Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2017/2018

Pedro Miguel André Alagoa João

Identificação de Aplicações de Vídeo em Canais Protegidos com Aprendizagem Automática

Identification of Video Applications over Protected Channels with Machine Learning

Universidade de Aveiro
2017/2018

Departamento de Eletrónica,
Telecomunicações e Informática

Pedro Miguel André Alagoa João

Identificação de Aplicações de Vídeo em Canais Protegidos com Aprendizagem Automática

Identification of Video Applications over Protected Channels with Machine Learning

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Paulo Jorge Salvador Serra Ferreira, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutora Ana Maria Perfeito Tomé
professora associada da Universidade do Porto

vogais / examiners committee

Prof. Doutor Mário João Gonçalves Antunes
professor adjunto do Instituto Politécnico de Leiria

Prof. Doutor Paulo Jorge Salvador Serra Ferreira
professor auxiliar da Universidade de Aveiro

Palavras Chave aprendizagem automática, percepção em redes, classificação de tráfego.

Resumo

Com a adoção de tráfego cifrado a tornar-se a norma e a crescente utilização de técnicas de obfuscação de tráfego, as empresas têm cada vez mais dificuldades em aplicar políticas de uso nas suas redes, bem como garantir o seu bom funcionamento. Os utilizadores têm mais conhecimentos tecnológicos, sendo facilmente capazes de contornar ferramentas de filtros de conteúdo online com a utilização de túneis protegidos como VPNs. Consequentemente, técnicas como DPI, que já estão ultrapassadas devido à sua impraticabilidade, tornam-se cada vez mais ineficazes. Além disso, todos os regulamentos que têm vindo a ser estabelecidos por governos e organizações internacionais sobre a privacidade dos cidadãos tornam a tarefa de monitorização de uma rede cada vez mais difícil. Este documento apresenta uma plataforma escalável e facilmente instalável para identificação de aplicações numa rede empresarial, focando-se em aplicações de vídeo. Esta abordagem deve ser eficaz independentemente do contexto e organização da rede, com o objectivo de ser uma ferramenta útil no processo de supervisão de redes. O modelo proposto oferece um compromisso entre a capacidade de supervisionar uma rede e assegurar a privacidade dos trabalhadores. A avaliação de resultados indica que é possível identificar serviços web em ligações estabelecidas sobre canais protegidos com uma precisão geral de 95%, usando informações de baixo-nível dos pacotes que não comprometem informação sensível dos trabalhadores.

Keywords machine-learning, network awareness, traffic classification.

Abstract

As encrypted traffic is becoming a standard and traffic obfuscation techniques become more accessible and common, companies are struggling to enforce their network usage policies and ensure optimal operational network performance. Users are more technologically knowledgeable, being able to circumvent web content filtering tools with the usage of protected tunnels such as VPNs. Consequently, techniques such as DPI, which already were considered outdated due to their impracticality, become even more ineffective. Furthermore, the continuous regulations being established by governments and international unions regarding citizen privacy rights makes network monitoring increasingly challenging. This work presents a scalable and easily deployable network-based framework for application identification in a corporate environment, focusing on video applications. This framework should be effective regardless of the environment and network setup, with the objective of being a useful tool in the network monitoring process. The proposed framework offers a compromise between allowing network supervision and assuring workers' privacy. The results evaluation indicates that we can identify web services that are running over a protected channel with an accuracy of 95%, using low-level packet information that does not jeopardize sensitive worker data.

Contents

List of Figures

List of Tables

Glossary

ACL	Access Control List
AI	Artificial Intelligence
AMQP	Advanced Message Queueing Protocol
ANN	Artificial Neural Networks
API	Application Programming Interface
BLINC	BLINd Classification
CAIDA	Center for Applied Internet Data Analysis
CPU	Central Processing Unit
CV	Cross-validation
CWT	Continuous Wavelet Transform
DBSCAN	Density-based spatial clustering of applications with noise
DNS	Domain Name Server
DPI	Deep Packet Inspection
EU	European Union
FN	False Negative
FP	False Positive
FTP	File Transfer Protocol
GDPR	General Data Protection Regulation
GIL	Global Interpreter Lock
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure or HTTP over TLS
IANA	Internet Assigned Number Authority
IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IPSec	Internet Protocol Security
ISP	Internet Service Provider
kNN	k-Nearest Neighbor
MAC	Message Authentication Code
ML	Machine Learning
NN	Neural Networks
NSA	National Security Agency
OvO	One vs. One
OvR	One vs. Rest
P2P	Peer to Peer
PCA	Principal Components Analysis
POP3	Post Office Protocol
QoS	Quality of Service
REST	Representational State Transfer
RFC	Request for Comments
ROC	Receiver Operating Characteristic
RSA	Rivest-Shamir-Adleman
RTP	Real-time Transport Protocol
SCP	Secure copy
SFTP	Secure File Transfer Protocol
SIP	Session Initiation Protocol
SNAP	Stanford Network Analysis Platform
SPAN	Switch Port Analyzer
SPID	Statistical Protocol IDentification
SSH	Secure Shell
SSL	Secure Sockets Layer
SSLv3	Secure Sockets Layer version 3
SVM	Support Vector Machines
TCP	Transmission Control Protocol
TLS	Transport Security Layer
TAP	Test Access Point
TN	True Negative
TP	True Positive
UDP	User Datagram Protocol
VFDT	Very Fast Decision Tree
VM	virtual machine
VPN	Virtual Private Network

CHAPTER 1

Introduction

Traffic monitoring has always been a vital task for network management. It is used to understand the behavior of traffic flows, to expose security breaches and policy enforcement. Identification of network services using traffic classification is a conventional method to enforce corporate policies, such as blocking certain websites or checking personal e-mail. It can also be utilized to detect attacks at the application level. Although methods like Deep Packet Inspection (DPI) are widely used, they rely on expensive computational power and are not viable for a real-time system. Furthermore, governments and international unions are establishing new regulations regarding citizen privacy rights[1], as described in Section 2.4. Thus, with the need for protection of data and user privacy becoming exponentially important, making approaches that depend on content inspection raises ethical issues. Additionally, massive data breaches are becoming a trend, with events such as the National Security Agency (NSA) disclosures of mass surveillance programs and the Equifax data leaks throughout the year of 2017 becoming more and more common. For this reason, data encryption is becoming increasingly adopted, undermining the efficiency of traditional network monitoring methods.

One of the most essential methods of encryption, Transport Security Layer (TLS), is becoming more prominent. Although designed to be used in critical tasks such as banking transactions and sending secure e-mails, with the ascending amount of web services replacing offline services, TLS application has spread to a wide range of services. According to the Google Transparency Report[2], around 95% of web pages accessed via Google Chrome were loaded over Hyper Text Transfer Protocol Secure or HTTP over TLS (HTTPS) as of August 2018.

1.1 Motivation

In light of the above, companies are struggling to deploy monitoring systems that are effective in applying Quality of Service (QoS) policies and estimating the performance of network applications in a way that complies with the security and privacy standards. On the other

hand, users of a network can take advantage of these limitations to access illicit content that does not conform to the company rules. For instance, according to a survey conducted by the AMA/ePolicy Institute[3], the primary concerns of employers that actively block access to the web are workers visiting adult sites with pornographic content, game sites and social networking services (96%, 61%, 50%, respectively). In another recent survey[4], about 37% of respondents admitted watching TV shows and movies at work. Despite the fact this type of content can be blocked with the utilization of specific software such as OpenDNS, sometimes this is not the ideal practice, as knowledgeable users can easily bypass these systems through the use of proxies or by merely changing the local machine's Domain Name Server (DNS). Plus, with the usage of protected channels such as SSH tunnels or Virtual Private Network (VPN), the effort of enforcing network policies becomes hugely complex. However, it is an absolute necessity, particularly in companies with a large workforce, as the corporate resources should be spent on productive work for the company. Other methods of traffic monitoring/blocking and their limitations will be discussed in Section 2.3.

1.2 Contributions

In this dissertation, a network-based framework for service identification will be presented, focusing on video applications. Since this approach should consider both security and privacy, the collected data should be non-intrusive, using only low-level traffic statistics to make predictions. It should be effective in identifying a given service in a timely way, regardless of the network setup and underlying protocols, with a strong focus on protected channels. Although traffic classification problems have been studied extensively throughout the years, there is a lack of contributions regarding the context of VPNs and protected tunnels. This study aims to help fill that gap in the field, by providing insight into how modern classification techniques behave when the network environment becomes unfamiliar. Despite the fact that this work will focus on video application classification, the proposed framework should be able to adapt to any type of application, which is proposed when future work is discussed in **??**

Besides proposing an ML model for identifying video traffic and video applications, this work also presents a framework for live traffic analysis, using the results from the creation of the classification model. This framework should be scalable, easily deployable and should work in real-time. Although this framework is designed to be used at a corporate level, it can also be adapted and used by internet service providers for traffic shaping purposes.

In Chapter 2, the most common encryption protocols will be presented (Section 2.1). After establishing how networks can be monitored (Section 2.2), we go into detail about the evolution of traffic classification and the challenges the field is currently facing. We conclude this chapter by discussing individual privacy rights in the light of recent regulations in Section 2.4.

In Chapter 3, we begin to explain the proposed framework for service identification in real time. Firstly, we describe the process of creating ML models and how to approach every component of the development. Secondly, we describe the architecture of the developed

real-time system, along with its requirements, used technologies and explain each decision made during development.

In Chapter 4, we describe the experiment to build an accurate traffic classifier with detail. It opens with the used methods to build the private dataset, describing its composition and characteristics of each class. Then the results are presented and discussed.

We conclude with Chapter 5, explaining the contributions of our work and how it could be enhanced. All the developed code for this dissertation is available online.[1]

[1] https://github.com/alagoa/application-identification

CHAPTER 2

State of the Art

2.1 Overview of encryption protocols

This section provides a brief description of some of the most common encryption protocols that are currently used. Since our goal is to identify network services no matter the context, it is important to establish the behavior of these protocols, both in the initialization of the connection and during the exchange of encrypted data. We describe each protocol's properties, dynamics and packet structure. The chosen protocols are **Internet Protocol Security (IPSec)**, **TLS**, **SSH** and **OpenVPN**, since they are among the most used protocols for increasing the security of communications. Although **OpenVPN** is not a protocol itself, it is becoming the most widely used open-source implementation of VPN techniques, using a custom protocol based on TLS.

2.1.1 IPSec

IPSec protocol suite is a set of protocols defined by the Internet Engineering Task Force (IETF)[5]. It provides encryption, authentication and data integrity in the network layer, as well as anti-replay. As a result of being a Layer 3 protocol, it protects both the payload of the packet and its network layer information.

The purpose of IPSec is to provide a secure network connection, independently of the application on each node. It is usually used for inter-site connections in corporate networks. The fact that is not required to change any application used in a particular connection for IPSec to work is its main advantage. Yet, this creates difficulties in ensuring the protection of specific applications. Furthermore, some of the algorithms used in the IPSec cryptographic suite are outdated. Plus, the fact that it has to encrypt an extremely high amount of data, adding overhead to the network, is a major drawback of the protocol.

The most critical part of IPSec is the ESP protocol, which provides the security features previously described. It adds a header and a trailer to the packet, according to the mode of

operation. In *transport* mode, only the payload of the original packet is encrypted, which means that the original IP header is used. In *tunnel* mode, both the payload and the original IP header are encrypted. A new IP header is created with the endpoints of the established tunnel.

2.1.2 TLS/SSL

TLS is a protocol based on Secure Sockets Layer version 3 (SSLv3) that provides secure communications when using the Transmission Control Protocol (TCP) protocol. As previously referred, it has become one of the most widely used encryption protocols, with an increase of 40% of services using HTTPS since March 2015.

TLS contains two levels of protocols, as described in Figure 2.1. The first level, called the *handshake layer*, contains the initialization sub-protocols that deal with the initial handshake, while the second level, called *record layer* contains the Record protocol, which serves as an envelope for application data. The Record protocol handles the encryption, authentication and splitting of the transmitted data. The structure of a TLS packet can be seen in Figure 2.1.

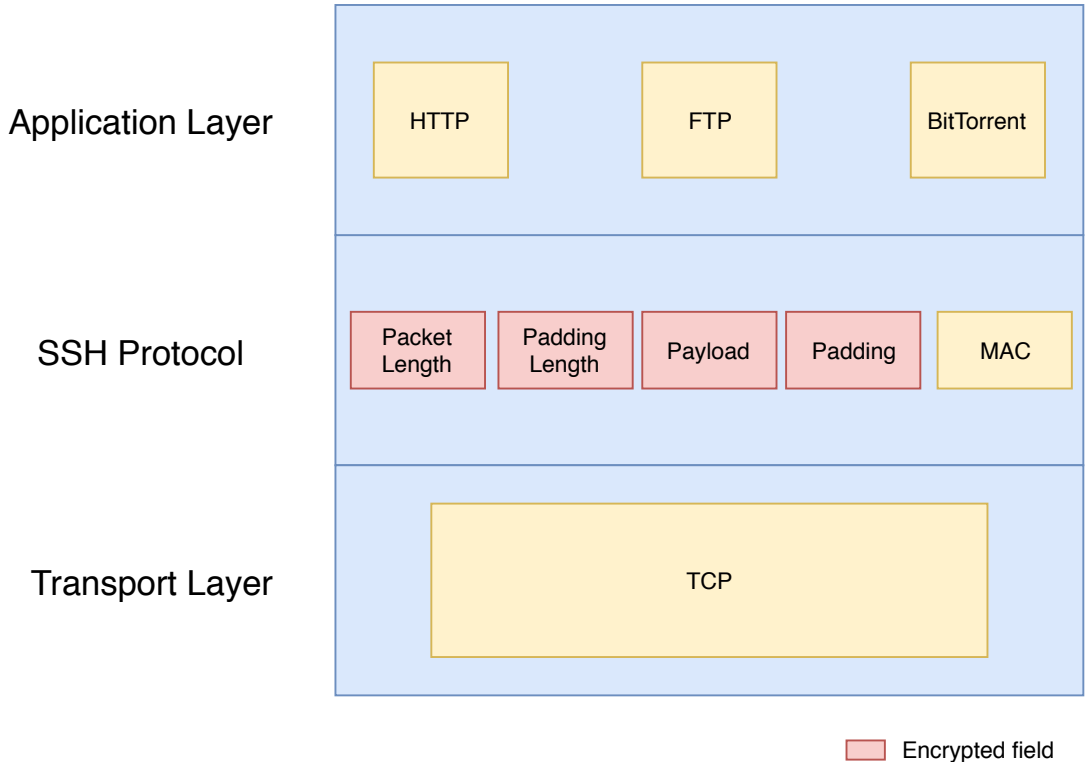


Figure 2.1: TLS/SSL packet structure.

A crucial part of the TLS protocol is the *handshake*. It is in this section that both parties define critical aspects such as authentication using an X.509 certificate chain, cipher suite negotiation and session key establishment. The messages exchanged in this process are unencrypted, up until a Change Cipher Spec message is sent, which causes a change in the encryption algorithm used in the messages.

2.1.3 SSH

The SSH protocol acts on the application layer and provides a secure channel in client-server communications. While it is best known for remote login in computer systems, it can also be used to securely transfer files, to tunneling applications and set up a VPN, usually using the OpenSSH suite. It provides encryption, client/server authentication and data integrity and is essential in most corporate environments. The SSH protocol packet format, seen in Figure 2.2 consists of an encrypted payload that contains the packet and padding length, the application data and the padding. The Message Authentication Code (MAC) at the end of the packet is not encrypted, as SSH uses an *encrypt-and-mac* approach.

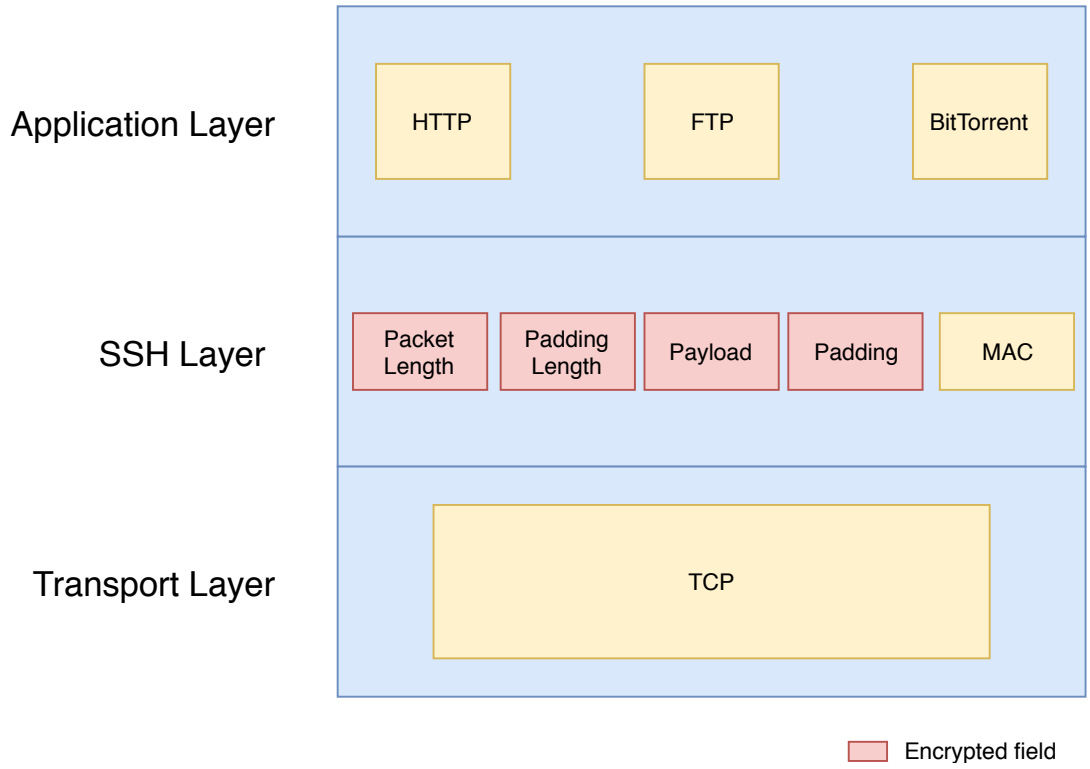


Figure 2.2: SSH packet structure.

An SSH connection begins with the establishment of a TCP connection. After the preferred encryption algorithms are exchanged, the client/servers keys are verified, and a shared key is created. After assuring its legitimacy, both parties start to send secure messages.

Since SSH allows the tunneling of any application running over TCP, network administrators that allow the use of this protocol would not be able to restrict what application the users' access, potentially exposing the network to insecure traffic. It is important to note that, in the case of an HTTP over SSH connection using port forwarding, only the IP addresses of the tunneled connection are visible in the IP headers, making separate application flows virtually impossible to differentiate.

2.1.4 OpenVPN

OpenVPN is an open-source software that provides VPN solutions. It can run either on TCP or User Datagram Protocol (UDP). Its security features are based on TLS and it uses the OpenSSL[6] cryptographic library. Since it is one of the most used open-source VPN implementations, it will be one of the contexts we will apply the developed framework.

2.2 Network monitoring

Since this dissertation will deal with network data, the way that data is acquired is a vital part of the process. Several methods of capturing data exist and they all have their own perks, but there are many guidelines to be followed that guarantee that the data is captured more faithfully and efficiently. In this section, these methods will be presented.

There are two types of monitoring:

- **Active monitoring**: Which requires the injection of dummy traffic and follow its flow in the network.
- **Passive monitoring**: It involves monitoring traffic that is present in the network. As such, it encompasses the need for a device in the network that can capture the traffic, such as a probe.

Active monitoring is regularly used when testing the performance of a given network, giving insight into the QoS and see how the network reacts to specific scenarios. Passive monitoring is viewed as the best method to analyze a network over a period of time, providing a holistic view of the network and useful when taking into account large amounts of data. This dissertation will focus more strongly on passive monitoring, exposing its specific methods along with their advantages and drawbacks.

The first type of passive monitoring is hardware-based monitoring, which is traditionally performed by hardware devices such as a Test Access Point (TAP) or using a mirror-port on a switch.

A TAP is a device which can capture traffic between two nodes on a network. They are often utilized in security contexts, as they cannot be detected in the network. Since they mostly work in full-duplex, the sniffed data arrives at all devices in real time[7]. In Figure 2.3 it can be observed how a TAP is typically located in a simple network.

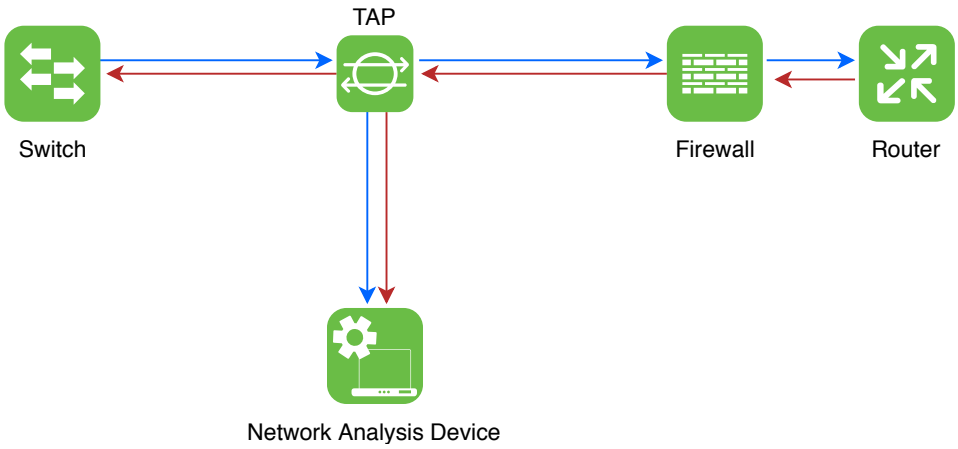


Figure 2.3: Network TAP diagram.

A TAP is often advantageous in situations where network performance is the goal, as it provides an overview of the state of a connection. It has also the perk of having a negligible packet loss rate. On the other hand, it cannot monitor intra-device traffic and it comes with the additional cost of the purchase of the device.

Another method of hardware-based monitoring is to use port mirroring, or Switch Port Analyzer (SPAN). This approach consists of forwarding replicates of the incoming/outgoing traffic of a given device interface to another that is connected to the machine that will analyze the traffic. Figure 2.4 shows an example of the network setup that uses this technique for monitoring the traffic.

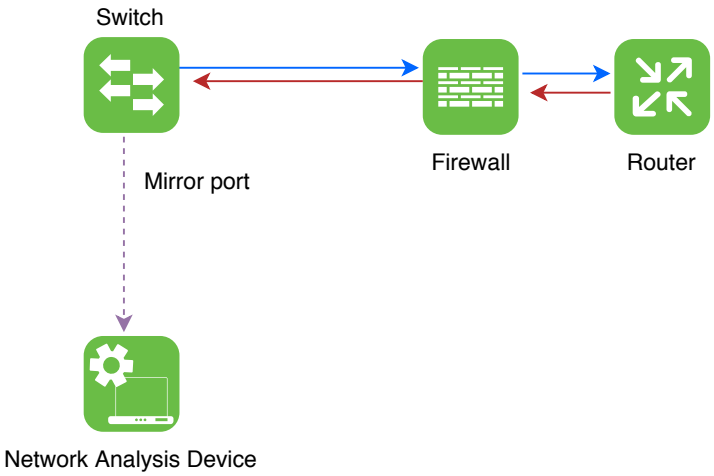


Figure 2.4: Port mirroring example diagram.

Port mirroring has numerous perks such as being capable of being configured remotely, does not require additional hardware, it can capture intra-device information and several device interfaces, in case the switch has that capability. However, it is usually a computationally expensive task for the switch's Central Processing Unit (CPU), which makes the packet loss rate considerably higher when compared to TAPs. Plus, since the CPU is so overloaded, some timestamps attributed to packets can be inaccurate, putting the reliability of the system at

risk.

Finally, there are specific software solutions, which can be used in terminals, that are much more versatile than the methods described above. One popular example is Wireshark[1], a powerful packet sniffing tool that can be used for monitoring both wired and wireless connections. One of the factors that are pushing software-based monitoring is the emergence of virtualized network environments, which makes hardware-based approaches not applicable in most cases.

These type of packet captures are usually conducted using devices such as network cards from computer machines such as desktops and laptops. Although this can be useful to monitor traffic more straightforwardly, usually these devices have a high packet loss rate, since they are not optimized for that task. However, this is usually not a problem in contexts where the loss of a small percentage of the packets is inconsequential, like building network application profiles. In situations such as testing the network performance, packet loss usually cannot be afforded.

2.3 Network traffic classification approaches

Network traffic classification has a significant role both in the research community and the industry. This section describes how the classification techniques evolved over the years, dividing them into their respective categories, stating their application and limitations. Even though many of the following examples apply to application protocols and not to specific application services such as a web service, it is essential to understand how these methods evolved throughout the years.

2.3.1 Port-based classification

The port-based approach is one of the first traffic classification methods to be used. It consists of the analysis of what ports a given connection is using, by analyzing the TCP (or UDP) header. An application protocol is usually associated with a well-known port number, which makes port-based classification a popular method to use in Access Control List (ACL) and firewalls. Table 2.1 shows the correspondence of network applications to their protocol numbers, assigned by the Internet Assigned Number Authority (IANA)[8].

Protocol	**Port**
File Transfer Protocol (FTP)	21
DNS	52
HyperText Transfer Protocol (HTTP)	80
HTTPS	443

Table 2.1: Official port numbers of some well-known protocols.

[1] https://www.wireshark.org/

Albeit fast and privacy-minded, these techniques quickly became obsolete, as applications could merely use unregistered port numbers or use a dynamic port. For example, the Session Initiation Protocol (SIP) parameter negotiation for an internet telephone call uses Real-time Transport Protocol (RTP) on random port numbers. The usage of dynamic ports is also common in Peer to Peer (P2P) applications[9], [10]. Some applications also may use well-known port numbers (such as port 80) to circumvent detection mechanisms. Thus, with the low accuracy of port-based methods, new approaches were needed to overcome these shortcomings.

2.3.2 Deep packet inspection

The first alternative to the port-based classification was DPI, also commonly called *payload-based inspection*, which scans through the application packet payload in order to match specific patterns with the signature of a network application. This means that the signature of applications/web services must be stored in a database, must be kept updated and maintained in order to keep up with the evolution of the service's *fingerprint*. DPI techniques can be applied to individual packets, but they are more effective when performed on a packet flow, in order to identify intricate patterns. DPI methods quickly became the standard for many traffic classification and intrusion detection tools.

OpenDPI was an open source solution for applying DPI techniques. The application/protocol signatures that are used to classify traffic are embedded in the pattern detection mechanism code, which means that if a pattern needs to be added or updated, the source code needs to be recompiled. Although no longer maintained (it has been forked under the name of *nDPI*), many studies were made on *OpenDPI*. Particularly, a study by Finsterbusch et al.[11] found that both *OpenDPI* and *nDPI* managed to classify TLS traffic with an accuracy of 100%.

L7-filter is an open-source package for Linux that classifies IP packets based on their application data. It is not a *pure* DPI classifier, as it takes into account non-payload data such as the used ports and the number of transferred bytes in a flow. The payload data inspection is done by using regular expressions to predict the network protocol. Although not as accurate as *OpenDPI*, it is more suitable for in industrial practice, as the signatures are independent of the software[12].

Libprotoident[13] is a C library which inspects only the first 4 bytes of a packet payload, greatly reducing the time that it takes to make a decision. Because of this, *Libprotoident* is generally perceived to be not as accurate than other DPI tools, which is not always the case. In a survey conducted by Finsterbusch et al.[11] that compared several DPI tools, *Libprotoident* showed the highest accuracy for the whole dataset, with the downside of demanding more processing power.

Several studies were conducted in order to identify application protocols using DPI. Bernaille et al.[14] managed to detect TLS traffic only by detecting the occurrence of *ServerHello* packets. Bujlow et al.[15] conducted an experiment which compared the most popular DPI

tools for traffic classification, comparing open source software such as the libraries presented above, as well as commercial solutions, like *PACE*[16]. The researchers concluded that *PACE* was the most reliable solution for the tested protocols and applications.

In general, the differences between these approaches differed in the amount of data needed to make a decision, whether they used information about incoming/outgoing traffic.

Because of the high accuracy of DPI methods, they are often used as the ideal method to build the ground truth, being the standard validation method when comparing to other approaches.

Even though DPI methods at the time showed near perfect accuracy, some drawbacks needed to be addressed. Firstly, as previously stated, most of the software was slow and required a significant amount of processing power. Secondly, the application signatures these tools relied on were volatile and it was not practical to update them. Thirdly, some of the libraries will not produce accurate results when encryption is used. Lastly, in places such as a corporate network, tools that examine the packet content may come across sensitive worker data when used for monitoring the network. This raises ethical issues related to data privacy, which will be discussed more thoroughly in Section 2.4.

Although most DPI libraries relied on pattern matching and regular expressions, some modifications were made to the existing tools in order to increase the accuracy and area of applications. Methods that combine techniques from different areas are called *hybrid methods*, which will be covered in Section 2.3.4.4. For instance, Hjelmvik[17] developed Statistical Protocol IDentification (SPID), an application protocol identification scheme which combined statistical data from the network flow with application layer data. This, study along with other studies in this area, hinted that the traffic classification approaches were shifting to engines that relied more on statistical properties of a connection rather than a thorough inspection of the actual data that was being transmitted.

2.3.3 Statistical based classification

Statistical based methods rely on the principle that there are distinctive properties and patterns in the traffic generated by specific applications. Among the first studies that established this, a particular study conducted by Paxson indicated that features such as flow duration and the number of transmitted bytes were deeply related to the type of application[18]. These studies were the basis for the creation of classifiers based on statistics. This type of classifier does not count on payload analysis, making its predictions only using low-level traffic features. These characteristics are usually unique to each application and even when the application is updated these features tend to remain invariable. Contrary to DPI tools, this is true even when encryption is introduced[19], [20].

For example, Wang et al.[21] managed to differentiate protocols such as FTP, HTTPS, Internet Message Access Protocol (IMAP), SSH, among others. This approach relied on packet size distributions and achieved an overall accuracy of around 87%. Crotti et al. [22] also attained over 90% of accuracy in their work using probability density functions based on

packets inter-arrival time.

However, these basic techniques proved to be inefficient when used in more complex scenarios, such as identifying applications with intricate dynamics such as web services and P2P applications. So more complex rules, called *heuristics*, were introduced. A notable example is the work conducted by Karagiannis et al.[23], in which the researchers managed to detect P2P applications through the use of a heuristic that defined that a port using both TCP and UDP hints at the usage of a P2P application.

Statistical based techniques were first applied to identify protocols such as HTTP, FTP and Post Office Protocol (POP3). In order to identify a protocol, the incoming traffic was compared to the *fingerprint*(also called *profile*) that the developed algorithms built for each protocol. This is a case of an *application profile*, which is constructed using statistical properties of flows generated by a particular application. For instance, Wang et al.[24] managed to achieve perfect accuracy and over 90% of recall when attempting to differentiate P2P applications. Their approach employed the longest common sub-sequence to detect the determining packets of a flow, which tries to identify the longest streak of common data that is the same in two packets from different flows.

In [25], the authors propose a method that uses *stochastic fingerprints* for applications running over TLS, using first-order Markov chains on the packet's time distribution. These applications included Twitter, PayPal, Dropbox and Skype. The researchers concluded that many protocols implementations do not follow the respective Request for Comments (RFC) and behave differently from the standard TLS stacks. This indicates that methods that rely on the analysis of the protocol's initiation phases and specific fields of data to identify an application could be useless in some cases.

Another type of profile is based on the host. These *host profiles* are created when taking into account attributes such as the number of flows a host is generating[26] and other host behavior. The most remarkable application of these techniques was BLINd Classification (BLINC)[27], a methodology that divided host behavior into three different levels: *social*, *network* and *application*. The *social level* focused on establishing the *popularity* of the host, by taking into account the diversity of its connections. The *network level*, also called *functional* level, captures the behavior of the host regarding its role on a network. For example, the host may be a service provider or a consumer. Finally, at the *application level*, the focus shifts to identifying what kind of services is the host interacting with. They achieved this by having empirically derived patterns represented by graphs. This multi-level approach leads to an accuracy of around 90% when classifying types of traffic. However, the approach is not as accurate when identifying specific application subtypes. In addition, one of the most debilitating limitations is that this method entirely relies on the assumption that the packet headers are not encrypted, which means this approach would not function in a protected channel environment.

Despite DPI being more accurate than the methods presented above, some drawbacks

inherent to them, such as the low speed and the computational overhead, are not present in statistical based techniques. This makes techniques that rely on flow characteristics to be much more suitable for real-time systems, where the quickness of a prediction is a crucial factor. Nonetheless, most of the purely statistical based methods also have their limitations. For instance, BLINC[27] was not able to detect many different applications, which implicates that some services may display identical profiles, making it difficult to distinguish them from each other. Plus, it has been proven that these approaches do not scale well when these methods are faced with unexpected objects. Although this can be solved by manual adjustments of the developed tools (by adding or modifying rules), it is not practical and usually is a tedious and lengthy process. In order to automate this and make developed tools more reliable for a broader range of applications, a subset of Artificial Intelligence, called ML was introduced to IP traffic classification.

2.3.4 Machine learning techniques

As stated at the end of section 2.3.3, ML is a subset of Artificial Intelligence (AI). The two terms are often confused with each other, but while AI is the concept of a machine or a program being able to act in a way that makes its goals achievable, ML is the group of methodologies that allow computers to take data and transform it into that intelligence.

One of the first application of ML was the *spam filter*, back in the 1990s[28], which is a program that tags an email as *spam* when given examples of *spam* emails and regular emails.

Although ML applied to traffic classification only became mainstream at the end of the 2000s, back in 1990 a network traffic controller that aimed at maximizing call completion was developed[29]. After this study, several others followed, particularly in the area of intrusion and anomaly detection[30]–[32]. These studies paved the way for much of the ML work that would be done in the following century.

2.3.4.1 Basic ML concepts

In this section, basic ML concepts will be introduced. These concepts exist across all fields of ML, and not only in the traffic classification area.

As established at the beginning of this section, ML transforms data into intelligence. The input data that an ML program uses to build its model is called the *training set*. Taking the *spam filter* as an example, the set of emails from which the program will learn from is called the *training set*, which is composed of training examples, called *samples*, or *instances*. Each instance is characterized by its *features*, attributes which values, as a whole, represent a *sample*. Features can be binary values, numeric values, text values or nominal values. For instance, usually in the networking field, subsequent packets form an instance, with its features being the number of bytes in the packets, the inter-arrival times, the standard deviation of packet lengths, among others.

Usually, the performance of these methods uses *accuracy* as a measure. In the case of the *spam filter*, the accuracy is the ratio of correctly classified emails, while in traffic classification is the ratio of correctly classified traffic. We go into a bit more detail about performance measurement in Section 2.3.4.2.

2.3.4.2 Performance measurement

To assess the performance of a ML model, it is necessary to have a *testing set*, which are sets of labeled data. The model built with the training set is applied to the testing set and, with the results, a confusion matrix is created. A confusion matrix[33] is a visualization of all possible outcomes of a prediction.

In Table 2.2 represents a confusion matrix of a binary classifier. As can be observed, there are four possible results:

- True Positive (TP) - A true positive occurs when the predicted class of the sample was *true* is the actual class is also *true*.
- True Negative (TN) - A true negative occurs when the predicted class of the sample was *false* is the actual class is also *false*.
- False Positive (FP) - A false positive occurs when the predicted class of the sample was *true* is the actual class is also *false*.
- False Negative (FN) - A false negative occurs when the predicted class of the sample was *false* is the actual class is also *true*.

		Predicted	
		Spam	Not spam
Actual	Spam	90	15
	Not spam	10	80

Table 2.2: Example of a binary confusion matrix.

Table 2.2 shows a confusion matrix for a *spam filter*. By observing the table, we can see that there were 90 TP, 80 TN, 10 FP and 15 FN.

The values of the confusion matrix can be combined to create more representative metrics such as *precision*, *recall*, *f-measure* (which combines *precision* and *recall*) and *accuracy*. Accuracy is usually the chosen metric to compare different approaches.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \qquad Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2.3.4.3 Types of learning

A survey from 2008 on ML techniques in traffic classification categorized learning types as *supervised* learning, *unsupervised* learning (or *clustering*) and *semi-supervised* learning[34].

2.3.4.3.1 Supervised learning

In *supervised learning*, the *training* data the system uses includes an attribute which is the solution, called *label*. This is illustrated in Figure 2.5.

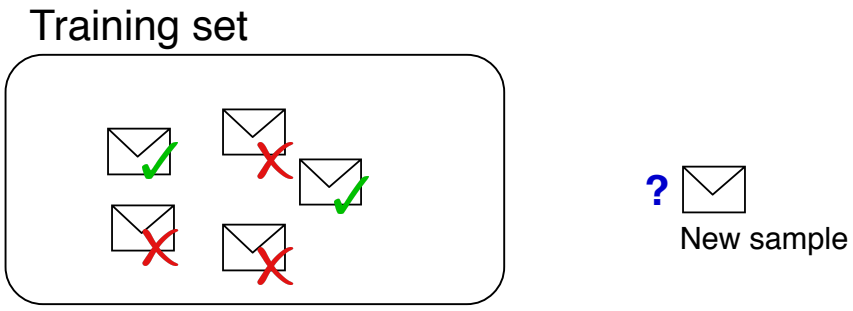


Figure 2.5: A labeled training set for supervised learning (e.g., spam classification)[35].

This example describes a *spam filter* training set. Some samples are marked as *spam* and others are marked as *not spam*. This way, a model is created, being able to classify new instances. But supervised learning is not used only in classification problems.

There are two types of *supervised learning* tasks: classification and regression. In classification, the model is trained with examples in which the *label* is their class. The model then *classifies* a given sample based on its features (e.g., Figure 2.5).For example, network application identification is a *classification* task in which the *label* of a given *sample* is the application identifier. In regression, the task is to predict a target numeric value, such as the value of a household.

Supervised learning is one of the most common methods of traffic classification and includes many algorithms like k-Nearest Neighbor (kNN), Neural Networks (NN), Support Vector Machines (SVM), Decision Trees:

k-Nearest Neighbor kNN is an approach that, by using feature similarity, determines the distance between samples, where instances from the same class should be near each other. In order to calculate those distances, it must be given a proximity function, such as the Euclidean distance[36] or the Mahalanobis distance[37]. One of the main advantages of kNN is that the algorithm does not require training time since the instances fed to the model are stored. Nevertheless, it requires higher computation in the testing phase, since it has to calculate the distance of the test point to all training samples in order to classify it.

In [38], Roughan et al. presented a study in which kNN was applied to distinguish application categorized as *interactive* (applications that require multiple interactions from a user), *bulk*

data transfer (such as FTP or P2P applications), *streaming* and *transactional*. The developed model managed to have an overall accuracy of around 95%. However, when a new application of a certain category is introduced the accuracy was not as high.

Wright et al. [39] used packet sizes, their direction and timing attributes to identify application protocols, achieving 100% accuracy for HTTP, HTTPS and FTP, although there was a considerable quantity of false positives.

However, kNN is usually sensitive to outliers and the use of irrelevant features. Its accuracy also can change drastically if the k value or the proximity function is changed[37].

Neural Networks NN are mathematical models for creating complex relationships between the data input and data output. It is based on how the biological nervous systems, like the brain, process information. Despite being introduced long ago in 1943 [40], only in the last two decades promising practical systems using Artificial Neural Networks (ANN) have been developed, because of factors such as the appearance of more massive datasets and the higher computational power. Other causes include more efficient algorithms and studies that indicated that some of the theoretical limitations would not be relevant in practice.

Auld et al.[41] developed a Bayesian trained neural network to classify traffic in categories like P2P, games, multimedia and attack (such as worms and virus). Their approach achieved 95% of accuracy on a dataset extracted eight months later to the construction of the model.

A recent technical report [42], using the same dataset as [41] found an optimal configuration for a Bayesian neural network that obtained over 99% of accuracy, although in very specific contexts.

A promising study conducted by Wang et al.[43] aimed at classifying tunneled traffic using one-dimensional convolution neural networks. Their approach achieved >90% of accuracy when classifying encrypted traffic. However, the traffic was split into vast categories (like *email*, *streaming*, *VoIP* and *file transfer*). Nevertheless, when compared to state of the art methods such as C4.5, the accuracy improved over 10% for VPN traffic.

Despite seeming promising, most NN approaches have some disadvantages. For instance, they are not suitable for small datasets where each instance has a large number of features. They also require high computational power and the training time be usually long.

Support Vector Machine An SVM is an algorithm whose objective is to find one (or more) hyperplane(s) that divide data in two or more classes. One of the main advantages of SVM is that it usually produces excellent results with minimal datasets, as can be evidenced by [44], where the researchers proposed a protocol classifier with an accuracy of over 90% in most cases.

Another notable study, Yang et al.[45], aimed at building a model for P2P traffic identification and application level classification. They achieved an accuracy of 97.3% for their binary classifier. Plus, this approach aimed to be successful in a real-time system environment, so it

only considered the first few seconds of a connection. Thus, SVM was proved to be a good solution for identifying traffic in large networks.

Despite indicating to be a promising alternative for traffic classification, SVM has two huge limitations. Firstly, SVM models are prone to overfitting, because of their complex algorithms and likelihood of misestimating the importance of features. Although this can be solved by parameter tuning and feeding more data to the algorithm, it regularly leads to the second problem - SVM algorithms often have long training times[46]. By giving a larger dataset to the model, the training phase can become impractical.

Decision Trees Decision Trees are algorithms that classify samples by arranging them based on the values of its features so that the best feature is at the root of the tree. There are various metrics to define the importance of each feature, such as the *gini impurity*, *entropy* and *variance reduction*. Their main advantage is being able to handle large datasets[47], [48].

In [49], Jun et al. compared two Decision Trees algorithms, J48 and REPTree, and achieved around 95% accuracy identifying known and unknown P2P traffic, using features such as flow duration, packets inter-arrival time and total of transmitted bytes.

A study conducted by McCarthy et al. used C4.5 decision trees along with AdaBoost to identify Secure Sockets Layer (SSL) applications, achieving around 98% of accuracy.

A recent study [50] proposes a multi-level framework to identify application services in HTTPS using the C4.5 algorithm, achieving around 88% accuracy. This method divides the network packets in separate flows and analyzes each flow individually.

Despite these promising results, these algorithms usually don't generalize well, which commonly leads to an overfitted model. Moreover, it is hard for these algorithms to update an already built model. Therefore, they may not be ideal for a system with constant retraining. That being said, at the beginning of the decade, faster variations of decision trees were developed, called Very Fast Decision Tree (VFDT), which can be more suitable for real-time systems. For instance, in [51], the developed model attained an accuracy above 98%, while consuming three times less memory than C4.5. Plus, its update time was remarkably faster than other static decision trees.

2.3.4.3.2 Unsupervised learning

In *unsupervised learning*, the training data is *unlabeled*. Concerning classifications, this means that the system should divide the dataset into classes (or *clusters*) without help. Figure 2.6 illustrates a classification example using *unsupervised learning*.

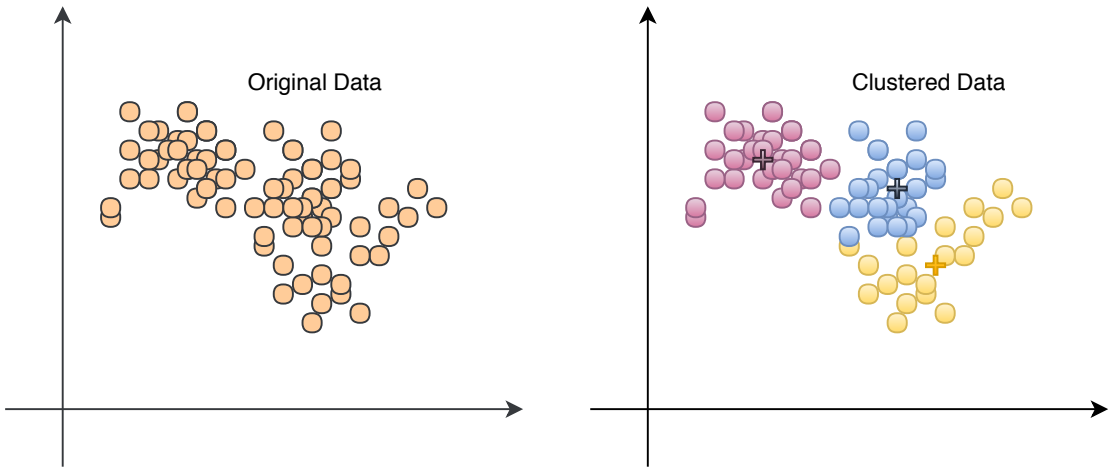


Figure 2.6: Unsupervised learning example.

The most notable of the unsupervised learning algorithms is *k-Means*, but researchers developed other valid unsupervised solutions specifically for traffic classification.

Autoclass was a method first applied to application identification by Zander et al.[52], in which the researchers attained 86.5% accuracy overall.

Maolini et al.[53] analyzed SSH traffic and managed to determine which underlying application protocol was running, using a k-Means algorithm. The analyzed protocols included Secure copy (SCP), Secure File Transfer Protocol (SFTP), HTTP). The achieved accuracy was around 99.88% for the underlying protocols running over SSH.

Erman et al.[54] identified P2P applications with an accuracy of 95% using k-Means, using only data from server-to-client flows. Some of the researchers from that study also compared the algorithm to AutoClass, verifying that k-Means performed better and is faster[55]. In the same study, another clustering algorithm called Density-based spatial clustering of applications with noise (DBSCAN) proved to be the most reliable of the three.

In 2013, Zhang et al.[56] proposed an enhanced version of the k-Means algorithm, they were able to improve the accuracy of the algorithm for classifying encrypted traffic.

Despite not having impressive accuracy when compared to some supervised learning algorithms, clustering methods can be hugely effective when combined with other techniques such as Markov models, heuristics, or supervised learning algorithms, as shown in Section 2.3.4.3.3 and Section 2.3.4.4.

2.3.4.3.3 Semi-supervised learning

Semi-supervised learning algorithms are trained on labeled and unlabeled data. Since in many situations it is hard for all the data to be labeled, either by financial costs or human incapability, these methods proved to be useful in these situations. Plus, in some cases, labeled data could inflict human biases to the created model, making semi-supervised methods more accurate.This is depicted in Figure 2.7. In most semi-supervised learning algorithms, the

samples are exposed to an unsupervised training algorithm in the first phase. In the second phase. Then, after all of the data is separated by clusters, it is only necessary to label one instance per cluster.[2]

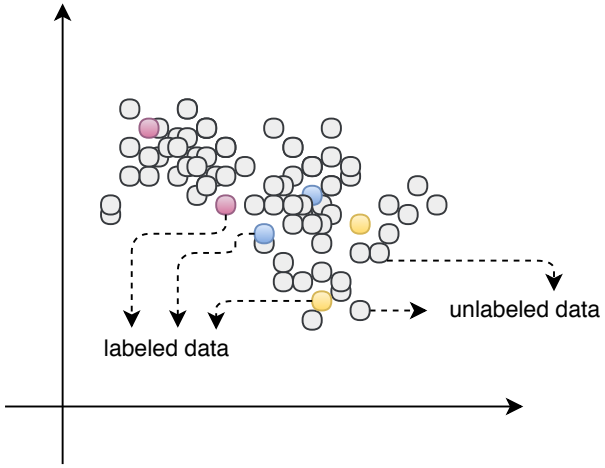


Figure 2.7: Semi-supervised learning example.

Semi-supervised learning algorithms have been commonly used in network related classification problems. For instance, Bar-Yanai et al.[57] constructed a protocol classifier by combining the k-Means and kNN algorithms, making a fast model that is resistant to encryption and packet ordering.

Bernaille and Teixeira[58] developed a model that recognized applications in SSL, with an accuracy of 85%, using a three-step method. Firstly, they identified an SSL connection by applying a clustering algorithm based on the Gaussian Mixture Model[59] on the first three packets of a TCP flows. Secondly, they detect the first data packet by inspected the contents of the packet. Finally, the underlying application is recognized by applying another clustering algorithm to the sizes of the application packets and an euristic involving utilized ports.

In [60], Erman et al. fed a combination of labeled and unlabeled flows to a clustering algorithm. Afterward, they mapped each cluster with labeled flows to an application category, using a simple probabilistic assignment. With this approach, they managed to build a model with 94% accuracy.

As can be concluded by the studies presented above, for the most part, semi-supervised methods are more complex that other methods, as they combine several approaches. When comparing the accuracy to supervised methods, they tend to be not as accurate. However, usually, semi-supervised methods are adopted for achieving high efficiency and are preferred when the training data is limited.

[2]Except when a desired class is slipt in several clusters, or when a single cluster has more than one desired class.

2.3.4.3.4 Ensemble methods

Ensemble methods use multiple algorithms to make the accuracy of the prediction higher. One prevalent example are *Random Forests*[61], which consist in having several Decision Trees in the training process. The predtiction of a Random Forest is the mode of the Decision Trees decision that compose it.

The most common usage of ensemble learning is *boosting*, particularly AdaBoost[62](short for Adaptive Boosting). For instance, in [63], the authors developed a method to detect applications using AdaBoost and achieved over 90% of accuracy. However, the authors admit that this method is not suitable for real-time systems.

Although not prevalent in traffic classification, some surveys [64][65] reveal that methods that use multiple classifiers are starting to achieve better accuracy than simplified ML approaches.

2.3.4.4 Hybrid methods

To minimize the disadvantages of some ML algorithms, a standard approach is to combine a number of classification techniques that can complement each other, achieving better results. This is called *hybrid classification*.

Wright et.al[39] managed to achieve over 90% accuracy when identifying application protocols in encrypted tunnels. They used a combination of kNN and Hidden Markov Models to classify the applications. Plus, they also show that it is possible to identify the number of flows in an encrypted tunnel, with an accuracy better than 20%. It is important to note that, unlike most studies in this area, this method did not perform a *flow analysis*, as it is unviable to pinpoint different flows in an encrypted tunnel, making this a *stream analysis*.

As stated at the end of Section 2.3.2, Hjelmvik[17] developed SPID, an application protocol identification technique that combined DPI with statistics. It measured byte frequency in a data flow and made assumptions based on that statistical data.

2.4 Privacy rights and ethical aspects of network monitoring

In nowadays society, privacy issues have been one of the most prominent topics of debate. The global surveillance disclosures[66] by Edward Snowden, in 2013, raised the public's awareness of the subject and put government's trustworthiness at stake. It was also proven that particular social networking services were also involved in the monitoring systems. More recently, with the Facebook-Cambridge Analytica data breach[67] which contained over data of 87 million users, the public's distrust is continuing to extend to companies. The notion that corporations instead of individuals now own knowledge and data creates a conflict of knowledge ownership.[68].

2.4.1 Information privacy laws

In light of the above, governments have been adopting data protection laws in order to minimize the exposure of sensitive citizen data. Notably, the General Data Protection Regulation (GDPR) has received plenty of attention.

GDPR is a regulation on data protection that affects all European Union citizens. Its main goal is to give control of data back to the individuals. It requires companies to handle European Union (EU) citizens' personal information responsibly. This includes actions such as asking the individuals consent when using personal data, clearly stating its purpose and usage, while giving them the right to ask for the correction of that data, as well as its destruction. Note that this applies to all companies that handle personal data from an EU citizen, even if the company itself is not based in the EU geographical area[69]. Corporations that do not comply with this standard when audits are carried out may face extremely harsh penalties, up to 4% of the company's annual revenue[70]. Therefore, companies are re-writing their procedures and adapting in order to avoid eventual fines.

Even though when one talks about GDPR's requirements it usually concerns *user privacy*, the regulation clearly states that it applies to *citizen privacy*. This, by default, covers *worker privacy* as well, since companies keep worker data.

2.4.2 Network and device surveillance in the workplace

So how does traffic monitoring comes into play regarding privacy? Well, employees may consider that network surveillance in the workplace a breach of their privacy. While the purpose of monitoring systems is to collect work-related information, they may come across sensitive data. Most of the time, since the employer owns the network and the computer terminals, monitoring is allowed. However, regarding personal devices, there is a gray area where no concrete regulations are established. In these situations, it is up to each corporation to have its own policies and inform their employees about how they will be enforced. Furthermore, with all the controversies of the recent years and the rise of the public's awareness, more and more regulations are coming into practice, universalizing workplace policies on monitoring.

For instance, the IETF has released the RFC 7258[71] on pervasive monitoring. This RFC states that practices such as subverting the cryptographic keys in secure communications, collecting application content and even protocol metadata can be considered an attack on the privacy of users. Yet, the authors admit that some of these procedures are necessary for performance and security reasons. They conclude by saying that there is the risk of these monitoring mechanisms to be abused, so a trade-off between worker privacy and having a healthy, manageable network is necessary.

To sum up, network monitoring is a sensitive subject, where one has to consider both the network management and the privacy of the individuals who use it. The framework developed in this dissertation presents a reasonable compromise between these two mindsets, as the utilized data complies with the standards presented in this section.

CHAPTER 3

A Framework for Network Service Identification

As stated throughout this document, the objective of this dissertation is to develop a framework for application services identification on a given network. The first step is to create a ML model. This is done by collecting data when using those services, labeling the data, extracting relevant features from it and build a classifier from the acquired dataset. The second step is to set up a system where that ML model can be used to identify traffic in real-time.

The designed system has two levels of classification. The first level tries to classify traffic into two groups - **video** and **non-video**. If an observation is classified as **video** by the first classifier, it is sent to the second level of classification, which will try to identify what specific video application the observation belongs to. Figure 3.1 illustrates this two-level framework.

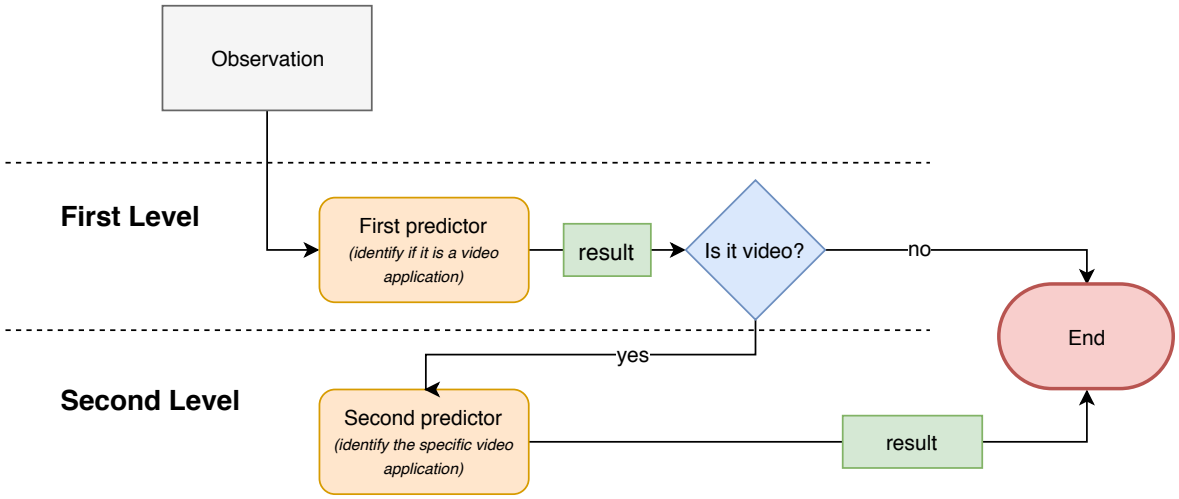


Figure 3.1: Diagram of the two-level prediction system.

In this chapter, we will give an overview of the developed framework and we will go into detail about each component that forms the system. It also describes how new ML models

can be built and integrated with the real-time system. This process includes the acquisition of data and the steps one has to take to transform it into a useful predictive tool for a live traffic analysis.

3.1 System overview

As described above, the system has two main components, the building of the ML model and the live analysis system. A simple overview of the framework can be observed in Figure 3.2.

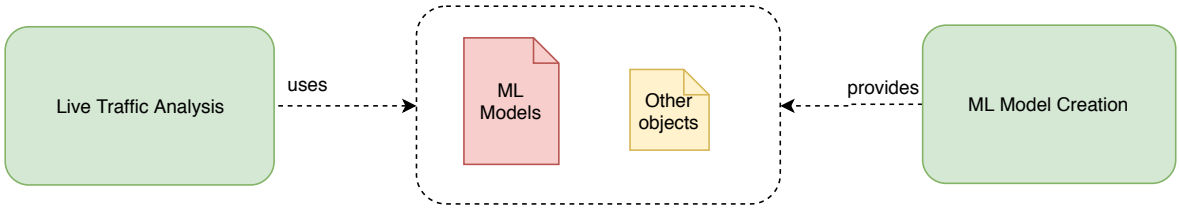


Figure 3.2: Framework overview.

The purpose of the **ML model creation** process is to create classifiers to use in the live traffic analysis. This process will also create other objects (explained in detail in Section 3.3.1.2) that will need to be given to the live traffic analysis.

The **Live Traffic Analysis** is a fully fledged system for application identification that uses the classifiers created in the **ML Model Creation** process. Section 3.3 goes into detail about its requirements, composition and decisions made when choosing the technologies used in the development phase.

3.2 Building the ML model

The process of building an ML model that is reliable and, above all else, accurate, can be extremely complex. To simplify the procedure, it can be divided in simpler parts, as it can be seen in Figure 3.3.

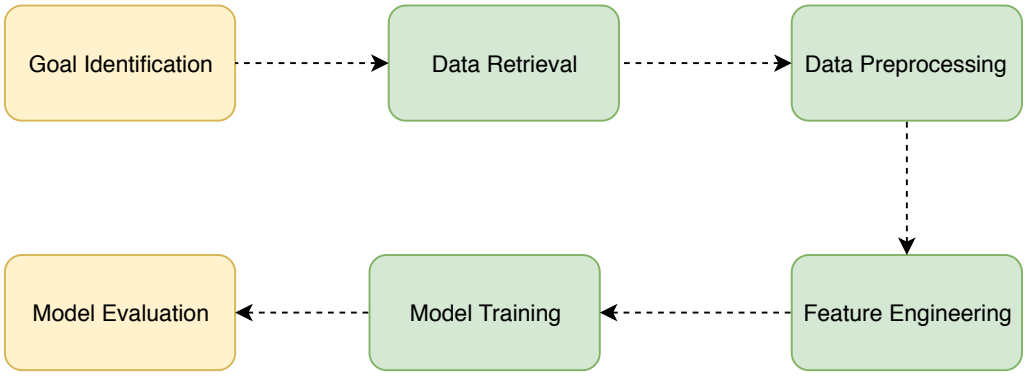


Figure 3.3: Building an ML model process.

3.2.1 Goal identification

The first phase consists in identifying the problem to be solved. In the case of this dissertation, the problem is that it does not exist an accurate, non-intrusive and real-time method to identify what network application service is being used by a machine without relying on expensive computational power. Thus, the **goal** is to build an ML model to classify traffic with those characteristics in mind. This leads to the definition of the classes that will be created for the ML model (labels), which in these case are the names of the application services that will be identified.

The next step is to define what is the input data that will be used to achieve the goal. This input data does not necessarily need to be exactly what is given to the ML model for training - additional processing is usually required before feeding the ML model the data. Since this system will be analyzing network traffic in real time, the input should be network data captures.

Now that the goal is defined, the next step is to fetch the data.

3.2.2 Data retrieval

In this stage, the input data should be collected to be processed to the desired format in the next phase.

As referred in the section above, the data used to train the ML model is network data that was captured when using the application services we want to identify.

In order to build reliable ML models for each service, the collected data must be as comprehensive as possible. To achieve this, four crucial requirements must be met. Firstly, the captured data should be captured in a wide variety of settings, whether geographically or regarding the network environment itself. Secondly, the times at which the captures are conducted. Thirdly, since most traffic is influenced by human behavior, it is optimal to capture data of several different individuals, to decrease the skewness of the dataset. Finally, the volume of the captured data and how it should be well distributed across all the analyzed services.

There are several open datasets of network captures that are widely used in this type of research. Notable examples include the Center for Applied Internet Data Analysis (CAIDA)[72] and the datasets from Stanford Network Analysis Platform (SNAP)[73].

Even though building a private dataset is time-consuming, it can be the best option to obtain data that complies with the requirements above, since it can be hard to determine if the open datasets meet those requirements. Plus, in a private dataset one can include reliable data from tunneled traffic, which is hard to find available online.

The labeling of the data could be done manually - which is more time consuming but leaves no room for error - or with the use of techniques such as DPI, as presented in Section 2.3.2, which is a popular method of determining the ground truth. The use of *software agents* (or *bots*) that are programmed to access certain applications (which become automatically labeled) is

also an option, although it may be difficult to emulate the human behaviour, possibly making the samples unreliable. Other methods also include the use of controlled environments where the users/machines are restricted to use the targeted applications.

3.2.3 Data preprocessing

When the data is acquired, it is usually not ready to be given to an ML algorithm. The raw data should first be *translated* into **features** that are easier for the ML models to take as an training input. This process is called data preprocessing.

For instance, in this specific case, one could argue that the raw packets data fields (such as the timestamps, used ports, packet size) could be used as inputs for the algorithm. However, this would make the training process awfully slow, since a high number of packet data is required to build a reliable ML model. Plus, the probability of some applications having packets with similar properties is exceptionally high. Along these lines and taking into account other researches conducted in the area, that were presented in Chapter 2, it was decided that statistics from the raw packet data over a fixed period of time would be used as **features** for the ML model. Figure 3.4 represents the developed method of taking capture files and extracting the relevant data from them, ultimately creating a single file with the samples ready to be fed to the ML algorithm. This process has two parts, extracting the relevant data fields from the raw packets and calculating statistical data to create samples.

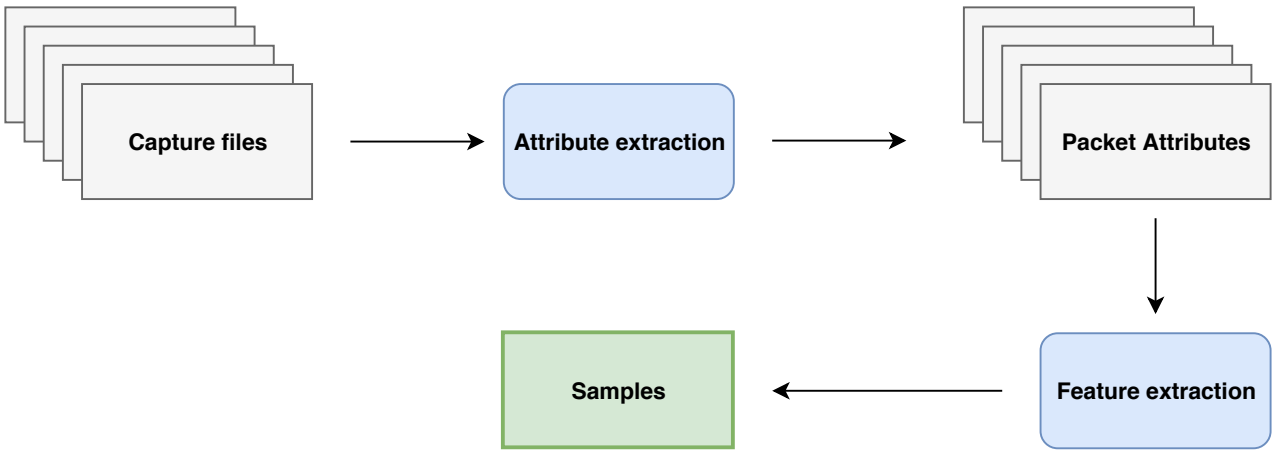


Figure 3.4: Data preprocessing pipeline.

The first developed tool's purpose is to extract relevant data fields from the packets. The objective is to count the number of packets and number of bytes, and divide them into fixed length time segments (or *time buckets*), based on the packets' timestamp. The only data fields taken from each packet were:

- Source and Destination IP address - in order to determine if the traffic was outgoing or incoming.
- The packet's timestamp - in order to divide the packets into *time buckets*.
- The packet's length.

- The SYN flag value, in the case of a TCP packet.

These fields are processed and divided into *time buckets*. The attributes of each *time bucket* can be found in Table 3.1.

Attribute	**Description**
up_packets	Number of uploaded packets
up_bytes	Number of uploaded bytes
up_syn	Number of outgoing SYN flags
down_packets	Number of downloaded packets
down_bytes	Number of downloaded bytes
down_syn	Number of incomming SYN flags

Table 3.1: *Time bucket* attributes.

As referred above, these fields are processed and divided in time segments and saved to a text file with the *.dat* extension. The structure of a *.dat* file can be observed in Table 3.2. Each line represents the amount of data in a *time bucket*. For instance, if we assume the *time bucket* is 1 second, we can observe that in the specific file represented in Table 3.2, there were 1540 outgoing packets and 980 incoming packets in the first second of the capture.

	Outgoing			Incoming		
	# packets	# bytes	# SYN flags	# packets	# bytes	# SYN flags
1	1540	875588	70	980	1218	6
2	1468	893158	56	1112	1230544	22
	...	...	...	...	...	...
N	970	376598	58	938	1236668	6

Table 3.2: Structure of a packet attribute file.

The next step is to use the *feature extraction* tool, which takes the created *.dat* files, applying statistical calculations on them in order to extract features. The result is a single file with samples to use as input for ML models.

However, before that transformation takes place, the data needs to be *standardized*.

3.2.3.1 Standardization

Standardization, also known as feature scaling, is a method of making all categories of data have a standard range[35]. It is commonly used because some ML algorithms do not perform well with features in different scales. Figure 3.5 shows an example of standardization.

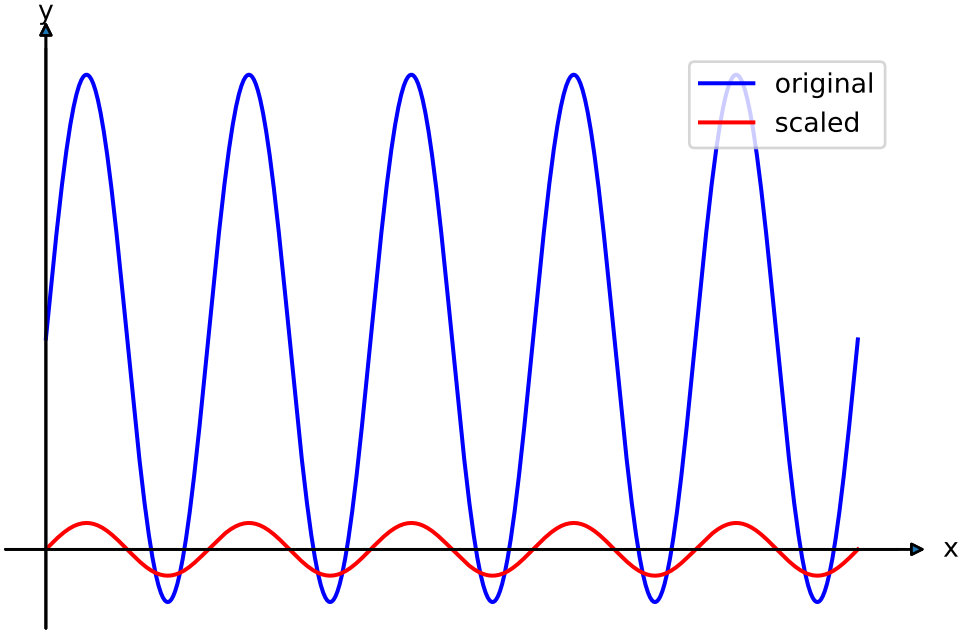


Figure 3.5: An example of standardization in a sine wave.

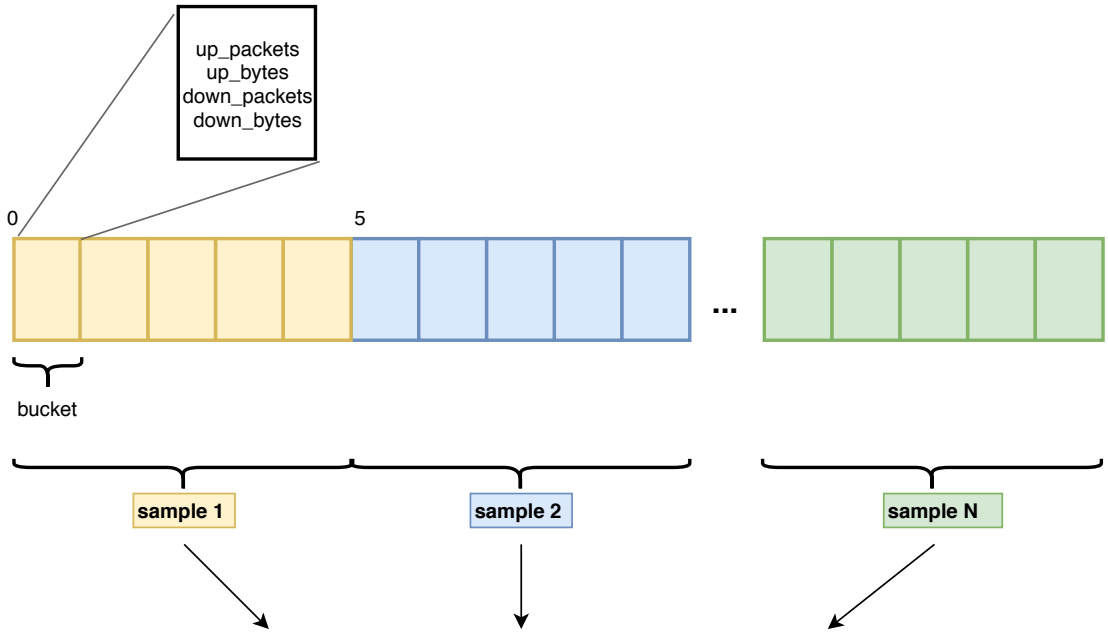
The standardization method used in this dissertation is StandardScaler[74] from the Scikit Learn library[75]. This implements a scaler in which the transformed data distribution has mean value 0 and standard deviation of value 1. After calculating the mean and the standard deviation for an attribute, the feature is scaled by using the following calculation:

$$\frac{X_i - \bar{x}}{\sigma}$$

Usually, standardization is used in the data **feature engineering** step (Section 3.2.4), applying the standardization to the features. However, in this dissertation, it was also used directly on the data from the packets. Since different applications have substantially different upload/download rate (see Section 4.2.1), there will be a significant difference in values such as *byte count*. Plus, most video streaming applications can detect if a connection has low bandwidth and adjust the quality of the video to the connection, making some characteristics of the flow (such as the packet size) differ from a flow of a standard, high-quality connection. For example, the byte volume attributes of a network packet capture of a 1080p video are widely different than the capture of a 480p video, even if it is the same application. The process of standardization of this data is done to dampen the effect of this type of situation.

3.2.3.2 Sample size definition and silence values

As previously stated, each *time bucket* was formed by dividing the packets into time segments. The next step is to choose a sample size for the dataset. As it can be seen in Figure 3.6, each sample is formed by several *time buckets*.



ID	up_packet_mean	down_bytes_median	...	up_packet_silences
1	0.3	0.4	...	0
2	0.21	0.42	...	2
...	...	...	...	...
N	0.86	0.47	...	1

Figure 3.6: Samples structure.

For instance, in Figure 3.6, assuming each *bucket* in corresponds to 1 second and that a sample has five *buckets*, the sample size is 5 seconds. Each sample is created by doing calculations on each group of buckets in order to extract relevant statistics, which will then be used as features for the ML model.

Another particularity in this work is that the number of *time buckets* without activity (when the upload or download attributes are 0) has to be accounted for in order to extract time-related features, as described in Section 3.2.3.3. However, since these values are scaled upon the statistics extraction phase, their value will not be 0. Figure 3.7 illustrates this problem.

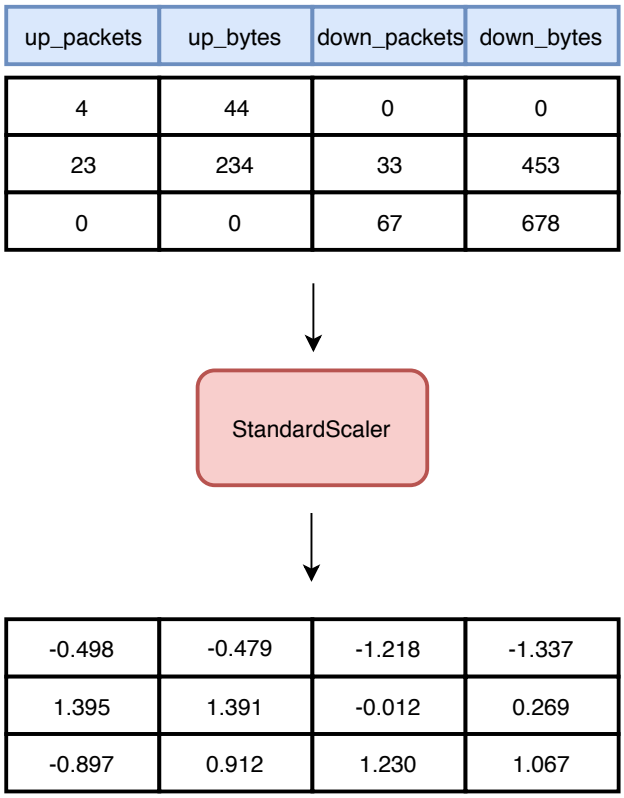


Figure 3.7: Normalization example.

As it can be seen in Figure 3.7, the 0 values are transformed into the scaled data, which makes it impossible to identify in which *time bucket* there was no activity.

A simple method of solving this issue is described in Figure 3.8.

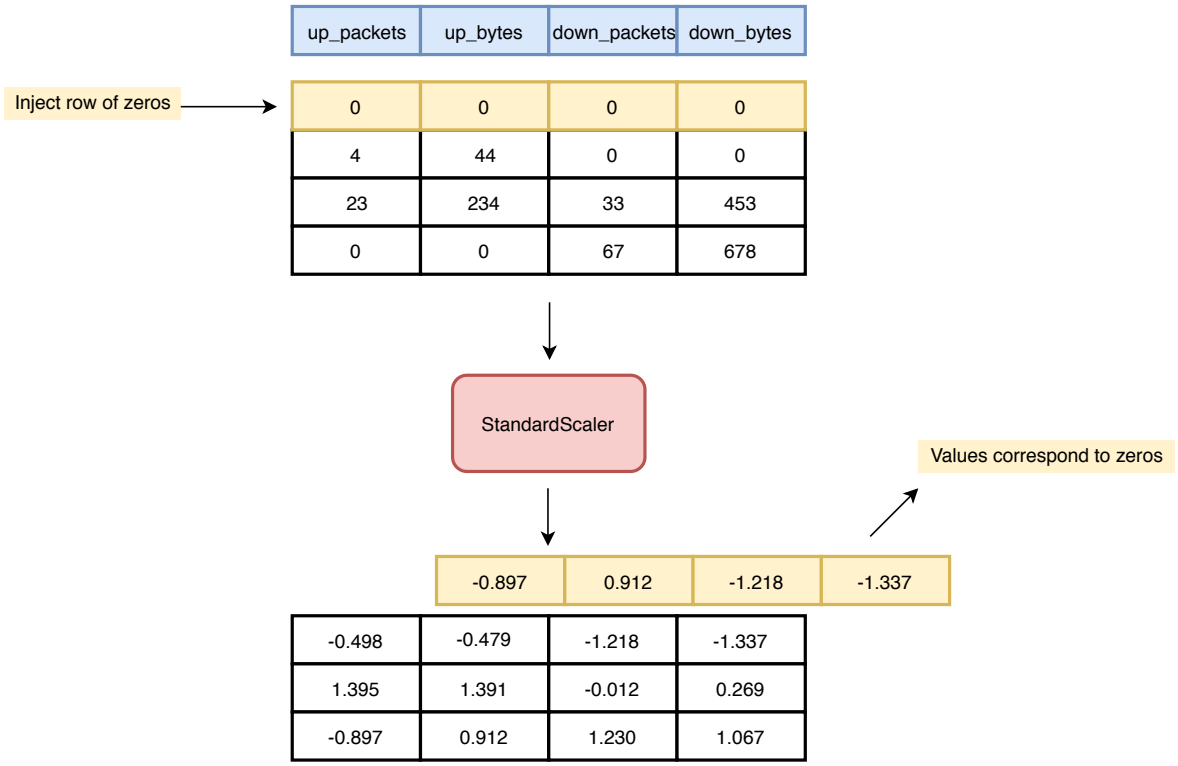


Figure 3.8: Normalization example with inactive *time buckets* identification.

By injecting a row of zeros at the start of the data, that row will also be scaled. Knowing that the values of that scaled row correspond to periods of inactivity, we can use it to identify those periods when calculating the statistical features in the step of creating samples.

It is important to note that these attributes are not *features* of the ML model, but the attributes of each processed *time bucket*. The normalization of *features* is described in Section 3.2.4.

3.2.3.3 Features

As previously presented in Section 3.2.3, the relevant packet attributes are divided into *time buckets* which will then be processed to create *features*. There are 134 features used in this work and they are all numerical. They can be divided into 3 large groups:

- **Basic statistics** - These features are calculated by directly applying functions to the input data (Table 3.1). They should be the most important features, as they provide insight into the actual data that is being transferred. For instance, a *video streaming* service should have a more distinct pattern than a newspaper website, since the latter is much more prone to be influenced by user behavior.
- **Silence periods** - These features are used to identify time-related patterns. Some applications may have consistent silence periods, some may have no silence periods at all. The goal of these features is to create a timing *fingerprint* for each class, making them more differentiable.
- **Pseudo-periodic components** - Wavelet transforms are used in this dissertation to bring out events of different periodicity[76]. For instance, events caused by human

behavior, such as *clicks*, are usually associated with low-frequency components, while protocol specific procedures, such as handshakes and control channels, are medium frequency components. High-frequency events include the traffic flow or packet bursts. In this work, we use the Continuous Wavelet Transform (CWT), since it provides insights into the time and frequency aspects of a given packet flow. CWTs are also commonly used in data compression and edge detection[77]. The remainder of this section describes how the pseudo-periodic components is extracted and what data is used as features for the classifier.

After obtaining the CWT for each attribute(*up_packets*, *up_bytes*, *down_packets*, *down_bytes*), the **scalogram** of the CWT is computed. The **scalogram** is the normalized modulus squared of the CWT, averaged over time. The result can be seen in Figure 3.9

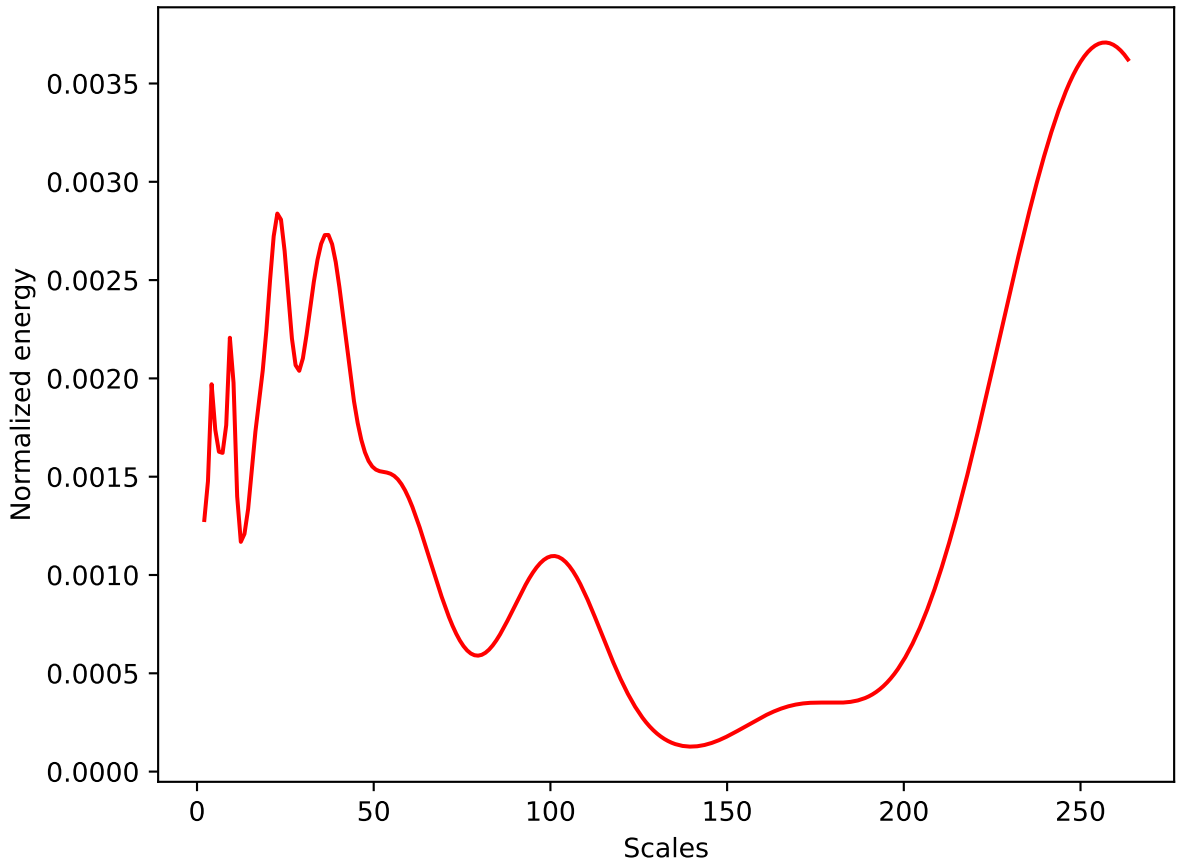


Figure 3.9: Scalogram of the *down_bytes* attribute of a video sample.

After the scalogram is obtained, both the highest maximum and lowest minimum peaks are extracted, up to 5 each. Figure 3.10 shows the points that are extracted. It is common practice to extract the points in fixed intervals (every 10 units, for example), but this method will theoretically extract more relevant points.

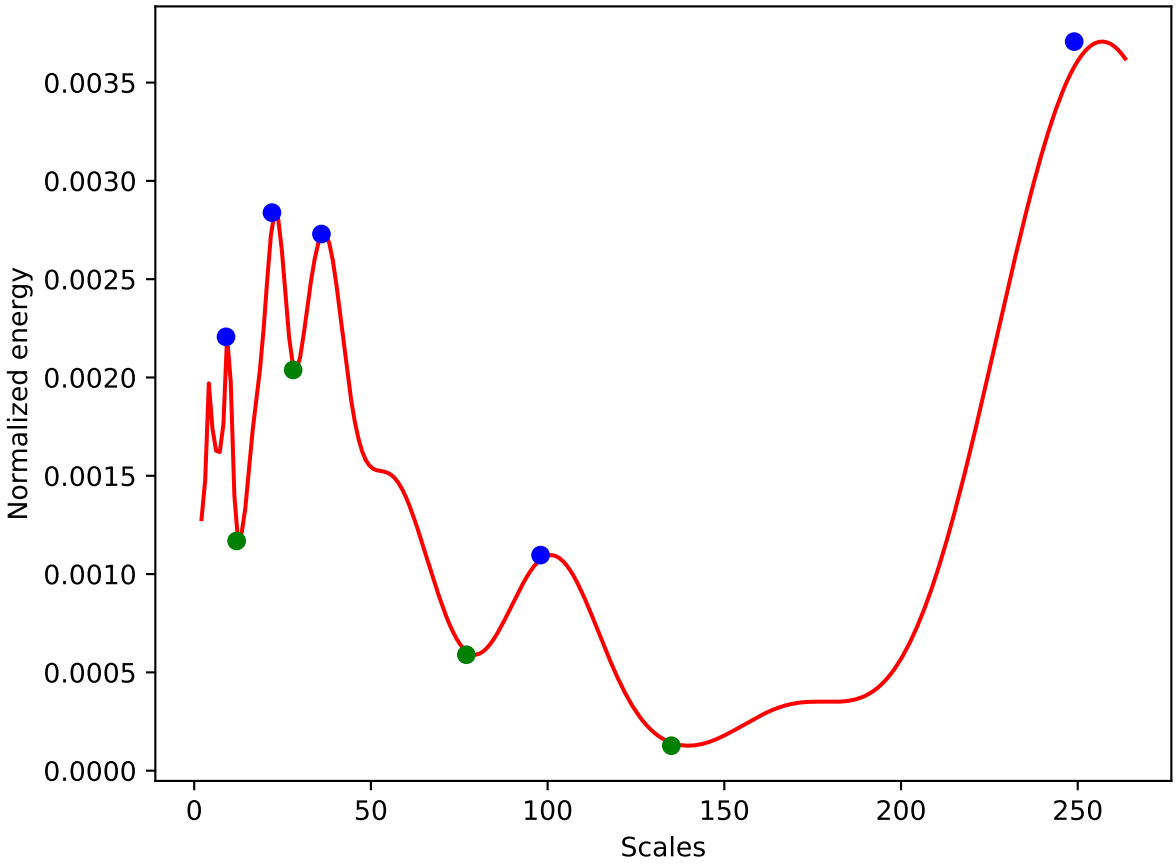


Figure 3.10: Detection of spikes in a scalogram.

It can easily be observed that not all spikes were extracted. For instance, there is a local maximum and a local minimum between the [150, 200] values of the x axis, but since they are much flatter than the marked spikes, they are not as relevant. The same applies for the first local maximum and the first local minimum in the example, but for a different reason - there are spikes with a higher (or lower, in the case of the minimums) value that are relatively close to these points. Consequently, these first two points will not be considered as spikes. This is done because if the five maximums and minimums were extracted by merely taking into account the y value, some relevant points would be ignored, e.g., the fourth maximum in the example in Figure 3.10. The algorithm for detecting the spikes is presented in Chapter 5.

The spikes coordinates are returned sorted by the y value. This way, the scalogram features will be ordered by the amplitude of the data, and not by order of occurrence in the scalogram.

The pseudo-periodic components features are the x and the y coordinates corresponding to the detected spikes, so there are 2 features per spike. As mentioned earlier, this is done for every *time bucket* attribute, in total, there will be 80 pseudo-periodic component features.

3.2.3.4 Feature dimensionality and reduction

A classifier that has 200 features is not necessarily better than a classifier that has 20 features, for example. This is because the features must be representative of a given label. Although there is not an optimal number of features, a large number of features usually require a large number of samples. Otherwise, the ML model could have problems such as overfitting. Usually, when one increases the number of features without increasing the number of observations, the results tend to be worse[78].

However, in many ML problems, there is not a simple method of determining if a given feature will be important for a given ML model without training the model with and without the sample and performing a results evaluation. Since this is not practical for problems that have a high number of features, other solutions are usually chosen.

A standard method of increasing the accuracy of ML models is through *feature reduction and combination*, the most used method being PCA[79]. PCA is a dimension-reduction technique that is used to reduce the number of features of a dataset, combining them in a smaller group that still contains the information of the original dataset. The resulting features of the PCA process are called the *principal components*. A comparison of spaces before and after PCA reduction can be found in Figure 3.11

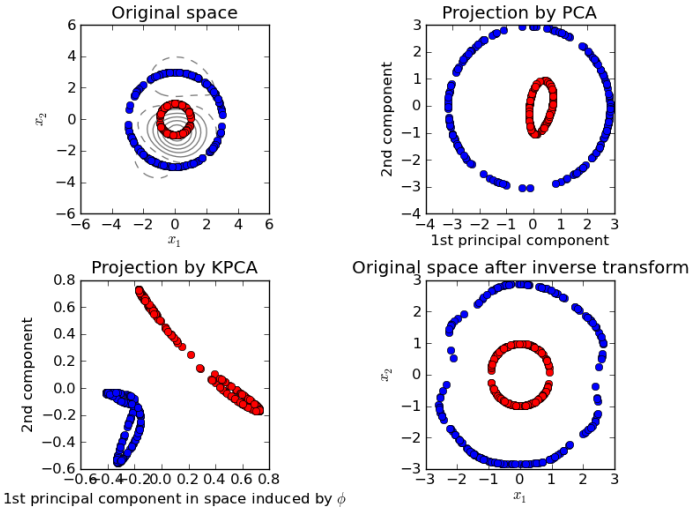


Figure 3.11: Comparison of dimensional spaces before and after applying PCA reduction [80].

3.2.4 Feature engineering

In the stage of data preprocessing, data is subjected to transformations in order to make the ML model have better results. It is also used to make data comply with the requirements of certain ML algorithms. For instance, Random Forest's input features can never have a *null* value. This specific case could be treated with data imputation or by merely disregarding the sample, for example. In Scikit Learn, this is achieved by using the *Imputer* class[81].

One common technique used in data preprocessing is feature scaling, which consists of stan-

dardizing the limits of the values of features. It is the same process described in Section 3.2.3.1, only applied to **features** instead of the packet data. This is done because most ML algorithms do not perform well when numerical attributes have different scales. For instance, two possible features for this project are *up_packet_median* and *up_bytes_median* which represents the median of outgoing packets and the median of outgoing bytes, respectively. In this case, the *up_bytes_median* feature has a much broader range than the *up_packet_median*, so it is generally a good idea to scale the features in a dataset.

PCA is also used as an optional step in this phase, since reducing the number of features can prevent the algorithms from overfitting and the resulting *principal components* can have as much information as the original dataset, but with less noise from non-relevant features.

3.2.5 ML Model training and evaluation

After going through the feature engineering step, the samples are split into two groups: the *training set* and the *test set*.

After the dataset splitting, the data is ready to be used for training ML models. As exposed in Section 2.3.4, many algorithms have been used successfully in traffic classification, such as SVM, Neural Networks and Decision Trees, although no solution is better than another for all cases. For instance, a comparative study conducted by Shafiq et al.[82] concluded that Decision Trees and SVM performed the better than Naive Bayes Bayes Network on both encrypted and unencrypted traffic.

Since all our data is labeled, the chosen algorithms to be tested were all *supervised learning* algorithms. As stated at the beginning of Chapter 3, two ML models will be developed. Since the first level of classification will be binary (identifying if a given sample is from a video capture or not) and the second level of classification will have multiple classes, it was decided that several algorithms should be tested to compare how they perform in different situations. The chosen algorithms were Decision Tree, SVM, Neural Networks and Random Forest. Additionally, the AdaBoost algorithm was used along with both Decision Tree and Random Forest to try to achieve better results.

As for the evaluation, in addition to the traditional measures presented in Section 2.3.4.2, we also use CV and ROC[83] to analyze the performance of the resulting classifiers. CV is a widely used technique that estimates how the classifier will behave when classifying unseen samples. It usually works by splitting up the training set in **K** parts, called *folds*, training the ML model with **K** times, using a different part for testing every time, training with the remainder **K-1** parts. Figure 3.12 illustrates this process.

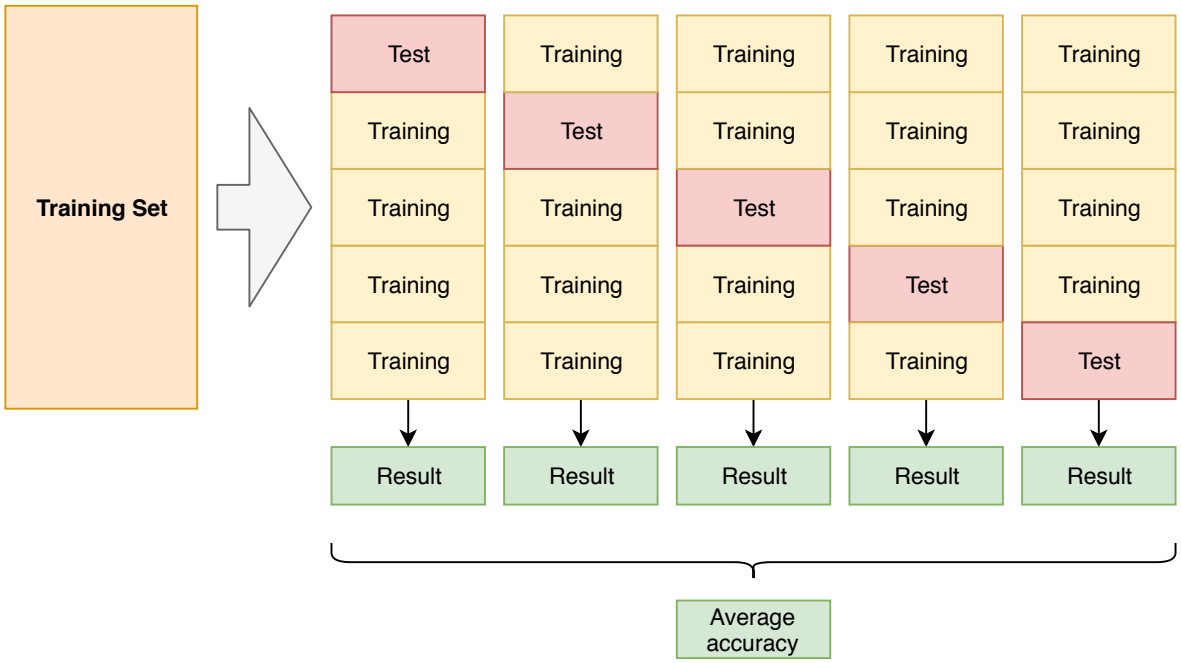


Figure 3.12: Example of a 5-fold CV.

ROC is used to plot TP rate against the FP rate, serving as another method, along with the confusion matrix, of visualizing the accuracy and reliability of an ML model.

3.3 Live Traffic Analysis

As stated at the beginning of this chapter, the point of developing ML models is to use them to perform real-time traffic classification. Since the system is aimed to be used in settings such as a corporate environment or as an Internet Service Provider (ISP) monitoring tool, one should consider some requirements when designing the system.

The most important aspect is that the system should be able to identify traffic in pseudo-real time. As stated in Chapter 4, the sample size that achieved the best results was 60 seconds, even if other sample sizes achieved good results (with an accuracy of over 90%). When taking into account the processing time of the samples, a prediction should be available not much time after the traffic is captured.

Another requirement is that the system should be easily scalable, which means it should be simple to add components to the system without having to change its architecture, especially when adding machines that will be monitored. Consequently, it should also be easy to manage and the interface should not be complex. Plus, since the system is composed by several parts that presumably will be deployed in different devices, each component should be easily deployable.

Finally, since it is likely that there will be several terminals to monitor, the processing of the acquired packets should preferably be done asynchronously to avoid delays in the predictions.

This section outlines the architecture and describes the components, how they interact and the technologies that were used to develop the system.

3.3.1 System architecture

As it can be seen in Figure 3.13, the system is composed by 4 parts:

- **Capture module** - The purpose of this component is to capture traffic in the device and redirect it to the respective processing unit. Since the used capture method is software based (for reasons explained in Section 4.1.1), it is simpler than the component that captures the traffic is located in the device itself.
- **Processing module** - the processing module is in charge of taking the network packets from the capture module and process them to form a sample that will be tested with the developed ML models (see Section 3.2). After the **Worker** makes a prediction, the result is sent to the server side through an HTTP POST request.
- **Server side** - the server side is composed by the Representational State Transfer (REST) server and the HTTP server that acts as the bridge from the REST server to the external clients and serves the web pages requested by them.
- **Client side** - the client side is simply composed by an application that makes HTTP requests, such as a web browser, in order to visualize the status of all the monitored devices.

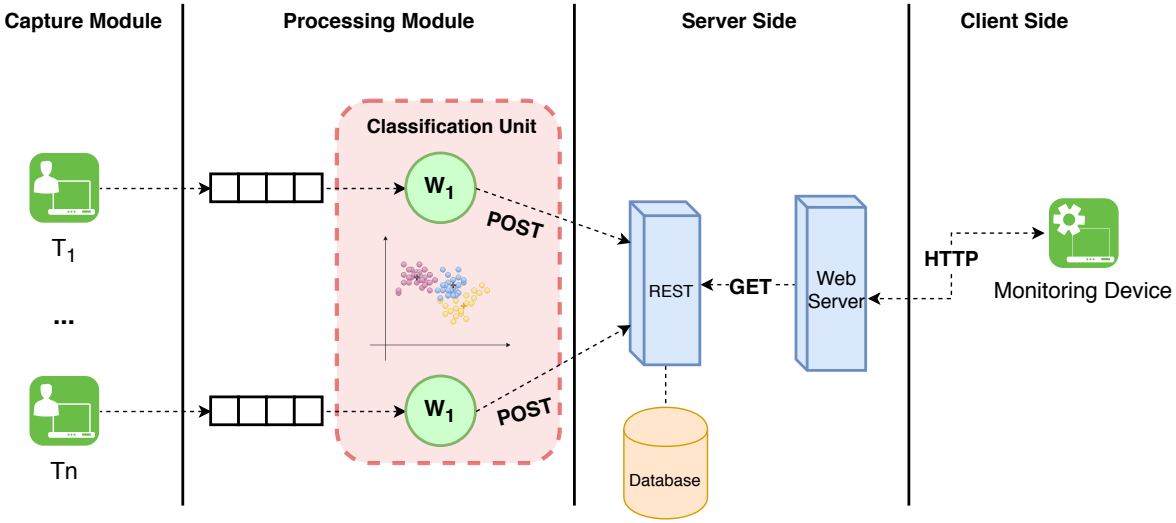


Figure 3.13: Live system architecture.

In order to meet the requirements exposed in the beginning of this chapter, the process of choosing what technologies and frameworks to use to develop the system is crucial. The scalability of the system has to be considered, as well as its speed and performance. It is also essential that each component not have a big tools ecosystem, such as libraries and other requirements. This makes the whole system much lighter and easy to deploy.

All the components of the real-time system were put in its own software packages using **Docker**[84], a program that performs operating-system-level virtualization, making the

deploying process much easier.

Figure 3.14 shows what frameworks were used in the development of each component of the system. Since the ML models were developed using Scikit-Learn[85], Python was the chosen language for the development of the live system.

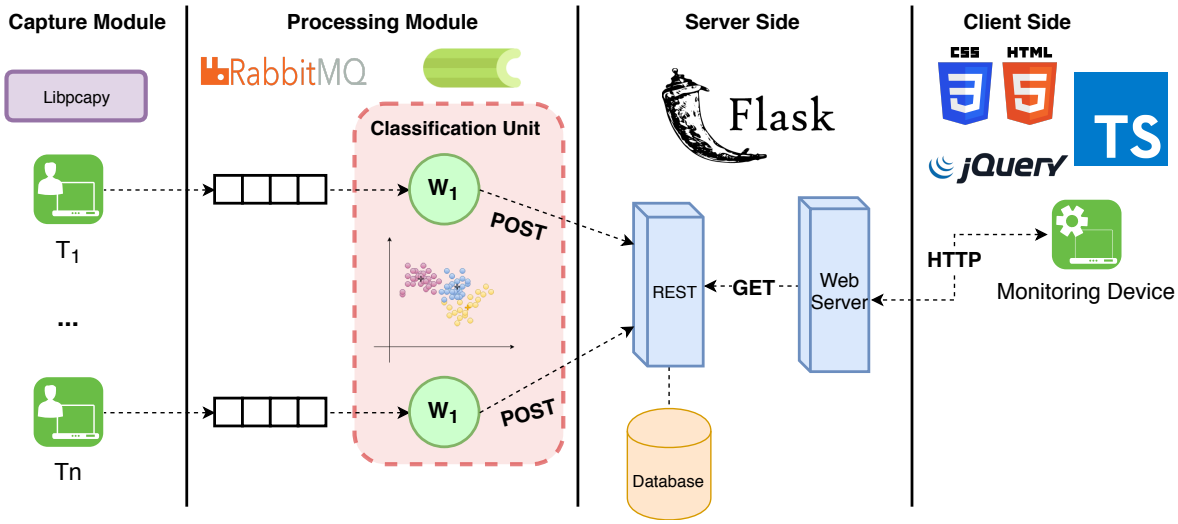


Figure 3.14: Technologies used in each component of the system.

The remainder of this chapter presents each part in detail, describing the purpose of each component, explaining why each framework was chosen to develop the component and how they interact with one another.

3.3.1.1 Capture module

The first step is to decide how the packet data will be captured. As discussed throughout the document (see Section 4.1.1), a software-based approach was taken to acquire the data. In the case of live capture, it is not efficient to save the captured data in files, as it makes the data processing slow and unpractical. Because of this, one has to deal with packets as an *object stream*. The ideal solution should be code based (to be easily connected to other components of the system), fast and should minimize packet loss, since it could potentially alter the prediction result.

Several Python libraries provide processing of live captures of network traffic, such as Pyshark[86], Impacket[87] and Scapy[88]. However, none of those met all the requirements above. For instance, Scapy is useful for creating and injecting packets into a network, but its *sniffing* capabilities are limited and not well documented. Pyshark is extremely easy to use, but achieving complex tasks is difficult, as its documentation is scarce. Plus, since it is written in Python, it is slower than libraries writer in C such as Impacket. Impacket is the fastest capture library for Python, but it is hard to use, as there are few examples and the documentation is limited.

For these reasons, it was decided to create our own library that met the requirements exposed

above. This library is called **Libpcapy**[89]. It is a Python library that serves as a wrapper for Libpcap[90], an interface written in C for user-level packet captures developed by The Tcpdump Group[90]. It achieves this by using **ctypes** [89], a Python library that allows calling functions of shared libraries, providing compatibility with C data types. With this solution, the performance issues are solved, as the functions from the *Libpcap* library are called directly, which means that the program executes C code, which is much faster than pure Python code.

To use *ctypes* to interact with C shared libraries, it is necessary to map the data structures and constants to Python. This is done by creating Python classes to mirror those structures. For instance, Listing 3.1 is the code for the packet header data structure, developed by The Tcpdump Group.

```
struct pcap_pkthdr {
        struct timeval ts;      /* time stamp */
        bpf_u_int32 caplen;     /* length of portion present */
        bpf_u_int32 len;        /* length this packet (off wire) */
};
```

Listing 3.1: C data structure for packet headers.

Listing 3.2 shows the developed code to map the *pcap_pkthdr* class to Python.

```
class pcap_pkthdr(ctypes.Structure):
    _fields_ = [('ts', timeval),
                ('caplen', ctypes.c_uint),
                ('len', ctypes.c_uint)]
```

Listing 3.2: Python class for mirroring *pcap_pkthdr*.

In order to use data structures from *libpcap*, they had to be mapped to Python similarly to the above representation. Because of time constraints, not all structures and constants were mapped. Consequently, as of now, *Libpcapy* does not support all *libpcap* features, such as file capture processing. Since the focus was primarily on live capture settings, all the necessary structures and functions were mapped to make that feature available.

Additionally to data structures, functions also had to be mapped to Python. One issue that occurred during development was during the *pcap_loop* function. This function processes packets from a live capture until an ending condition occurs, typically until *cnt* packages are processed. However, if the *cnt* value is set to -1, the capture can run infinitely. A stopping condition is, for example, a keyboard interrupt. However, since the C loop is running, the interrupt will not have any effect, since the C code does not provide SIGINT handling, which causes the program to run forever. This is because the Global Interpreter Lock (GIL) is released during *cytpes* calls. For this reason, the solution was to run the loop on a background thread, allowing the main thread (the Python thread) to handle interrupts.

Libpcapy also has features such as packet filtering and device scanning.

After establishing what the developed library can do, the question is what is done to the

captured packets. Firstly, it is important to note that the monitored terminals will most likely be quite active, with several programs and background tasks being executed simultaneously. Since packet loss can affect the final result (the prediction), it is crucial that the machines are not overloaded with processing tasks. For this reason, it was decided to minimize the amount of tasks to be done by the capture unit. As a result, the capture module is simply in charge of capturing the packets, retrieving the relevant information, and send it to the **processing module**, which should ideally be located in another machine.

The relevant information to be extracted from the packets is the **timestamp** of the packet, which is in the packet header, the **source** and **destination Internet Protocol (IP) addresses**, and the **length** of the packet, which are contained in the IP header. These are the only attributes that are needed from the packets to create a sample. The **timestamp** is needed to divide the packets into their respective *time buckets*, the **IP addresses** are needed for determining if it is an outgoing or an incoming packet, and the **length** is necessary to calculate the statistics that will be used to create a sample.

With these attributes, a **Pkt Info** object is created (corresponding to a unique packet) and will be sent to the **processing module**. Since all the information that will be traded between the **capture** and **processing** modules is the **Pkt Info** object, which can be serialized, the model that fit the best with the problem of sending information from a component to another is the *producer* and *consumer* model.

Section 3.3.1.2 presents with more detail what message broker was chosen to serve this purpose and describes the rest of the processing module.

3.3.1.2 Processing module

As mentioned in Section 3.3.1, the **processing module** is in charge of taking the packet info from the message queue that is shared with the **capture module**, make a prediction of the created sample and send the result to the server. This process is complex and involves several parts, so each one will be described in detail.

Firstly, the message broker that was chosen was *RabbitMQ*[91]. *RabbitMQ* is a lightweight and highly-scalable message broker that uses the Advanced Message Queueing Protocol (AMQP)[92]. AMQP is a network protocol, so all the *producers* and *consumers* can be executed in different machines.

So, since there are messages (in this case, *Pkt Info* objects) in the queue, a *consumer* is needed to process them. However, there are some requirements to take into consideration before explaining how the *consumer*(or *worker*) is implemented.

For instance, there may be some cases, mainly when the sample size is low and the number of packets to process is large, where the rate of packets being produced is larger than the rate of the packets being consumed. For example, assuming that each sample is 10 seconds of traffic, the *worker* should take less than 10 seconds to process the packets, create a sample, and make a prediction. If it takes longer than that, the producer will be flooding the queue

with more messages while the *worker* still has not processed all messages from the previous sample, potentially propagating the overall lag of the system, which can become significant on successive delays.

For this reason, it was decided that each sample should have a dedicated process assigned to it. This way, if one process encounters a delay, the subsequent samples will not get delayed, as there is a unique process that will handle them. Figure 3.15 illustrates the behaviour of the processes when a delay occurs, in the case when a sample is 60 seconds long.

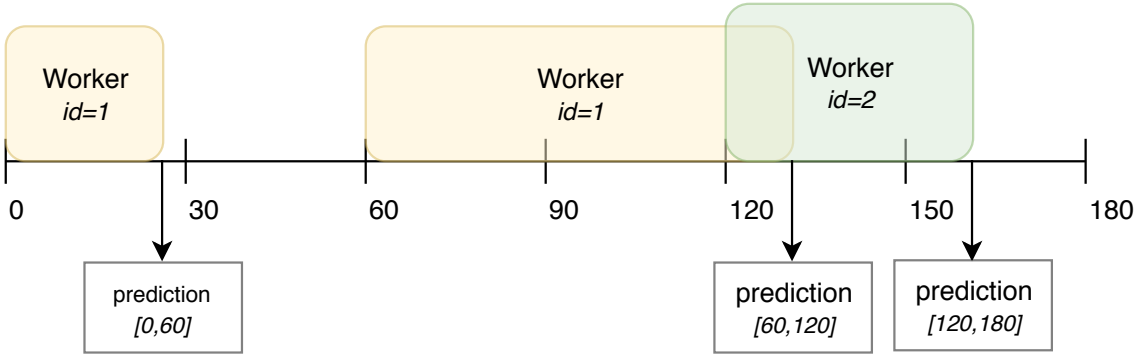


Figure 3.15: Time diagram describing the processing of the messages when a delay occurs.

In Figure 3.15, a delay occurs when *Worker 1* is processing messages during the [60, 120] segment. If *Worker 1* managed to finish the processing before the 120 mark, there would not be two simultaneously active *workers*. Since this happened, *Worker 2* is executed to process the incoming messages. This solves the delay problem presented previously.

However, the delays are unpredictable, so there is no way of knowing how many processes should be executed to deal with these cases. Plus, most of the processes would only be useful during the processing of the packet info. The remainder of their execution time they would be blocked. Although this generally does not consume CPU resources, it may add some overhead to the kernel process management system. This particular case, allied to the fact that there is no method of knowing how much processes will be enough to cover delays, are the main reasons why the processing of the packets is so complex.

There is, however, an elegant way of dealing with these problems - a **task scheduler**. The most famous example is the *cron daemon*[93] on Linux, which adds crontab tasks to a system that will execute them automatically. It is often used for tasks such as sending emails, make data backups or system maintenance. Figure 3.16 shows how this solution would apply to the current problem.

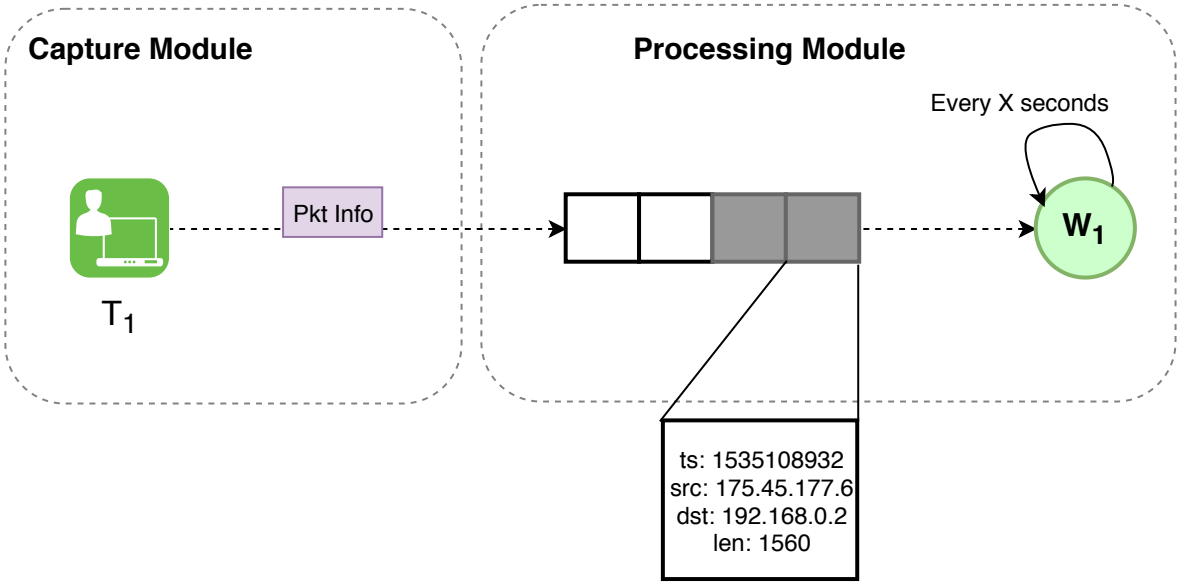


Figure 3.16: Interaction of the **Capture** and **Processing** modules in detail, with scheduled processing.

In this dissertation, the chosen scheduler was **celery beat**[94]. **Celery**[95] is an asynchronous task queue based on distributed message passing. It uses RabbitMQ and Redis[96] as a message broker and database, respectively. However, **Celery** was chosen because of a particular feature called **celery beat**.

Celery beat is a scheduler that allows processes to start at regular intervals to undertake certain tasks that are processed asynchronously. There are several reasons why we decided to use **celery beat** and not a simple crontabtab job.

- **Integration** - the tasks for **celery beat** are written in Python, which facilitates the process of integrating it with the whole system.
- **Portability** - **celery beat** can be executed in any system that supports a Python environment.
- **Privilege** - **celery beat** does not need root access to be executed in a machine, unlike most crontab tasks.
- **Adaptability** - since **Celery** is written to be used along with **RabbitMQ**, it is the solution that fit the best with the system.

Having decided what scheduler to use, the next step was to create the tasks for the workers. Since the objective is to get **Pkt Info** objects from the message queue that correspond to a certain period of time (the sample size), **celery beat** was configured to launch a worker every *sample size* seconds. For instance, in the case where the developed ML models had better accuracy, the sample size was 60 seconds (see Chapter 4). In this case, a task would be scheduled to run every 60 seconds. This task is retrieving all the messages in the *RabbitMQ* queue, create a sample from the data, make a prediction on the sample and send the result to the *Server side*. Figure 3.17 shows the developed algorithm used in the *worker* tasks.

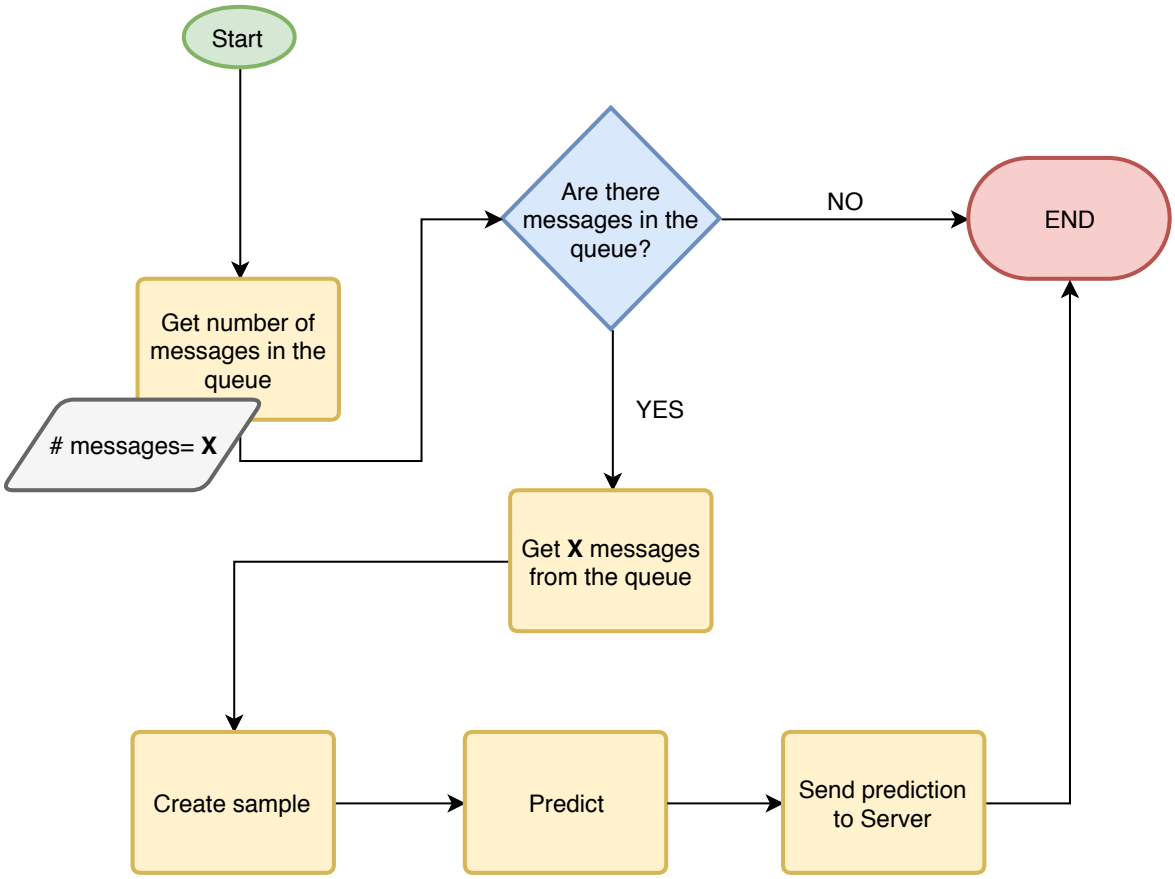


Figure 3.17: *Worker* task algorithm.

It is important to note that to process a sample and make a prediction, certain objects are necessary. These objects are created during the creation of the ML model and then saved to be used by the live framework, as illustrated in Figure 3.18.

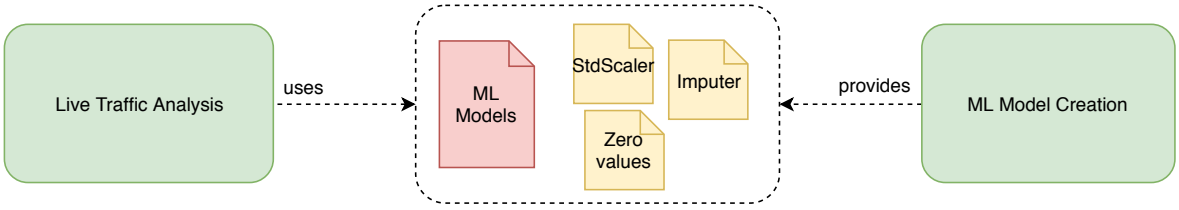


Figure 3.18: Framework overview with shared objects.

Firstly, the most important one is the ML model itself. Secondly, other objects such as the StandardScaler, which was used to scale the training data, as well as the *Imputer* object used to fill any *null* values that may occur (see Section 3.2.4). Finally, the values corresponding to silence, as described in Section 3.2.3.1.

After making a prediction, the *worker* needs to send it to the server. This is done through an HTTP POST request to a REST Application Programming Interface (API), which will be described in more detail in Section 3.3.1.3.

3.3.1.3 Server Side

The server side consists of three parts: the REST server, the web server and the database. The REST server serves as the gateway to the database. Predictions made by the *workers*, as well as other metrics such as the *number of uploaded bytes per minute* are sent to it through POST requests and saved in the database. These values are then accessed by the web server, which makes requests to fulfill the requests from the client side.

For instance, the last prediction of a given terminal can be accessed by making an HTTP GET request to the following endpoint:

```
http://server_address.com/api/predictions/<terminal_id>
```

The chosen technology to develop the server side was **Flask**, a web application framework that can be used easily to create a REST API and that also has the ability to serve static content such as HyperText Markup Language (HTML) pages.

Because of time constraints, the database was not developed, although we suggest how to approach the development in Chapter 5.

3.3.1.4 Client Side

The client side consists of a simple page to visualize the network status and each machine's application usage report. It uses **Typescript** along with **jQuery** as a client-side language. **Typescript** is a superset of the Javascript language and was chosen because it adds static typing to Javascript, making its code more readable, easy to maintain and support, and less prone to errors since it is a compiled language.

CHAPTER 4

Experimental setup, results and analysis

This chapter describes the conducted experiment to test the methods proposed throughout this document. It starts by describing the built dataset and how it was acquired, as well as the method for labeling the samples that compose the dataset. It concludes with the result presentation, the evaluation of the classifiers and their analysis, giving possible explanations for the obtained results.

4.1 Network traffic acquisition

As stated in Section 3.2.2, the collected data should be as broader as possible, i.e., it should consider different network environments and conditions, as well as different users. For instance, even in services such as video streaming, where most of the time the user is not actively interacting with the service, human behavior can influence the upload/download of packets. Actions such as pausing, skipping to another point of the video or opening several browser tabs at once can have a significant influence on how packets flow in the network.

The data was captured in several different settings, such as home networks and the campus network. These connections also had different bandwidth capacity, in order to have the widest variety of captured data. The data was also acquired across six months, at different periods of the day (morning, afternoon and late night), at different days of the week and using different capture devices. Finally, we asked several people to capture network traffic when using the tested applications in order to encompass multiple human behaviors.

4.1.1 Capture devices and configuration

As stated in Section 2.2, it is usually more advisable to capture data using specialized equipment, such as a TAP or the port mirroring capabilities some switches have. This is

especially true in scenarios where the loss of a packet or inconsistencies between the packets' arrival times can have a significant impact on the end result.

That being said, in this work we decided to use a software-based approach since it makes the post-sniffing process simpler. Additionally, the amount of data collected when building a profile for each application is not usually enough to cause significant packet loss. However, this method may not be optimal in a context where the quickness of results is a priority, as discussed in Section 3.3.

All of the data was always captured on the endpoint device that was sending/receiving the data. The machines ran Wireshark in *promiscuous mode*, while the application services ran in the background using browsers like Mozilla Firefox and Google Chrome[1]. Both cabled and wireless connections were tested, so the interface in which to capture was chosen accordingly. No filters were applied when capturing.

Table 4.1 describes the machines used for network data acquisition. The reason most of the stations used virtual machine (VM) for capturing the data was to sanitize the capturing environment, so that there was a smaller chance of programs running in the background interfering with the captures and ultimately change the *fingerprint* of the application that was being captured at the time. Using a VM may introduce some delays in the packet flow[97], but they should be negligible.

Machine ID	**Description**	**Type of connection (wired or wireless)**	**Browsers (Chrome or Firefox)**	**Setting (home network or campus)**
Lab1	Laptop running Fedora 27 VM	Both	Both	Both
Lab2	Laptop running Windows 10	Wireless	Chrome	Home Network
Lab3	Desktop running Ubuntu 18.04 VM(low bandwidth)	Wired	Both	Home network

Table 4.1: Used machines for capturing network data.

The station Lab3 has the particularity of having very low bandwidth available (5 Mbps of download and 1Mbps of upload). This was done by configuring the virtualization software to allocate limited bandwidth for the VM. The objective is to cover situations in which a terminal with poor quality internet access tries to access the tested application services. For example, as mentioned in Section 3.2.3.1, most popular video streaming applications can detect if a connection has low bandwidth and adjust the quality of the video to the connection, which may change some characteristics of the packet flow.

Figure 4.1 shows the number of packets each machine captured over the course of this project. The total amount of captured packets was 15,700,377 packets. These packets include video applications, non-video applications and network noise from background applications as well. The protocol distribution is presented in Figure 4.4.

[1]Except in the case of a particular application which uses its own desktop interface

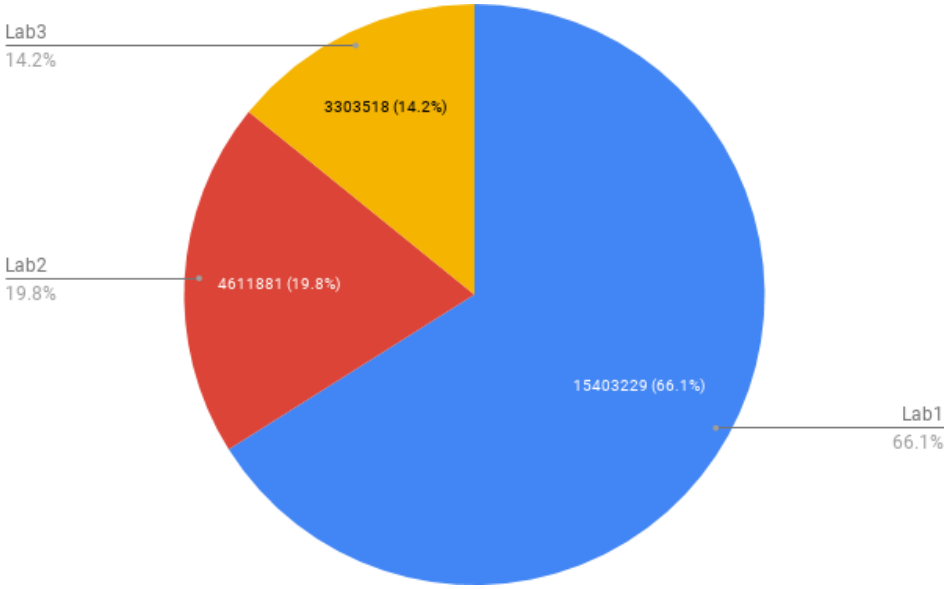


Figure 4.1: Captured packets by each machine.

4.1.2 Tunneled traffic

Since the objective of this work is to identify applications over protected channels, there was a need to setup network environments to acquire traffic to test these particular cases. The chosen protocols to test were SSH and OpenVPN, since they are relatively easy to set up and, because of that, are two of the most popular tunneling protocols.

4.1.2.1 SSH tunnel with dynamic port forwarding

A simple method of redirecting traffic, particularly HTTP traffic, to a protected channel is to use a SOCKS proxy. SOCKS is an internet protocol which redirects packets between a client and a server through a proxy server.

Figure 4.2 depicts a setup of an HTTP over SSH tunnel using SOCKS.

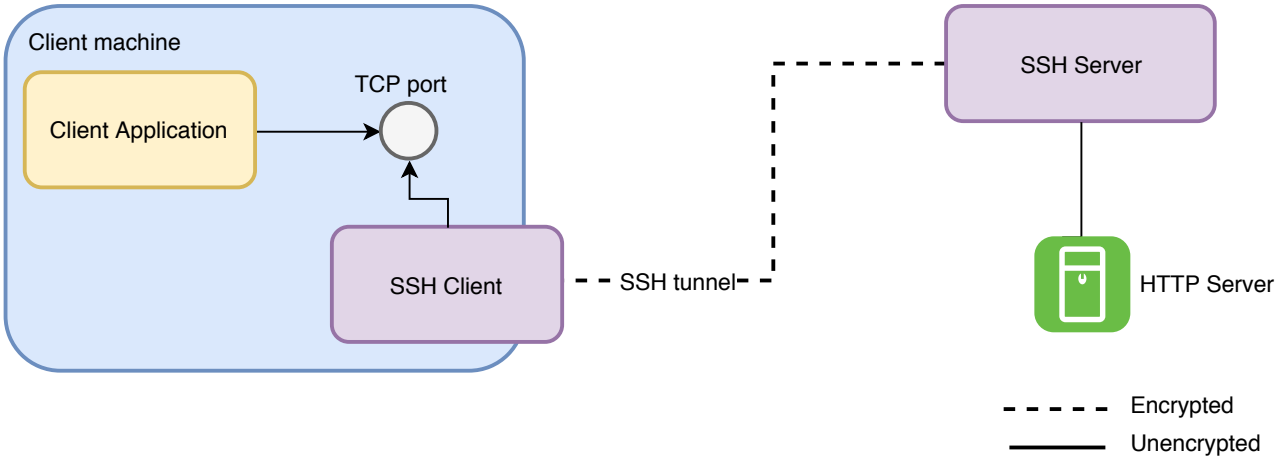


Figure 4.2: Diagram of an SSH tunnel using a SOCKS proxy.

The user application (e.g. browser) forwards its traffic a port opened by the SOCKS, where the tunnel is. The traffic is then forwarded to the SSH tunnel and, on the server, it is forwarded to the internet. A SOCKS proxy has to be configured on each application on the client, unlike a VPN, where all traffic is forwarded to the tunnel. As stated previously, the used applications were Google Chrome and Mozilla Firefox. The SSH server runs on a Raspberry Pi 3B[98].

4.1.2.2 OpenVPN tunnel

Another common method of encrypting traffic is to use OpenVPN. OpenVPN is a popular choice, as it is easy to set up and, unlike an SSH tunnel, there is not a need to configure each application to forward the traffic to a specific port.

Figure 4.3 shows, in a simplified way, how an OpenVPN tunnel works.

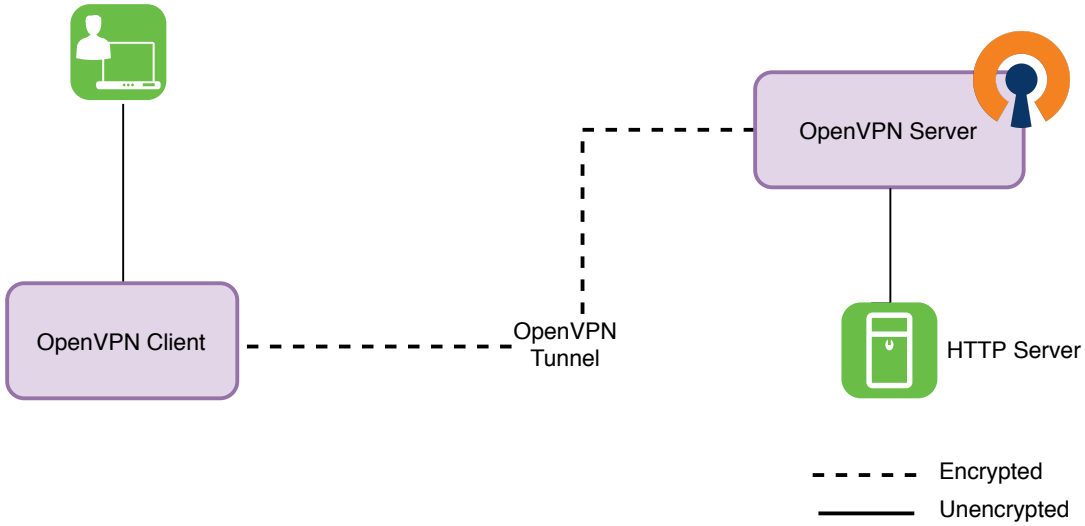


Figure 4.3: Diagram of a OpenVPN tunnel.

All network requests are forwarded to the OpenVPN Client, which can be located on the client terminal itself or outside, acting as a gateway. The OpenVPN client then sends the requests to the OpenVPN Server, through an encrypted VPN tunnel. The OpenVPN server decrypts the traffic and sends the request to the destination server, which responds to the OpenVPN Server.

The OpenVPN server was set up in a Raspberry Pi 3B. The server was setup using PiVPN[99], and the chosen parameters for the tunnel were the default values, such as using UDP as a transport protocol and having a 2048-bit Rivest-Shamir-Adleman (RSA) key.

4.1.3 Labeling

As stated in Section 2.3.2, DPI is one of the most common methods of obtaining the ground truth for network captures. However, this method is computationally expensive and does not guarantee 100% accuracy. For this reason, allied to the fact that all captures were conducted

by ourselves, all of our data was labeled manually, according to the application that was being used between certain timestamps.

4.2 Dataset composition

Overall, a total of 15,700,377 packets were captured and the dataset comprises of about 26,89GB of data, both upload and download. The total time of capture was 64,252 seconds.

Figure 4.4 provides a view of the composition of the dataset separated by the protocol.

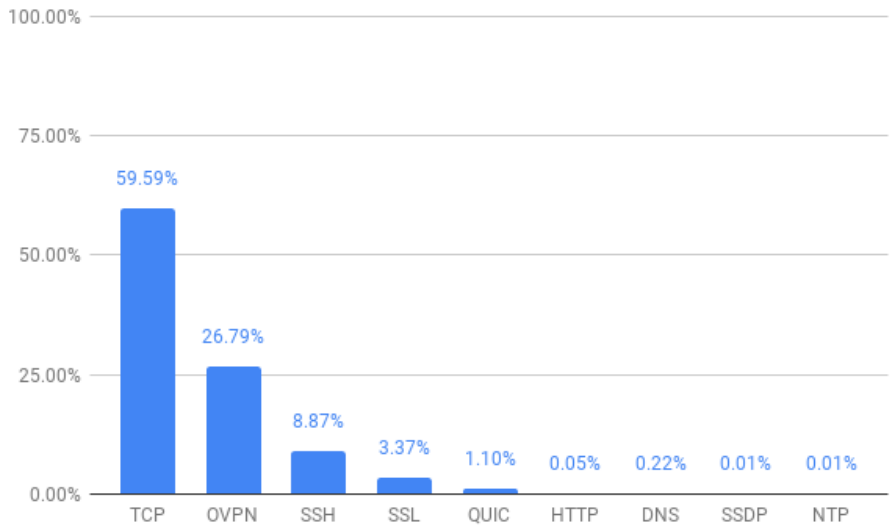


Figure 4.4: Distribution of the protocols contained in the dataset.

4.2.1 Considered classes

As stated previously in Chapter 3, two ML models were developed - one for determining if a service was a video application or a non-video application, and the other for identifying from which specific video application a sample is. Together, these two ML models form a **two-level identification model**.

The first level consists of identifying traffic by classifying it into two large groups: **video** and **non-video**. The **non-video** applications included news websites, forums, social networks, file sharing and online games. Some of these services contain video traffic, but it is not the main focus of the service. The applications chosen to be part of the **video** category will be presented in detail later in this section.

The second level consists of identifying what specific video application was being accessed when the capture was made. A group of four popular video streaming applications was chosen to be a part of this experiment. Those applications are *Acestream*, *Youtube*, *Netflix* and *Twitch*.

4.2.1.1 Acestream

Acestream[100], formerly known as *TorrentStream*, is a P2P multimedia streaming protocol. It is particularly used for broadcasting sports live streams and it is based on the BitTorrent protocol[101].

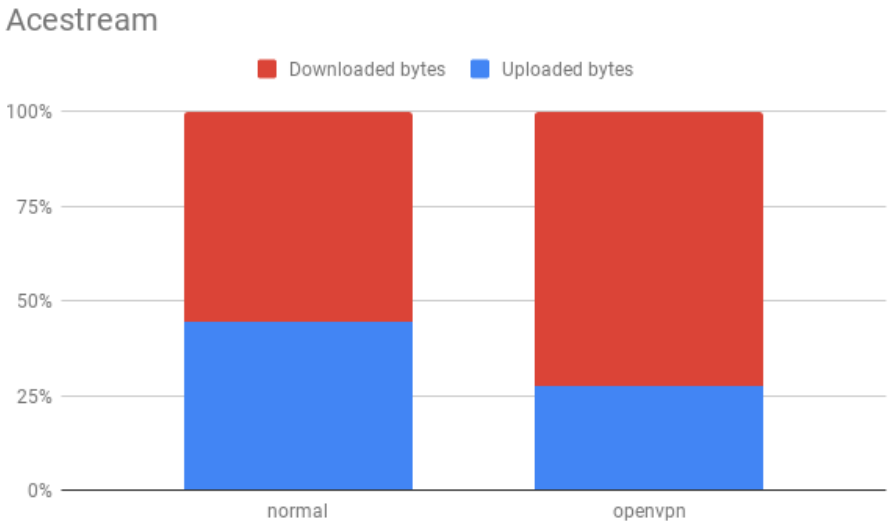


Figure 4.5: Uploaded and downloaded bytes of Acestream across all network settings

As it can be observed in Figure 4.5, the number of downloaded bytes is only slightly larger than the number of uploaded bytes. This behavior is expected, since this is a P2P application, in which every user, in addition to being a client, acts as a server for other peers.

It was not possible to capture traffic using the setup described in Section 4.1.2.1, as the application did not offer an option to redirect the traffic to a specific port.

4.2.1.2 Netflix

Netflix is one of most popular subscription-based TV series and movie streaming services. Unlike *Acestream* and *Twitch*, they do not provide a live stream service.

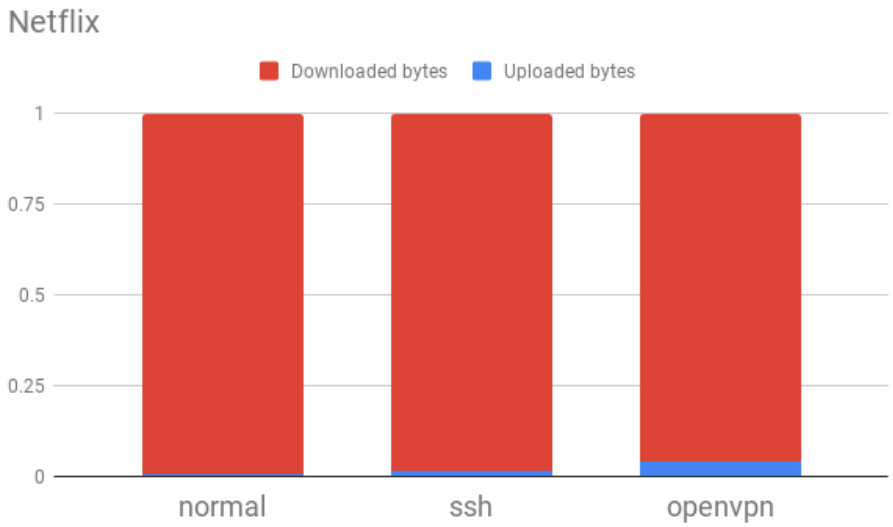


Figure 4.6: Uploaded and downloaded bytes of Netflix across all network settings.

Unlike *Acestream*, *Netflix* is not a P2P application. Consequently, the volume of downloaded data is massive when compared to the volume of uploaded data.

4.2.1.3 Youtube

Youtube is the second most visited website in the world[102]. It is a video-sharing website with millions of daily users. Although it also provides live video streams, this dissertation did not focus on that service, since generally *Youtube* is used for watching non-live videos.

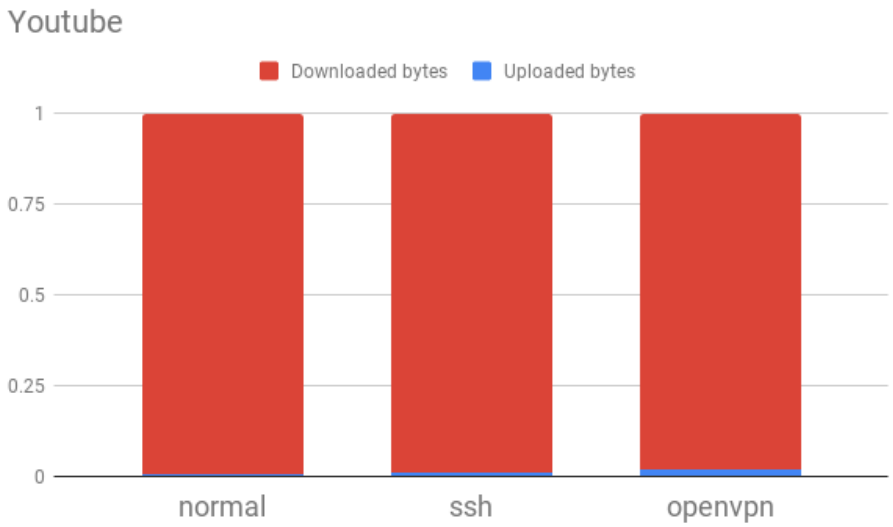


Figure 4.7: Uploaded and downloaded bytes of Youtube across all network settings

Youtube's download/upload byte ratio is similar to *Netflix*'s, which makes sense since they are both very similar video streaming services.

4.2.1.4 Twitch

Twitch is a live streaming video platform that primarily focuses on video game live streaming. It is the most popular website for streaming video games. It also has features such as chat and subscriptions.

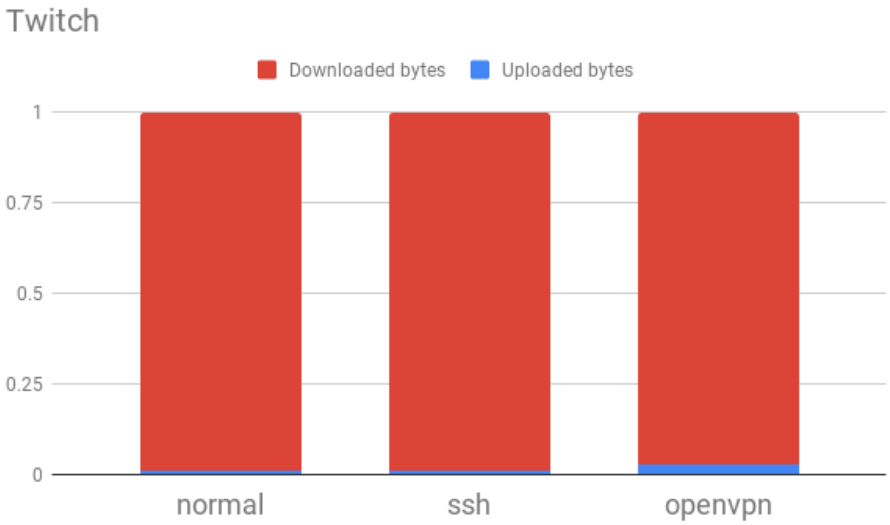


Figure 4.8: Uploaded and downloaded bytes of Twitch across all network settings.

Although *Twitch* is primarily a live streaming service, its ratio is very similar to the one of both *Netflix* and *Youtube*.

4.2.1.5 Overview of all classes

	Normal		SSH		OpenVPN		**Total**	
	Packets	Bytes	Packets	Bytes	Packets	Bytes	Packets	Bytes
Acestream	0.74	1.25	-	-	1.15	2.67	0.78	1.33
Netflix	1.29	118.71	1.18	73.34	1.91	22.62	1.58	50.22
Youtube	1.24	141.42	1.16	106.12	6.21	57.39	3.18	86.83
Twitch	1.27	103.01	1.31	112.02	3.22	37.37	1.88	75.96
Non-video	1.58	19.91	-	-	1.36	9.65	1.57	18.99

Table 4.2: Download/Upload ratio for each class.

By analyzing Table 4.2 we can extract some characteristics that may be useful when choosing the features to use for the ML models. For instance, we can observe that the packet information does not vary that much across the applications (except for Acestream, which is below 1 most of the time). On the other hand, byte ratio can vary substantially. For example, the values are much higher in the video applications than in the non-video applications (except for *Acestream*, for reasons explained in Section 4.2.1.1). This means that outgoing traffic features may have a great influence in categorizing *Acestream*, as it is incredibly different from other

applications, which may lead to a very accurate identification of this class.

Another useful metric for extracting knowledge from packet data is to compare the packet size across all services and network settings. Figure 4.9 shows the average packet size for each application.

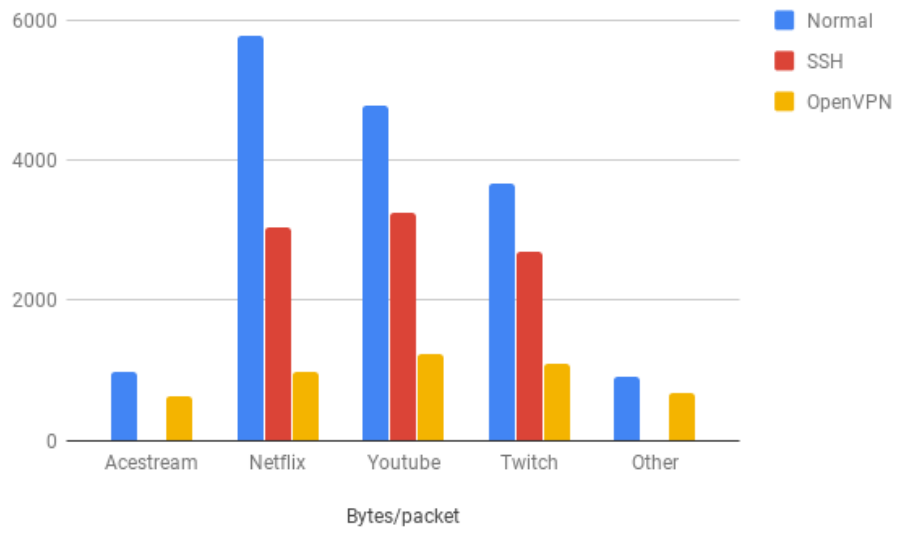


Figure 4.9: Average packet size for each service (bytes)

It is interesting to note that for the tunneled traffic, the average packet size is similar across all applications, which indicates that the network environment has a great influence in the traffic flow. We can observe that non-tunneled traffic has, the largest packet size, followed by SSH traffic and finally OpenVPN.

4.2.2 Processed dataset characteristics

After the acquired data was processed using the methods described in Section 3.2, the result was a single *.csv* in which each row is a sample and each column is a feature. As described in Section 3.2.3, the sample size is the number of *time buckets* in a single sample. The fact that this number is arbitrary can change characteristics of the data after the data preprocessing phase (Section 3.2.3). For this reason, from this point on, the dataset descriptions will be of the processed data with a *time bucket* of 1 second and a sample size of 60 seconds, since those were the values that achieved the best results overall, as can be seen in Section 4.3.

As stated in Section 3.2.3.3, features can be divided in 3 groups - **basic stats**, **silence periods** and **pseudo-periodic components**. However, after preliminary tests, it was noted that the **pseudo-periodic components** did not contribute to a better result. In fact, in most cases, having the scalogram data slightly decreases the accuracy of the ML models.

For this reason, **pseudo-periodic components** were discarded from the feature list, except in one specific situation described in Section 4.3.2. That being said, the number of features used in most of the tests is 52. Table 4.3 shows five random samples from the dataset.

	up_bytes_mean	up_bytes_median	up_bytes_std	...	label
432	-0.176328	-0.176933	0.001749	...	*youtube*
466	-0.17206	-0.174558	0.005345	...	*netflix*
745	-0.170881	-0.175935	0.011627	...	*browsing*
749	-0.162876	-0.170104	0.018725	...	*browsing*
846	-0.173766	-0.174598	0.002649	...	*twitch*

Table 4.3: Random samples from the dataset.

Since the dataset is mostly composed of statistical features, a useful method to estimate the impact of each feature and to understand how they are related is through data visualization. For example , Figure 4.10 depicts the scatter matrix of some important silence period features, taking the whole dataset into account.

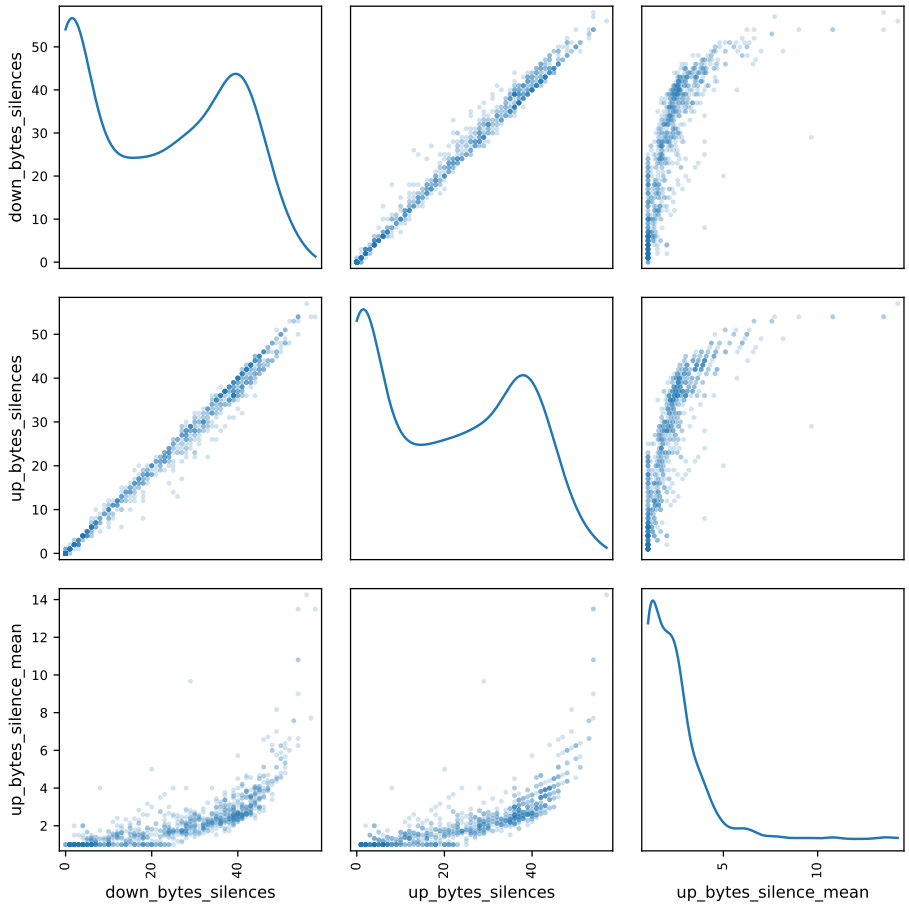


Figure 4.10: Scatter matrix of some silence period related features.

From this matrix, particularly the relation of the number of *silences* in outgoing and incoming traffic, we can conclude that they are both closely related, showing a strong positive linear relation between each other, as one would expect. From that, we can conclude that for the captured applications, the number of silent periods (upload or download) is similar for the same sample.

While Figure 4.10 indicates a positive relation between features, it does not help us conclude if different applications have different time-related characteristics, a fact that would certainly increase the accuracy of the classification process. Figure 4.11 shows a histogram of the number of download and upload silences for video traffic, while Figure 4.12 shows the same statistic for non-video applications.

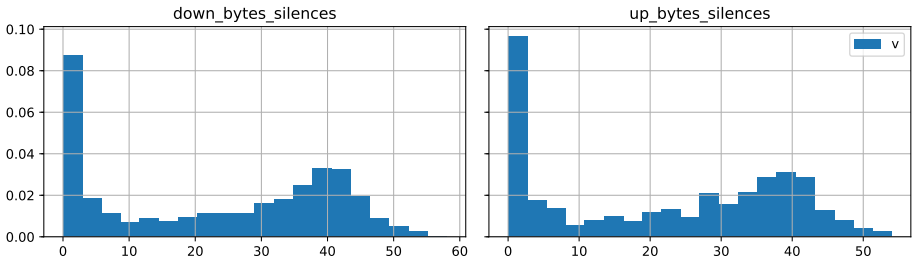


Figure 4.11: Histogram of silence periods for video applications.

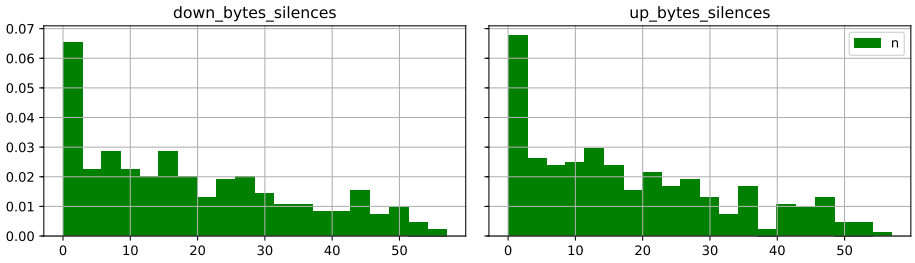


Figure 4.12: Histogram of silence periods for non-video applications.

Upon analysis and comparison of the two histograms, two facts can be concluded. Firstly, as expected, video traffic has more periods of activity than non-video traffic. This is probably due to the fact that for non-video traffic such as news websites or internet forums, a user may spend lots of time reading articles or comments without interacting with the application, which translates itself into more silence periods for non-video applications. Secondly, the silence periods for non-video applications are more diverse than for video applications, which has a slight peak around 35-40 seconds. This may be because user behavior is more volatile than the behavior of video streaming. For instance, when a user is browsing an internet forum, he is more likely to influence the way the traffic flows (by clicking on images and hyperlinks) than he is when watching a video stream. This is because each video application has its own method of transferring the video. For example, with Youtube, most videos have an initial *burst* of incoming traffic, and the rest of the video is downloaded in segments that occur almost periodically. The fact that there is this difference between the two histograms may be as a result of the dynamic explained above.

We have established that time period characteristics can help differentiate between categories of traffic. As for basic statistics such as mean, median, variance, etc (see Section 3.2.3.3), they can also be determinative when classifying traffic. Figure 4.13 shows two parallel coordinates

plot for outgoing packets features and outgoing bytes features, respectively.

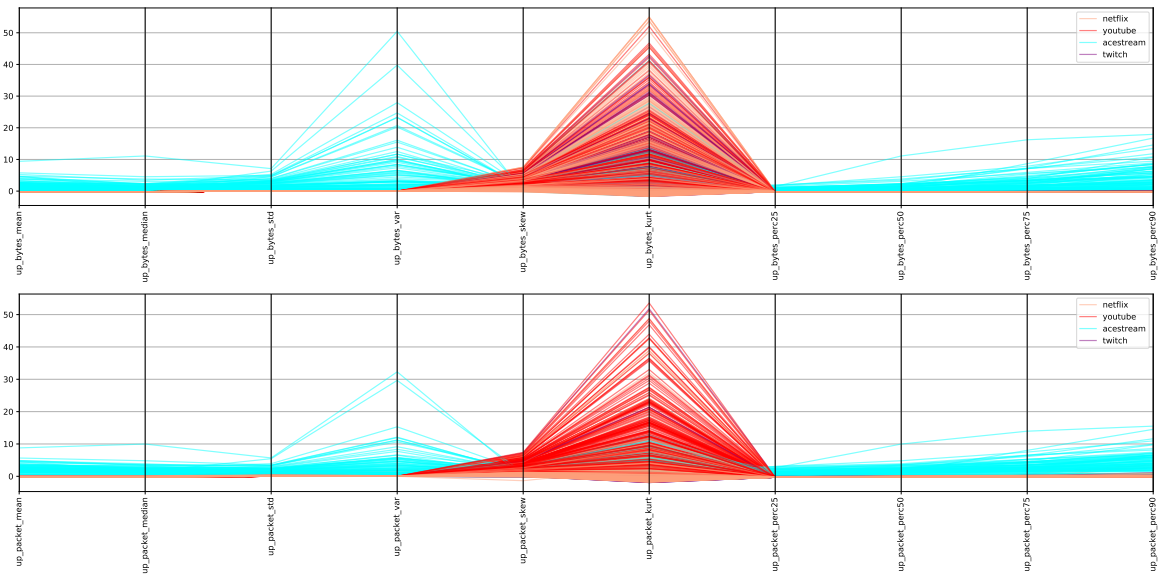


Figure 4.13: Parallel coordinates plot for outgoing packet features and outgoing bytes features.

By analyzing Figure 4.13, the statements in Section 4.2.1.5 regarding the importance of upload features in identifying *Acestream* are strengthened, since its distribution is very different from the other video classes.

Analysing the difference between the two charts more closely, it can be observed that there is a spike in the kurtosis of both upload packet and bytes. However, *Twitch*, for instance, has much higher values in the bytes features than the packets features, which may indicate that the length of the packets varies highly, although infrequently, while maintaining rougly the same number of packets.

Figure 4.14 shows the same chart for outgoing bytes features.

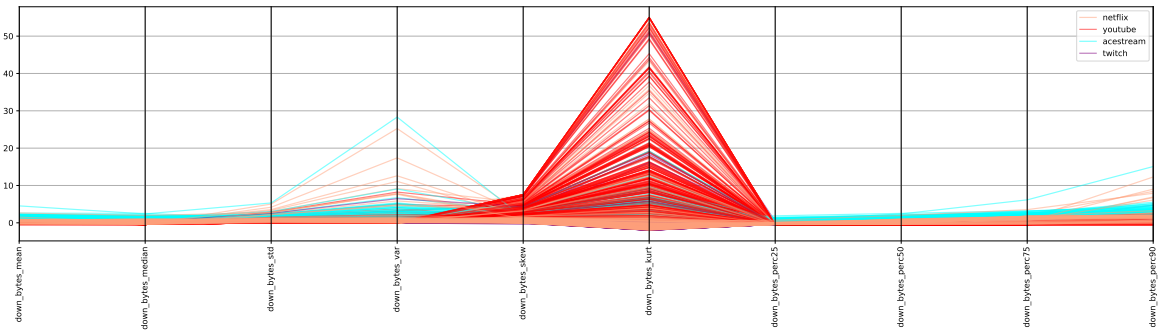


Figure 4.14: Parallel coordinates plot for incoming bytes features.

The incoming bytes features do not highlight differences in the *Acestream* traffic as well as the incoming bytes, but it can still be analyzed to reach useful hints. For instance, if a stream of incoming traffic features a high variance, there is a good chance it is *Netflix traffic*, as well

as if the stream had low kurtosis values.

4.3 Classification and results evaluation

As stated throughout the document, the two-level classification model is composed by the first level - identifying if a given sample is from a video application or not - and the second level - identifying which video application the sample belongs, in case it is a video application.

In addition, since the developed framework is designed to be used at a corporate level, it should also consider situations where tunneled traffic is hard to capture, as it requires setting up the specific network environment and other requirements exposed in Section 3.2.2. Thus, this research also contemplated the situations in which there is no such traffic available for training the ML model. Therefore, there are two more situations to test: training with tunneled traffic (which should achieve better results) and training without tunneled traffic. Figure 4.15 depicts the process of spliting the dataset for the latter case.

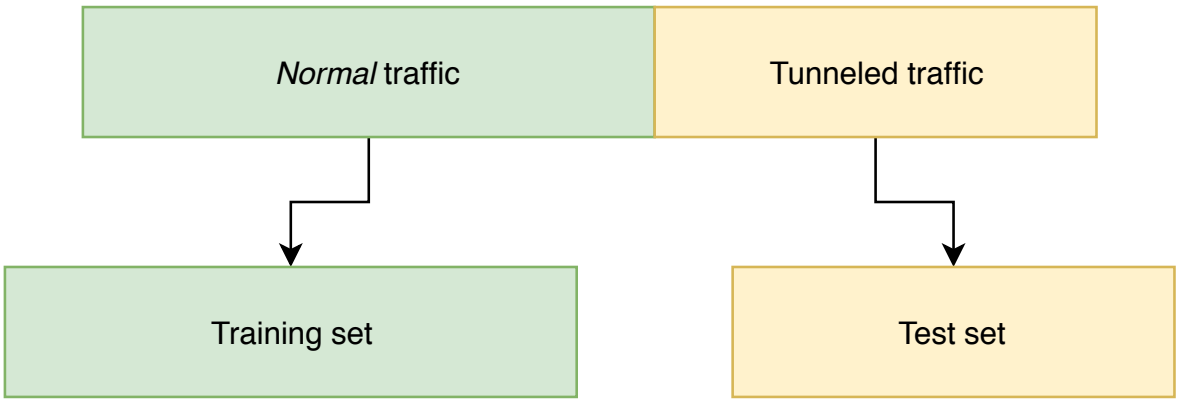


Figure 4.15: Training set and test set for the *training without tunnel traffic* scenario.

As stated in Section 3.2.3.2, both the *time bucket* size and sample size can be chosen when building the dataset. Since the optimal sizes that result in a better classifier were unknown, it was decided to build different datasets from the same packet data with different *time bucket* sizes and sample sizes. The chosen values were 1 second and 0.1 seconds for the *time bucket* size and 10, 30 and 60 units for the sample size.

As stated in Section 3.2.5, the selected ML algorithms to use were **Decision Tree**, **SVM**, **Neural Networks**, **Random Forest**, **Decision Tree + AdaBoost** and **Random Forest + AdaBoost**.

The **One vs. One (OvO)** and **One vs. Rest (OvR)** strategies were also applied to the **Random Forest** algorithm for the multiclass classifier. The purpose of these strategies is to transform a multiclass classification in multiple binary classifications. When using **OvO**, a binary classifier is trained for each pair of labels, which means that for **N** labels there would be $\frac{N(N-1)}{2}$ classifiers. The result is the mode of the result of each binary classifier. The **OvR** strategy consists of training one binary classifier per label, in which it considers the label in

question as positive and the others as negative.

Since some of the chosen algorithms do not usually perform well with unscaled features, particularly SVM and Neural Networks, we did tests with both unscaled and scaled datasets. Plus, we applied PCA reduction (see Section 3.2.3.4) with different values for the number of principal components.

The separation of the dataset in *training set* and *test set* was done with done with the *train_test_split()* function from the *sklearn.model_selection* library. The size of the test set is 20% of the whole dataset. For evaluating the results, in addition to the test set accuracy, we did a 10-fold CV on the training set.
All of the results are presented in chapter 5.

4.3.1 Training with tunneled traffic

In this section, we present the classification results achieved when training the ML model with all types of traffic (tunneled and non-tunneled).

4.3.1.1 Identifying video category

In this case, the classifier aims to identify whether a given observation corresponds to video or not. Although we can observe that almost every combination of parameters obtained good results (over 90% for the most cases), after analyzing the results, we concluded that the best algorithm for almost every case, was **Random Forest with boosting**.

Time bucket (s)	Sample time (s)	Scaled	CV score	Accuracy test set
0.1	1	No	**0.93176±0.00395**	**0.94429±0.00519**
0.1	1	Yes	**0.94484±0.00438**	**0.95093±0.00520**
1	10	No	**0.94150±0.00340**	**0.93531±0.00611**
1	10	Yes	**0.94091±0.00343**	**0.93531±0.00610**
0.1	3	No	**0.92561±0.00392**	**0.93171±0.00588**
0.1	3	Yes	**0.92602±0.00393**	**0.92984±0.00586**
1	30	No	**0.94611±0.00438**	**0.96028±0.00612**
1	30	Yes	**0.94374±0.00398**	**0.94761±0.00554**
0.1	6	No	**0.94374±0.00438**	**0.94761±0.00612**
0.1	6	Yes	**0.94222±0.00338**	**0.94715±0.00483**
1	60	No	**0.94960±0.00429**	**0.95327±0.00612**
1	60	Yes	**0.95080±0.00554**	**0.94393±0.00612**

Table 4.4: Random Forest Classifier with AdaBoost results for identifying video traffic.

All of the results in Table 4.4 have around 94% accuracy, with the best tested case being 96% accuracy in the test set. As expected, the longer the sample, the better is the performance of the classifier, since traffic behaviour characteristics are not as evident in short periods of time. Even though the longest samples achieved better results, they are only slightly better when compared to shorter samples. This may be because since the classes are so broad, only a small amount of traffic is necessary to make an accurate classification. Consequently, in

practice, it may be better to have the ML model predict 60 samples of 1 second than to have it predict one 60 second sample.

It can also be observed that the Random Forest classifier does not significantly change its accuracy when using scaled features. This behaviour is expected, as Random Forests do not usually need normalized data because of their partitioning nature. Consequently, applying any monotonic function to the input data should not affect the performance of the ML model.

Figure 4.16 presents the ROC curve for the case that achieved the best results (1 second bucket size and 60 second sample). For the rest of this section, the presented results will also be for that case.

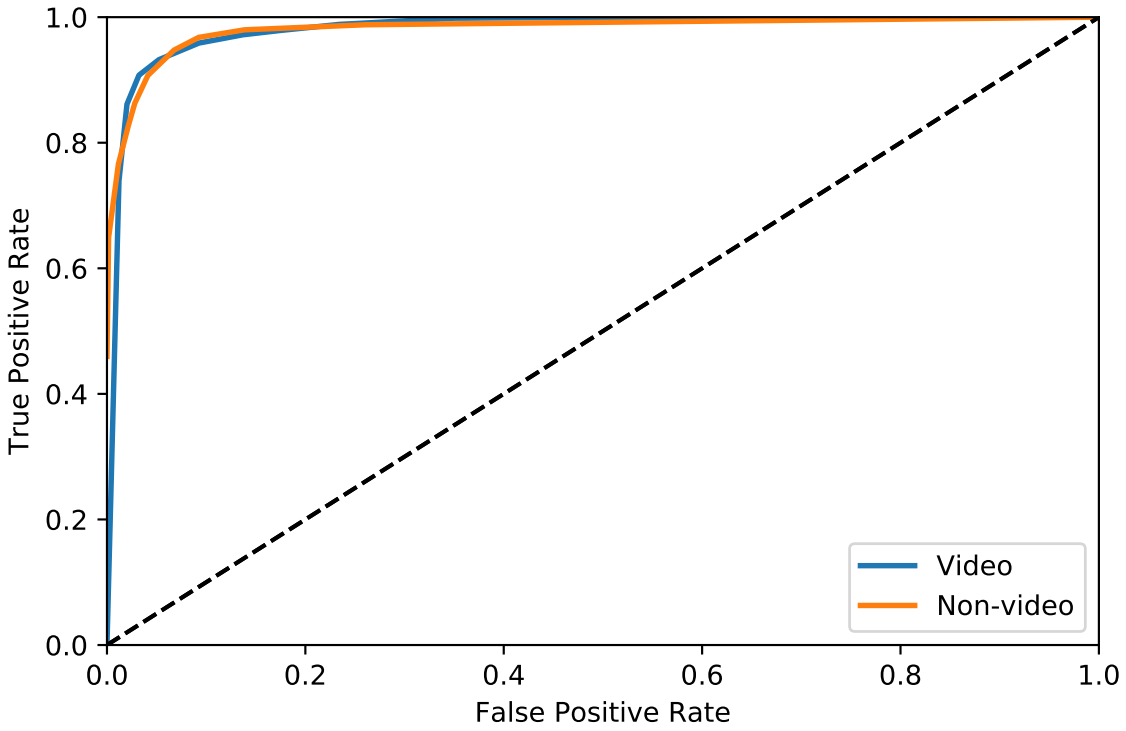


Figure 4.16: ROC curve for the best CV score of Random Forest classifier video traffic identification (best case).

The ROC area under curve score is 98.65% for both classes. By analyzing the ROC curve, we can see that both classes have roughly the same TP and FP rate, meaning that the classifier does not favour one class over another.

Since the two classes are so comprehensive, it was expected that the performance of the ML model would be as high as the results show. Even though the ML model has a good performance, there are instances where it failed to accurately predict the class of an observation This means that the misclassified observations have traits the classifier interpreted as belonging to the incorrect class, which can happen for multiple reasons. For instance, some online games have similar timing characteristics to video streaming services, since they can also have periodic network events. Similarly, a user that performs many actions such as pausing a video

or skipping to another point in the video affects the periodic nature of the video traffic, which can explain why some of the observations were misclassified.

To determine the importance of the group of features that were used, we also conducted tests where the ML model was trained only with a specific subset of features. The results can be observed in table 4.8.

Feature group	**CV score**	**Accuracy test set**
Basic Stats	0.94611±0.00520	0.94392±0.00520
Silence periods	0.77632±0.00960	0.77160±0.00832
Basic Stats + silence periods	0.95080±0.00554	0.94393±0.00612

Table 4.5: Random Forest Classifier with AdaBoost results using different feature groups for video identification (best case).

From Table 4.8, we can conclude that the **basic statistics** are the most important features for this binary classifier. This may be because there are only 12 **silence period** features, which may be a too small number to train the classifier. On the other hand, there are 44 **basic stats**, which is enough for the classifier to have a good performance. When adding the **silence period** features to the **basic stats**, the performance of the model does not have a significant increase. Although **silence period** features do not have a great influence when differentiating video traffic from non-video traffic, they could be useful to distinguish between video applications, since the similarities between classes will be higher.

The algorithms also were trained after applying PCA with a different number of principal components to see how the performance would be affected.

# principal components	**Random Forest + AdaBoost**	**SVM**	**Neural Network**
No PCA	0.95080±0.00521	0.92493±0.00615	0.93206±0.00010
5	0.90040±0.00614	0.92036±0.00609	0.89912±0.00571
10	0.91566±0.00570	0.91556±0.00520	0.92499±0.00521
20	0.91440±0.00577	0.92142±0.00007	0.93203±0.00519
40	0.92385±0.00568	0.92611±0.00520	0.92965±0.00615

Table 4.6: CV accuracy after applying standardization and PCA to the feature set.

As it can be seen from Table 4.6, the performance of the classifier is affected when applying PCA. In the case of Random Forest with boosting, applying PCA hurts the performance. One possible cause is that Random Forests need enough features to find good *splits* (like the branches of a tree) more easily[103]. By decreasing the number of features, the process of finding a *split* becomes harder.

On the other hand, applying PCA increases the accuracy when using algorithms such as SVM and Neural Networks. This does not happen when using only 5 principal components, probably because much information from the original features has been lost. However, when using 10 or more principal components, the performance of the classifiers increases.

4.3.1.2 Identifying specific video application

In this case, the classifier's objective is to predict what video class a given observation belongs to. The considered applications are presented in Section 4.2.1.

The first thing that can be concluded from the results is that most of the classifiers CV score is, to a small degree, higher than the test score, which indicates **overfitting**. A probable reason for this is the small size of the dataset - there may not be enough samples of each class. A common cause of overfitting is also a large number of labels, but that does not seem to be the case here since the higher CV score also happens after applying *feature reduction* to the dataset.

It can also be observed that **Random Forests** were the algorithms that performed better, with the OvR strategy obtaining slightly better results, as some studies in the area would indicate[104].

Nevertheless, the obtained results are good across all parameters combinations. That being said, there is a small increase of accuracy when the sample time is 60 seconds, which is expected since there is more information. Table 4.7 presents the most relevant results when applying feature standardization and PCA.

# principal components	Random Forest OvR	SVM	Neural Network
No PCA	0.94839±0.00808	0.91613±0.00020	0.93548±0.00637
5	0.88387±0.00877	0.91613±0.00733	0.91613±0.00733
10	0.90968±0.00903	0.92258±0.00819	0.90968±0.00559
20	0.92903±0.00879	0.92903±0.00822	0.92258±0.00877
40	0.92258±0.00880	0.90968±0.00734	0.94193±0.00720

Table 4.7: Test accuracy after applying standardization and PCA to the feature set.

These results tie well with the results presented in Table 4.6, as the PCA application decreases the performance of Random Forest, while making SVM and Neural Networks perform better, as long as there are enough features. Unlike the previous case, the improvement of using PCA for those two algorithms is negligible.

Another important detail to take from the results is that the performance did not improve when applying boosting techniques to the Random Forest classifiers, unlike in the first classifier (Section 4.3.2.1). A probable cause for this is that since the first base classifier is already overfitted, applying a boosting technique hurts its performance.

As it can be seen from Table 4.7, the best case achieved 94.8% accuracy in the test set. Figure 4.17 shows the normalized confusion matrix for this case.

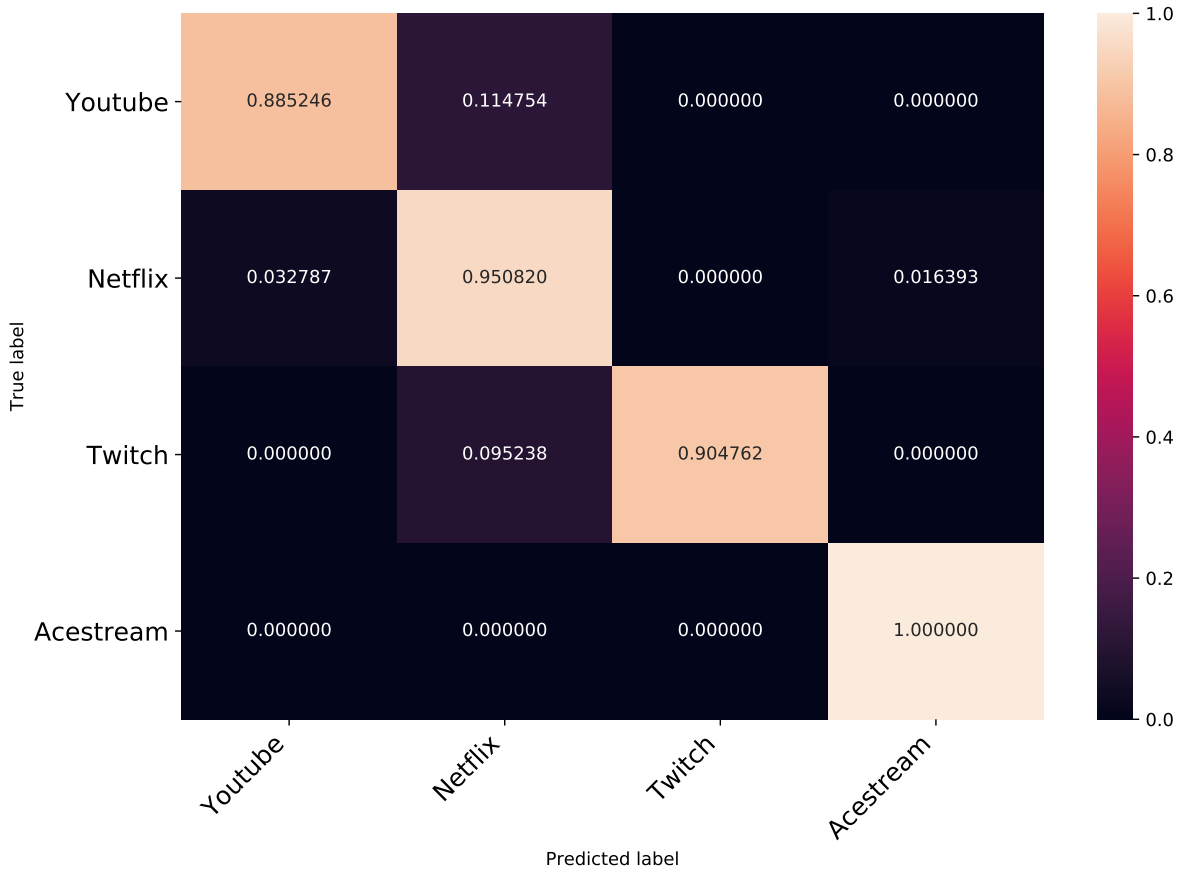


Figure 4.17: Normalized confusion matrix for video application classification (best case).

From Figure 4.17 we can observe that the classifier did not make any mistake when classifying *Acestream* samples. This confirms the presumption exposed in Section 4.2.1.5 where we stated that the P2P nature of this application would make it very distinct from the others. However, for the other classes, the ML model did not perform perfectly.

Figure 4.18 shows the error rates, providing a clearer analysis of the situations where the classifier makes prediction mistakes.

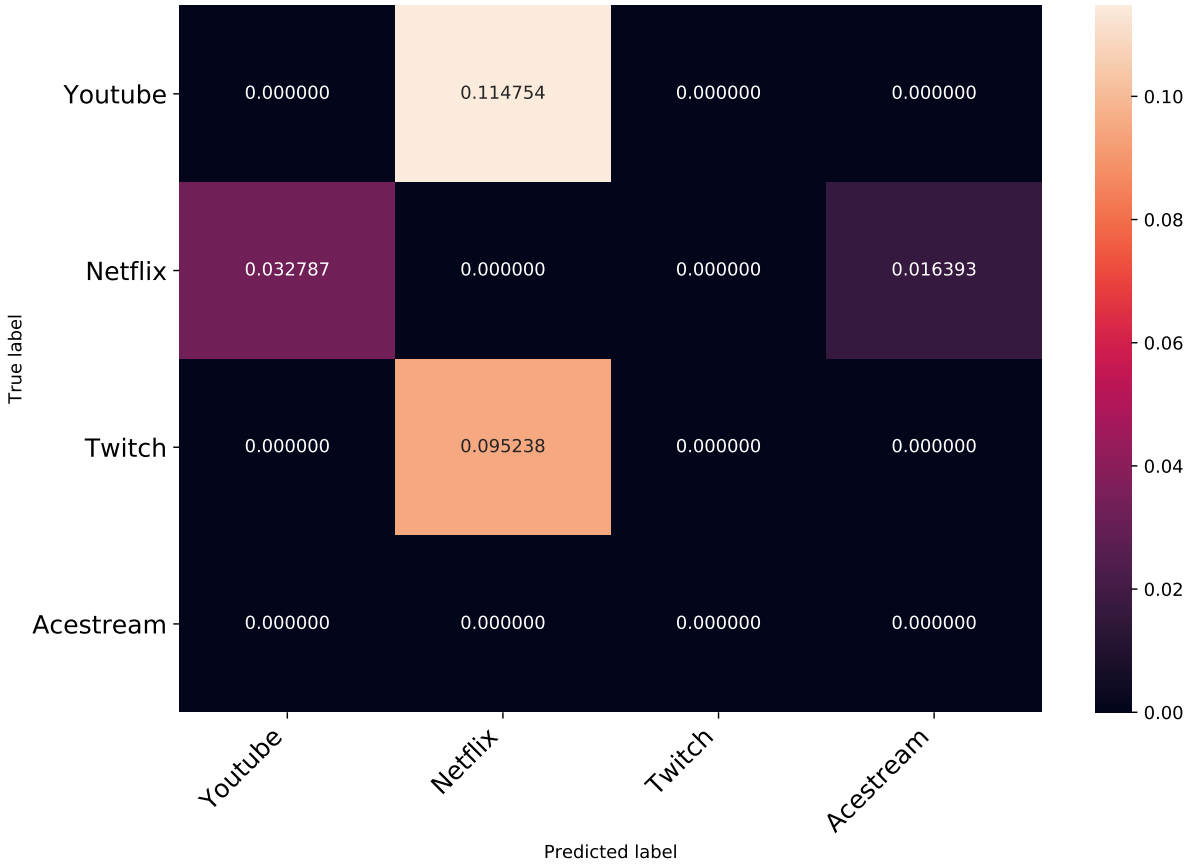


Figure 4.18: Normalized errors of confusion matrix for video application classification (best case).

From the normalized errors matrix, it can be concluded that the classifier struggles to differentiate between *Youtube* and *Netflix*, with over 10% of *Youtube* samples being misclassified as *Netflix*. This also occurs with *Twitch*, although the error rate is not as high. The only circumstance of the classifier misclassifying an application across multiple classes is with *Netflix*, but the amount of mistakes is low when compared to the other classes.

A probable reason for the confusion between *Youtube* and *Netflix* is that those two applications are very similar in concept, since they are both non-live streaming video services.

To estimate the importance of the used features, the same test where the ML model was trained only with a specific subset of features was also applied in this context.

Feature group	**CV score**	**Accuracy test set**
Basic Stats	0.99889±0.00438	0.91832±0.00693
Silence periods	0.89390±0.00735	0.82133±0.00854
Basic Stats + silence periods	0.94483±0.00472	0.94839±0.00808

Table 4.8: Random Forest Classifier with AdaBoost results using different feature groups for video application identification (best case).

By comparing the performance of the classifier using different feature groups, we can conclude that, just like in Section 4.3.1.1, using the **basic statistics** leads to a more accurate

prediction. Although the **silence period** features by themselves do not have a great performance, when used together with the **basic statistics** the classifier's performance noticeably increases. This increase did not happen in the context exposed in Section 4.3.1.1, which indicates that for distinguishing between classes that have more similarities (such as video applications), **silence periods** have an important part.

It is important to note that the CV score when using only **basic statistics** as features was close to 100%, while having a test accuracy of about 92%, which strengthens the notion that the classifier is overfitted because of the feature-sample ratio.

4.3.2 Training without tunneled traffic

In this section, we present the classification results achieved when training the classifiers without tunneled traffi, while testing the ML models only with tunneled traffic. Consequently, the training set comprises of all non-tunneled traffic that was captured, while the test set has all SSH and OpenVPN Traffic.

4.3.2.1 Identifying video category

As expected, the overall results were not as good as before (Section 4.3.1.1). Although we have already established that the longer the sample, the better the classifier will behave, this is more evident in this case, as the results vary dramatically with the sample time (Table 4.9).

Time bucket (s)	Sample time (s)	Accuracy test set
0.1	1	77,9%
0.1	3	73,6%
0.1	6	74,4%
1	10	77,9%
1	30	81,1%
1	60	94,6%

Table 4.9: Random Forest Classifier results for idetifying video traffic (training without tunneled traffic).

As it can be observed in Table 4.9, the classifier that performed the best had **94,6%** accuracy on the test set, which is in line with the overall results from the same experiment with tunneled traffic (Section 4.3.1.1). The fact that the best classifier has roughly the same accuracy in both cases is an indicator that the two categories are broad enough to the point that the underlying protocols that control the packet flow do not affect each group's *fingerprint*. Although this happens when the classes are very comprehensive, it does not happen when they are more specific, as it will be discussed in Section 4.3.2.2.

In this case, CV tests were not conducted, since the training test did not contain tunneled traffic.

4.3.2.2 Identifying specific video application

In the case of identifying a video service without training the classifier with tunneled traffic, the overall accuracy of the ML models was considerably lower when compared to training with tunneled traffic. The best classifier achieved **75,5%** accuracy and the used algorithm was Random Forest with boosting.

This decrease in accuracy was expected, since the patterns generated by a normal traffic flow and a tunneled traffic flow have differences. Although the video applications have characteristics that help classifiers differentiate them from one another, the way that SSH and OpenVPN affects the packet flow still has a significant impact in the classification process.

Furthermore, this was the only case where the **pseudo-periodic components** features helped improve the accuracy of the ML model, even if the accuracy increase was not significant.

Feature group	**Best accuracy test set**
Basic Stats + silence periods	75,5%
Basic Stats + silence periods + pseudo-periodic components	77,9%

Table 4.10: Comparison of the accuracy of the classifier when adding pseudo-periodic components features.

Although there is no apparent reason for this small performance increase, one possible explanation is that the pseudo-periodic components encompass more information about the application itself and are not significantly affected by encountering an observation where the traffic was shaped differentely because of the tunnel. This way, even when it is tested with an observation of a network setup it is not familiar with, the classifier can still identify the application with more success.

4.4 Summary

In this chapter, we presented a proof of concept for the developed framework by conducting an experiment tackling the real-time identification of video applications. The results indicate that it is possible to identify specific video applications with an accuracy of over 94% even when using protected channels. If the ML models are presented with traffic captured in network setups that are unknown for them (Section 4.3.2), the results are not as good, which shows that the influence of the use of VPNs is still substantial.

CHAPTER 5

Conclusions and future work

Many studies and reports show that the increasing use of encryption and the urge to apply encryption methods to ensure secure communications. The leading cause of this phenomenon is the continuous awareness of individual privacy and all the laws and regulations that are established to protect it. Every company with benevolent practices wants to respect the privacy of its workers, but for it is also fundamental that the network of a corporate environment is monitored to ensure its health, which generates a conflict of interests. Although some companies apply policies to what can and cannot be accessed by the workers, knowledgeable users may be capable of circumventing such restrictions with the usage of protected channels (Section 4.1.2).

The work developed in this dissertation aimed to develop a method to classify traffic of different applications in real time, regardless of the network setup. The presented solution achieved great results when compared to state of the art methods, all this in the relatively unexplored field of traffic classification in protected channels. Plus, it showed to be a sensible compromise between user privacy and network management, as it provides a non-intrusive insight into the usage of the network's resources.

Besides the research and development of ML models to classify traffic (Section 3.2), a real-time system for traffic analysis that uses the produced classifiers was also developed. As described in Section 3.3, the architecture of this system was designed to make it scalable and easily deployable in corporate environments. Although the system is designed to be applicable to a company, the concept and implementation could also be adapted and used by ISPs, since a non-intrusive inference of what service clients are using could be useful for traffic shaping purposes.

This work dove further into traffic classification problems, while raising questions and opening perspectives into other approaches and contexts to be used in network application identification, showing promising results. Although many contexts and classification methods were tested in this dissertaion, some improvements can be made in future studies, particularly in the context of protected channels.

Firstly, regarding the actual work conducted in this dissertation, it is necessary to have a larger dataset to train and test the classifiers. Although one could apply a sliding window technique (see Figure 5.1) when creating samples from traffic to get much more observations to work with, they would be less comprehensive than observations created by new traffic.

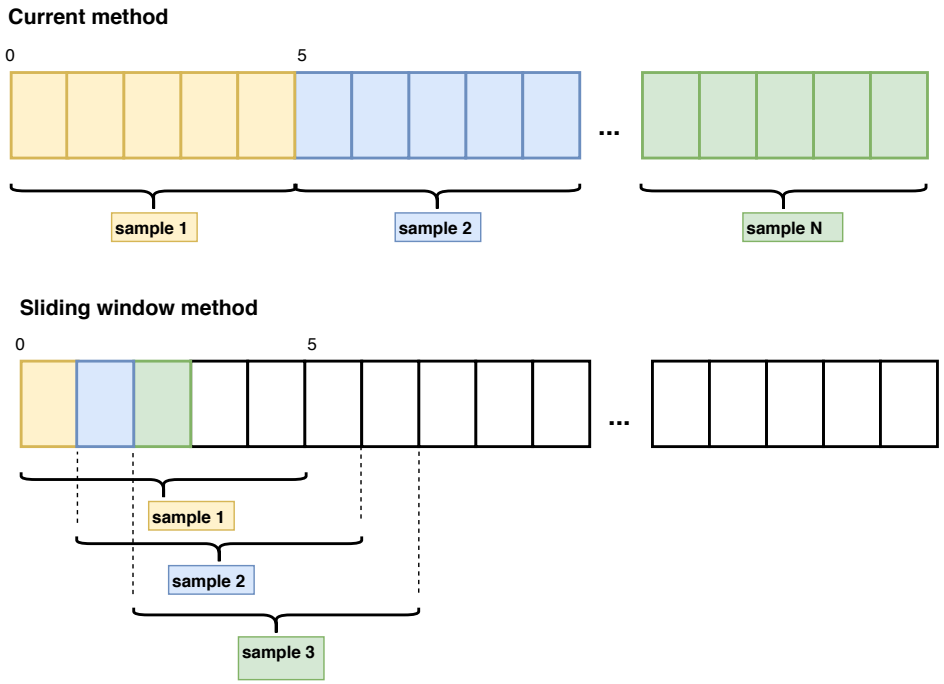


Figure 5.1: Comparison of the sliding window method and the current method for extracting samples from packet captures.

Besides increasing the number of samples, the proposed framework should be tested for more applications. Other services such as online gaming, music streaming and tasks like database access could also be in their own class in order to achieve a more fine-grained analysis of the traffic.

It is also essential to identify the importance of each feature, mainly the reason why pseudo-periodic components did not help the classifier achieve a better performance in most cases. It would also be interesting to find other features that could have valuable information (e.g. variance of the silence periods). It could also be useful to try to identify the number of TCP flows in a connection[39], since it is a valid identifier for web applications.

One aspect of the study that was only superficially addressed was hyperparameter tuning[105]. Although it was used in the cases that achieved the best accuracy using Scikit Learn implementation[106], its application should be studied further in order to achieve classifiers with better performance.

Although many contexts were covered, both concerning the considered protected channels and the incapability of training the ML model with tunneled traffic, there are other situations worth looking further into. For instance, the usage of traffic morphing methods, such as the Tor network[107], which may affect the applications' fingerprint. One possible solution for

this is to identify the occurrence of traffic morphing and compensating by using features that are more robust against those changes. In order to do that, a more thorough study of the features is necessary to determine their importance in specific contexts. Another situation worth studying is when there is more than one application being accessed at a given time. A simple way of dealing with this problem is to have a binary classifier for each class instead of a multiclass model with the services combinations.

Regarding the real-time system itself, the database was not implemented due to time constraints. InfluxDB[108] is a time series database that indexes its entries by using timestamps, which could be a good fit to the problem at hand.

In conclusion, the work developed in this dissertation set the ground for many enhancements and has a lot to offer regarding the contexts it could be applied in. It is proof that a concession between network management and privacy is possible without sacrificing performance.

References

[1] European Parliament & Council, "Regulation (EU) 2016/679 of the European Parliament and of the Council", *Official Journal of the European Union*, 2016. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L:2016:119:FULL%7B%5C&%7Dfrom=EN.

[2] *Google Transparency Report*. [Online]. Available: https://transparencyreport.google.com/https/overview (visited on 03/21/2018).

[3] American Management Association / ePolicy Institute, "Electronic Monitoring & Surveillance Survey", *American Management Association*, pp. 1–11, 2007. [Online]. Available: http://www.plattgroupllc.com/jun08/2007ElectronicMonitoringSurveillanceSurvey.pdf.

[4] *Americans Are Watching Netflix at Work and in the Bathroom - The New York Times*. [Online]. Available: https://www.nytimes.com/2017/11/17/business/media/watch-netflix-at-work.html (visited on 09/11/2018).

[5] *IETF | Internet Engineering Task Force*. [Online]. Available: https://www.ietf.org/ (visited on 09/11/2018).

[6] *OpenSSL: The Open Source toolkit for SSL/TLS | BibSonomy*. [Online]. Available: https://www.bibsonomy.org/bibtex/28796dfa7899f1eeab5e921051d6c3eed/ragibhasan (visited on 09/11/2018).

[7] *Observer Network TAPs*. [Online]. Available: https://www.viavisolutions.com/pt-br/node/58398 (visited on 07/04/2018).

[8] *IANA Port Numbers*. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml (visited on 04/03/2018).

[9] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "Is P2P dying or just hiding?", in *IEEE Global Telecommunications Conference, 2004. GLOBECOM '04.*, vol. 3, IEEE, pp. 1532–1538, ISBN: 0-7803-8794-5. DOI: 10.1109/GLOCOM.2004.1378239. [Online]. Available: http://ieeexplore.ieee.org/document/1378239/.

[10] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, "Transport layer identification of P2P traffic", in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*, New York, New York, USA: ACM Press, 2004, p. 121, ISBN: 1581138210. DOI: 10.1145/1028788.1028804. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1028788.1028804.

[11] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A Survey of Payload-Based Traffic Classification Approaches", *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1135–1156, 2014, ISSN: 1553-877X. DOI: 10.1109/SURV.2013.100613.00161. [Online]. Available: http://ieeexplore.ieee.org/document/6644335/.

[12] C. Shen and L. Huang, "On Detection Accuracy of L7-filter and OpenDPI", in *2012 Third International Conference on Networking and Distributed Computing*, IEEE, Oct. 2012, pp. 119–123, ISBN: 978-1-4673-2858-6. DOI: 10.1109/ICNDC.2012.36. [Online]. Available: http://ieeexplore.ieee.org/document/6386665/.

[13] S. Alcock and R. Nelson, "Libprotoident: Traffic Classification Using Lightweight Packet Inspection", [Online]. Available: https://wand.net.nz/sites/default/files/lpi.pdf.

[14] L. Bernaille and R. Teixeira, "Early Recognition of Encrypted Applications", [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.5232%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[15] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, "Independent comparison of popular DPI tools for traffic classification", *Computer Networks*, vol. 76, pp. 75–89, Jan. 2015, ISSN: 1389-1286. DOI: 10.1016/J.COMNET.2014.11.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128614003909.

[16] R&S Cybersecurity ipoque, *R&S®PACE 2 | Deep Packet Inspection*. [Online]. Available: https://www.ipoque.com/products/dpi-engine-rsrpace-2 (visited on 04/05/2018).

[17] E. Hjelmvik, "The SPID Algorithm Statistical Protocol IDentification", 2008. [Online]. Available: https://www.iis.se/docs/The%7B%5C_%7DSPID%7B%5C_%7DAlgorithm%7B%5C_%7D-%7B%5C_%7DStatistical%7B%5C_%7DProtocol%7B%5C_%7DIDentification.pdf.

[18] V. Paxson, "Empirically derived analytic models of wide-area TCP connections", *IEEE/ACM Transactions on Networking*, vol. 2, no. 4, pp. 316–336, 1994, ISSN: 10636692. DOI: 10.1109/90.330413. [Online]. Available: http://ieeexplore.ieee.org/document/330413/.

[19] C. McCarthy and A. N. Zincir-Heywood, "An investigation on identifying SSL traffic", in *2011 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, IEEE, Apr. 2011, pp. 115–122, ISBN: 978-1-4244-9939-7. DOI: 10.1109/CISDA.2011.5945943. [Online]. Available: http://ieeexplore.ieee.org/document/5945943/.

[20] T. Yildirim and P. Radcliffe, "A Framework for Tunneled Traffic Analysis", [Online]. Available: http://www.icact.org/upload/2010/0169/20100169%7B%5C_%7Dfinalpaper.pdf.

[21] X. Wang and D. J. Parish, "Optimised Multi-stage TCP Traffic Classifier Based on Packet Size Distributions", in *2010 Third International Conference on Communication Theory, Reliability, and Quality of Service*, IEEE, 2010, pp. 98–103, ISBN: 978-1-4244-7273-4. DOI: 10.1109/CTRQ.2010.24. [Online]. Available: http://ieeexplore.ieee.org/document/5532778/.

[22] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting", *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, p. 5, Jan. 2007, ISSN: 01464833. DOI: 10.1145/1198255.1198257. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1198255.1198257.

[23] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, "Transport layer identification of P2P traffic", in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*, New York, New York, USA: ACM Press, 2004, p. 121, ISBN: 1581138210. DOI: 10.1145/1028788.1028804. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1028788.1028804.

[24] P. Wang, X. Guan, and T. Qin, "P2P Traffic Identification Based on the Signatures of Key Packets", in *2009 IEEE 14th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks*, IEEE, Jun. 2009, pp. 1–5, ISBN: 978-1-4244-3532-6. DOI: 10.1109/CAMAD.2009.5161471. [Online]. Available: http://ieeexplore.ieee.org/document/5161471/.

[25] M. Korczynski and A. Duda, "Markov chain fingerprinting to classify encrypted traffic", in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, IEEE, Apr. 2014, pp. 781–789, ISBN: 978-1-4799-3360-0. DOI: 10.1109/INFOCOM.2014.6848005. [Online]. Available: http://ieeexplore.ieee.org/document/6848005/.

[26] D. Lee and N. Brownlee, "A Methodology for Finding Significant Network Hosts", in *32nd IEEE Conference on Local Computer Networks (LCN 2007)*, IEEE, Oct. 2007, pp. 981–988, ISBN: 0-7695-3000-1. DOI: 10.1109/LCN.2007.21. [Online]. Available: http://ieeexplore.ieee.org/document/4367941/.

[27] T. Karagiannis, K. Papagiannaki, M. Faloutsos, T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC", in *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '05*, vol. 35, New York, New York, USA: ACM Press, 2005, p. 229, ISBN: 1595930094. DOI: 10.1145/1080091.1080119. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1080091.1080119.

[28] M. Sahami, S. Dumais, D. Heckerman, E. Horvitz, and G. Building, "A Bayesian Approach to Filtering Junk E-Mail", [Online]. Available: http://robotics.stanford.edu/users/sahami/papers-dir/spam.pdf.

[29] B. Silver and Bernard, "Netman: a learning network traffic controller", in *Proceedings of the third international conference on Industrial and engineering applications of artificial intelligence and expert*

systems - IEA/AIE '90, vol. 2, New York, New York, USA: ACM Press, 1990, pp. 923–931, ISBN: 0897913728. DOI: 10.1145/98894.99101. [Online]. Available: http://portal.acm.org/citation.cfm?doid=98894.99101.

[30] J. F. Rank, "Artificial Intelligence and Intrusion Detection: Current and Future Directions", 1994. [Online]. Available: http://home.eng.iastate.edu/%7B~%7Dguan/course/backup/CprE-592-YG-Fall-2002/paper/intrusion/ai-id.pdf.

[31] A. K. Ghosh, A. Schwartzbard, M. Schatz, and M. I. C. H. Schatz, "Learning Program Behavior Profiles for Intrusion Detection", [Online]. Available: http://www.usenix.org%20www.rstcorp.com.

[32] T. Lane and C. E. Brodley, "Detecting the Abnormal: Machine Learning in Computer Security", 1997. [Online]. Available: http://docs.lib.purdue.edu/ecetr%20http://docs.lib.purdue.edu/ecetr/74.

[33] T. Fawcett, "An introduction to ROC analysis", 2005. DOI: 10.1016/j.patrec.2005.10.010. [Online]. Available: http://people.inf.elte.hu/kiss/11dwhdm/roc.pdf.

[34] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning", *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008, ISSN: 1553-877X. DOI: 10.1109/SURV.2008.080406. [Online]. Available: http://ieeexplore.ieee.org/document/4738466/.

[35] Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools ...* 2017. [Online]. Available: https://books.google.pt/books?hl=pt-PT%7B%5C&%7Dlr=%7B%5C&%7Did=khpYDgAAQBAJ%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PP1%7B%5C&%7Ddq=hands+on+ml+with+scikit+learn+and+tensorflow+%7B%5C&%7Dots=kLCyOOAri2%7B%5C&%7Dsig=nivfwA%7B%5C_%7DHnbvbpS-AoaKXCVLVz5M%7B%5C&%7Dredir%7B%5C_%7Desc=y%7B%5C#%7Dv=onepage%7B%5C&%7Dq=hands%20on%20ml%20with%20scikit%20learn%20and%20tensorflow%7B%5C&%7Df=f.

[36] Li Jun, Z. Shunyi, Lu Yanqing, and Z. Zailong, "Internet Traffic Classification Using Machine Learning", in *2007 Second International Conference on Communications and Networking in China*, IEEE, Aug. 2007, pp. 239–243, ISBN: 978-1-4244-1008-8. DOI: 10.1109/CHINACOM.2007.4469372. [Online]. Available: http://ieeexplore.ieee.org/document/4469372/.

[37] Shijun Huang, Kai Chen, Chao Liu, A. Liang, and Haibing Guan, "A statistical-feature-based approach to internet traffic classification using Machine Learning", in *2009 International Conference on Ultra Modern Telecommunications & Workshops*, IEEE, Oct. 2009, pp. 1–6, ISBN: 978-1-4244-3942-3. DOI: 10.1109/ICUMT.2009.5345539. [Online]. Available: http://ieeexplore.ieee.org/document/5345539/.

[38] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS", in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*, New York, New York, USA: ACM Press, 2004, p. 135, ISBN: 1581138210. DOI: 10.1145/1028788.1028805. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1028788.1028805.

[39] C. V. Wright, F. Monrose, G. M. Masson, and M. Edu, "On Inferring Application Protocol Behaviors in Encrypted Network Traffic", *Journal of Machine Learning Research*, vol. 7, pp. 2745–2769, 2006. [Online]. Available: http://www.jmlr.org/papers/volume7/wright06a/wright06a.pdf.

[40] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943, ISSN: 0007-4985. DOI: 10.1007/BF02478259. [Online]. Available: http://link.springer.com/10.1007/BF02478259.

[41] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian Neural Networks for Internet Traffic Classification", *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 223–239, Jan. 2007, ISSN: 1045-9227. DOI: 10.1109/TNN.2006.883010. [Online]. Available: http://ieeexplore.ieee.org/document/4049810/.

[42] A. K. J. Michael, E. Valla, N. S. Neggatu, and A. W. Moore, *Network traffic classification via neural networks*, 2017. [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-912.html.

[43] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks", in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, IEEE, Jul. 2017, pp. 43–48, ISBN: 978-1-5090-6727-5. DOI: 10.1109/ISI.2017.8004872. [Online]. Available: http://ieeexplore.ieee.org/document/8004872/.

[44] A. Este, F. Gringoli, and L. Salgarelli, "Support Vector Machines for TCP traffic classification", *Computer Networks*, vol. 53, no. 14, pp. 2476–2490, Sep. 2009, ISSN: 1389-1286. DOI: 10.1016/J.COMNET.2009.05.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128609001649.

[45] Y.-x. Yang, R. Wang, Y. Liu, S.-z. Li, and X.-y. Zhou, "Solving P2P Traffic Identification Problems Via Optimized Support Vector Machines", in *2007 IEEE/ACS International Conference on Computer Systems and Applications*, IEEE, May 2007, pp. 165–171, ISBN: 1-4244-1030-4. DOI: 10.1109/AICCSA.2007.370879. [Online]. Available: http://ieeexplore.ieee.org/document/4230954/.

[46] G. C. Cawley and N. L. C. Talbot, "On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation", *Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010. [Online]. Available: http://jmlr.csail.mit.edu/papers/volume11/cawley10a/cawley10a.pdf.

[47] L. 0. Hall, N. Chawla, and K. W. Bowyer, "Decision Tree Learning on Very Large Data Sets", [Online]. Available: https://www3.nd.edu/%7E~%7Ddial/publications/hall1998decision.pdf.

[48] K. Alsabti, S. Ranka, and V. Singh, "CLOUDS: A Decision Tree Classifier for Large Datasets", [Online]. Available: https://pdfs.semanticscholar.org/e0e7/31805c073c4589375c8b8f65769834201114.pdf.

[49] L. Jun, Z. Shunyi, L. Shidong, and X. Ye, "P2P Traffic Identification Technique", in *2007 International Conference on Computational Intelligence and Security (CIS 2007)*, IEEE, Dec. 2007, pp. 37–41, ISBN: 0-7695-3072-9. DOI: 10.1109/CIS.2007.81. [Online]. Available: http://ieeexplore.ieee.org/document/4415297/.

[50] W. M. Shbair, T. Cholez, J. Francois, and I. Chrisment, "A multi-level framework to identify HTTPS services", in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, IEEE, Apr. 2016, pp. 240–248, ISBN: 978-1-5090-0223-8. DOI: 10.1109/NOMS.2016.7502818. [Online]. Available: http://ieeexplore.ieee.org/document/7502818/.

[51] X. Tian, Q. Sun, X. Huang, and Y. Ma, "Dynamic Online Traffic Classification Using Data Stream Mining", in *2008 International Conference on MultiMedia and Information Technology*, IEEE, Dec. 2008, pp. 104–107, ISBN: 978-0-7695-3556-2. DOI: 10.1109/MMIT.2008.185. [Online]. Available: http://ieeexplore.ieee.org/document/5089070/.

[52] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning", in *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)l*, IEEE, 2005, pp. 250–257, ISBN: 0-7695-2421-4. DOI: 10.1109/LCN.2005.35. [Online]. Available: http://ieeexplore.ieee.org/document/1550864/.

[53] G. Maiolini, A. Baiocchi, A. Iacovazzi, and A. Rizzi, "Real Time Identification of SSH Encrypted Application Flows by Using Cluster Analysis Techniques", in, Springer, Berlin, Heidelberg, 2009, pp. 182–194. DOI: 10.1007/978-3-642-01399-7_15. [Online]. Available: http://link.springer.com/10.1007/978-3-642-01399-7%7B%5C_%7D15.

[54] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and peer-to-peer traffic in the network core", in *Proceedings of the 16th international conference on World Wide Web - WWW '07*, New York, New York, USA: ACM Press, 2007, p. 883, ISBN: 9781595936547. DOI: 10.1145/1242572.1242692. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1242572.1242692.

[55] J. Erman, M. Arlitt, and A. Mahanti, "Traffic Classification Using Clustering Algorithms", [Online]. Available: https://pages.cpsc.ucalgary.ca/%7E~%7Dmahanti/papers/clustering.pdf.

[56] M. Zhang, H. Zhang, B. Zhang, and G. Lu, "Encrypted Traffic Classification Based on an Improved Clustering Algorithm", in, Springer, Berlin, Heidelberg, 2013, pp. 124–131. DOI: 10.1007/978-3-642-35795-4_16. [Online]. Available: http://link.springer.com/10.1007/978-3-642-35795-4%7B%5C_%7D16.

[57] R. Bar - Yanai, M. Langberg, D. Peleg, and L. Roditty, "Realtime Classification for Encrypted Traffic", in, Springer, Berlin, Heidelberg, 2010, pp. 373–385. DOI: 10.1007/978-3-642-13193-6_32. [Online]. Available: http://link.springer.com/10.1007/978-3-642-13193-6%7B%5C_%7D32.

[58] L. Bernaille and R. Teixeira, "Early Recognition of Encrypted Applications", [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.5232%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[59] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification", in *Proceedings of the 2006 ACM CoNEXT conference on - CoNEXT '06*, New York, New York, USA: ACM Press, 2006, p. 1, ISBN: 1595934561. DOI: 10.1145/1368436.1368445. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1368436.1368445.

[60] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Semi-Supervised Network Traffic Classification", [Online]. Available: https://pages.cpsc.ucalgary.ca/%7B~%7Dmahanti/papers/metrics152-erman.pdf.

[61] International Association for Pattern Recognition. Technical Committee 11. and T. Kam, *Proceedings of the third International Conference on Document Analysis and Recognition : August 14-16, 1995, Montréal, Canada*. IEEE Computer Society Press, 1995, p. 278, ISBN: 0818671289. [Online]. Available: https://dl.acm.org/citation.cfm?id=844379.844681.

[62] Y. Freund, Y. Freund, and R. E. Schapire, "A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting", 1995. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.9855.

[63] E. N. de Souza, S. Matwin, and S. Fernandes, "Network traffic classification using AdaBoost Dynamic", in *2013 IEEE International Conference on Communications Workshops (ICC)*, IEEE, Jun. 2013, pp. 1319–1324, ISBN: 978-1-4673-5753-1. DOI: 10.1109/ICCW.2013.6649441. [Online]. Available: http://ieeexplore.ieee.org/document/6649441/.

[64] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. L. Woniak, "Ensemble learning for data stream analysis: a survey", 2017. [Online]. Available: http://www.cs.le.ac.uk/people/llm11/publications/KRAWXZYKINFFUS2017.pdf.

[65] E. N. de Souza, S. Matwin, and S. Fernandes, "Traffic classification with on-line ensemble method", in *2014 Global Information Infrastructure and Networking Symposium (GIIS)*, IEEE, Sep. 2014, pp. 1–4, ISBN: 978-1-4799-5490-2. DOI: 10.1109/GIIS.2014.6934280. [Online]. Available: http://ieeexplore.ieee.org/document/6934280/.

[66] B. Gellman, *Edward Snowden, after months of NSA revelations, says his mission's accomplished - The Washington Post*. [Online]. Available: https://www.washingtonpost.com/world/national-security/edward-snowden-after-months-of-nsa-revelations-says-his-missions-accomplished/2013/12/23/49fc36de-6c1c-11e3-a523-fe73f0ff6b8d%7B%5C_%7Dstory.html?utm%7B%5C_%7Dterm=.966aa9811276 (visited on 04/18/2018).

[67] A. Hern, *Far more than 87m Facebook users had data compromised, MPs told | UK news | The Guardian*. [Online]. Available: https://www.theguardian.com/uk-news/2018/apr/17/facebook-users-data-compromised-far-more-than-87m-mps-told-cambridge-analytica?utm%7B%5C_%7Dsource=esp%7B%5C&%7Dutm%7B%5C_%7Dmedium=Email%7B%5C&%7Dutm%7B%5C_%7Dcampaign=GU+Today+main+NEW+H+categories%7B%5C&%7Dutm%7B%5C_%7Dterm=271764%7B%5C&%7Dsubid=18350085%7B%5C&%7DCMP=EMCNEWEML6619I2 (visited on 04/18/2018).

[68] I. Rechberg and J. Syed, "Ethical issues in knowledge management: conflict of knowledge ownership", *Journal of Knowledge Management*, vol. 17, no. 6, pp. 828–847, Oct. 2013, ISSN: 1367-3270. DOI: 10.1108/JKM-06-2013-0232. [Online]. Available: http://www.emeraldinsight.com/doi/10.1108/JKM-06-2013-0232.

[69] *Art. 3 GDPR – Territorial scope | General Data Protection Regulation (GDPR)*. [Online]. Available: https://gdpr-info.eu/art-3-gdpr/ (visited on 04/18/2018).

[70] *Fines and Penalties – GDPR EU.org*. [Online]. Available: https://www.gdpreu.org/compliance/fines-and-penalties/ (visited on 04/18/2018).

[71] S. Farrell and H. Tschofenig, *RFC 7258 - Pervasive Monitoring Is an Attack*. Kenchiku Setsubi Iji Hozen Suishin Kyōkai, 2004. [Online]. Available: https://tools.ietf.org/html/rfc7258.

[72] *Center for Applied Internet Data Analysis*. [Online]. Available: http://www.caida.org/data/ (visited on 07/02/2018).

[73] *SNAP: Stanford Network Analysis Project*. [Online]. Available: https://snap.stanford.edu/index.html (visited on 07/02/2018).

[74] *sklearn.preprocessing.StandardScaler*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (visited on 08/03/2018).

[75] *scikit-learn: machine learning in Python — scikit-learn 0.19.2 documentation*. [Online]. Available: http://scikit-learn.org/stable/ (visited on 08/03/2018).

[76] J. A. Gubner and W.-B. Chang, "Wavelet transforms for discrete-time periodic signals", *Signal Processing*, vol. 42, no. 2, pp. 167–180, Mar. 1995, ISSN: 0165-1684. DOI: 10.1016/0165-1684(94)00125-J. [Online]. Available: https://www.sciencedirect.com/science/article/pii/016516849400125J.

[77] "CS448: Topics in Computer Graphics Mathematical Models for Computer Graphics Introduction to Wavelets", Tech. Rep., 1997. [Online]. Available: http://cva.stanford.edu/classes/ee482a/docs/lect01%7B%5C_%7Dsample.pdf.

[78] G. V. Trunk, "A Problem of Dimensionality: A Simple Example", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 3, pp. 306–307, Jul. 1979, ISSN: 0162-8828. DOI: 10.1109/TPAMI.1979.4766926. [Online]. Available: http://ieeexplore.ieee.org/document/4766926/.

[79] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments", *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20 150 202, Apr. 2016, ISSN: 1364-503X. DOI: 10.1098/rsta.2015.0202. [Online]. Available: http://rsta.royalsocietypublishing.org/lookup/doi/10.1098/rsta.2015.0202.

[80] *Decomposing signals in components (matrix factorization problems) — scikits.learn 0.8 documentation*. [Online]. Available: http://scikit-learn.sourceforge.net/0.8/modules/decomposition.html (visited on 09/11/2018).

[81] *sklearn.preprocessing.Imputer — scikit-learn 0.19.2 documentation*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Imputer.html (visited on 09/11/2018).

[82] M. Shafiq, Xiangzhan Yu, A. A. Laghari, Lu Yao, N. K. Karn, and F. Abdessamia, "Network Traffic Classification techniques and comparative analysis using Machine Learning algorithms", in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, IEEE, Oct. 2016, pp. 2451–2455, ISBN: 978-1-4673-9026-2. DOI: 10.1109/CompComm.2016.7925139. [Online]. Available: http://ieeexplore.ieee.org/document/7925139/.

[83] A. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms", ISSN: 0031-3203. DOI: 10.1016/S0031-3203(96)00142-2. [Online]. Available: www.sciencedirect.com/science/article/pii/S0031320396001422.

[84] D. Merkel, *Docker: lightweight Linux containers for consistent development and deployment*, 2014. [Online]. Available: https://dl.acm.org/citation.cfm?id=2600241.

[85] 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 É. Duchesnay, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011, ISSN: ISSN 1533-7928. [Online]. Available: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html.

[86] *PyShark - Python packet parser using wireshark's tshark*. [Online]. Available: https://kiminewt.github.io/pyshark/ (visited on 09/11/2018).

[87] *Impacket | Core Security*. [Online]. Available: https://www.coresecurity.com/corelabs-research/open-source-tools/impacket (visited on 09/11/2018).

[88] *Scapy*. [Online]. Available: https://scapy.net/ (visited on 09/11/2018).

[89] *ctypes — A foreign function library for Python — Python 2.7.15 documentation*. [Online]. Available: https://docs.python.org/2/library/ctypes.html (visited on 09/11/2018).

[90] Tcpdump, "Tcpdump/Libpcap public repository", [Online]. Available: http://www.tcpdump.org/.

[91] *RabbitMQ - Messaging that just works.* [Online]. Available: https://www.rabbitmq.com/ (visited on 09/11/2018).

[92] S. Vinoski and Steve, "Advanced Message Queuing Protocol", *IEEE Internet Computing*, vol. 10, no. 6, pp. 87–89, Nov. 2006, ISSN: 1089-7801. DOI: 10.1109/MIC.2006.116. [Online]. Available: http://ieeexplore.ieee.org/document/4012603/.

[93] Keller and M. S., *Linux journal.* 65es. Robert F. Young, 1994, vol. 1999, p. 15. [Online]. Available: https://dl.acm.org/citation.cfm?id=327981.

[94] *Periodic Tasks — Celery 4.2.0 documentation.* [Online]. Available: http://docs.celeryproject.org/en/latest/userguide/periodic-tasks.html (visited on 09/11/2018).

[95] *Homepage | Celery: Distributed Task Queue.* [Online]. Available: http://www.celeryproject.org/ (visited on 09/11/2018).

[96] *Redis.* [Online]. Available: https://redis.io/documentation (visited on 09/11/2018).

[97] G. Lettieri, V. Maffione, and L. Rizzo, "A Study of I/O Performance of Virtual Machines", *The Computer Journal*, vol. 61, no. 6, pp. 808–831, Jun. 2018, ISSN: 0010-4620. DOI: 10.1093/comjnl/bxx092. [Online]. Available: https://academic.oup.com/comjnl/article/61/6/808/4259797.

[98] *Raspberry Pi Documentation.* [Online]. Available: https://www.raspberrypi.org/documentation/ (visited on 08/01/2018).

[99] *PiVPN: Simplest setup of OpenVPN.* [Online]. Available: http://www.pivpn.io/%7B%5C#%7Dtech (visited on 08/01/2018).

[100] *Ace Stream.* [Online]. Available: http://info.acestream.org/%7B%5C#%7D/about/acestream (visited on 08/02/2018).

[101] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The Bittorrent P2P File-Sharing System: Measurements and Analysis", in *Proceedings of the 4th international conference on Peer-to-Peer Systems*, Springer-Verlag, 2005, pp. 205–216, ISBN: 3-540-29068-0, 978-3-540-29068-1. DOI: 10.1007/11558989_19. [Online]. Available: http://link.springer.com/10.1007/11558989%7B%5C_%7D19.

[102] *Youtube.com Traffic, Demographics and Competitors - Alexa.* [Online]. Available: https://www.alexa.com/siteinfo/youtube.com (visited on 08/02/2018).

[103] H. Ishwaran, "The effect of splitting on random forests", *Machine Learning*, vol. 99, no. 1, pp. 75–118, Apr. 2015, ISSN: 0885-6125. DOI: 10.1007/s10994-014-5451-2. [Online]. Available: http://link.springer.com/10.1007/s10994-014-5451-2.

[104] M. Verleysen, Université Catholique de Louvain, Katholieke Universiteit Leuven, C. I. European Symposium on Artificial Neural Networks, M. L. 2. 2.-2. Bruges, and ESANN 23 2015.04.23-25 Bruges, *Proceedings / 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2015, Bruges, Belgium, April 22-23-24, 2015.* Ciaco, 2015, ISBN: 9782875870148. [Online]. Available: https://www.researchgate.net/publication/301295119/%7B%5C_%7DOne-Vs-All%7B%5C_%7DBinarization%7B%5C_%7Din%7B%5C_%7Dthe%7B%5C_%7DContext%7B%5C_%7Dof%7B%5C_%7DRandom%7B%5C_%7DForest.

[105] M. Claesen and B. D. Moor, "Hyperparameter Search in Machine Learning", *undefined*, 2015. [Online]. Available: https://www.semanticscholar.org/paper/Hyperparameter-Search-in-Machine-Learning-Claesen-Moor/0173ca962e4ab3d084c89568345e06f67d3d7efc.

[106] *sklearn.model_selection.GridSearchCV — scikit-learn 0.19.2 documentation.* [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.model%7B%5C_%7Dselection.GridSearchCV.html (visited on 09/11/2018).

[107] R. Dingledine, N. Mathewson, and P. Syverson, *Tor: the second-generation onion router*, 2004. [Online]. Available: https://dl.acm.org/citation.cfm?id=1251396.

[108] *InfluxDB | The Time Series Database in the TICK Stack | InfluxData.* [Online]. Available: https://www.influxdata.com/time-series-platform/influxdb/ (visited on 09/11/2018).

Appendix A

Video category identification results

Table on the back of the sheet. In the PCA column, "-" means unscaled features.

Time bucket(s)	Time (s)	PCA	RF		SVM		NN		RF Adaboost		DT	
			CV score	Test score	CV score	Test score	CV score	Test score	CV score	Test score	CV score	Test score
1	30	-	0.9344292046	0.9556074766	0.8812213502	0.9018691589	0.9080954952	0.9228971963	0.9461117911	0.9602803738	0.9320796634	0.9439252336
1	30	0	0.9402841577	0.953271028	0.9203559972	0.9392523364	0.9221071031	0.9392523364	0.9437412597	0.9476145931	0.9285946512	0.9439252336
1	30	5	0.8893541252	0.9205607477	0.8911016312	0.9275700935	0.8858349133	0.9182242991	0.9010537916	0.9275700935	0.8694535244	0.8855140187
1	30	10	0.8952124383	0.9252336449	0.9215529876	0.9369158879	0.9185879713	0.9392523364	0.9198088015	0.941588785	0.8794292446	0.8948598131
1	30	20	0.8993130055	0.9299065421	0.9347282697	0.941588785	0.9267821857	0.9392523364	0.9180167759	0.9369158879	0.8736016512	0.8808411215
1	30	40	0.9010263518	0.9322429907	0.9215153479	0.9439252336	0.9349729602	0.9579439252	0.9197712418	0.9509345794	0.8572271022	0.8761682243
0.1	3	-	0.9171928738	0.9270346118	0.8406425794	0.8533676333	0.8643865025	0.8711412535	0.9256134825	0.9317118803	0.9007593036	0.9092609916
0.1	3	0	0.9153797005	0.923292797	0.8692983732	0.8790926099	0.8752048406	0.8823666978	0.9260230787	0.9298409729	0.8984203958	0.9080916745
0.1	3	5	0.8482453296	0.8566417212	0.8549702717	0.8629560337	0.8477783144	0.8571094481	0.9290631524	0.9252336449	-	-
0.1	3	10	0.8599999558	0.8659962582	0.8628066319	0.8695042095	0.8640348391	0.8662301216	0.9234443751	0.9392523364	-	-
0.1	3	20	0.8692391758	0.8795603368	0.8697662436	0.8769878391	0.8746786614	0.881898971	0.9238534949	0.9345794393	-	-
0.1	3	40	0.8764324664	0.8821328344	0.8705263751	0.8786248831	0.8839766341	0.8917212348	0.926261807	0.9347282697	-	-
1	10	-	0.8787374189	0.8794635896	0.8892464129	0.8916601715	0.9075746291	0.9080280592	0.9414974465	0.9353078722	0.922967838	0.9088074825
1	10	0	0.9292056785	0.9290724864	0.8999650413	0.9033515199	0.908542832	0.9142634451	0.940911509	0.9353078722	0.9254970209	0.9033515199
1	10	5	0.8734450997	0.8682774747	0.8849525778	0.8823070928	0.8769508451	0.8846453624	0.9003774998	0.9158878505	-	-
1	10	10	0.879296115	0.8869836321	0.8902191756	0.8939984412	0.897629651	0.900233827	0.8981343235	0.9299065421	-	-
1	10	20	0.8742316695	0.8846453624	0.8931458232	0.8939984412	0.8987992461	0.8994544037	0.8987946863	0.900233827	0.8547216987	0.8542478566
1	10	40	0.8863060858	0.8916601715	0.8939278332	0.9010132502	0.906015169	0.9041309431	0.8987992461	0.9010132502	-	-
0.1	1	-	0.8787374189	0.8794635896	0.8892464129	0.8916601715	0.9075746291	0.9080280592	0.9285946512	0.9345794393	-	-
0.1	1	0	0.8803162637	0.8790737564	0.8910027055	0.8986749805	0.9007455314	0.9064692128	0.9498844844	0.9508963367	0.9155520428	0.9212782541
0.1	1	5	0.8863060858	0.8916601715	0.8980519184	0.9158878505	0.8984592421	0.9016090157	0.91338903	0.9018691589	-	-
0.1	1	10	0.8790737564	0.8823070928	0.8845407566	0.8862298453	0.8980519184	0.9016090157	0.9256314312	0.9158878505	-	-
0.1	1	20	0.8993130055	0.9299065421	0.8863762653	0.8823666978	0.8921132174	0.9256314312	0.9279757362	0.9275700935	-	-
0.1	1	40	0.8980057026	0.9003741815	0.8989751098	0.8925233645	0.9092609916	0.953271028	0.9279757362	0.9270346118	-	-
1	60	-	0.943675982	0.9299065421	0.8242583545	0.8925233645	0.9120744577	0.9205607477	0.9496010032	0.953271028	0.9250713309	0.9299065421
1	60	0	0.9448387727	0.9252336449	0.9249338805	0.9299065421	0.9320624064	0.9299065421	0.9508048336	0.9439252336	0.926261807	0.9299065421
1	60	5	0.8981343235	0.9205607477	0.9109113413	0.9252336449	0.8991173213	0.9112149533	0.9004045339	0.9158878505	0.8851778386	0.8878504673
1	60	10	0.91338903	0.9018691589	0.9155628298	0.9392523364	0.9249895772	0.9112149533	0.9156589147	0.9252336449	0.8827558465	0.8925233645
1	60	20	0.9003224546	0.9158878505	0.9214181487	0.9485981308	0.9320347209	0.9205607477	0.9143997134	0.9252336449	0.8980519184	0.9158878505
1	60	40	0.9003774998	0.9018691589	0.9261103511	0.9345794393	0.9296537685	0.9345794393	0.9238534949	0.9205607477	0.8746580027	0.9065420561
0.1	6	-	0.9331005732	0.9391955098	0.8153247183	0.8391019645	0.8818642429	0.8713751169	0.9437412597	0.9476145931	0.9114569674	0.9092609916
0.1	6	0	0.9320796634	0.9439252336	0.8849525778	0.8823070928	0.8839766341	0.8917212348	0.9421512457	0.9267342167	-	-
0.1	6	5	0.8604650818	0.8709073901	0.8788259017	0.8872778297	0.8790613234	0.8863423761	0.8735661571	0.8779232928	0.8347320077	0.8489242283
0.1	6	10	0.8846760484	0.8826005613	0.8882985012	0.8942937325	0.8916912743	0.8905519177	0.8787374189	0.8916601715	-	-
0.1	6	20	0.889353597	0.8994387278	0.8967213406	0.9022450889	0.902101327	0.911131899	0.8882985012	0.8942937325	-	-
0.1	6	40	0.8980057026	0.9003741815	0.8966039716	0.9036482694	0.9086537778	0.9139382601	0.8863060858	0.8862298453	-	-

Table 1: Video category identification results.

Time buckets(s)	Time (s)	PCA	RF	SVM	NN	RF Adaboost	DT
1	30	-	0.8106725146	0.7385719152	0.7938596491	0.8165204678	0.7536549708
1	30	0	0.8345829735	0.7299488678	0.7883718396	0.8217219819	0.7395871934
1	30	5	0.8932748538	0.6830533236	0.8589181287	0.8859649123	0.7828947368
1	30	10	0.8786549708	0.6899196494	0.8245614035	0.8296783626	0.8004385965
1	30	20	0.8676900585	0.6978086194	0.7843567251	0.8450292398	0.8216374269
1	30	40	0.855994152	0.7963955188	0.7514619883	0.8355263158	0.7953216374
0.1	3	-	0.7360116874	0.7818496811	0.6550766983	0.7283418554	0.7008035062
0.1	3	0	0.7837481712	0.8010228933	0.73619681	0.8893748581	0.7258127124
0.1	3	5	0.7758217677	0.7674135412	0.750036523	0.7758948137	0.7585829072
0.1	3	10	0.7541271001	0.7586458841	0.6574141709	0.8282163743	0.7891654466
0.1	3	20	0.7313367421	0.7450073064	0.6644265888	0.8491947291	0.7741417093
0.1	3	40	0.7510591673	0.7963955188	0.6748721695	0.8653001464	0.7975146199
1	10	-	0.7788602046	0.7491859107	0.7004383829	0.7403799318	0.7447637604
1	10	0	0.7501287211	0.6668776936	0.7092332497	0.7573859823	0.7282871431
1	10	5	0.8519240136	0.7676570872	0.7963955188	0.8619094009	0.798343887
1	10	10	0.7786166585	0.7586458841	0.7221139795	0.7924987823	0.7308816366
1	10	20	0.8051631758	0.7450073064	0.7045786654	0.7842182172	0.7359961033
1	10	40	0.7932294204	0.8199121523	0.6965416464	0.7450073064	0.7536549708
0.1	1	-	0.7788602046	0.8618286816	0.6980029226	0.8566417212	0.7450073064
0.1	1	0	0.7283758189	0.8360175695	0.6938671345	0.8018134791	0.8878504673
0.1	1	5	0.7197886484	0.8565153734	0.6675351238	0.7299488678	0.7008035062
0.1	1	10	0.7949342426	0.850658858	0.6970287384	0.6675351238	0.7123571021
0.1	1	20	0.798100341	0.8491947291	0.7301509985	0.7313367421	0.7188872355
0.1	1	40	0.7932294204	0.7723886048	0.6977593765	0.750036523	0.7123571021
1	60	-	0.9458272328	0.7539127435	0.8125915081	0.8667642753	0.7467057101
1	60	0	0.9248375827	0.7749634681	0.803747195	0.8872398135	0.8878504673
1	60	5	0.8989751098	0.7107377648	0.8491947291	0.9106881406	0.859443631
1	60	10	0.8301610542	0.7221139795	0.8462664714	0.7467057101	0.8491947291
1	60	20	0.8565153734	0.7536549708	0.8579795022	0.8169838946	0.7877013177
1	60	40	0.8199121523	0.7461891416	0.7803806735	0.7803806735	0.8213762811
0.1	6	-	0.7444850256	0.7371201062	0.7208181154	0.753688824	0.7557341125
0.1	6	0	0.7584354513	0.7589814871	0.7389491685	0.7847371757	0.8831775701
0.1	6	5	0.7301509985	0.7109081092	0.7359961033	0.7510591673	0.6970287384
0.1	6	10	0.7375456538	0.7202'983916	0.7021183346	0.7748393752	0.7156166847
0.1	6	20	0.7403944485	0.7209180711	0.7467057101	0.7896585233	0.7355985217
0.1	6	40	0.7891654466	0.7430917069	0.7450073064	0.7785416639	0.7365546845

Table 2: Video category identification results (training without tunneled traffic.)

Video application identification results

Table on the back of the sheet. In the PCA column, "-" means unscaled features.

		RF		RF OvO		RF OvR		SVM		NN		RF Adaboost	
Time	PCA	CV score	Test score	CV score	Test score	CV score	Test score	CV score	Test score	CV score	Test score	CV score	Test score
10	-	0.9439014745	0.9309600863	0.9417516068	0.9309600863	0.947140433	0.9374325782	0.9085551276	0.8953613808	0.9080081636	0.8942826321	0.8935794404	0.9029126214
10	0	0.9411965626	0.9363538296	0.9420153714	0.932038835	0.9471324429	0.9406688242	0.9088108473	0.8953613808	0.9204123376	0.9050701187	0.8908200747	0.9083063646
10	5	0.8683657551	0.8338727077	0.8599656036	0.8403451996	0.8767057777	0.8414239482	0.8416719114	0.8263214671	0.8451629314	0.811218986	0.8829719825	0.8912789183
10	10	0.8783099582	0.8522114347	0.875085558	0.8608414239	0.8845407566	0.8640776699	0.8654029106	0.8457389428	0.8807859131	0.8511326861	0.9028418935	0.9348726592
10	20	0.8807387874	0.854368932	0.8759043708	0.8457389428	0.8845277218	0.8608414239	0.8664694804	0.8457389428	0.8893686401	0.8845738943	0.9177478203	0.9225806452
10	40	0.9012736924	0.882416397	0.8907439696	0.8856526429	0.89962597	0.8759439051	0.8618885901	0.836030205	0.9045010575	0.8932038835	0.9038375779	0.9093851133
30	-	0.9587581432	0.925566343	0.9636427485	0.9352750809	0.9611841595	0.925566343	0.9295941254	0.8867313916	0.9424848676	0.8996763754	0.7571516391	0.7475728155
30	0	0.9595711513	0.925566343	0.9628297404	0.9385113269	0.9619841595	0.928802589	0.9465891424	0.8964401294	0.9474021505	0.8996763754	0.7037809599	0.7313915858
30	5	0.913405455	0.9093851133	0.9215030685	0.8932038835	0.9231095201	0.8932038835	0.9247485445	0.8867313916	0.919850931	0.8932038835	0.9053859953	0.8802588997
30	10	0.9368850774	0.8964401294	0.9441436664	0.8964401294	0.9474021505	0.8964401294	0.9474021505	0.8964401294	0.9465631261	0.9126213592	0.9522280619	0.8996763754
30	20	0.9157988985	0.8511326861	0.9019584055	0.8673139159	0.9149795437	0.8673139159	0.9093339628	0.857605178	0.9231029635	0.8867313916	0.9189663257	0.8737864078
30	40	0.9239485445	0.8770226537	0.9093209546	0.8673139159	0.9279353265	0.8737864077	0.8979583006	0.8608414239	0.9271417781	0.8899676375	0.926296407	0.8770226537
60	-	0.9152306666	0.9032258065	0.9445077351	0.9419354839	0.9255429226	0.9225806452	0.889466017	0.864516129	0.9152306666	0.9032258065	0.9445077351	0.9419354839
60	0	0.9250923338	0.935483871	0.9480271209	0.935483871	0.9255429226	0.9225806452	0.9266796354	0.9161290323	0.9250923338	0.935483871	0.9480271209	0.935483871
60	5	0.9138224925	0.9161290323	0.8859518353	0.8774193548	0.871560273	0.864516129	0.918662881	0.9161290323	0.9138224925	0.9161290323	0.8859518353	0.8774193548
60	10	0.9251187748	0.9096774194	0.9137201279	0.9225806452	0.8795488236	0.9032258065	0.9331338084	0.9225806452	0.9251187748	0.9096774194	0.9137201279	0.9225806452
60	20	0.9314150991	0.9225806452	0.9394574131	0.9290322581	0.8469384218	0.8709677419	0.9347228308	0.9290322581	0.9314150991	0.9225806452	0.9394574131	0.9290322581
60	40	0.9235059136	0.9419354839	0.9233233025	0.9290322581	0.8442647293	0.8516129032	0.9364398194	0.9096774194	0.9235059136	0.9419354839	0.9233233025	0.9290322581

Table 3: Video application identification results.

Time buckets(s)	Time (s)	PCA	RF	SVM	NN	RF Adaboost	DT
1	30	-	0.6722580645	0.7019354839	0.7174193548	0.7161290323	0.7174193548
1	30	0	0.6619719825	0.6998471	0.7083841915	0.7038159196	0.6719193554
1	30	5	0.6012903226	0.6412903226	0.6012903226	0.5277419355	0.6735483871
1	30	10	0.6619354839	0.6387096774	0.6258064516	0.6012903226	0.7187096774
1	30	20	0.6451612903	0.6503225806	0.6374193548	0.6541935484	0.6490322581
1	30	40	0.5638709677	0.5522580645	0.5948387097	0.695483871	0.6180645161
0.1	3	-	0.5392872477	0.4993559468	0.5392872477	0.7093173036	0.6401889223
0.1	3	0	0.5401459854	0.5297157623	0.5038591959	0.7281058919	0.5783919592
0.1	3	5	0.5719192787	0.5285530271	0.5641906398	0.5641906398	0.6139974238
0.1	3	10	0.5787891799	0.5873765565	0.5869471876	0.6019750966	0.5547445255
0.1	3	20	0.5697724345	0.5762129669	0.5654787462	0.6500644053	0.533705453
0.1	3	40	0.5113782739	0.5444396737	0.5191069128	0.6753971662	0.5401459854
1	10	-	0.7183462532	0.6201550388	0.6925064599	0.6330749354	0.4496124031
1	10	0	0.601938194	0.5718310535	0.6013951055	0.6593818506	0.6392951831
1	10	5	0.6511627907	0.6072351421	0.5943152455	0.5426356589	0.677002584
1	10	10	0.5684754522	0.6485788114	0.6072351421	0.5891472868	0.677002584
1	10	20	0.6279069767	0.6149870801	0.5503875969	0.6330749354	0.684754522
1	10	40	0.5943152455	0.5348837209	0.5142118863	0.661498708	0.6175710594
0.1	1	-	0.7788602046	0.8618286816	0.6980029226	0.8566417212	0.7450073064
0.1	1	0	0.7283758189	0.8360175695	0.6938671345	0.8018134791	0.8878504673
0.1	1	5	0.7197886484	0.8565153734	0.6675351238	0.7299488678	0.7008035062
0.1	1	10	0.7949342426	0.850658858	0.6970287384	0.6675351238	0.7123571021
0.1	1	20	0.798100341	0.8491947291	0.7301509985	0.7313367421	0.7188872355
0.1	1	40	0.7932294204	0.7723886048	0.6977593765	0.750036523	0.7123571021
1	60	-	0.9458272328	0.7539127435	0.8125915081	0.8667642753	0.7467057101
1	60	0	0.9248375827	0.7749634681	0.803747195	0.8872398135	0.8878504673
1	60	5	0.8989751098	0.7107377648	0.8491947291	0.9106881406	0.859443631
1	60	10	0.8301610542	0.7221139795	0.8462664714	0.7467057101	0.8491947291
1	60	20	0.8565153734	0.7536549708	0.8579795022	0.8169838946	0.7877013177
1	60	40	0.8199121523	0.7461891416	0.7803806735	0.7803806735	0.8213762811
0.1	6	-	0.7444850256	0.7371201062	0.7208181154	0.753688824	0.7557341125
0.1	6	0	0.7584354513	0.7589814871	0.7389491685	0.7847371757	0.8831775701
0.1	6	5	0.7301509985	0.7109081092	0.7359961033	0.7510591673	0.6970287384
0.1	6	10	0.7375456538	0.7202’983916	0.7021183346	0.7748393752	0.7156166847
0.1	6	20	0.7403944485	0.7209180711	0.7467057101	0.7896585233	0.7355985217
0.1	6	40	0.7891654466	0.7430917069	0.7450073064	0.7785416639	0.7365546845

Table 4: Video application identification results (training without tunneled traffic.)

Appendix B

```
from collections import deque
from scipy.signal import argrelextrema

RANGE_VALUE = 30
N_SPIKES = 5
DEFAULT_VALUE = -1

def get_spikes(values, comparator, check_range = len(values)/(RANGE_VALUE)):
    """Returns the top N_SPIKES spikes of a given value array

    Args:
        values: The y values of the plot
        comparator: numpy.greater for maximums, npnumpy.less for minimums
        check_range: number of points to check before and after each maximum

    Returns:
        The coordinates of each spike, ordered by Y value.
    """

    spikes = deque([(DEFAULT_VALUE,DEFAULT_VALUE)] * N_SPIKES, maxlen=N_SPIKES)
    aux = argrelextrema(scalo, comparator, order=check_range)
    if aux[0].size:
        for e in np.nditer(aux) or []:
            spikes.append((values[e], scales[e]))
    ordered = sorted(spikes, key=lambda x: x[1], reverse=True)
    values = np.hstack(zip(*ordered))
    return values
```

Listing 1: Algorithm for detecting the spikes in the scalograms.