Union.
- **KNN**. K Nearest Neighbours.
- **KVM**. Kernel-Based Virtual Machines.
- **LAN**. Local Area Network.
- **LDA**. Linear Discriminant Analysis.
- **LXC**. Linux Containers.
- **MAC**. Media Access Control.
- **ML**. Machine Learning.
- **MLA**. Machine Learning Algorithm.
- **MLP**. Multilayer Perceptrons.
- **MPLS**. Multiprotocol Label Switching.
- **OS**. Open System.
- **OSI**. Open System Interconnection.
- **PaaS**. Platform as a Service.
- **PID**. Process Identifier.
- **QDA**. Quadratic Discriminant Analysis.

- **QoS**. Quality of Service.
- **RMA**. Reliability, Maintainability, and Availability.
- **RBF**. Radial Basis Function.
- **RFC**. Request for Comments.
- **SaaS**. Software as a Service.
- **SMOTE**. Synthetic Minority Over-sampling Technique.
- **SMOTEENN**. Synthetic Minority Over-sampling Technique with Edited Nearest Neighbours.
- **SMTP**. Simple Mail Transfer Protocol.
- **SSH**. Secure Shell.
- **SVM**. Support Vector Machines.
- **TCP**. Transmission Control Protocol.
- **UDP**. User Datagram Protocol.
- **VE**. Virtual Environment.
- **VM**. Virtual Machine.

8.2 Appendix of Scenarios

Algorithm 7 Browsing

Require: Selenium API

1: Generate text input
2: Open Browser with Selenium
3: Search the text generated in 1 in a search engine
4: Open first five results in different tabs
5: Wait for 10 seconds
6: Close Browser

Algorithm 8 Streaming

Require: Selenium API, BeautifulSoup

1: Set a timer for a random duration
2: Generate link to video using BeautifulSoup
3: Open link generated in 2 in browser using selenium
4: **while** timer is not reached **do**
5: Stream video
6: **end while**
7: Close Browser

Algorithm 9 File Transfer

Require: Public and Private Key Pair

1: Send a file to another computer through SCP
2: Request a file from another computer through SCP

Algorithm 10 P2P

Require: eDonkey, link for a file, agent's timer

```
Request a download using file
Wait for 5 minutes
Check status on download
if download has finished AND agent's timer has not run up then
    Remove downloaded file
    Repeat 1
else if agent's timer has not run up then
    Repeat 2
end if
```

Algorithm 11 VoIP

```
procedure CALLER(Selenium with permission to use the microphone)
    Set a timer for a random duration
    Open Browser with Selenium
    Login in to Facebook
    Open Chat
    Call contact
    while Phone rings do
        Wait
    end while
    if Picker picked up the call then
        Wait for a next window to pop-up
    end if
    Close Browser
end procedure
procedure PICKER(Selenium with permission to use the microphone)
    Set a timer for a random duration
    Open Browser with Selenium
    Login in to Facebook
    while Phone doesn't ring do
        Wait
    end while
    if Phone rings then
        Start running 16
        Pick up the call
        while Timer has not run up do Wait
        end while
    end if
    Close Browser
end procedure
```

Algorithm 12 Chat

Require: Sellenium, timer

1: Generate text input
2: Open Browser with Selenium
3: Login in to Facebook
4: Open Chat
5: **while** timer has not reached its limit **do**
6: Write the random text from 1
7: Wait for a number of seconds.
8: **end while**

Algorithm 13 Email

Require: Python with SMTPlib

1: Structure email
2: Send it

References

[1] Kristell Aguilar and Riad Sadok. Emulating application and user behavior on a cloud platform for later traffic analysis. Research project, Université de Pau et des Pays de l'Adour, March 2018.

[2] Michael Hogan, Fang Liu, Annie Sokol, and Jin Tong. Nist cloud computing standards roadmap. *NIST special publication*, 35:6–11, 2011.

[3] Jean-Luc Voirin. *Model-based System and Architecture Engineering with the Arcadia Method (Implementation of Model Based System Engineering)*. ISTE Press - Elsevier, 2017. ISBN 9780081017944.

[4] James D. McCabe. *Network Analysis, Architecture, and Design (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann, 2007. ISBN 0123704804.

[5] Telecommunication Standardization Sector of ITU. Ict facts and figures, 2018. URL `https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2017.pdf`.

[6] A. Clark and B. Claise. Guidelines for considering new performance metric development. RFC 6930, RFC Editor, October 2011. URL `https://tools.ietf.org/html/rfc6390`.

[7] Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys & Tutorials*, 2018. doi: 10.1109/COMST.2018.2883147. URL `http://dx.doi.org/10.1109/COMST.2018.2883147`.

[8] Alberto Dainotti, Antonio Pescape, and Kimberly Claffy. Issues and future directions in traffic classification. *IEEE Network*, 26(1):35–40, January 2012. ISSN 0890-8044. doi: 10.1109/MNET.2012.6135854.

[9] Hassan Alizadeh and André Zúquete. Traffic classification for managing applications' networking profiles. *Security and Communication Networks*, 9(14):2557–2575, 2016. doi: 10.1002/sec.1516. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1516`.

[10] T. Bujlow, T. Riaz, and J. M. Pedersen. A method for classification of network traffic based on c5.0 machine learning algorithm. In *2012 International Conference on Computing, Networking and Communications (ICNC)*, pages 237–241, Jan 2012.

[11] Tomasz Bujlow, Valentin Carela-espa, and Pere Barlet-ros. Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification Technical Report Department of Computer Architecture (DAC). Technical report, Universitat Politècnica de Catalunya, Department of Computer Architecture (DAC), June 2013.

[12] W. Li and A. W. Moore. A machine learning approach for efficient traffic classification. In *2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 310–317, October 2007. doi: 10.1109/MASCOTS.2007.2.

[13] Douglas C. Sicker, Paul Ohm, and Dirk Grunwald. Legal issues surrounding monitoring during network research. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 141–148, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-908-1. doi: 10.1145/1298306.1298307. URL `http://doi.acm.org/10.1145/1298306.1298307`.

[14] Andrew Ng. Cs229: Machine learning - lecture notes, 2017. URL `http://cs229.stanford.edu/syllabus.html`.

[15] Scott Krig. *Computer Vision Metrics: Survey, Taxonomy, and Analysis*. Apress, 2014.

[16] Shahbaz Rezaei and Xin Liu. Deep learning for encrypted traffic classification: An overview. *arXiv preprint arXiv:1810.07906*, 2018.

[17] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 281–286. ACM, 2006.

[18] Zhu Li, Ruixi Yuan, and Xiaohong Guan. Accurate classification of the internet traffic based on the svm method. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 1373–1378. IEEE, 2007.

[19] Alice Este, Francesco Gringoli, and Luca Salgarelli. Support vector machines for tcp traffic classification. *Computer Networks*, 53(14):2476–2490, 2009.

[20] C. Wang, T. Xu, and X. Qin. Network traffic classification with improved random forest. In *2015 11th International Conference on Computational Intelligence and Security (CIS)*, pages 78–81, December 2015. doi: 10.1109/CIS.2015.27.

[21] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, 64(9-12):1194–1213, 2007.

[22] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Semi-supervised network traffic classification. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 369–370. ACM, 2007.

[23] K. Singh and S. Agrawal. Comparative analysis of five machine learning algorithms for ip traffic classification. In *2011 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC)*, pages 33–38, April 2011. doi: 10.1109/ETNCC.2011.5958481.

[24] H. Singh. Performance analysis of unsupervised machine learning techniques for network traffic classification. In *2015 Fifth International Conference on Advanced Computing Communication Technologies*, pages 401–404, Feb 2015. doi: 10.1109/ACCT.2015.54.

[25] Z. Aouini, A. Kortebi, Y. Ghamri-Doudane, and I. L. Cherif. Early classification of residential networks traffic using c5.0 machine learning algorithm. In *2018 Wireless Days (WD)*, pages 46–53, April 2018. doi: 10.1109/WD.2018.8361693.

[26] E. Exposito F. Pacheco and M. Gineste. A wearable machine learning solution for internet traffic classification in satellite communications. In *The 17th International Conference on Service-Oriented Computing ICSOC*, Toulouse France, October 2019.

[27] [Online]. Mawi working group traffic archive. URL `http://mawi.wide.ad.jp/mawi/`.

[28] Valentín Carela-Español, Pere Barlet-Ros, Albert Cabellos-Aparicio, and Josep Solé-Pareta. Analysis of the impact of sampling on NetFlow traffic classification. *Computer Networks*, 55(5):1083–1099, April 2011. ISSN 13891286. doi: 10.1016/j.comnet.2010.11.002. URL `http://linkinghub.elsevier.com/retrieve/pii/S1389128610003439`.

[29] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. *National institute of standards and technology*, 53(6):50, 2009.

[30] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[31] TM Mitchell. Machine learning, mcgraw-hill higher education. *New York*, 1997.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, pages 98–227. MIT Press, 2016. `http://www.deeplearningbook.org`.

[33] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press. ISBN 978-1-58603-780-2. URL `http://dl.acm.org/citation.cfm?id=1566770.1566773`.

[34] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23, apr 2006. doi: 10.1145/1129582.1129589.

[35] Z. Fan and R. Liu. Investigation of machine learning based network traffic classification. In *2017 International Symposium on Wireless Communication Systems (ISWCS)*, pages 1–6, Aug 2017. doi: 10.1109/ISWCS.2017.8108090.

[36] Chris Piech. K means. Retrieved May 22, 2019 from `https://stanford.edu/~cpiech/cs221/handouts/kmeans.html`, 2013.

[37] Harry Zhang. The optimality of naive bayes. *AA*, 1(2):3, 2004.

[38] 1.9. naive bayes. Retrieved May 22, 2019 from `https://scikit-learn.org/stable/modules/naive_bayes.html`, 2019.

[39] Andrew W Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 50–60. ACM, 2005.

[40] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[41] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: A statistical signature-based approach to ip traffic classification. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*. ACM Press, 2004. doi: 10.1145/1028788.1028805. URL `https://doi.org/10.1145/1028788.1028805`.

[42] 1.2. linear and quadratic discriminant analysis. Retrieved May 22, 2019 from `https://scikit-learn.org/stable/modules/lda_qda.html`, 2019.

[43] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997. ISSN 0022-0000. doi: 10.1006/jcss.1997.1504. URL `http://dx.doi.org/10.1006/jcss.1997.1504`.

[44] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL `https://doi.org/10.1023/A:1010933404324`.

[45] Matthew Bernstein. Notes. Retrieved May 22, 2019 from http://pages.cs.wisc.edu/~matthewb/pages/notes/notes.html.

[46] R. C. Jaiswal and S. D. Lokhande. Machine learning based internet traffic recognition with statistical approach. In *2013 Annual IEEE India Conference (INDICON)*, pages 1–6, Dec 2013. doi: 10.1109/INDCON.2013.6726074.

[47] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An architecture for differentiated services. RFC 2475, RFC Editor, December 1998. URL http://www.rfc-editor.org/rfc/rfc2475.txt.

[48] J. Korhonen, H. Tschofenig, M. Arumaithurai, and A. Lior. Traffic classification and quality of service (qos) attributes for diameter. RFC 5777, RFC Editor, February 2010. URL https://www.rfc-editor.org/rfc/rfc5777.txt.

[49] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. RFC 2474, RFC Editor, December 1998. URL http://www.rfc-editor.org/rfc/rfc2474.txt.

[50] Seyong Park, Kyungtae Kim, Doug C Kim, Sunghyun Choi, and Sangjin Hong. Collaborative qos architecture between diffserv and 802.11 e wireless lan. In *Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual*, volume 2, pages 945–949. IEEE, 2003.

[51] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. RFC 3031, RFC Editor, January 2001. URL http://www.rfc-editor.org/rfc/rfc3031.txt.

[52] Thuy T. T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4): 56–76, 2008. ISSN 1553-877X. doi: 10.1109/SURV.2008.080406.

[53] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 229–240. ACM, 2005.

[54] Telecommunication Standardization Sector of ITU. Series e: Overall network operation, telephone service, service operation and human factors quality of telecommunication services: concepts, models, objectives and dependability planning – terms and definitions related to the quality of telecommunication services. definitions of terms related to quality of service. Recommendation E.800, International Telecommunication Union, Geneva,Switzerland, November 2008.

[55] Holger Dreger, Anja Feldmann, Michael Mai, Vern Paxson, and Robin Sommer. Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection. In *15th USENIX Security Symposium*, pages 257–272. Usenix, 2006.

[56] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5, January 2007. doi: 10.1145/1198255.1198257. URL `https://doi.org/10.1145/1198255.1198257`.

[57] Telecommunication Standardization Sector of ITU. Series y: Global information infrastructure, internet protocol aspects and next-generation networks next generation networks – security requirements for deep packet inspection in next generation networks. Recommendation Y.2770, International Telecommunication Union, Geneva,Switzerland, 2013.

[58] Niccolò Cascarano, Luigi Ciminiera, and Fulvio Risso. Improving cost and accuracy of dpi traffic classifiers. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 641–646. ACM, 2010.

[59] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*. ACM Press, 2004. doi: 10.1145/1028788.1028804.

[60] Jun Li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. Real-time p2p traffic identification. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–5. IEEE, 2008.

[61] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, 25(5):355–374, sep 2015. ISSN 10557148. doi: 10.1002/nem.1901. URL http://onlinelibrary.wiley.com/doi/10.1002/nem.604/abstracthttp://doi.wiley.com/10.1002/nem.1901.

[62] Silvio Valenti, Dario Rossi, Alberto Dainotti, Antonio Pescapè, Alessandro Finamore, and Marco Mellia. Reviewing traffic classification. In *Data Traffic Monitoring and Analysis*, pages 123–147. Springer, 2013.

[63] Andrew W. Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *Lecture Notes in Computer Science*, pages 41–54. Springer Berlin Heidelberg, 2005. doi: 10.1007/978-3-540-31966-5_4. URL https://doi.org/10.1007/978-3-540-31966-5_4.

[64] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT Conference on - CONEXT '08*, Madrid, Spain, December 2008. ACM Press. doi: 10.1145/1544012.1544023.

[65] Roni Bar-Yanai, Michael Langberg, David Peleg, and Liam Roditty. Realtime classification for encrypted traffic. In *International Symposium on Experimental Algorithms*, pages 373–385. Springer, 2010.

[66] Alina Vlăduţu, Dragoş Comăneci, and Ciprian Dobre. Internet traffic classification based on flows' statistical properties with machine learning. *International Journal of Network Management*, 27(3):e1929, 2017.

[67] Jun Zhang, Yang Xiang, Wanlei Zhou, and Yu Wang. Unsupervised traffic classification using flow statistical properties and ip packet payload. *J. Comput. Syst. Sci.*, 79(5):573–585, August 2013. ISSN 0022-0000. doi: 10.1016/j.jcss.2012.11.004. URL http://dx.doi.org/10.1016/j.jcss.2012.11.004.

[68] Taimur Bakhshi and Bogdan Ghita. On Internet Traffic Classification: A Two-Phased Machine Learning Approach. *Journal of Computer Networks and Communications*, 2016 (May):1–21, 2016. ISSN 2090-7141. doi: 10.1155/2016/2048302. URL http://www.hindawi.com/journals/jcnc/2016/2048302/.

[69] Pascal Roques. Mbse with the arcadia method and the capella tool. In *8th European Congress on EMbedded Real Time Software and Systems (ERTS 2016)*, 2016.

[70] Capella. Capella, 2019. URL https://wiki.polarsys.org/Capella.

[71] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. Nist cloud computing reference architecture. *NIST special publication*, 500(2011):1–28, 2011.

[72] S. Amante, B. Carpenter, S. Jiang, and J. Rajahalme. Ipv6 flow label specification. RFC 6437, RFC Editor, November 2011. URL http://www.rfc-editor.org/rfc/rfc6437.txt.

[73] Andrew Moore, Denis Zuev, and Michael Crogan. Discriminators for use in flow-based classification. Technical report, University of London, Department of Computer Science, 2013.

[74] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–37, 1996.

[75] An architectural blueprint for autonomic computing. Technical report, IBM, June 2005.

[76] Proxmox. Proxmox Open-Source Virtualization Platform, 2018. URL https://www.proxmox.com/en/proxmox-ve.

[77] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[78] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL `http://jmlr.org/papers/v18/16-365`.

[79] Gustavo EAPA Batista, Ana LC Bazzan, and Maria Carolina Monard. Balancing training data for automated annotation of keywords: a case study. In *WOB*, pages 10–18, 2003.