NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

JOÃO PEDRO DE FIGUEIREDO HENRIQUES

MSc in Electrical and Computer Engineering

MONITORING TRAFFIC CLASSIFICATION IN AN ISP

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon

March, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

MONITORING TRAFFIC CLASSIFICATION IN AN ISP

JOÃO PEDRO DE FIGUEIREDO HENRIQUES

MSc in Electrical and Computer Engineering

Adviser: Dr. Pedro Miguel Figueiredo Amaral
Auxiliar Professor, NOVA University Lisbon

Examination Committee

Chair: Dr. Daniel de Matos Silvestre
Auxiliar Professor, NOVA University Lisbon

Rapporteur: Dr. Luís Filipe Lourenço Bernardo
Associate Professor, NOVA University Lisbon

Adviser: Dr. Pedro Miguel Figueiredo Amaral
Auxiliar Professor, NOVA University Lisbon

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING
SPECIALIZATION IN ⟨SPECIALITY NAME⟩

NOVA University Lisbon
March, 2023

Monitoring Traffic Classification in an ISP

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.9.4) [25].

Acknowledgements

I would like to start by expressing my gratitude to my supervisor, professor Pedro Amaral, first of all for having stimulated a fascination for telecommunications with the subjects taught throughout the course, and for the opportunity and help provided in the development of this dissertation.

I am also grateful to Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa for these 5 years of much growth and learning that I will take to my professional and personal life.

Thanks should also go to NOS SGPS for the opportunity they gave me to develop this project and for the way they treated me during the whole process. A special thanks must go to Mário Batista who played a key role in the development of this work during the time we were working together at NOS, providing all the necessary help and giving me freedom to explore more and more.

A big thanks also to my closest group of colleagues who became great friends for life for all the support during the elaboration of this document.

And last but not least, I would like to thank my family and friends for never letting me give up and for helping me through the hardest times.

Abstract

The use of mobile applications has been increasing exponentially in recent years, which in turn leads to the need to manage a large flow of Internet traffic. Service providers offer their customers plans that include exemption from taxation of traffic consumption for some applications and certain quality guarantees for them, therefore monitoring the traffic classification is important to maintain the quality of the service offered. In this dissertation, a prototype was developed in order to monitor the accuracy of the traffic classification by the 4G network core for some of the most popular mobile applications. The solution presented for monitoring the core classification was to measure the amount of data consumed by the application directly on the test devices (only Android mobile phones were used in this work) and compare it with the result given by the core classification. The project was developed in a 4G network of a service provider. This project was also an automation work, since the entire system works automatically and there is no need for human interaction. Real devices were used to test the classification. Navigation in applications was automated using tools such as Robot Framework, Appium, and ADB. The Robot Framework proved to be very strong in automating tasks, both at the mobile level and in validating the core classification. A dashboard with graphs was built to analyze the performance of the classifiers over time, using the Grafana tool. To manage the launch of tests periodically and automatically, the Jenkins tool was used. A network data collection mechanism for mobile applications was developed, using automation techniques, which can be used to implement other types of classifiers, such as ML or DL.

Keywords: Traffic Classification, Monitoring, Automation, Robot Framework, Appium, ADB, Grafana, Jenkins, Machine Learning, Deep Learning

Resumo

A utilização de aplicações móveis tem vindo a aumentar exponencialmente nos últimos anos o que, por sua vez, leva a necessidades de gestão de um grande fluxo de tráfego de Internet. Os fornecedores de serviço oferecem aos seus clientes, planos que incluem isenção de taxação do consumo de tráfego para algumas aplicações e certas garantias de qualidade para as mesmas, portanto a monitorização da classificação de tráfego é importante para manter a qualidade do serviço oferecido. Nesta dissertação desenvolveu-se um protótipo com o objetivo de monitorizar a precisão da classificação de tráfego por parte do core da rede 4G para algumas das aplicações móveis mais populares. A solução apresentada para a monitorização da classificação do core foi medir a quantidade de dados consumidos pela aplicação diretamente nos dispositivos de teste (neste trabalho apenas foram usados telemóveis Android) e comparar com o resultado dado pela classificação do core. O projeto foi desenvolvido numa rede 4G de um fornecedor de serviço. Este projeto foi também um trabalho de automação uma vez que todo o funcionamento do sistema é automático não havendo a necessidade de interação humana. Foram usados dispositivos reais para testar a classificação. A navegação nas aplicações foi automatizada usando ferramentas como Robot Framework, Appium e ADB. O Robot Framework provou ser muito forte na automatização de tarefas, tanto a nível do mobile como na validação da classificação do core. Foram construídos painéis de monitorização com gráficos para analisar o desempenho dos classificadores ao longo do tempo, usando a ferramenta Grafana. Para a gestão de lançamento de testes de forma periódica e automática, foi usada a ferramenta Jenkins. Foi desenvolvido um mecanismo de coleção de dados de rede das aplicações móveis, usando técnicas de automação, que pode ser usado para a implementação de outros tipos de classificadores, como por exemplo ML ou DL.

Palavras-chave: Classificação de tráfego, Monitorização, Automação, Robot Framework, Appium, ADB, Grafana, Jenkins, Machine Learning, Deep Learning

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

Internet traffic has been increasing exponentially in recent years, much due to the easy access to internet services offered by the Internet service providers (ISPs) and the mass use of smartphones and other devices. Managing all this huge flow of data over the network has become an essential factor, especially in terms of Quality of Service (QoS), Quality of Experience (QoE), security, capacity planning, among others. In the case of an ISP, for example, certain applications need to have QoS guarantees depending on the service subscribed by customers or the exemption from taxation of internet traffic consumption.

Traffic can be classified based on the protocol (e.g. TCP, UDP), type (e.g. voice, video, audio, download), and the application that generated the data stream (e.g. Youtube, Facebook, WhatsApp), etc. The task of finding a model, dataset, or approach capable to identify and classify all applications is complicated. The emergence of new Web protocols such as HTTP/2 and QUIC, has brought new challenges to traffic classification, mainly due to the fact that communication between clients on the Internet has become almost entirely encrypted. The rapid release of new updates to the most popular applications, often at an almost weekly rate, and the appearance of new applications make the task of classification difficult for models.

There are a variety of techniques that have been used over the years to identify and classify traffic in telecommunications networks. Initially, the simplest and most popular technique was the port based approach, since applications used well-known registered ports. Later, payload based techniques, also known as Deep Packet Inspection (DPI), gained popularity as they were able to get extremely accurate results by comparing the packet payload with known expressions/signatures of applications. Nowadays, these two approaches are becoming ineffective. Many applications may use the same port or use random ports. DPI techniques have become very hard (or impossible) to use due to large amounts of encrypted traffic.

In order to overcome these difficulties, Machine Learning and Deep Learning techniques for traffic classification have been widely studied in recent years and promising

results have been obtained. These techniques are mostly based on using only statistical features of the packets and therefore do not require packet payload, allowing the identification of encrypted traffic. One of the major challenges present in these models is the lack of real, comprehensive, and up-to-date traffic datasets to train the models.

Data collection and labeling are the most time-consuming tasks. It is important to start developing methods that can automate these tasks, in order to obtain data sets that can serve as input to develop new classifiers as well as to verify the accuracy of currently used approaches. In this thesis, we will start by automating traffic collection for some specific applications using some open source tools. We then develop a software prototype that automates the verification of a classifier's accuracy.

1.2 Motivation

This thesis was developed in partnership with an Internet service provider, NOS SGPS. As a service provider, traffic classification is an essential task for a good operation of all services offered to customers. New mobile plans subscribed by clients tend to offer free network traffic for specific popular applications, so correct identification of these applications is very important. Almost every week new updates are launched for these applications and new ones enter the list of the applications of interest. This brings challenges to the classification due to the changes in the network traffic generated by the applications. These changes may downgrade the accuracy of the existing classifier.

In order to monitor if the classification is being correctly done by the existing classifiers at the core of the network, there is a need for automation in obtaining labelled datasets that follow the changes in traffic patterns of applications like Instagram, WhatsApp, Facebook, etc. Therefore, it is interesting to be able to generate data from those mobile applications in an automatic way, since a manual approach is error-prone and has no scalability for all the applications. In this work, we will start by implementing automatic traffic generation methods with the use of some open source tools like Appium and ADB (android debug bridge). This will be the starting point for the automation of network traffic monitoring.

When it comes to data collection, the generation of data can be done in two different ways: synthetic data or real-world data. Synthetic data is the traffic that is generated using automatisms like scripts, simulators, and emulators. The big advantage is the ease of labeling because the traffic is known a priori. However, the traffic may differ from the real-world data which is the traffic from real users in the network. In this case, the labeling, privacy, and security concerns are an obstacle to data collection.

Once we have the process of generation and capturing traffic working, we are able to build a prototype system for the automatic monitoring of the existing classifiers. This system allows the autonomous verification of the accuracy of the core classification since we can generate new labeled traffic as applications evolve. This data can be used to evaluate the use of other types of classifiers, such as ML and DL techniques.

1.3 Work Contribution

This thesis was developed in partnership with NOS, resulting in the creation of a system that autonomously controls the accuracy of core network traffic classification. The system has been successfully deployed on the network and is fully operational, providing valuable insights for the teams involved and saving time previously spent on manual monitoring.

Additionally, we implemented a data collection system, which provides a reliable source of data for the implementation of ML/DL-based network traffic classifiers. Collecting reliable data on the core network of an ISP is crucial for these new techniques, and our data collection system provides an essential starting point for their development.

1.4 Dissertation Outline

The dissertation's organization is divided into the following chapters:

- **Chapter 2**: State-of-the-art regarding traffic classification techniques and existing challenges in the real world networks. Related work is presented throughout this chapter.
- **Chapter 3**: Network Architecture used for the implementation of the project. It is given an overview of the evolution of mobile networks and an explanation of how 4G networks work. It is also explored how traffic classification is done in the core.
- **Chapter 4**: Implementation of the Autonomous Monitoring System. This chapter explores all the parts that compose the developed system and how it was implemented.
- **Chapter 5**: Analysis of the results from the Monitoring System regarding the Core DPI Classification Results. The challenges for the DPI classification are also discussed.
- **Chapter 6**: This chapter shows the implementation of a Data Collection System and that can be a starting point for the implementation of ML/DL algorithms for traffic classification.
- **Chapter 7**: This final chapter draws conclusions from the developed work and presents interesting points to be explored in future work.

2

State-of-the-Art

2.1 Network Traffic Classification: Overview

The complexity of knowing from where the traffic comes in a telecommunications network keeps increasing every day. For the ISPs it is important to identify the application that generated the data in the network for QoS, network management, security, etc. This is not an easy task since most of the techniques that have been used over the years are becoming very limited.

There are two different categories of traffic classification: online and offline. Online classification refers to cases where classification needs to be done as quickly as possible because the classification output influences the current flow decisions (e.g. QoS guarantees, routing). For other applications, such as billing systems, or traffic information for further analysis, the classification can be offline [33].

2.1.1 Port based IP traffic classification

Port based techniques were very popular in the past because many applications used fixed port numbers assigned by the Internet Assigned Numbers Authority (IANA)[4][27]. These methods are simple and do not require strong computation power. The port numbers can be easily accessed and are usually not affected by encryption, thus allowing a fast flow classification [15], which is suitable for online classification.

Table 2.1: IANA assigned Port Numbers for some Internet applications. Adapted from [31]

Application	Port Number
FTP	21
Telnet	23
SMTP	25
DNS	53
HTTP	80
HTTPS	443

These techniques are no longer a good solution since most applications can use the same port and use dynamic port negotiation.

2.1.2 Payload based IP traffic classification

Another approach is the payload based IP traffic classification also known as Deep Packet Inspection (DPI). These techniques inspect the real content of the packets and try to match them with known signatures/expressions. Over the years, these methods showed very good accuracy and overcame the port number dependency problem.

Today, payload based approaches have become almost impossible to use due to the high number of encrypted data that is exchanged which makes it impossible to identify the signatures of applications. There is an overhead in keeping the application's signatures up to date, since the application's updates may change the known expressions and therefore the identification of the classifier. Also, there are scalability problems with DPI techniques since it requires large amounts of memory. Another obstacle with these methods is that access to the information can violate privacy policies and regulations.

2.1.3 Machine Learning classification

The most recent solutions to overcome the difficulties faced by the above methods is the use of Machine Learning techniques applied to the statistical features of the flows. In this approach there is no longer the need to inspect the payload of the packets, leading to the reduction of the computational costs that were present with DPIs and permitting the identification of encrypted traffic.

In ML techniques we can use supervised, unsupervised, or semi-supervised learning. For supervised learning, we need to obtain (pre-classified) labeled datasets. Obtaining this annotated data in a real network is not an easy task. Unsupervised learning does not require labeled data and it's normally used to group data by their proximities (without knowledge about the output) and clustering it. Semi-supervised learning is a combination of the supervised and unsupervised methods. It can work with only a few labeled data and the rest unlabeled. This approach is a solution to overcome the hard task of gathering labeled data [3].

2.1.3.1 Supervised Learning

As mentioned before, supervised learning requires pre-classified labeled data. It aims to build a model capable of predicting the class to which a given input with specific features belongs. It means that the supervised technique trains the model with known labeled data, and then this model will produce prediction output to new data samples as illustrated in Figure 2.1.

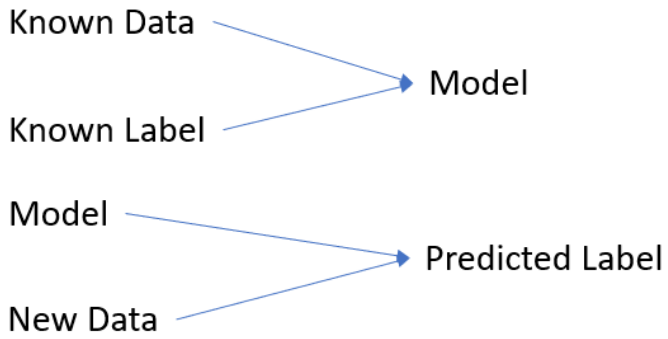


Figure 2.1: Supervised Learning

To train the classifier we use a training dataset, and to validate if the classifier is performing correct classification, we use a test dataset that will verify if the labels given by the classifier are correct. The construction of the full classification method can be divided into three phases: training, classification, and validation.

In the training phase, the optimal approach would be to use a training dataset that it's collected from real traffic, so the classifier can be more accurate. Sometimes when the training dataset is too controlled and detailed it can induce overfitting in the classification model, which means that the classifier gets too adapted to the training set and becomes unable to classify new data that may differ slightly from the training set. The classification phase is where the model predicts and attributes the labels to the new data.

Lastly, the validation phase evaluates the precision of the classifier. Some metrics such as Accuracy, Precision, Recall, F1-score are the most used to get insights from the classifier.

1. Accuracy: How many right predictions were obtained

$$Accuracy = \frac{TrueNegatives + TruePositives}{TruePositives + FalsePositives + TrueNegatives + FalseNegatives} \tag{2.1}$$

2. Precision: How correct is the classifier

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{2.2}$$

3. Recall: How many True Positives were classified

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2.3}$$

4. F1-score: Balance between precision and recall

$$F1 - score = \frac{2 \times (Precision \times Recall)}{Precision + Recall} \tag{2.4}$$

Some of the most used supervised algorithms in traffic classification are:

- **SVM (Support Vector Machines):** SVM is one of the most used algorithms for classification. Since it is a supervised algorithm, it requires having pre-classified labeled data for training the model. The objective is to find hyperplanes in an N-dimensional space (where N is the number of features) that differentiate the data in classes based on the pre-defined labels. The data points that are closest to the hyperplane are the Support Vectors. The success of the classes created by the SVM is dependent on a good choice of a kernel function. Different kernel functions will generate different features in the data. There are four types of kernel functions: linear, polynomial, radial basis function, and sigmoid.

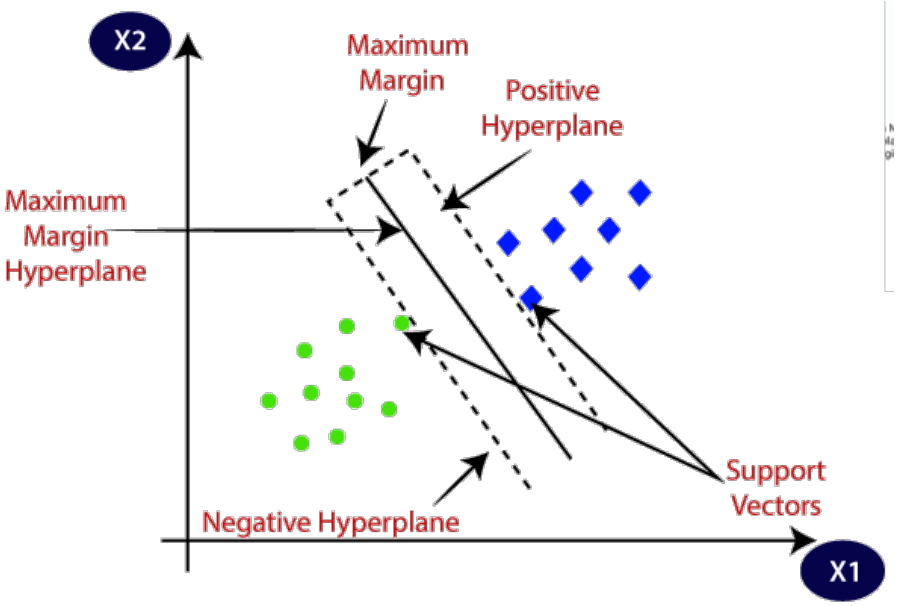


Figure 2.2: SVM classification. Adopted from [40]

In [14] Zhong Fan and Ran Liu showed that the use of the SVM model based on radial base kernel functions achieved better results in traffic classification and it is computationally more efficient. The authors made a test for the stability of the model using a new dataset separated by over one year, and even though the results were downgraded, it continued to give a strong accuracy for the major application categories. In [20] the SVM algorithm was used to develop an automatic traffic detection system to identify attack network flows.

- **Decision Trees:** This algorithm, as its name implies, has a tree structure and it is built during the training phase using the labeled dataset. Each node of the tree represents a test on a feature of the dataset and the branches represent the decision rules. The algorithm uses a question-answer paradigm, so it advances to another node based on the answer to the tested feature. Decision trees have the advantage of being easily understood.

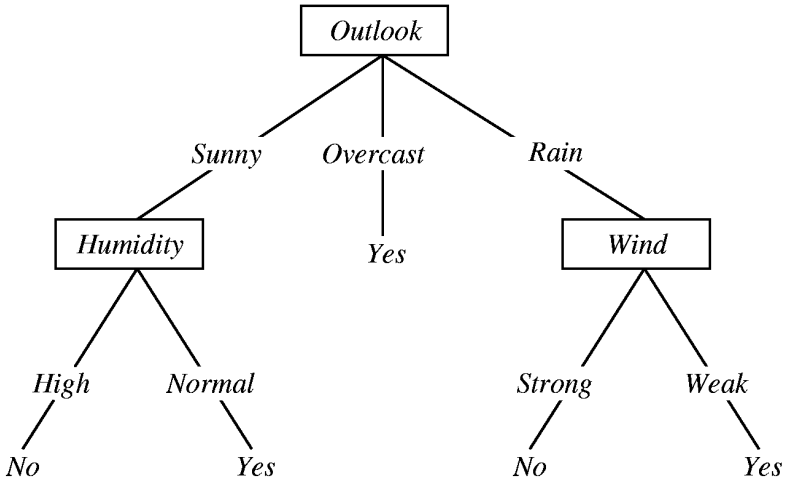


Figure 2.3: Decision Tree algorithm. Adopted from [43]

The most used decision trees algorithms are the C4.5 and the Random Forest. In [10] both algorithms were used for traffic classification with real-world traffic datasets. Under the same conditions, both algorithms achieved good results and proved that the models can classify traffic flow. In [28] the C4.5 algorithm outperformed Random Forest although with slower performance.

2.1.3.2 Unsupervised Learning

The main goal of unsupervised learning algorithms is to find patterns and correlations in the data and group the data according to similarities. These methods do not require pre-classified labeled data, so these algorithms bring the capacity to extract and collect information without knowing any corresponding output. They are usually used for clustering tasks where the data is grouped in clusters by their features. Unlike supervised learning, an unsupervised learning cluster approach can discover new applications by looking at the relationships that formed a new cluster. Nowadays, new applications and updates to existing applications are constantly appearing and these clustering techniques can help identify new patterns.

Some of the most used algorithms for clustering in traffic classification are:

- **K-Means:** This algorithm is probably the most used for clustering tasks because it is simple and fast. It groups points of the initial dataset according to their similarities by a fixed K number of clusters. Initially, the centers of the clusters are chosen randomly. At each iteration, the center of the cluster is recalculated and the algorithm is repeated. This process will be performed until it converges.

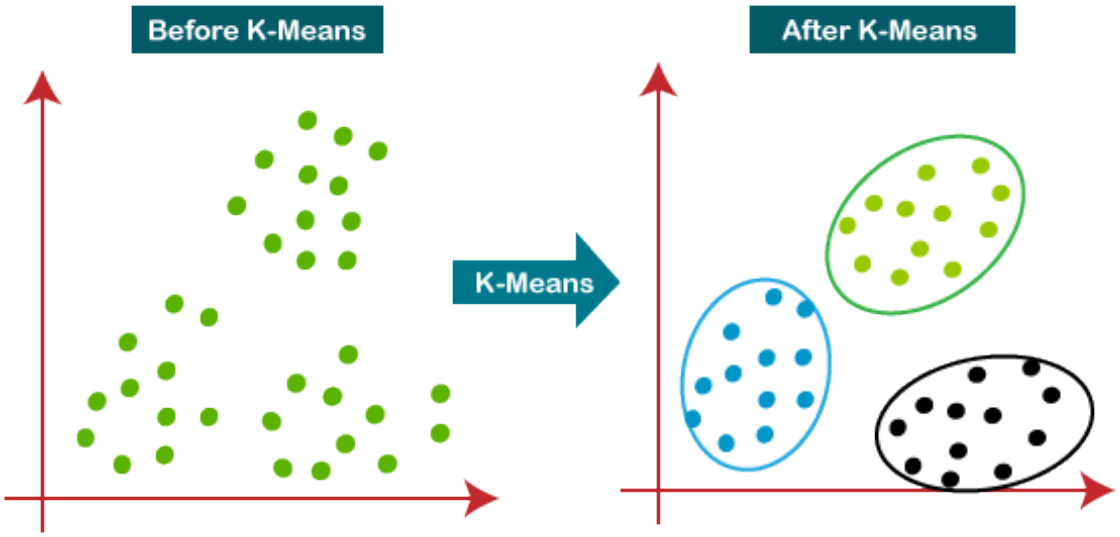


Figure 2.4: K-Means algorithm. Adopted from [18]

In [37] was demonstrated that K-Means can achieve very good results in producing clusters for a single traffic class which allows to isolate and identify different applications. Liu, Li, and Li [22] shown that the use of Log transformation of the data can improve the accuracy of the model, and the procedure of Log transformation is simple and non-time consuming. In [26] the implementation of a filtering phase after clustering proves to improve the performance of the classifier. The performance in all classifiers is dependent on the K number of clusters used, so for better performance of the classifier, it is necessary to find the optimal K number that gives the maximum homogeneously in the clusters. Obviously, increasing the number of clusters increases the overall time of the model.

- **DBSCAN (Density Based Spatial Clustering of Applications with Noise):** This algorithm has an advantage over partition-based algorithms because it can produce non-spherical clusters, unlike K-Means, which allow different shapes of clusters that can be more interesting. The DBSCAN algorithm is based on the concepts of density-reachability and density-connectivity, so a cluster is defined by these notions. There are two input parameters: a value for distance (D) that defines the neighbors of an object, and the minimum number of points (minPts). DBSCAN will find all the density-connected objects and those objects form a new cluster. All the objects that do not have a cluster assigned are considered noise. This also differs from K-Means which assigns all the objects [13].

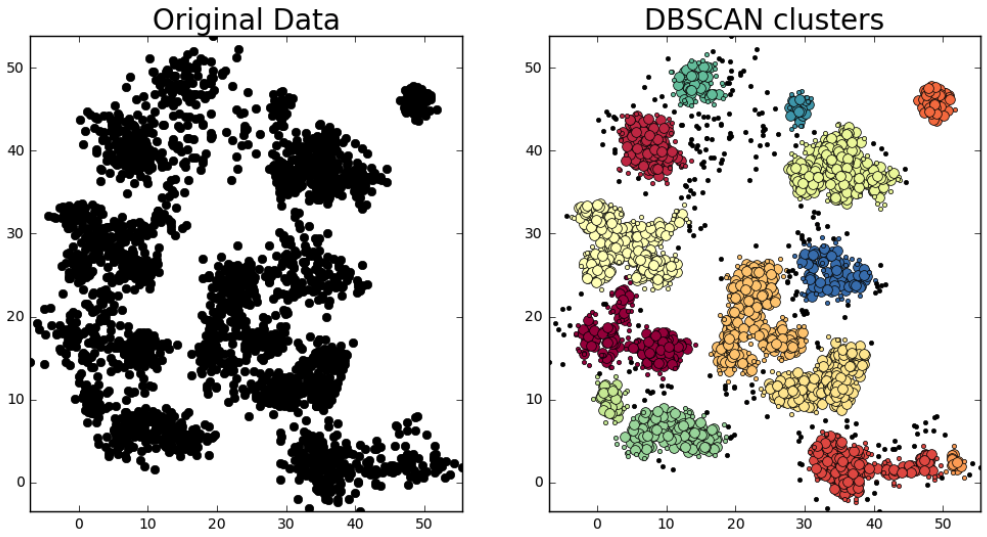


Figure 2.5: DBSCAN algorithm. Adopted from [8]

In [13] and [12] was made a comparison between K-Means, DBSCAN, and AutoClass. AutoClass has a big disadvantage over the other two algorithms because it takes much more time to construct the model. K-Means needs only 1 minute, DBSCAN needs 3 minutes, and AutoClass takes 4.5 hours to train the model. In [13] AutoClass achieved better overall accuracy but the trade-off of model building time makes it less suitable, and DBSCAN proved to have great potential because the clusters have high predictive power to isolate single categories of traffic. In [12] DBSCAN algorithm had the highest precision.

2.1.3.3 Semi-Supervised Learning

Semi-supervised learning methods are an alternative to overcome the difficulties of obtaining datasets with labeled data. Many works have been done using semi-supervised methods for traffic classification which makes this approach very useful because there is no need to have a huge dataset with labeled data. With only a few labeled data it is possible to obtain good results.

2.1.4 Deep Learning

The Machine Learning algorithms mentioned before have been extensively used in the last decade and have been successful. However, the complexity of the new modern applications and protocols has brought some challenges to these models. The constant updates that applications release and the appearance of new applications make some of these methods decrease the capabilities of classification.

In the last years, Deep Learning has been growing a lot in areas such as image detection and classification, speech recognition, natural language, social media, etc., and many

works and studies have already been done in the area of network traffic classification using these techniques.

Deep Learning is a subset of Machine Learning and it is based on artificial neural networks (ANN). These algorithms have multiple layers of linear and non-linear transformations. Each layer can extract more information than the last. These transformations are prepared in the training phase according to the labeled (or unlabeled) data. A big advantage of Deep Learning is that the selection of features is automatically done during the training of the network. This can overcome the hard task of identifying new applications that are constantly emerging and recognizing patterns of the old applications that are evolving. In comparison with ML methods, DL has a higher capacity for learning more complex patterns and nonlinear relationships between the input data and the output classes.

Once again, the big challenges are that DL requires a large amount of labeled data during the training phase and strong computation power. Since the training phase of a model can take from a few hours to a few days or weeks, re-training of models can be an interesting solution to this time-consuming task.

Some of the most used algorithms in DL for traffic classification are:

- **Convolutional Neural Networks (CNN):** CNNs are a type of Deep Learning model that were initially created to process images. Their primary purpose is to classify images by taking an image as input and producing a label as output that indicates which class the image belongs to. CNNs are made up of an input layer, a certain number of hidden layers, and an output layer. The hidden layers are responsible for carrying out the convolution operation (convolutional layer), which is then fed into an activation function. The output of the activation function is passed through a pooling operation, which helps to reduce the dimensionality of the output by downsampling it (pooling layer). Following the pooling layer, there are fully connected and normalization layers.

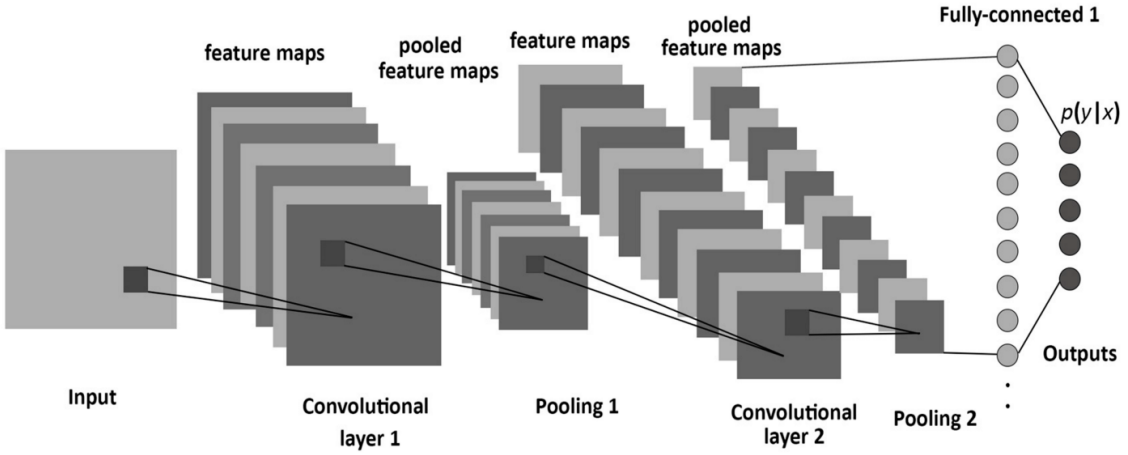


Figure 2.6: Convolutional Neural Network. Adopted from [45]

- **Long Short-term Memory (LSTM):** LSTM is a type of recurrent neural network (RNN) which is suitable for sequential data where the output of a LSTM model depends not only on the last inputs but also on the previous ones, therefore this capacity of memory brings more power in processing sequential data, which is normally the case when dealing with data from network traffic. It uses a mechanism of three gates: forget gate, input gate, and output gate. This controls how the information is saved or not in the LSTM unit regarding the features found in the input data.

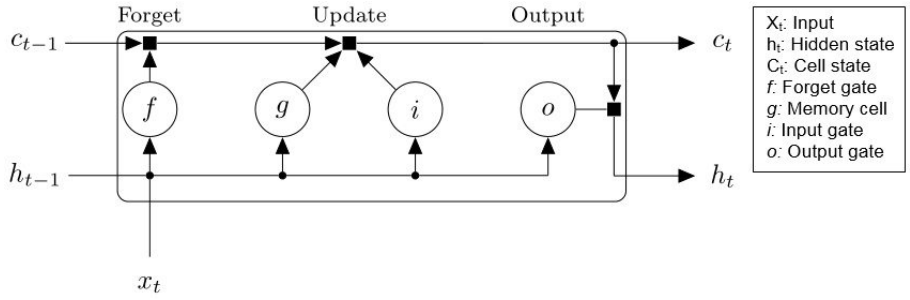


Figure 2.7: LSTM. Adopted from [23]

2.1.4.1 Related Works

A detailed overview of the use of Deep Learning methods for TC was made in [33]. The authors introduced a general framework for deep-learning-based TC. Furthermore, they discussed the process of data collection (data capturing, data pre-processing, features, training), covering some of the most common models used in deep learning classifiers, the process of evaluation, and the challenges that exist with these models, in particular, the requirement of large datasets for training.

Lopez-Martin et al. [24] implemented a model for traffic classification based on a combination of CNN+LSTM. It was demonstrated that it is not necessary to process a large number of packets per flow. They obtained good results using only 5-15 pakets per flow. This work proved that the combination of both models improves the accuracy of the classifier.

In [32][34] the authors used architectures with CNN + LSTM models and achieved good results classifying encrypted traffic. In [32] the accuracy of classification improved when both models were combined, particularly due to the significant amount of ambiguous flows generated by applications. They only used the first six packets from the flows. In [34] the authors used a semi-supervised approach to show how to obtain good classification with few labeled data with the use of CNN. First, the model was pre-trained on a large unlabeled dataset. Then the model is re-trained in a small labeled dataset using the learned weights of the first train. Good results were achieved inclusive with a QUIC dataset. Without the pre-training, the accuracy decreases.

2.2 New Web Protocols: Challenges

Most of the Internet traffic is over Hypertext Transfer Protocol (HTTP) and it is expected to continue this way since more and more applications keep migrating their services to the Web. With all the demand and evolution of the Web, over the past few years, there have been improvements to HTTP [27] to improve security and QoE. In this section, we describe how HTTP/2 and QUIC protocols bring challenges to traffic classification, as well as Domain Name Encryption.

2.2.1 HTTP/2

HTTP/2 is the upgraded version of HTTP/1.1 and was released in May 2015. It has been widely adopted by major internet providers since then [2]. This version addresses some of the issues present in the previous versions.

The primary aim of HTTP/2 is to reduce latency and minimize protocol overhead by compressing HTTP header fields efficiently. To accomplish this, it uses full request and response multiplexing of streams, server push, and binary message framing. HTTP/2 also offers payload encryption, but it does not replace any core concepts such as HTTP methods, status codes, URIs, and header fields [17].

The use of multiplexing, compression, and encryption complicates traffic classification by introducing unpredictability in the data [7].

2.2.2 QUIC

Quick UDP Internet Connections (QUIC), proposed by Google in 2012, is a transport protocol that aims to improve the browsing experience on the internet. It is supported by most browsers and heavily used by Google services like YouTube.

It provides low-latency communication, security, and quick deployment [9]. It is implemented in user space and runs over UDP. QUIC has several advantages, such as reduced establishment time of connections negotiating TLS in one RTT, congestion control, and multiplexing without the head-of-line blocking.

However, these benefits bring complexity to traffic classification, as with HTTP/2.

2.2.3 Domain Name Encryption

Although these days, almost all Internet traffic is encrypted, it is still possible to find information that can be used to identify data on the network, such as DNS and SNI.

Domain Name System, or DNS, is a protocol responsible for converting human-readable domain names into machine-readable IP addresses, which is why it is possible to access a website using a name instead of using its IP address. It is known as the phonebook of the Internet.

The Server Name Indication, or SNI, is an extension for the TLS protocol and is responsible for ensuring that the client receives the correct SSL certificate when asking for a specific website. This is necessary because many web servers host several domain names which makes the IP address not enough to identify the required domain.

The Domain Name System (DNS) has been used by many entities for traffic classification, and that is because there is no encryption in traditional DNS queries and responses between clients and servers. The same is true for the domain name information that is also visible and available without encryption just like DNS. The Server Name Indication (SNI) is transmitted during the TLS handshake in plaintext.

The ISPs have been using this information with DPI techniques that in the cases where there is no encryption, it proves to be a great way to identify traffic flows from known applications. The only thing needed is to know apriori some relevant SNIs for the interesting applications.

Internet usage has increased exponentially and all issues of security and privacy are becoming more and more important. In order to improve security, there have been advances in the DNS protocol. DNS-over-TLS (DoT), DNS-over-HTTPS (DoH), and Encrypted Server Name Indication (ESNI) are some new technologies that are being widely used.

2.2.3.1 DNS-over-TLS (DoT) and DNS-over-HTTPS (DoH)

DNS-over-TLS (DoT) is a security protocol for encrypting DNS queries and responses between clients and DNS resolvers. It uses the Transport Layer Security (TLS) protocol on top of the normal UDP (or TCP).

DNS-over-HTTPS, or DoH, just like DoT, encrypts DNS resolutions ensuring privacy and security. In contrast with DoT, the queries and responses are sent via the HTTP or HTTP/2 protocols.

Figure 2.8 shows how encrypted DNS hides information for third parties in a communication.

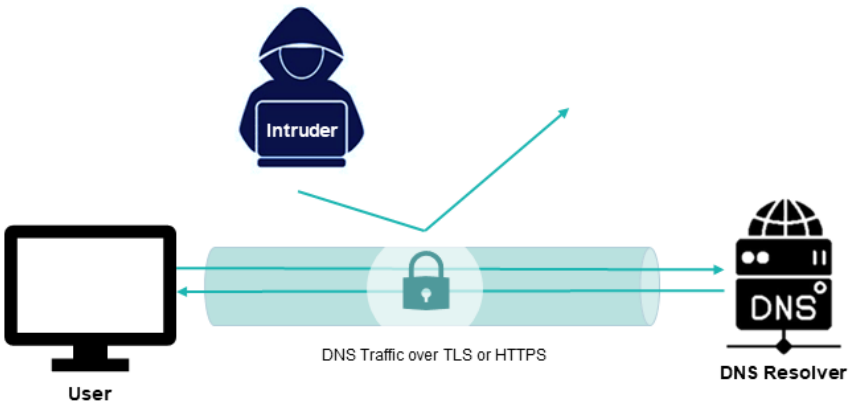


Figure 2.8: Encrypted DNS

The major difference between these two protocols is in the use of ports since DoT has a dedicated port (853), and DoH uses port 443 which is the port used for all the HTTPS traffic as well [11].

2.2.3.2 Encrypted Server Name Indication (ESNI)

Encrypted Server Name Indication (ESNI) is an extension to the TLS 1.3 protocol, and it is only compatible with this version. As the name implies, this feature deals with the encryption of the SNI. For encryption, both sides of communication must know the key for encrypting and decrypting information, therefore it was necessary to find a solution since the ClientHello message in the TLS handshake, is sent before the encryption key is negotiated between client and server.

To accomplish this objective, ESNI encrypts the SNI part of the ClientHello message. Figure 2.9 demonstrates the TLS 1.3 handshake with Encrypted SNI.

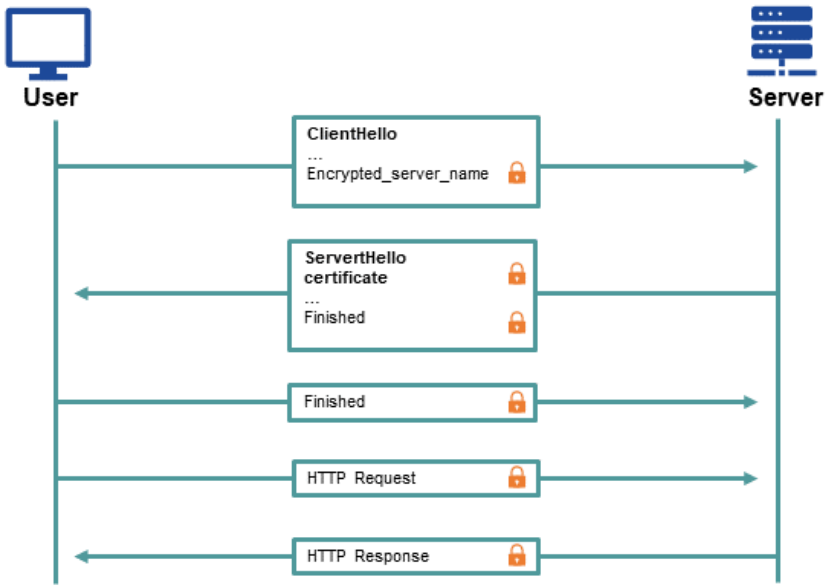


Figure 2.9: TLS 1.3 with Encrypted SNI. Adopted from [16]

For that, the server publishes a public key to its DNS record. The key is then sent in the DNS response from the server to the client. The ClientHello message sent by the client will now have the SNI extension replaced by an encrypted SNI extension that only the server can decrypt. Naturally, if the goal is encrypting SNI, this method will only be feasible by having encryption in the DNS protocol, therefore it is achieved using DoH, otherwise, we would have the encrypted SNI but the server name would be exposed to the network as unencrypted plaintext [16].

2.2.3.3 Encrypted Client Hello (ECH)

Encrypted Client Hello (ECH) is another extension to the TLS 1.3 protocol, and like ESNI it is only available in this version. Unlike ESNI, ECH encrypts the whole ClientHello message making all the TLS handshake encrypted. ECH is a work-in-progress born from ESNI in an attempt to close some security holes that are vulnerable to sophisticated attacks. Similar to ESNI, it also uses a public key via DNS, in this case through DoH for the first message exchange [30].

Figure 2.10 demonstrates the TLS 1.3 handshake with the ECH extension. As we can see, the CH message is divided into two different parts: unencrypted ClientHelloOuter (CHO) and encrypted ClientHelloInner (CHI).

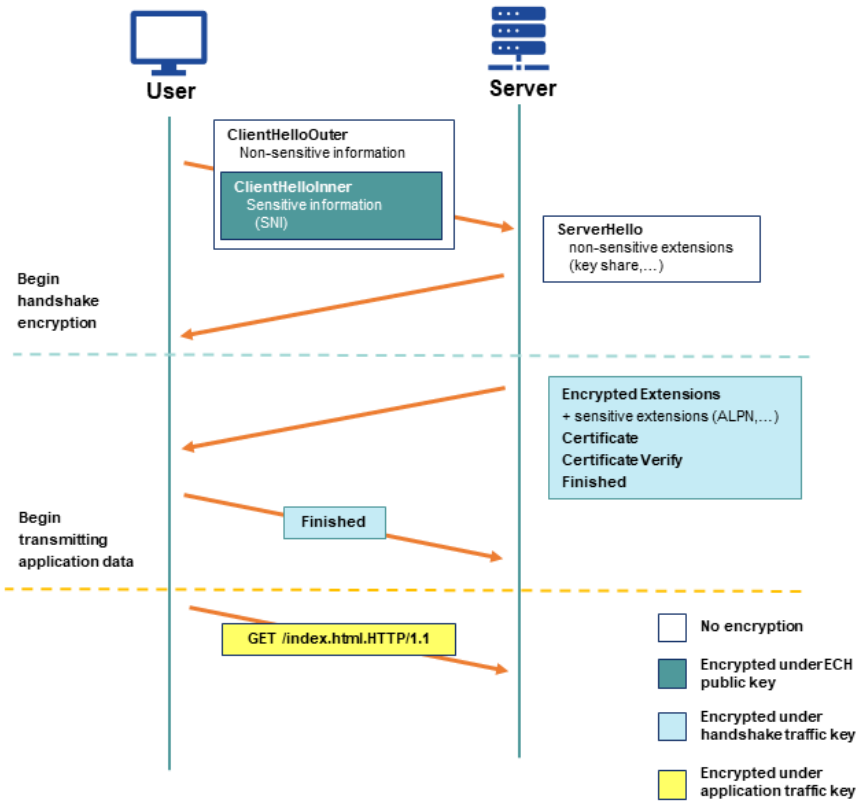


Figure 2.10: The TLS 1.3 handshake with the ECH extension. Adapted from [30]

It reveals to have improvements when compared to ESNI in terms of scalability, keeping the privacy of handshake parameters, connection establishment because it can retry to connect when decryption fails without stopping the handshake, and it has improvements to key distribution of the public keys making it more robust to DNS attacks.

2.2.4 Related Work

In order to know more about these new protocols and how they are changing the traffic over the Internet, there has been some work done in this area.

In [27] was studied how HTTP/2 is changing web traffic and the differences compared to HTTP/1. The authors presented an approach to detect the HTTP/2 encrypted traffic using machine learning algorithms and used only features found in NetFlow data. This study proved the differences between the protocols and how the new features of HTTP/2 result in distinct network fingerprints.

Not many studies have been made about QUIC protocol and how to detect its traffic. In [41], the authors implemented a classification method using convolutional neural networks. They captured five Google services: Google Hangout Chat, Google Hangout Voice Call, YouTube, File transfer, and Google play music. The results were very accurate but it isn't very reliable in a real-time classification as they consider it is necessary to have a larger dataset and investigate the features needed from the flows.

Shamsimukhametov et al. in [36] studied the impact that the Encrypted ClientHello has on the state-of-the-art traffic classification methods. This paper showed that ECH degrades the performance of the classifiers and presents new approaches to face these difficulties. The novel TC algorithms use the unencrypted part of the TLS Hello messages and the particular metadata of the handshake as features of the Random Forest algorithm.

2.3 Data Collection

One of the most important requirements and at the same time the most challenging one for TC is having a large and representative dataset. The task of collecting and labeling data is usually a hand-made job and very time-consuming. The lack of commonly accepted methods and automatisms for collecting and labeling data makes it difficult to build strong datasets. Also, the increasing number of applications and possible traffic classes makes it almost impossible to have datasets covering all traffic types. Privacy concerns are another obstacle to having real-world network traffic available to the public. Usually, the dataset is collected for a specific classification goal. For instance, for an ISP to control the traffic of their customers for free applications, it is more important to have a vast and representative dataset for those applications than other traffic not relevant for controlling operations.

Data collection location is very important because it affects the available features and labeling. It can happen at the client or server side, at any point of the network (the core of the network, edge of the network, etc.). Some popular packet capturing and analyzing tools are Wireshark, tcpdump, and tshark.

2.3.1 Features Extraction Selection

The collected raw data may contain noise and not relevant data, therefore a pre-processing phase of the datasets to clean the data may be interesting to perform. The goal of traffic classification is to attribute a label for each flow with the corresponding traffic class. A flow is a series of packets exchanged for a single application and usually, is determined

by having the same source IP, destination IP, source port, destination port, and transport protocol.

The choice of features to be used in the models for network traffic classification affects the accuracy, computational performance, and memory complexity of the classifiers. Since the majority of the traffic is encrypted, statistical based features are preferred for feature extraction of the packets.

The most common categories used in the TC models are:

- **Statistical Features:** Packet length, inter-arrival time, and direction of consecutive packets. Some studies and works achieved good results using only the first few packets of the flows.
- **Header:** All useful header fields (typically L3 and L4 information, when unencrypted).
- **Payload Data:** Content of the packets (hard to use with encrypted traffic). Some studies have achieved good results using only the packets from TLS handshake that are exposed.

In [44] the authors proposed a TC model for encrypted traffic using only the TLS handshake packets. During the handshake process of TLS, some information is exposed and not encrypted. The content of these packets was used as input features to construct a Bayesian neural network deep learning based classifier. This work refers to some limitations with the use of statistic-based approaches when network and application conditions change.

2.3.2 Data Generation: Automation

Aiming at our goal of monitoring the classification of mobile application traffic in an ISP using automation techniques, throughout this section we will present some of the tools and software that will be used for the automation of mobile application data generation, as well as the respective verification of the classification accuracy by the core of the provider's network.

The collection of the traffic to train and test the classifiers can be done in different ways. The most common procedures are:

- Real traffic acquisition by real users from the network
- Traffic generation that tries to simulate real traffic conditions through scripts
- Emulated traffic that sets scenarios as close as real ones for intentional emulated interactions in the network

Since the generation of the data is essential to test the accuracy of the classifier, there is a need for automation in traffic generation. The manual generation of mobile applications data requires a lot of time and human effort and is error-prone. Some open-source

tools can help alleviate these difficulties. These tools simulate the human use of the applications and table 2.2 lists some of them pointing out their strengths and limitations.

Table 2.2: Comparison between mobile automation tools, adapted from [42]

Tool	**Strengths**	**Limitations**
Monkey Runner	Coordinated based automation. Easy to setup.	Supports only Android. Requires different scripts for different devices which can break with slight changes in the UI.
UI Automator	Integrated with Android development IDE. Object based automation.	Supports only Android. No support for web-view, long press.
Monkey Talk	Open source. Eclipse based IDE. Supports iOS & Android apps.	Limited features in free version.
Appium [5]	Element based Navigation. Large Open source community.	iOS Apps requires source/debug binaries. No image based navigation.
GUITAR	Open Source. Supports iOS & Android apps. Image based navigation.	Small open source community. No element based navigation. iOS requires rooting of device.

2.3.2.1 Appium

Appium is an open-source tool for mobile application automation that is compatible with Android and iOS. Among the tools listed in table 2.2, we chose Appium as it seems to be the most suitable for the test cases we wanted to cover in some specific applications, and a big advantage of using Appium is the fact that it has great integration with the Robot Framework allowing more control over testing. The role of this tool in our work is to automate some use cases in different mobile applications in order to generate real network traffic through real cell phones, trying to make these automated tasks as similar as possible to human behavior.

Appium is based on a Client-Server architecture composed of three components: Appium Client, Appium Server, and the mobile device under test.

We can write our scripts in any programming language as long as it has an HTTP client API, offering flexibility to developers to use the language they are most comfortable with. There are a few Appium client libraries already implemented (in Python, Java, Ruby, PHP, C#, and JavaScript). The Appium server is written in Node.js and can run on a local machine or in the cloud. The server receives requests through JSON Wire Protocol sent by the Appium client libraries (the script/code) and then executes the commands on a mobile device. The server sends an HTTP response with the result of the command execution.

Figure 2.11 shows the architecture of Appium and illustrates its high level functioning.

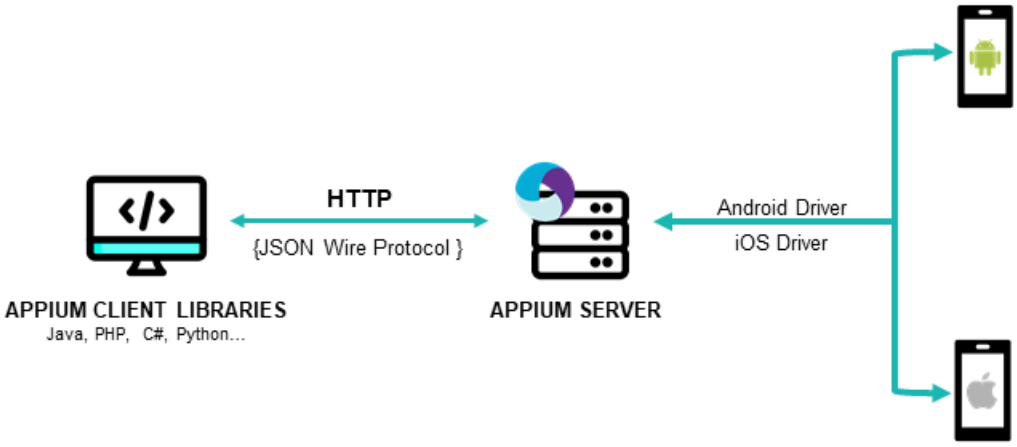


Figure 2.11: Appium architecture

2.3.2.2 Android Debug Brigde (adb)

Android Debug Brigde (adb) is a command-line (CLI) tool that can be used to interact with Android devices. It is at the lowest level of interaction we can have with the devices as opposed to Appium, since we can have access to a Unix shell where we can send commands directly to the device.

Similarly with Appium, it has a client-server architecture composed by three components:

- **Client:** Who sends the commands. Runs in the host machine.
- **Daemon:** Executes the commands on the devices. Runs in the devices as a background process.
- **Server:** Establishes and manages the connection between the client and the adb daemon. Acts as proxy between the client and the daemon. The server runs as a background process on the host machine.

Figure 2.12 shows the architecture of ADB.

2.3.2.3 Robot Framework

Robot Framework [35] is an open-source automation framework. It is mainly used for test automation and robotic process automation (RPA). It was inspired by Pekka Klärck's Master's Thesis [21] and was initially developed at Nokia Networks in 2005.

Robot Framework is implemented using the Python programming language and offers great freedom of functionality. It has a simple syntax utilizing human-readable keywords, keeping the programming and understanding of the scripts easy.

There is a rich array of existing libraries already integrated with RF implemented in Python, Java, among others. The real power of RF is the fact that it is open and extensible

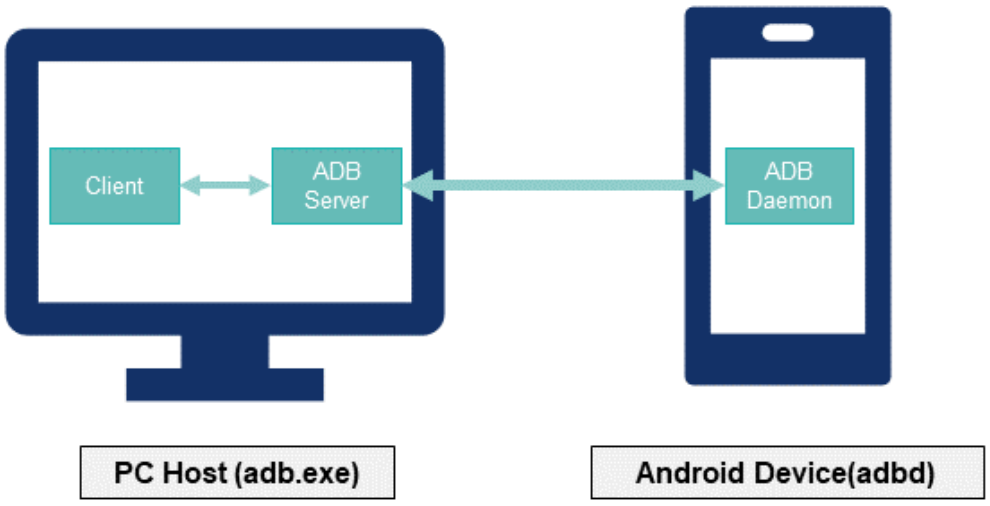


Figure 2.12: adb architecture

to implement and create suitable libraries and functions for the needed tasks, and it is quite simple.

Robot Framework provides good reporting analysis. Each test performed by RF generates files with detailed information (step-by-step) about the whole execution in HTML format, making interpretation easy. Provides statistics about the entire test (total test cases, passed, failed, skipped) and messages in case of error. All these features make this tool very powerful in the context of automation and in the case of our work, it reveals to be really strong when integrated with Appium for the mobile automation purpose, and Jenkins for CI/CD (continuous integration/continuous delivery) paradigm.

2.3.2.4 Jenkins

We introduced above some of the tools that can be used to create scripts that can automate the tasks of generating data and check the classification of mobile applications by the network core. Since we want to build a system that is capable of monitoring real-time traffic classification continuously over time, we need to have all the tasks related to building, testing, and deploying automated and running periodically. Jenkins can help to achieve these objectives.

Jenkins is an open-source automation server very useful for continuous integration and continuous delivery (CI/CD) and offers the possibility to schedule the execution of our tests in many ways. A vast array of plugins are available in Jenkins including a Robot Framework plugin making this tool interesting and suitable for our work.

3

Network Architecture

The system was implemented in a 4G - LTE laboratory network at NOS SGPS that is a replica of the network deployed in the production environment.

In this chapter we will give an overview of the used network architecture as well as an explanation of how 4G networks work and the evolution of mobile networks. We will also talk about the configurations done as part of this work and explain how traffic classification is performed in the network core.

3.1 Evolution of Cellular Mobile Networks

The first generation of mobile telecommunications networks, 1G, appeared in the 1980s. The systems from this generation were analog-based using FDMA (Frequency Division Multiple Access) technology and only supported voice data communications. These networks had security problems since they did not support information encryption allowing third parties intrusion.

The second generation, 2G, appeared in the 1990s. The systems were fully digital and made data and voice services available for the first time. These services were based on TDMA (Time Division Multiple Access) digital radio technology. This generation came to address the major security issues present in 1G networks. GSM (Global System for Mobile Communications) is the most popular example of the 2G technologies.

There was a transition generation between 2G and 3G networks that was commonly called 2.5G. Some popular services like SMS (Short Message Service) and GPRS (General Packet Radio Service) were introduced in this generation. These advances served as the basis for the implementation of 3G.

In GSM, the architecture relies on circuit-switching (CS), so the establishment of circuits between the two points of communication (caller and called) is necessary. The packet-switching (PS) was added to GPRS technology, so the core network is composed of two domains: circuit and packet. With PS, data is transported in packets, unlike GSM where it is necessary to establish dedicated circuits.

In the early 2000s the third generation of mobile communication systems, 3G coordinated by the 3GPP (3rd Generation Partnership Project) organism was launched. It brought great transformation to the world of mobile communications and much is due to the development of the WCDMA (Wideband Code Division Multiple Access) access technology. The principal objectives of 3G networks were to be able to increase data transfer rates and support a wider range of services like multimedia applications. One of the most important advances of this generation was the implementation of UMTS (Universal Mobile Telecommunication System).

Figure 3.1 shows the evolution of mobile communication network architectures in a very high-level way.

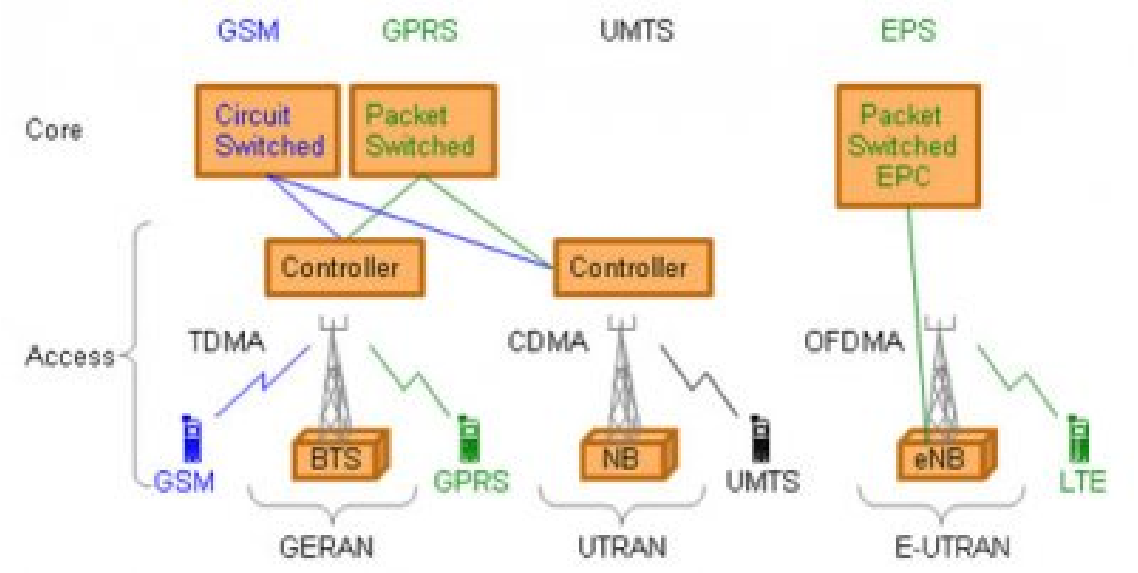


Figure 3.1: Evolution of mobile communication network architectures. Adopted from [1]

After 3G came 4G and 5G, which are the most emerging technologies today. The rest of this chapter will be dedicated to 4G networks since it was on this network that the project was developed.

3.2 4th Generation Networks

The first decade of the 21st century saw an exponential increase in the use of cellular mobile networks, leading to high demand and a need to improve these networks. Some of the main reasons that motivated the development of the 4th generation of mobile communication networks, 4G, are presented below:

- High demand for higher data rates and quality of service
- Low complexity and cost reduction
- Interoperability and easy roaming

- Scalability

The 3GPP community launched two working groups for the development of 4G systems: Long Term Evolution (LTE) for the radio access networks development and the System Architecture Evolution (SAE) group for the core network development.

For the access network was defined the Evolved UTRAN (E-UTRAN) and for the core network the Evolved Packet Core (EPC) that we will explore in more detail below. These two technologies form the Evolution Packet System (EPS) which is what we commonly call 4G or LTE.

3.2.1 The Architecture of 4G Networks

One of the big changes introduced with 4G is that was decided to use IP (Internet Protocol) as the key protocol to transport all services. It was also decided that the core would not have the circuit-switched domain. These implementations led to changes in the architecture and services that needed to be replaced for IP-based solutions.

Figure 3.2 shows a high-level network architecture of LTE networks. In comparison with the previous generations, the goal was to have a "flat architecture", reducing the complexity and the number of network nodes.

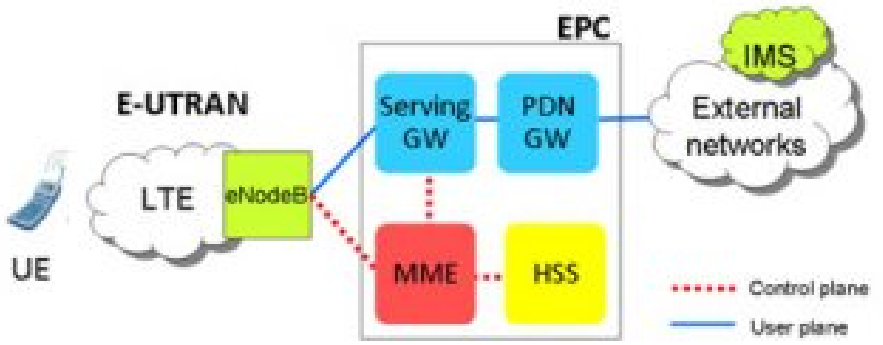


Figure 3.2: High-Level 4G Network Architecture. Adopted from [1]

Another feature added in 4G was the separation of the user plane (user data) and control plane (signaling). This offers more freedom for operators to make changes and adapt their networks easily. Therefore, allows scalability.

The devices, usually called User Equipment (UE) connect to the base stations (eNodeBs) via radio signals. The base stations transmit information between the mobile and the core network through IP packets.

3.2.2 Evolved UTRAN (E-UTRAN) - LTE Access Network

Evolved Universal Terrestrial Access Network (E-UTRAN) is the access part of the Evolved Packet System (EPS). The E-UTRAN handles the radio communications between the mobiles and the core.

To achieve the goal of having high data rates, the access solution is based on OFDMA (Orthogonal Frequency Division Multiple Access) for downlink and SC-FDMA (Single Carrier – Frequency Division Multiple Access) for uplink.

The LTE access network is basically a network of base stations, Evolved NodeBs (eNBs). Unlike 2G and 3G, there is no controller node between the eNodeBs and the EPC. The various eNBs are interconnected with each other through X2 interface. The eNBs are connected to the core (EPC) via the S1 interface.

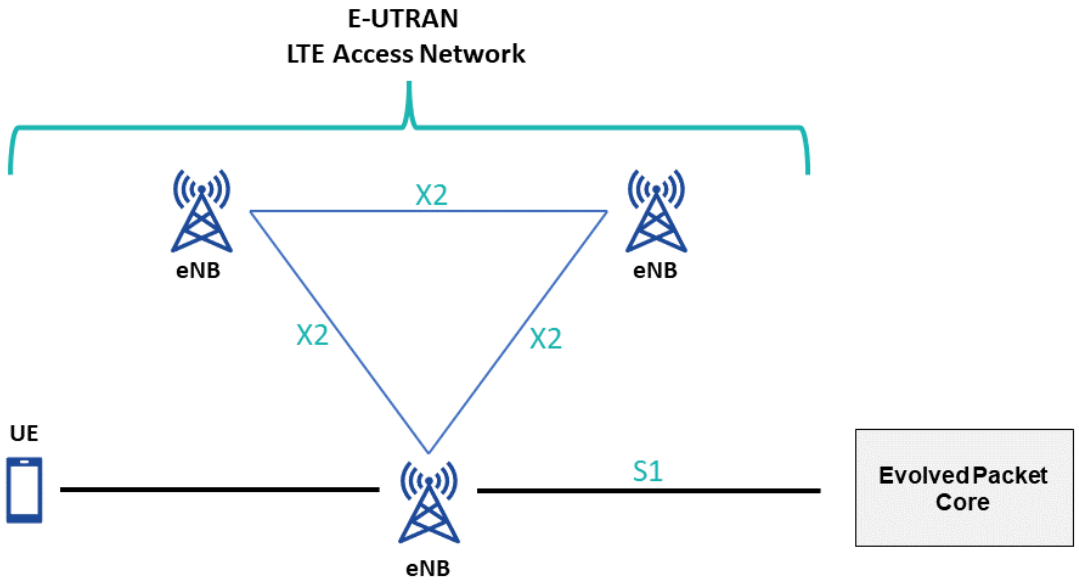


Figure 3.3: E-UTRAN - LTE Access Network

3.2.3 Evolved Packet Core (EPC)

The Evolved Packet Core, or EPC, is the core network of the 4G/LTE networks. It deals with the routing and computing part of the 4G network.

For our work, this is one of the most relevant parts since we are monitoring the traffic classification performed in the core of one ISP.

Figure 3.4 shows us the architecture of the EPC.

As mentioned before, the E-UTRAN is connected to the core via the S1 interface, more specifically, we can split into S1-MME for control plane data transfers between eNodeBs and MMEs, and S1-U for user plane data between eNodeBs and S-GW.

Figure 3.5 shows the protocol stack between the eNodeBs and the EPC.

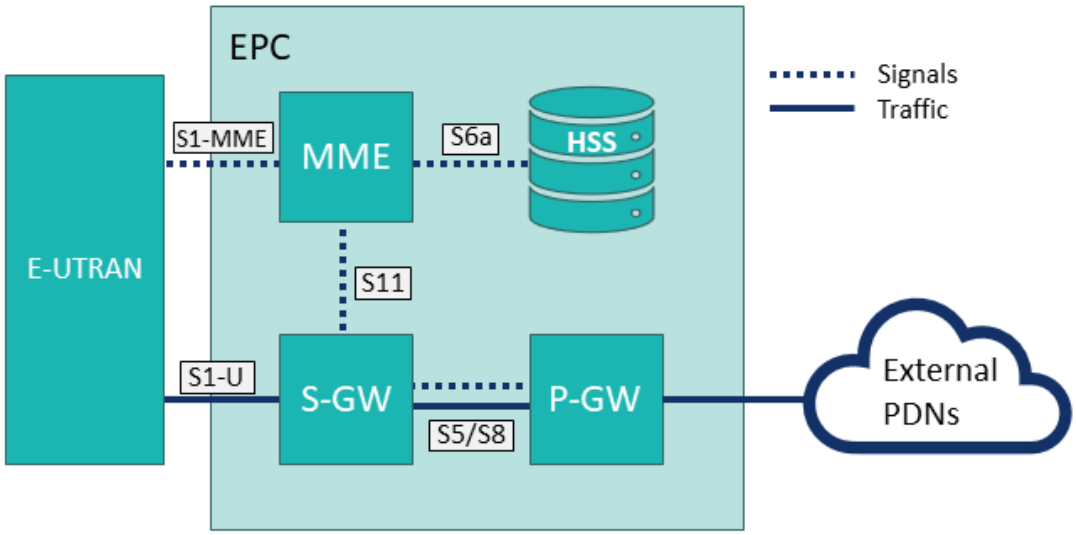


Figure 3.4: EPC Architecture

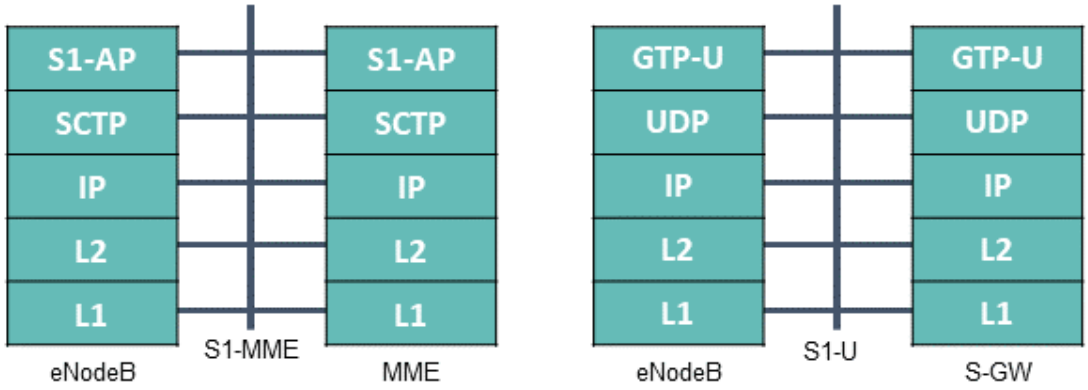


Figure 3.5: Protocol Stack of S1 interface

Below we explain in more detail each of the nodes that compose the core and some of their functions.

3.2.3.1 Mobility Management Entity (MME)

The MME (Mobility Management Entity) deals with the control plane. It handles the mobility and security questions for E-UTRAN access. It is also the MME that selects which S-GW to use.

S1-MME signaling data has the following main procedures:

- Bearer-level procedures
- Handover procedures

- NAS signaling transport
- Paging procedure

The S6a interface is used for the MME to retrieve information from Home Subscriber Server (HSS). Some of the procedures over the S6a interface are the following:

- Location Management
- Subscriber Data Handling
- Authentication
- Fault Recovery

3.2.3.2 Serving Gateway (S-GW)

The Serving Gateway (Serving GW or S-GW) deals with the user plane. It acts as a router forwarding the data between the UEs and the external networks (incoming and outgoing IP packets). It is connected with the other gateway, the P-GW.

A tunneling protocol called GPRS Tunneling Protocol (GTP) is used over the S1-U interface connecting the eNB to the S-GW, and over the S5/S8 interface connecting the S-GW with the P-GW. It provides user data encapsulation.

3.2.3.3 PDN Gateway (P-GW)

The Packet Data Network Gateway (PDN GW or P-GW) is the point that connects the EPC to the external networks (also called Packet Data Networks), either 3GPP or non-3GPP networks. It routes packets to and from external IP networks. It is also responsible for IP address allocation.

Some of the functions of the P-GW are the following:

- Policy Enforcement
- Packet Filtering
- Charging Support
- Lawful Interception

As these functions suggest, this node is where the traffic classification in the core network takes place. In the implementation of this work, we interacted directly with the P-GW to access the core's classification for the mobile applications.

The classification is achieved with the use of Deep Packet Inspection (DPI) techniques as mentioned in 2.1.2. We will explore in more detail how the DPI works in 3.4.

Each packet data network is identified by an access point name (APN), which we will also mention in 3.3 as we configured one dedicated APN for the implementation of our work.

3.2.3.4 Home Subscriber Server (HSS)

The Home Subscriber Server (HSS) is a database that stores user/subscriber-related information and provides support functions, mainly for the MME node via the S6a interface.

Some of the HSS support functions are the following:

- User Security support
- User Identification handling
- Mobility Management
- Access/Service authorization and provisioning support

Having the user information centralized in this single node allows the separation of signaling from the rest of the traffic which leads to a high-performing network.

3.2.3.5 Interfaces of the EPC: Protocol Stacks

In order for the different nodes to communicate with each other and transport user and control plane data, there are some interfaces connecting them. Figure 3.6 shows the protocol stacks of the interfaces inside the LTE core network.

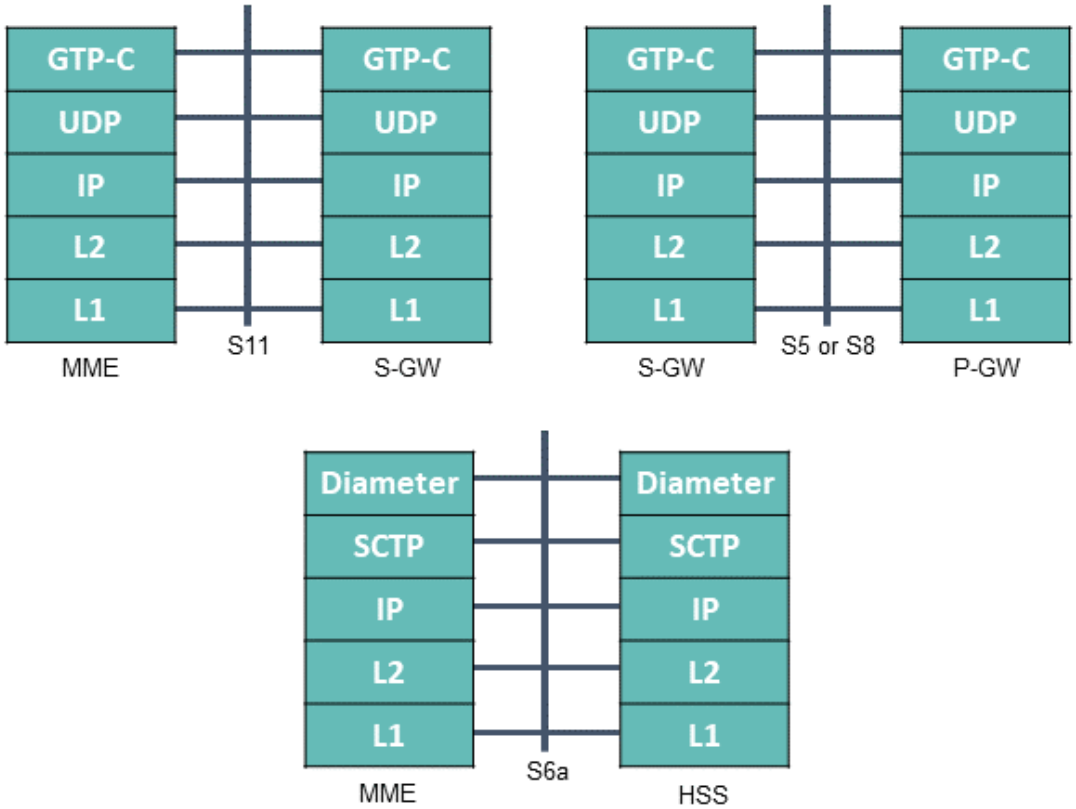


Figure 3.6: Protocol Stacks of core interface

3.3 Access Point Name (APN): Configuration

The access point name (APN) is a gateway between a mobile network and another network, such as the Internet. In a 4G network, it identifies the external Packet Data Network (PDN) that the user wants to access and determines the P-GW to use. Figure 3.7 helps to understand the APN functionality.

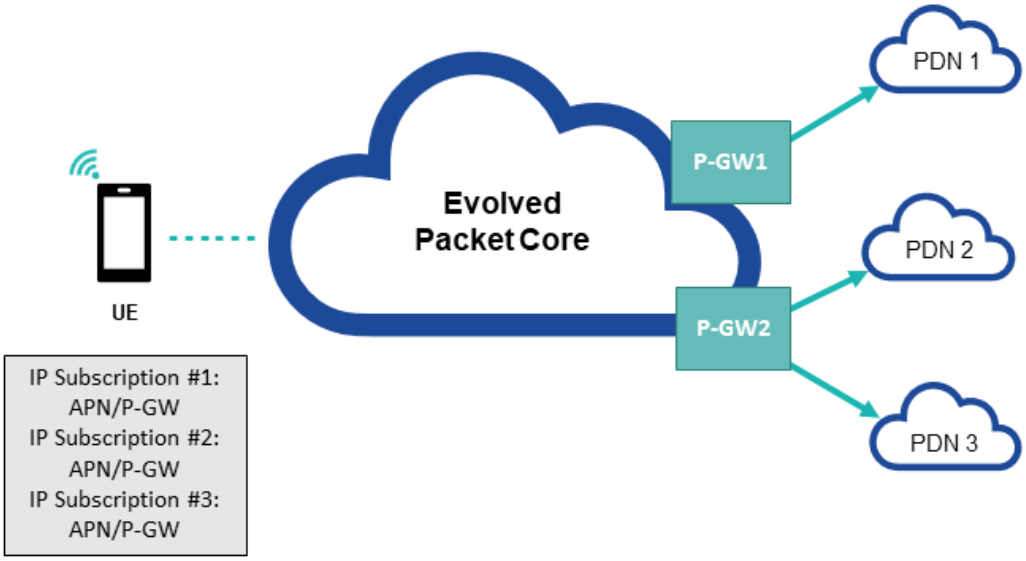


Figure 3.7: Relationship between APNs and P-GWs

The information about the user subscription and also about the PDNs to which the user can connect are stored in the HSS. In the case of the 4G network we were working on, this information is stored in the form of an APN.

One of the most important functions of APN is that it allows mobile operators to characterize the user session for that user when accessing the external PDN. Some functions of APNs are the following:

- Authorization
- Methods for address allocation
- Timeouts
- Charging type
- Policy Model (for example, if a policy and charging rules function [PCRF] is used for policy control)

Besides the APNs being defined by the operators, the user must have the APN selected on his UE. Therefore, for our work, we created and configured one dedicated APN in

the core network for the tests that we performed, and at the same time, we needed to configure this same APN on the devices that we used for the tests.

3.4 Core network classification: Deep Packet Inspection (DPI)

The objective of this work was to monitor the core classification of popular mobile applications. In this section, we will explore how the classification is achieved in the core of an ISP.

The classification of the traffic is performed at the P-GW node in the Evolved Packet Core, and it is based on packet inspection approaches, in this case, DPI techniques. The mechanism of the DPI techniques is based on the inspection of the various layers of the packets and analyzing the individual protocol fields and states. We can divide the packet inspection into two different types:

- **Shallow Packet Inspection (SPI)** - Inspection of the layer 3 (IP header) and layer 4 (TCP, UDP, etc. header) information.
- **Deep Packet Inspection (DPI) –** Inspection of layer 7 and 7+ information.

Once we are monitoring the traffic at the core level, the inspection of layers 3 and 4, in some cases, reveals to be insufficient, so layer 7 examination is required. One of the reasons why it is not enough is because of the encapsulation of upper layer protocol headers, which is the case of the LTE traffic over the core network with the IP-over GPRS.

It is only possible to know the true destination of the traffic in the case of terminating proxies with layer 7 examination since SPI is only capable to retrieve information about the destination IP address/port number of a terminating proxy.

For mobile application identification, is very important to access information that is only available at layer 7, which is the case of URI (Uniform Resource Identifier) information.

Figure 3.8 gives an overview of the information that the DPI can retrieve from the packets layer by layer.

3.4.1 How DPI works

The system uses Access Control Lists (ACLs) to redirect the subscriber traffic for inspection. These ACLs are configurable and are applied according to the APN profile parameters of the subscriber. The traffic can be inspected or simply forwarded, depending on the rules of the ACL. Figure 3.9 demonstrates the ACLs behavior. That is why we choose to use a dedicated APN for the development of this work.

The classification is made for a specific user session. A user session refers to the time interval in which the user is transferring and sending data over the mobile network. In practice, and so that the sessions are as controlled as possible for the tests developed in

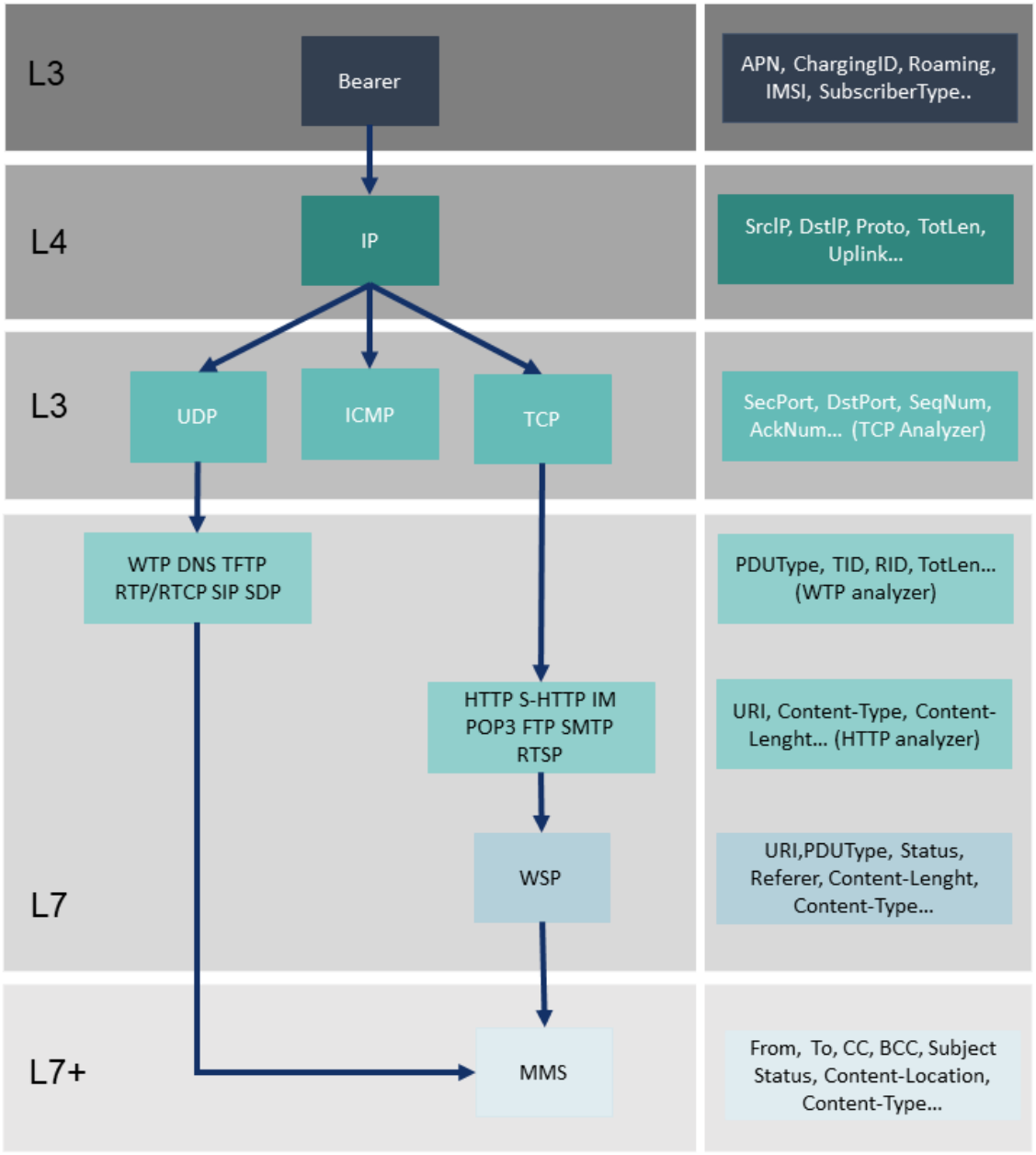


Figure 3.8: DPI layers analysis

this work, the session is started and finished with the turning on and off of flight mode on the UE. Therefore, with the terminal in flight mode, when flight mode is turned off, the session begins and ends when flight mode is put back on.

The examination is based on regular expression (regex) rule matching. There are rules defined (ruledefs) that are going to be compared with the incoming packet information in the DPI system. Then, if there is a match with the ruledefs, it takes the necessary actions to the packet. These actions also include the generation of records of the subscriber's session for billing purposes. For our work, we will use this Call Detail Records, CDR file, for the validation of the core classification (section 4.4.2).

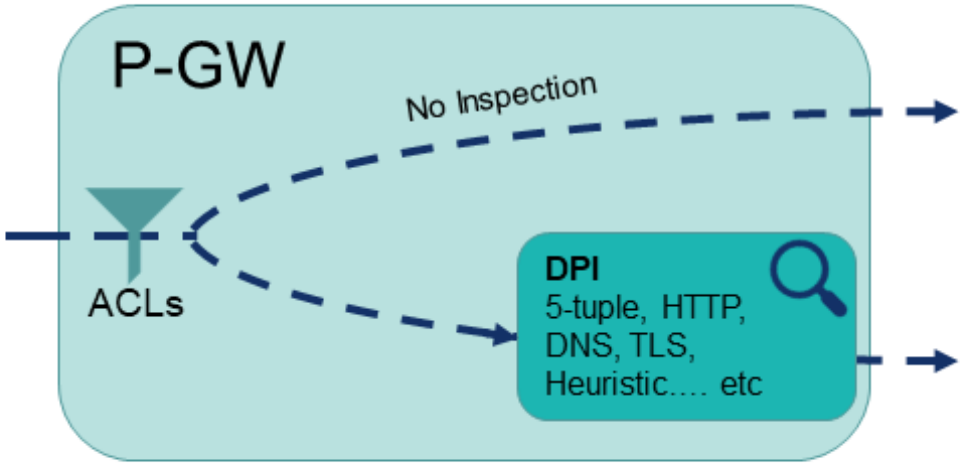


Figure 3.9: ACL filtering

Figure 3.10 exemplifies the behavior of DPI.

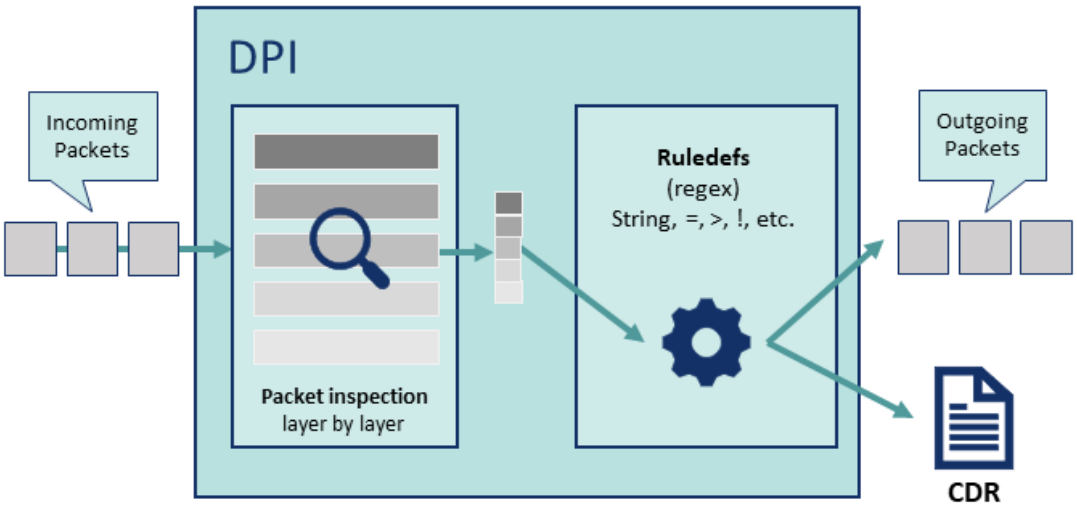


Figure 3.10: DPI behavior

We can summarize the DPI behavior in steps. The main steps in the DPI classification are:

1. Redirection of packets based on ACLs according to the user's APN.

2. Packet inspection layer by layer to retrieve protocol fields information (retrieved by analyzers of each of the supported protocols as we can see in figure 3.8).

3. Check for a match in the ruledefs for fields retrieved from the packets (regex).

4. Execution of the actions defined in the matched ruledefs and generation of the CDR. Also, no match can occur and no action will be applied to the packet at all.

3.4.1.1 Call Detail Records (CDR)

As mentioned before, the classification by the core is referent to a user session. Therefore, when a user realizes a data session (since flight mode is turned off until it is put back on), one of the actions of the DPI engine, is to generate records of that same session. We call it the Call Detail Record or CDR. We can generate a file with all the record information about that session.

The CDR can contain a lot of information for one session. For our work, the relevant ones are the number of bytes that the DPI identifies as a certain application or group of traffic, according to the rules that were defined.

With the mobile plans offered by the ISPs, it is necessary to differentiate not only types of traffic (for example video, gaming, etc.) but also individual applications. So, when defining the ruledefs for traffic identification by the ISP, it is necessary to have different IDs for every application that we want to have special billing characteristics. This way, we can look at the CDR file generated for one session, and know how many bytes were consumed in different applications or different traffic categories.

4

Autonomous Monitoring System: Implementation

The purpose of this work is to have a system capable of monitoring the classification of mobile applications by the core network, automatically, in this case, at NOS SGPS. For an Internet Service Provider like NOS, besides knowing which application generated the traffic, it is often necessary to know the amount of data consumed when using this same application for customer billing purposes.

For the monitoring purpose, we want to know if the classification is being well performed as well as the amount of data consumed. Therefore, it would be interesting if we could compare the core's classification with other reliable data. Since we want to monitor the network traffic classification from mobile applications, the proposed solution was to measure the amount of data consumed by the applications directly on the cell phone and compare it with the result given by the core.

In this chapter, we will discuss the various parts that make up the developed system.

4.1 Overview

The operation of the autonomous monitoring system is based on testing. The goal is to monitor the accuracy of the core classification for some mobile applications, so we developed tests for that applications to validate if the classification is being well performed. The tests were designed to run automatically and periodically without the need for human interaction.

The main features that the implemented system has are the following:

- Launching of the tests for each application under study
- Report generation about test execution
- Calculation of the accuracy of the core classification
- Database storage of the tests results
- Dashboard visualization of the accuracy obtained by the tests over time, by application

- Sending email reports about the results obtained in the core classification accuracy tests

Figure 4.1 gives us an overview of all the parts that compose this system. Throughout this chapter we will explore each part of this system in more detail.

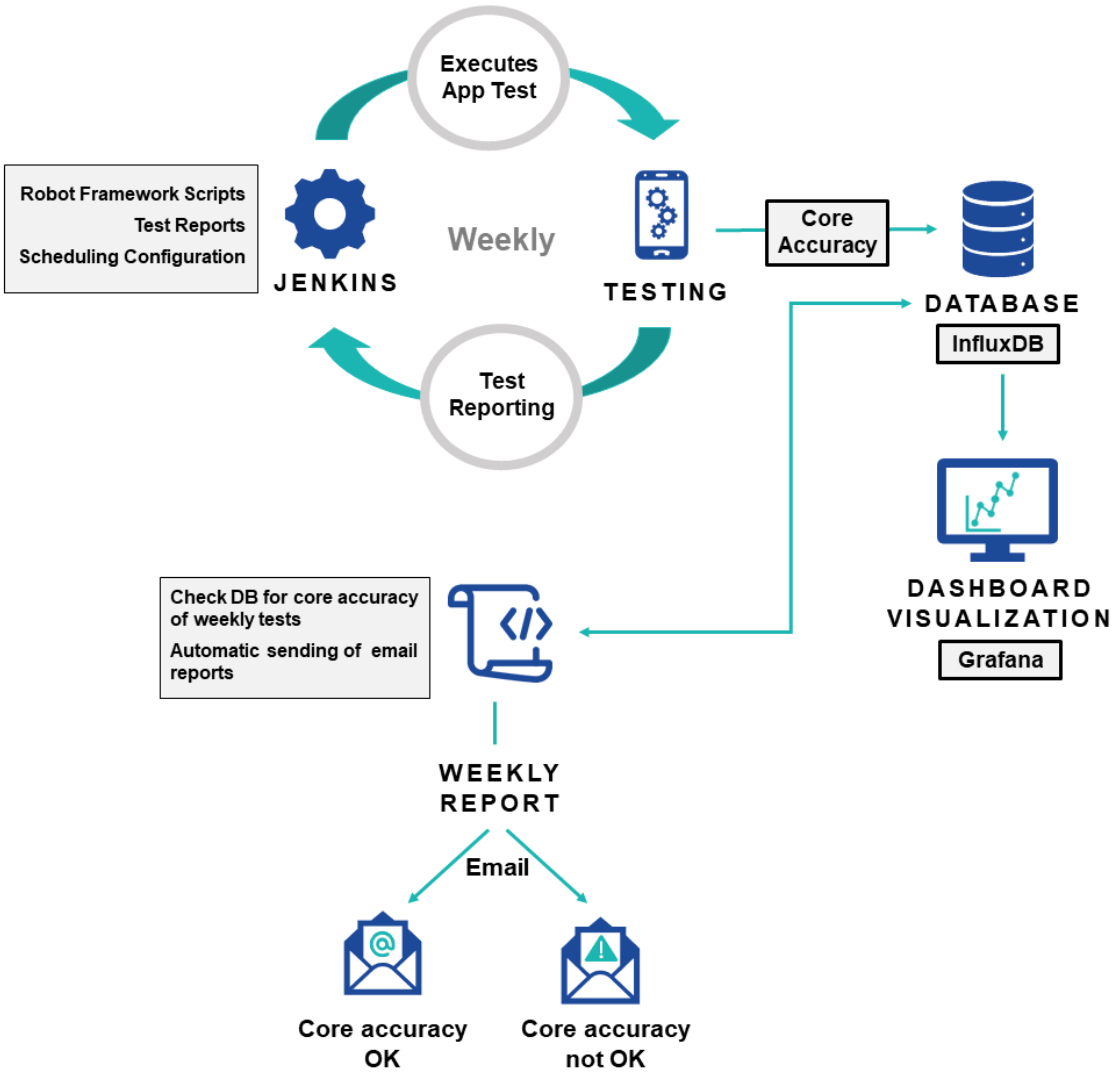


Figure 4.1: Autonomous Monitoring System overview

The tests are executed periodically, as was already said before. Since we are working with devices and core network nodes that are in a laboratory environment, we decided to run the tests every Saturday during the morning period in order to avoid work and tests from other colleagues and other teams.

The reporting email about the results of the weekly tests is sent every Monday to inform the team of the accuracy achieved by the core. In parallel, there are graphs showing the test results for each application over time.

4.2 System Environment

As presented before, the system has many features. In this section, we will show the environment of the system and how it was built to achieve all those features.

The system is deployed in an environment of multiple virtual machines. Figure 4.2 demonstrates the interconnections of the pieces that compose the system and that need to be integrated.

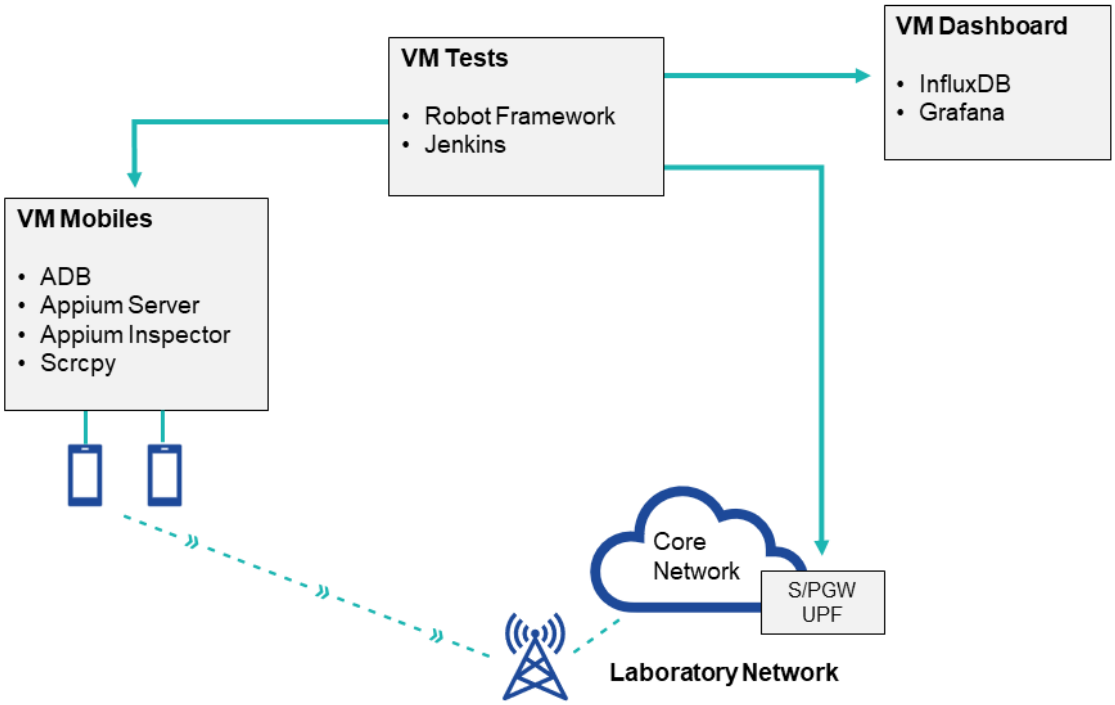


Figure 4.2: System Environment

All these pieces are inside the NOS network and in secure virtual machines. All these virtual machines run Linux-based operating systems.

The system needs to interact with the phones to get network information and navigate through the applications, needs to access the core network to get information about the classification, and also has to interact with the database.

Robot Framework is responsible for integrating all the parts and is the orchestrator of the tests.

4.2.1 Integration with Robot Framework

Robot Framework is installed and running in a virtual machine and has to interact with the other parts of the system that are not on the same machine, as we can see in figure 4.2.

In this section, we will explain how we handle the integration of all the parts of the tests using RF.

4.2.1.1 SSH with Robot Framework

A very useful way to access remote machines is the use of the SSH protocol. This protocol can be used to log into a remote machine and execute commands or run scripts.

SSHLibrary [39] is a Robot Framework library that supports multiple SSH connections to different machines and allows to automate the access to different hosts and perform the necessary actions on those hosts.

Figure 4.3 shows a simple example of how RF uses the SSHLibrary keywords to connect to remote machines and execute commands.

```
*** Keywords***

Connect to remote machine A
    Open Connection     IP_rmA   alias=rmA
    Login               username_rmA        password_rmA

Connect to remote machine B
    Open Connection     IP_rmB   alias=rmB
    Login               username_rmB        password_rmB

Do something
    Switch Connection   rmA
    #Do something on mobiles VM
    Execute Command    echo 'Hello World from remote machine A'

    Switch Connection   rmB
    #Do something on core node
    Execute Command    echo 'Hello world from remote machine B'
```

Figure 4.3: SSHLibrary example

Since during the tests we need to access different hosts, this library helps to automate and overcome that difficulty and allows us to have one single script capable to perform the entire test.

4.2.2 Mobiles Virtual Machine setup

Once we are monitoring the classification of mobile applications, one of the most important pieces of the system is the interaction with the mobile devices. Table 4.1 shows the characteristics of the phones used for the tests.

Table 4.1: Android Devices used for the tests

Vendor	Model	Android Version
Samsung	A12	Started with version 10 and upgraded to version 11 during the project development
Samsung	S21	12

4.2.2.1 USB Redirection

The devices are physically attached via USB to a server that hosts multiple virtual machines. To access the devices, the server is configured to redirect the data from the USB port where the devices are attached to the desired VM. In this case, we call it the Mobile's VM.

Figure 4.4 gives a simple overview of the environment.

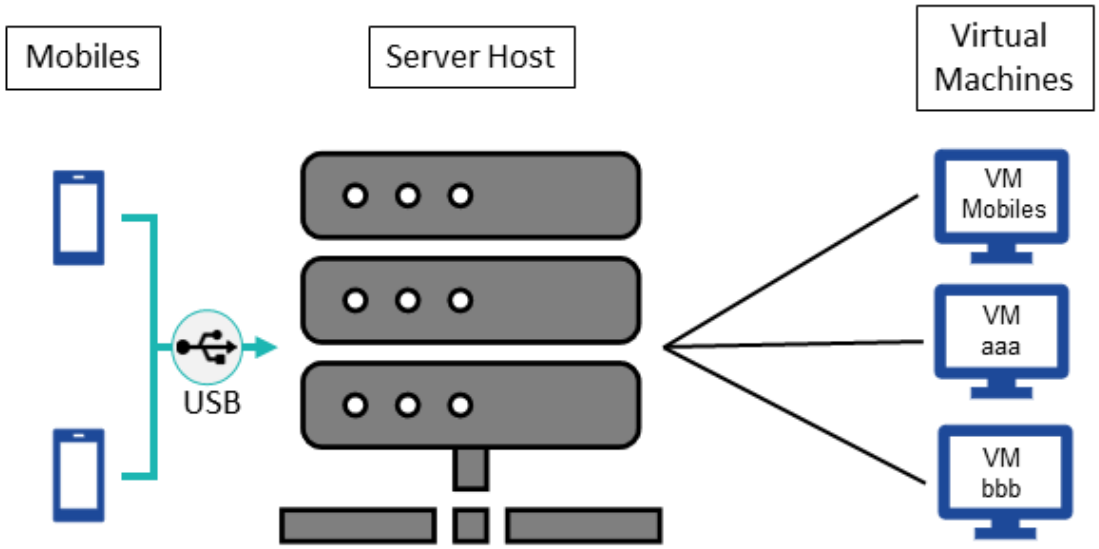


Figure 4.4: USB Redirection

4.2.2.2 Scrcpy (Screen Copy)

Scrcpy, an abbreviation for "Screen Copy", is an open-source screen mirroring tool that allows you to control an Android device connected via USB or over TCP/IP from a Windows, macOS, or Linux environment.

This application is installed in this VM to help troubleshoot the devices if needed, and also it is very useful to control and monitor the execution of the mobile application navigation tests during the development phase of the script tests.

4.2.2.3 ADB (Android Debug Bridge)

This Virtual Machine has the ADB tool installed to interact with the mobiles. We will explore later in this chapter the use of ADB on the tests.

As it was explained in 2.3.2.2, adb has a Client-Server architecture. The adb client and the adb server are installed and running in this VM.

Figure 4.5 shows the parts that compose the Scrcpy and ADB tools on the VM

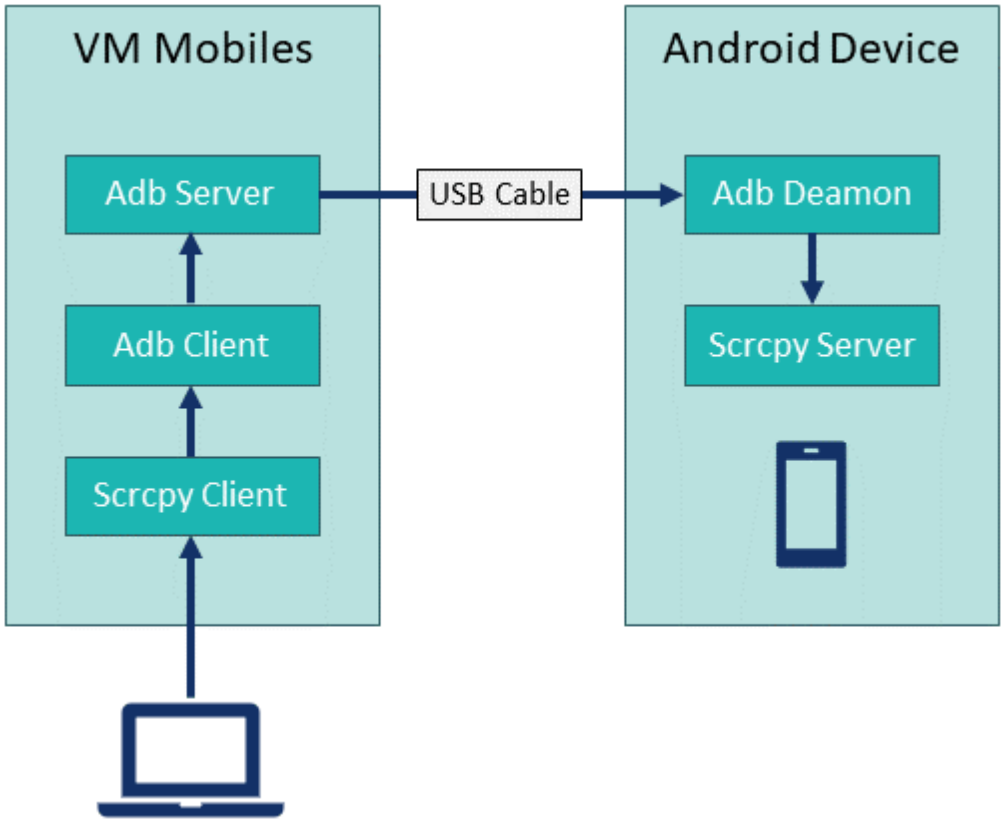


Figure 4.5: VM Mobiles

4.2.2.4 Appium

Another important tool for the testing phase is the mobile application automation that will be carried out with the use of Appium. Therefore, it was installed and configured in this VM.

Further in this chapter, we will explore the use of Appium for mobile application automation in this work.

Appium Server

As was explained in 2.3.2.1, Appium has a Client-Server architecture. In order to use two devices, we have two Appium Servers running in different ports on the VM, one for each android device. The first time Appium establishes a connection with the phone, it installs a service that keeps running on the mobile.

Appium Inspector

Appium Inspector was installed on this machine to help in the development phase of the RF + Appium scripts for the mobile application automation.

It works like an Appium Client and is used to inspect mobile applications and retrieve information about the elements on the screen.

4.2.3 Database and Dashboard Virtual Machine setup

As presented before, the system has a graphical visualization component to track the test results over time. For this, an InfluxDB database and a Grafana dashboard have been configured in this virtual machine.

4.2.3.1 InfluxDB

InfluDB is an open-source time series database and is very used for monitoring purposes. We installed and configured one InfluDB database that stores the accuracy values obtained in the tests.

4.2.3.2 Grafana

Grafana is an open-source platform very useful for monitoring. It allows us to see the data via charts and graphs and create dashboards. It has very easy integration with InfluxDB which makes these two tools very suitable for this work. So the data source for the Grafana charts is the InfluxDB database.

We have configured the Grafana server on this virtual machine and built a dashboard showing the results of the tests for each application.

4.3 Robot Framework: Test Flow

The operation of the system is based on weekly tests, as already mentioned. For each of the applications we are monitoring, a test is executed. In this section, we will show the flow of the developed tests.

The tests follow a step-by-step procedure orchestrated and controlled by Robot Framework scripts. The test scripts for each application are stored in the Jenkins workspace and they are executed from there.

Robot Framework is very powerful when we talk about automating tasks. In this work all the required tasks for the monitoring system were automated, thus, all the steps required for the tests are in the RF script. Figure 4.6 shows the flow of the tests. The test is the same for each of the applications to be monitored, only the navigation part of the application differs.

Before starting the test, the device must have the correct APN selected in its network settings. Since all tests, one for each application, are run sequentially, we have scheduled

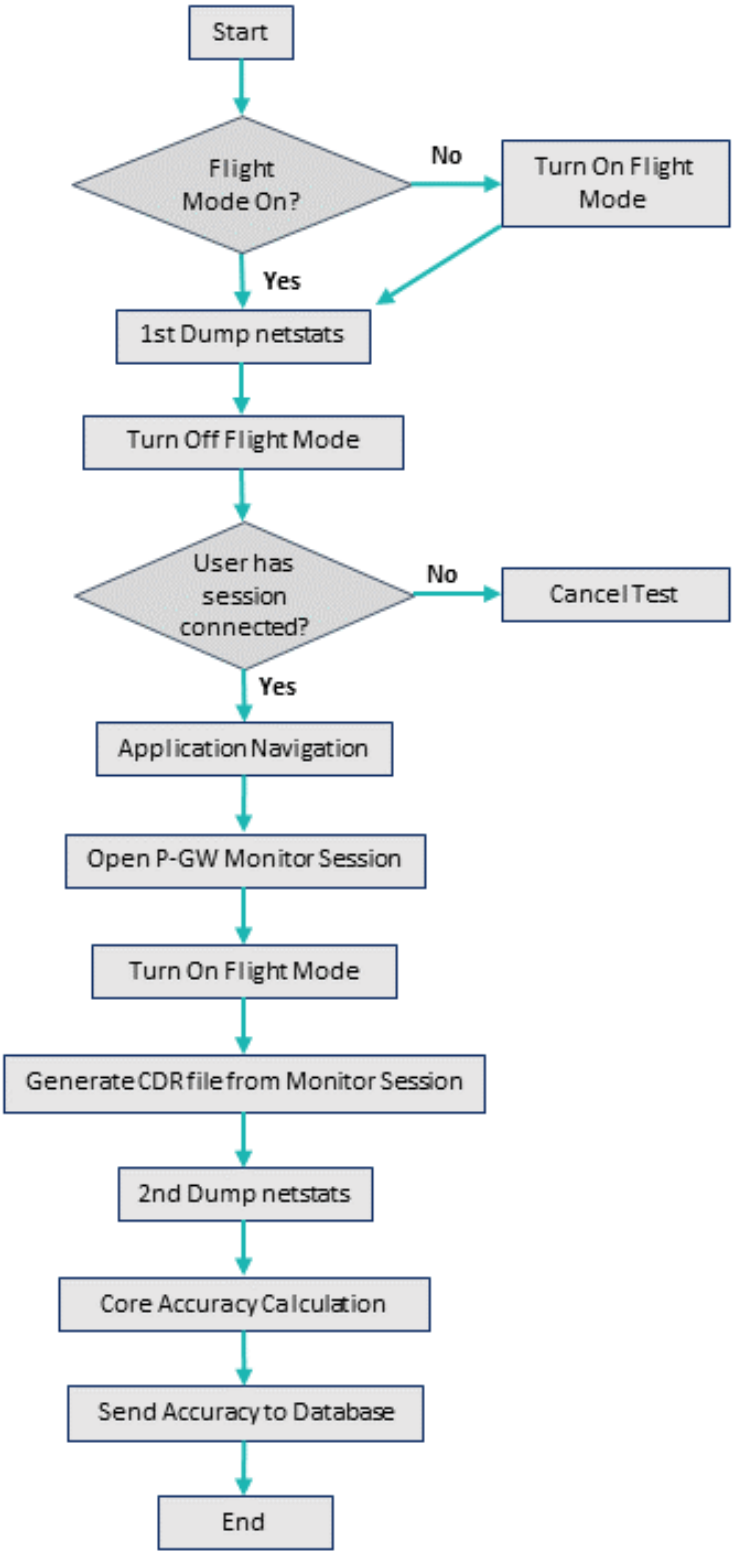


Figure 4.6: Test Flow

a script to run before the test execution to select the correct APN on the device, later in this chapter we will go into more detail about this script. We do this procedure to ensure that the phone has the correct APN selected because it may have been used for testing and the settings may have changed.

The test starts by checking if the phone has airplane mode turned on, if not, it turns it on. This is necessary to have the network statistics on the android controlled before the test, so we can accurately calculate the amount of data consumed by the mobile, and also to close any data session that may be open between the terminal and the core network.

When the airplane mode is turned off to proceed with the rest of the test, we validate if the user session was created in the core network. There could be some kind of problem with the network or some kind of intervention happening and the test would be unreliable. So if the session is not successfully created, there is no point in proceeding with the test, then it is canceled and marked as an error.

4.4 Accuracy of the core classification: Android VS DPI Core

As explained above, the chosen method to validate the core classification is knowing directly on the cell phone the amount of data consumed by the applications and then comparing it with the classification given by the core network.

The methodology used in each test is always the same regardless of the app being tested, the only difference is obviously the app being tested, and therefore, its use cases. Figure 4.7 demonstrates in a simple way the flow of the implemented tests. For this work only android devices were used, so the analysis of data consumed on the mobile side is done using the ADB tool.

We assume that the real amount of consumed traffic is given by inspecting the network statistics on the mobile, so the accuracy of the core classifier is measured according to the following expression:

$$Core\ Accuracy\ (\%) = \frac{Core}{ADB} \times 100 \tag{4.1}$$

Therefore, as it is shown in this expression, the tests produce a result in the form of a percentage. The closer the core value is to the mobile's value, the more accurate the classification is. The core classification can be incorrect from two different perspectives according to the value calculated by the core:

- If the total number of bytes given by the core is lower than the ADB value, the percentage of the accuracy will be a value below 100%, representing a lack of classification.

- If the value calculated by the core is higher than the ADB value, the percentage will be above 100%, representing an excess of classification.

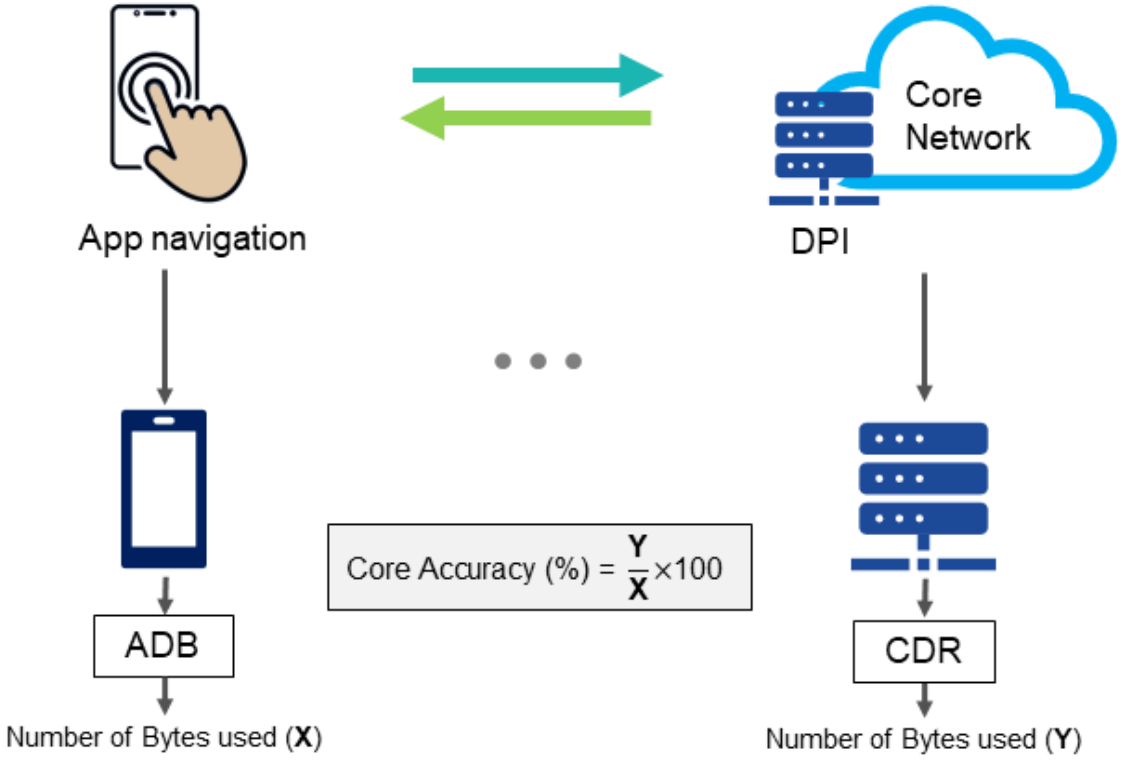


Figure 4.7: High-level flow of the tests for the core accuracy calculation

Both cases present an error in the classification. The error percentage can be calculated with the following expression:

$$Error\ (\%) = \left|100\ \% - Core\ Accuracy\ \%\right| \tag{4.2}$$

We opted to calculate the accuracy of the core classification without limiting the value to 100%, in order to be able to visually differentiate whether the classification is getting an error by excess or by lack.

4.4.1 Network statistics from Android

In order to validate the classification of the core network, the solution we used was to measure directly on the device the amount of data consumed by the application we are testing. We will assume this as the real value that we will use to compare with the core classifier.

Since we only used Android devices, the inspection of the network data is made using the ADB tool from the Android devices.

We use a tool that runs on Android devices called *dumpsys* that provides information about some of the services running on the device. In this case, we want to inspect the network usage statistics, so we can specify the *netstats* service to *dumpsys* to get the information about the network status. We call it a dump. Using ADB, we can run the following command on the device under test to perform a dump:

```
adb shell dumpsys netstats detail
```

Including the detail option, the output of the command provides detailed network statistics that allow us to inspect unique user ID (UID) information. Each application has its own UID, so we can look up the network statistics of the application we are testing filtering by the ID of that specific application.

To find out the identifier of our app we can run the following command on the device:

```
adb shell dumpsys package com.example.myapp | grep userId
```

There are available two types of connection information: mobile or WI-FI. The devices we used for testing are controlled and only have mobile connection records.

For each UID record, we get information for each two-hour window. The data that we used is the following:

- *rxBytes* and *rxPackets*: received bytes and received packets.
- *txBytes* and *txPackets*: sent bytes and sent packets.

4.4.1.1 Calculation of data consumed by the applications through ADB

The goal is to calculate how much data is consumed during the usage of a certain app. For that we need to know how much data is recorded before and after the app is used, then we can just subtract the final value from the initial one, and we get the total data corresponding to the usage of the application in question. Figure 4.8 shows the method used for the calculation.

As illustrated in the figure 4.8, we execute a dump before and a dump after manipulating the app, this way we can know the amount of data that the app we are testing had registered before the test and after the test.

The calculation is quite simple: we sum all the consumed byte values (uplink and downlink) referring to the application's statistics (by its userID). In practice, all the values for the rxBytes and txBytes fields for that userID are summed, and we get the total number of bytes registered before and after using the application. Then we subtract the final value (after the test) from the initial one (before the test) and we get the total number of bytes consumed by the application.

4.4.1.2 Automation with Robot Framework and ADB

As explained above, the calculation of data consumed by the applications is done using the ADB tool. We wrote bash scripts where we run these commands on the devices and where we calculate the number of bytes consumed by the application. These scripts are in the virtual machine where the mobiles are connected so that we can execute the ADB commands.

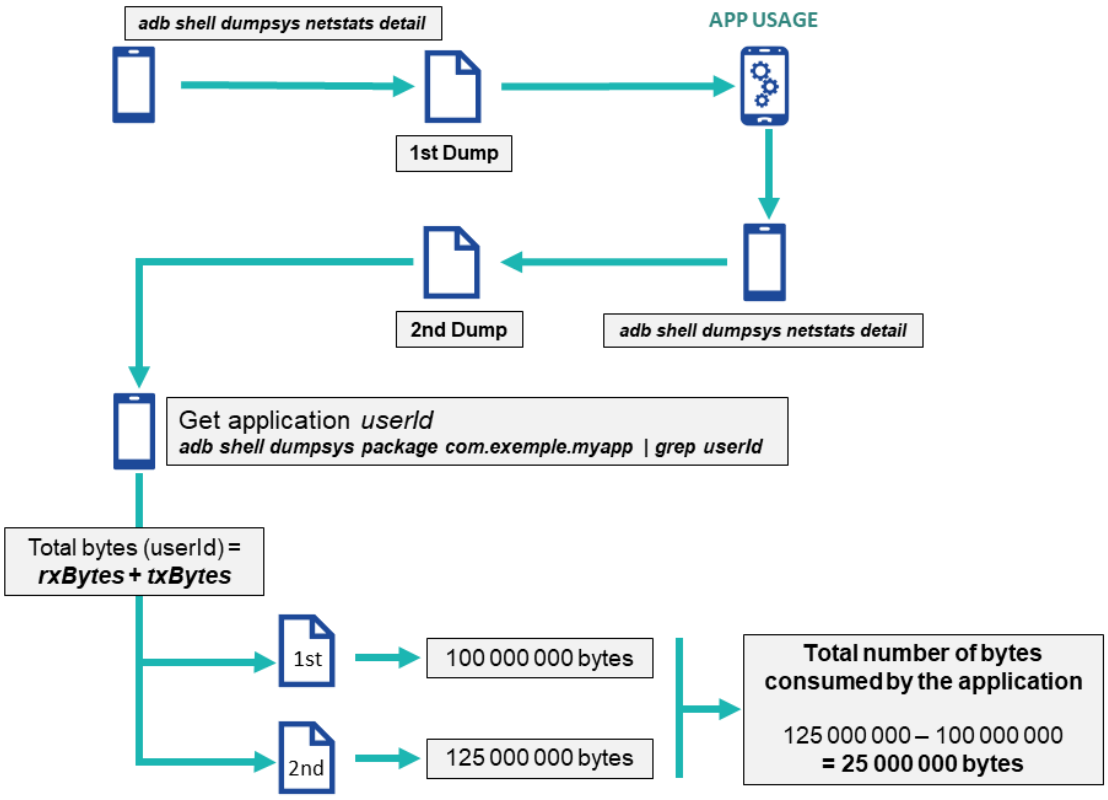


Figure 4.8: Android Bytes Calculation

The Robot Framework script, which is the main script of the test, is not in the same virtual machine as the mobiles, so for running the bash scripts we need to establish an SSH connection with the mobile's VM. For that, we use the functions provided by the SSHLibrary for RF which allow us to open a connection, log into the SSH server with the username and password and execute commands on the remote machine.

4.4.2 DPI Core Classification

In section 3.4 we presented the DPI classifier in the core network and how the classification is achieved. In this section we want to demonstrate how we access this classification. The objective is the same as on the Android side: calculate how much data is consumed during the usage of a certain application.

As it was already explained, the DPI counts the amount of traffic that a certain user consumed in a session. As we know, we can control a session by turning on and off the flight mode of the terminals. Also, as we have mentioned, we use the Call Detail Records of the session to get the value from the DPI.

Figure 4.9 demonstrates, in a simplified way, the steps required to obtain the classification made by DPI during the use of the applications.

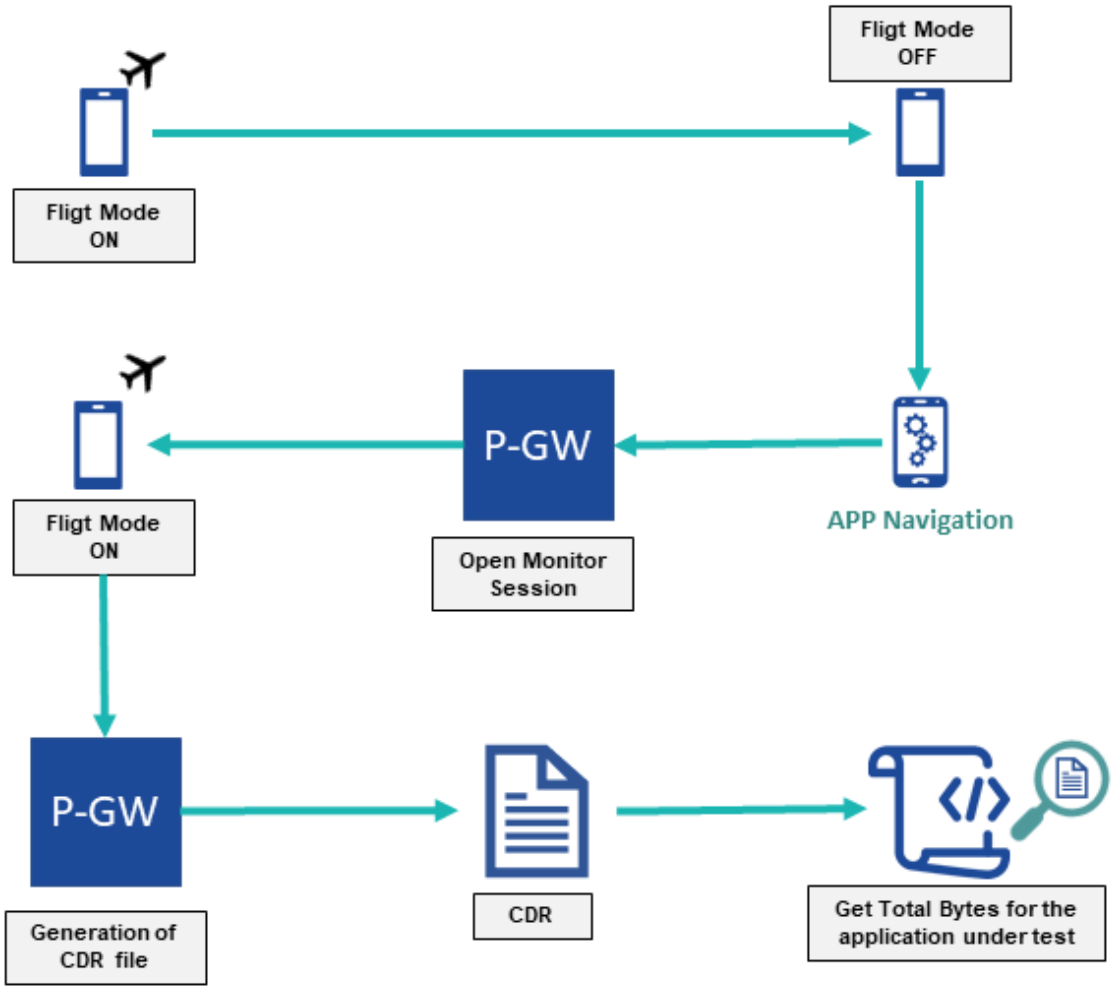


Figure 4.9: DPI bytes calculation

4.4.2.1 Automation with Robot Framework and Python

In order for the Robot Framework test script to access the CDR from the DPI, we use the SSHLibrary functions to access the P-GW node and execute CLI commands on it. As figure 4.9 shows, after the application navigation step of the test, we access the P-GW node and open a monitor session using CLI commands. The monitor session allows us to generate a file with the contents of the CDR, also with the use of some CLI commands.

Finally, after the terminal is put back into flight mode, the DPI generates the CDR. Since we have the monitor session opened, we can generate a file with the CDR content and analyze it. Similarly to what is done on the Android side, we sum the uplink and downlink number of bytes that the DPI classified as being traffic from the application we are testing. For this calculation, we use a Python script to ease the calculation.

4.5 Mobile Automation and Application Navigation

When we talk about monitoring traffic classification of mobile applications, one of the important pieces is the usage of the applications themselves. In this section, we will talk about the process of automating the usage of various mobile applications as well as some other tasks required for the testing.

The goal was to try to reproduce some different use cases in different applications as similar as possible to human behavior. It is important to keep the interactions similar to a human being to try to avoid the security mechanisms against bots and, of course, to simulate as much as possible the behavior that people in the real network have in their daily lives using these applications.

For this work we monitored the following mobile applications:

Table 4.2: Monitored Applications

Applications
Facebook and Facebook Messenger
Instagram
TikTok
Youtube
WhatsApp

These are some applications of interest to an ISP since they are the most used and have differentiated traffic charging conditions. For each one of them, we have automated different use cases. Table 4.3 shows the automated use cases developed for each of the applications. In order to try to simulate as close as possible with real usage of these applications, we automated the upload of contents in some of the applications.

Table 4.3: Developed use cases for each of the applications

Application	**Use Cases**
Facebook and Facebook Messenger	Scrolling through the application (swipes up and down) Multimedia content visualization (Stories) Sending text and audio messages (Facebook Messenger)
WhatsApp	Sending text and audio messages Sending photos
Instagram	Scrolling through the application (swipes up and down) Multimedia content visualization (Stories and Reels) Upload photos (Stories and Post)
TikTok	Scrolling through the application (swipes up and down)
Youtube	Scrolling through the application (swipes up and down)

The tasks that have been automated on the mobiles for the testing execution are the following:

- APN selection
- Turning ON/OFF flight mode

- Applications navigation (table 4.3)

For the mobile automation, we started by creating bash scripts using only ADB commands to interact with the devices. This method for automating the navigation of the applications is not user-friendly, since the code reading and the development of use cases are not easy tasks nor allow control over the tests. Therefore, we upgraded the application navigation automation with the use of Appium, however, there were still tasks that had to be performed using ADB directly.

The routine of turning on/off the flight mode was kept using ADB since it was not possible to find a solution that was not based on screen taps.

We will explore below the use of ADB and Appium, in order to get the mobile automation.

4.5.1 Mobile Automation with ADB

Initially, we started to automate application navigation using only ADB with bash scripts. So the interaction with the device was made by running ADB commands on the device.

To perform a tap on the screen we can run the following command on the device:

```
adb shell input tap $X_Coord $Y_Coord
```

To perform a swipe on the screen we can run the following command on the device:

```
adb shell input touchscreen swipe $X_Coord1 $Y_Coord1 $X_Coord2 $Y_Coord2
$duration(ms)
```

The mobile automation with ADB is based on screen coordinates. Therefore it requires extra effort to know which coordinates we want to click or swipe on the screen. Also, the fact that not all devices have the same screen resolution makes it even more difficult to build scalable tests across multiple devices.

To overcome this difficulty, we defined bash functions to perform taps and swipes based on percentages of the screen's vertical and horizontal axis in order to perform these actions on both test terminals. Figure 4.10 shows these functions. As we can see, these functions convert the percentage values of the horizontal and vertical axis into their respective coordinate values.

Once we can only navigate the applications through swipes and taps, the code is not very user-friendly to read since it is basically a sequence of taps and swipes. And as it was already said, it is not easy to build these scripts. Another big problem with this approach is that we can not control if the application navigation runs as intended. If the UI of the application does not have the expected behavior (whether it is because the application has been updated, or due to some delay in loading the page on the screen, among other situations), the script will continue to swipe and tap at the phone without doing what is

```
adbshell () {
  $SLEEP
  adb -s $PHONE shell $1
}

tap () {
  $SLEEP
  XCOORD=$(awk -v x=$1 -v n=$HORI 'BEGIN{printf "%0.0f\n", x/100*n}')
  YCOORD=$(awk -v x=$2 -v n=$VERT 'BEGIN{printf "%0.0f\n", x/100*n}')
  adbshell "input tap $XCOORD $YCOORD"
}

swipe () {
  $SLEEP
  XCOORD1=$(awk -v x=$1 -v n=$HORI 'BEGIN{printf "%0.0f\n", x/100*n}')
  YCOORD1=$(awk -v x=$2 -v n=$VERT 'BEGIN{printf "%0.0f\n", x/100*n}')
  XCOORD2=$(awk -v x=$3 -v n=$HORI 'BEGIN{printf "%0.0f\n", x/100*n}')
  YCOORD2=$(awk -v x=$4 -v n=$VERT 'BEGIN{printf "%0.0f\n", x/100*n}')
  adbshell "input touchscreen swipe $XCOORD1 $YCOORD1 $XCOORD2 $YCOORD2 $5"
}
```

Figure 4.10: Bash functions to perform taps and swipes on the devices

expected. Therefore, this approach does not allow to have control over the tests since it is a “blind navigation”.

Due to the high complexity of the scripts, the developed use cases using ADB only performed simple swipes and taps over the main pages of the applications. There was no upload of content.

4.5.2 Mobile Automation with Appium

After having produced some basic tests with the use of bash scripts, we decided to improve the number of use cases we had for applications and thus have more functionality, so that we could simulate traffic as close to reality as possible. Also, it would be good if we could have control during the tests in case of failure of the application navigation. Yet, we wanted to have scripts that could keep up with application updates without having to reprogram the script again. To get these upgrades it was not feasible to do it with ADB alone.

The solution was the use of the Appium tool. Once the tests are written in Robot Framework, the use of Appium makes every task easier, and this is due to the AppiumLibrary mobile application library for Robot Framework [6]. This library provides Keywords for RF that automate mobile application usage. The big advantage over ADB is that, unlike ADB, we can specify what elements on the screen we want to click, or swipe, among other tasks. Appium, therefore, allows more controlled navigation over the application, unlike navigation with ADB which is a "blind navigation"since it is based on

screen coordinators.

The applications we are monitoring are very popular which means they are constantly being updated which can lead to changes in their user interface. Having scripts based only on screen coordinates make them less robust to these updates. So, locating specific elements on the application screen is an important step to achieve more robust tests. Table 4.4 shows the main supported strategies used by AppiumLibrary to locate elements on the screen.

Table 4.4: AppiumLibrary locator strategies to specify elements

Strategy	Description	Example
identifier	Matches by @id attribute	Click Element | identifier=my_element
id	Matches by @resource-id attribute	Click Element | id=my_element
xpath	Searches the application XML	Click Element | xpath=//path/to/my_element
class	Matches by class	Click Element | class=android.widget.TextView
android	Uses the UI Automator API	Click Element | android=UiSelector().text('Send')

Figure 4.11 shows an example of how we can click on a specific element on the screen using Appium and Robot Framework with the use of keywords from the AppiumLibrary for RF.

```
Reels
    wait until element is visible  id=com.instagram.android:id/clips_tab
    Click Element  com.instagram.android:id/clips_tab
    FOR    ${i}    IN RANGE    10
      swipe by percent    50     50     50    10  2000
      Sleep  3s
      Exit For Loop If    ${i} == 9
      Log    ${i}
    END
```

Figure 4.11: Example of Robot Framework using AppiumLibrary keywords to automate mobile application usage

As we can see from figure 4.11, the use of RF keywords makes the code easy to understand, unlike ADB. In this example, we are pressing a button on the application and then we do ten swipes down. The use of the "Wait Until Element Is Visible" keyword from AppiumLibrary allows having more control over the tests. For example, if the application has been updated, this button may have disappeared from the screen or may have changed position, and with Appium we can know if it found the element with its identifier. If the element is found, even though in another position, we can still click on that element, in contrast with a script that only clicks on given screen coordinates that would click on the wrong place. Also, if for some reason the page takes longer to load the elements, we may still have a chance to click on the desired element once it is loaded using this keyword. This way we can have the test more "bulletproof".

4.5.2.1 Appium Inspector

In order to know the identifier of the elements on the screen, we used the Appium Inspector tool. This tool allows us to inspect mobile applications and retrieve information about the elements on the screen. Figure 4.12 shows an example of this tool. As we can see, it inspects the XML structure of the application.

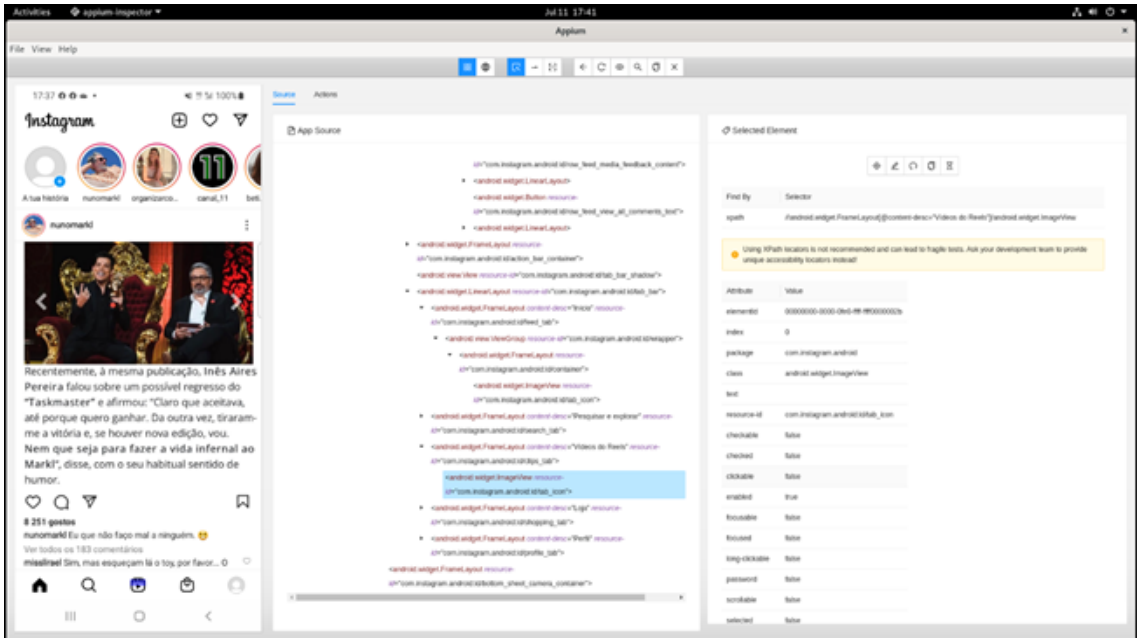


Figure 4.12: Appium Inspector

4.5.2.2 Challenges

The use of Appium improved the quality of the tests, introduced the capacity of having control over the tests, and made it easier to build the tests when compared to ADB. Nevertheless, there are some challenges. Using Appium Inspector to inspect the applications, we have noticed that sometimes there are no defined IDs for the elements on the screen. This can lead to complex XPath queries like the one presented in figure 4.13. It shows the automation of the APN selection with Appium using one XPath query to find the correct APN on the screen.

```
Open APN settings and select the correct one
    Start Activity  com.android.settings     com.android.settings.Settings$ApnSettingsActivity
    Wait Until Element Is Visible   xpath=//*[@text="automacao"]/../../android.widget.RadioButton
    Click Element   xpath=//*[@text="automacao"]/../../android.widget.RadioButton
    Sleep  3s
    Press Keycode  3   #home
```

Figure 4.13: Use of XPath to select the correct APN on the devices

Even though application navigation with Appium is more secure in case of updates, the scripts need to be maintained. Element IDs may change or disappear and the XML

structure can also change, causing XPath queries to become ineffective.

Once Appium has a Client-Server architecture, it is also necessary to control the Appium Server. Sometimes we detected some instability in the server which make us have to restart it.

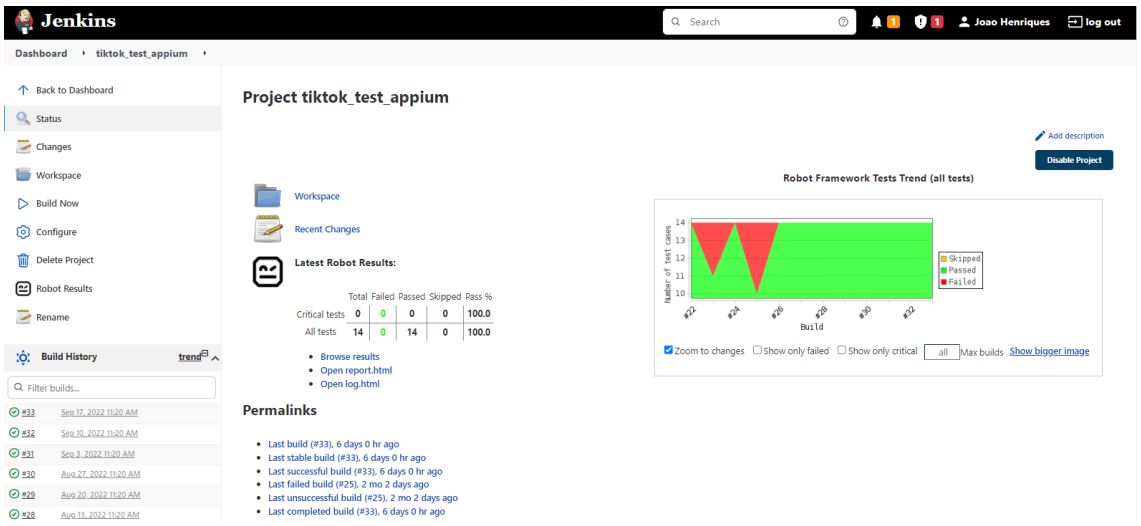
4.6 Jenkins

The mobile application DPI accuracy tests are stored in the Jenkins workspace. It is from there that they are automatically launched and produce reporting information due to the use of a RF plugin.

Figure 4.14 shows how Jenkins with the use of the Robot Framework plugin presents information about the tests. Figure 4.14a presents the list of tests. Figure 4.14b shows what we can see expanding the information about one of these tests.

S	W	Name ↓	Last Success	Last Failure	Last Duration	Robot Results + Duration Trend
✗	☁	facebook_test	13 days #34	6 days 1 hr #35	4 min 15 sec	10 / 11 pass
✗	☁	facebook_test_appium	3 mo 0 days #8	6 days 0 hr #24	3 min 2 sec	20 / 21 pass
✓	☁	instagram_test	6 days 0 hr #26	20 days #24	5 min 46 sec	11 / 11 pass
✗	☁	instagram_test_appium	12 days #21	5 days 23 hr #22	4 min 35 sec	19 / 20 pass
✗	☁	RobotTests	7 mo 3 days #5	5 mo 7 days #6	6.7 sec	31 / 32 pass
✓	☁	tiktok_test	6 days 0 hr #24	2 mo 2 days #16	2 min 31 sec	11 / 11 pass
✓	☁	tiktok_test_appium	5 days 23 hr #33	2 mo 1 day #25	2 min 47 sec	14 / 14 pass
✓	☁	tiktok_test_S21	2 mo 12 days #19	2 mo 12 days #18	3 min 0 sec	
✗	☁	whatsapp_A12	2 mo 13 days #8	6 days 0 hr #18	1 min 44 sec	13 / 17 pass
✓	☁	youtube_test	6 days 0 hr #45	2 mo 2 days #37	2 min 36 sec	11 / 11 pass
✓	☁	youtube_test_appium	5 days 23 hr #19	12 days #18	2 min 54 sec	14 / 14 pass

(a)



(b)

Figure 4.14: (a) presents the list of tests. (b) shows expanded information for one of the tests

For each test we have to configure the following parameters:

- The script we want to run. Since the tests are RF scripts, they are launched via a shell command. We configure the execution of the test to run this command in Jenkins.
- Data and time of test execution (scheduling).
- Post-test actions (generation of test execution report files).

4.6.1 Test Scheduling

As was already mentioned, the system runs the tests weekly, thus it was necessary to schedule the execution of the necessary scripts.

As said before, we scheduled the execution of a script that automatically selects the correct APN on the mobile to run before the tests. Also, all the application test scripts are scheduled in a sequential way.

In order to achieve this level of automation and coordination, we used cron-based expressions that allow scheduling the execution of the scripts.

Cron is a job scheduler for Unix-based operating systems suitable for automating repetitive tasks. We can specify the time we want to run our cron jobs and it will run our job periodically at the time we configured. The cron jobs are stored in a cron table (crontab) file. The syntax of a cron expression is composed of five fields that represent the time to execute the job, and by a shell command to be executed. Figure 4.15 shows the syntax of a cron expression.

```
# ┌──────────── minute (0 - 59)
# │ ┌────────── hour (0 - 23)
# │ │ ┌──────── day of the month (1 - 31)
# │ │ │ ┌────── month (1 - 12)
# │ │ │ │ ┌──── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
```

Figure 4.15: Cron expression syntax

Jenkins has a specific configuration parameter to introduce the cron expression and is not necessary to specify any shell command for the script once it is specified in another configuration parameter.

Figure 4.16 demonstrates how the scheduling of one of the test scripts was done in Jenkins. In this case, the script will be executed every Saturday at 11 am.

4.6.2 Test Execution Reporting

Besides the power that Jenkins gives with the easy scheduling procedure, it also has great reporting analysis integrated with the Robot Framework plugin. Since the script tests are

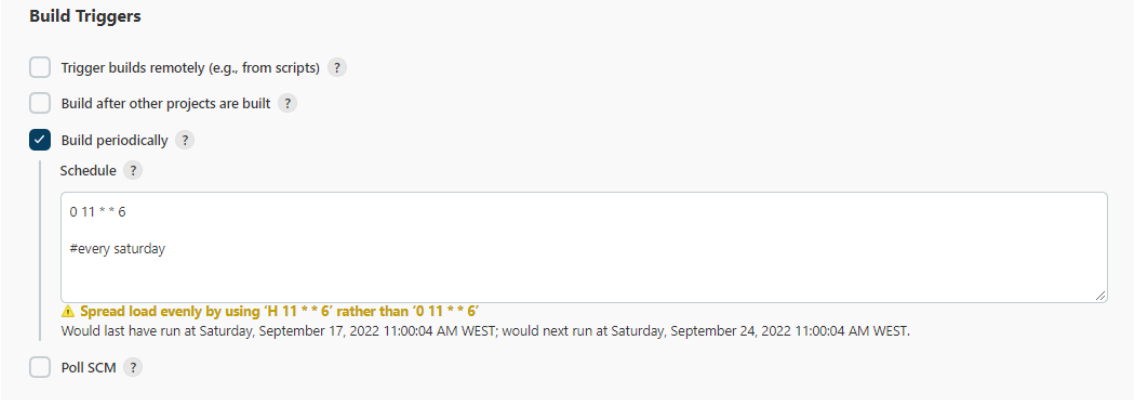


Figure 4.16: Jenkins Scheduling Configuration

written in Robot Framework, they produce HTML files with detailed information about the whole execution of the test for each of the performed tests, as introduced in 2.3.2.3.

This plugin, among other functions, displays and stores the test results from the RF scripts in a user-friendly way every time a test is executed. This allows us to have statistics about the tests.

The reports from RF give information about the errors that happen during the tests. Thus, it helps to understand what went wrong and why some tests may have failed. Since our tests have an autonomous navigation component in mobile applications, if any of the use cases in the application fail, we can figure out what failed in the test report. This way we can manage the state of the system and check if it needs some intervention or updates to the script tests.

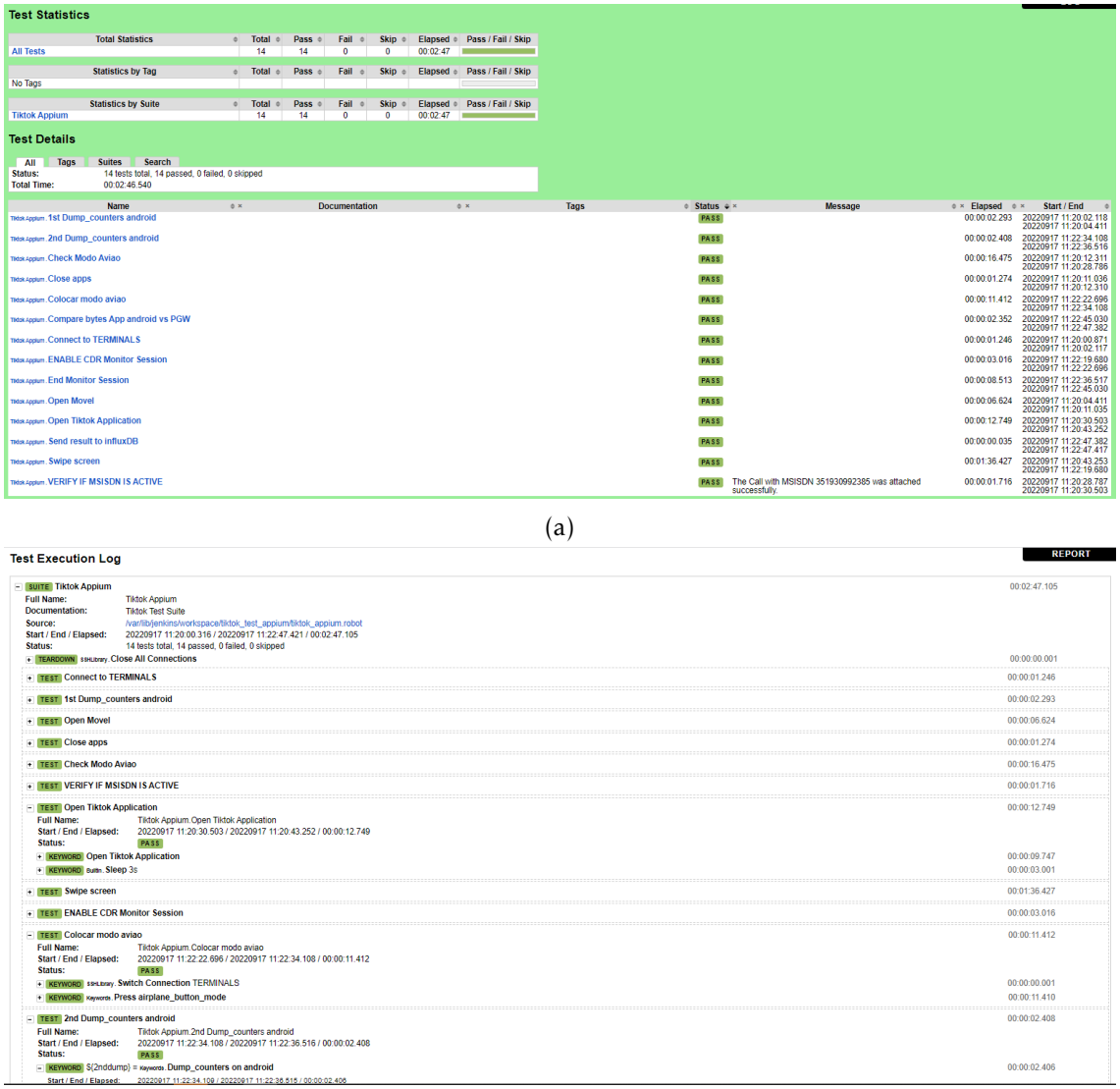
Figure 4.17 shows an example of the report files that are generated by RF and are accessible by Jenkins. Figure 4.17a shows the Report.html file where we can see what the status of each test case was. Figure 4.17b shows the Log.html file that gives us more detail about each test case. In cases of errors, the Log.html file helps to debug and understand what actually failed.

4.7 InfluxDB, Grafana, and Weekly Email Reporting

As part of the monitoring system, it is essential to have a way to visualize the results of the accuracy tests that we explored before throughout this chapter.

As we can see in figure 4.6, every test ends by sending the result of the core classification accuracy to a database, which is then available for visualization in a dashboard of graphs. In our work, we used InfluxDB and Grafana for the database and graphs, respectively.

Figure 4.18 shows the relations between the database and the exterior. In our system, the data is only stored in the database by the script tests. Grafana queries the database to display the values in charts. For the weekly email reports, we wrote a script that queries



(a)

(b)

Figure 4.17: (a) Report.html file. (b) Log.html file

the database and analyzes the results from the tests.

4.7.1 InfluxDB database

Since we are monitoring five different applications, we built a database with five different measurements, one for each application. An InfluxDB measurement is similar to an SQL table. Inside the measurement, we store the accuracy value in a field. An InfluxDB field is like an SQL unindexed column.

To store the accuracy value obtained from the test, we send an HTTP POST to the */write* endpoint of the InfluxDB API. Figure 4.19 shows how we make the POST request to our database with the accuracy value and the measurement name of the tested application in the ${result} variable, in the Robot Framework script. We use the Curl (standing for "Client URL") command line tool to make the HTTP POST request.

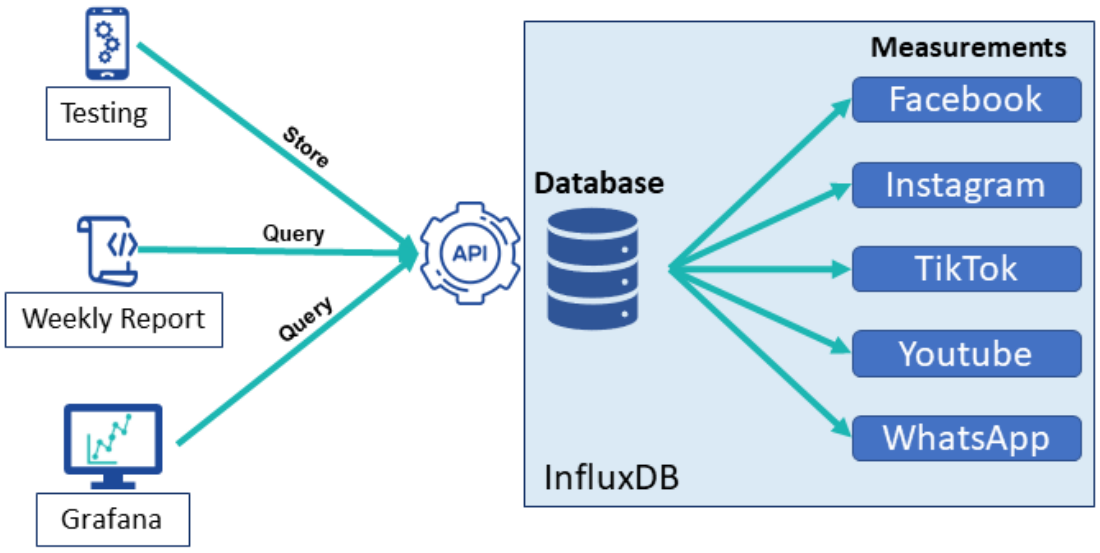


Figure 4.18: InfluxDB's relations with the exterior

```
Send result to influxDB
    ${result}=    Set Variable    ${data}${value}
    Log    ${result}    console=yes
    ${result}=  Run Process    curl -i -XPOST 'http://10.36.0.177:8086/write?db\=apps_tests' --data-binary '${result}'
    Log  ${result.stdout}  console=yes
```

Figure 4.19: Sending the accuracy obtained in the test

4.7.2 Grafana Dashboard

As we have already mentioned, Grafana has great integration with InfluxDB, so it is very simple to display charts with the data from our measurements in the InfluxDB database. Since the Grafana and InfluxDB instances are installed in the same Virtual Machine, we configured our InfluxDB as the data source for our Grafana Server, and with the use of InfluxQL query language, we perform queries to our measurements.

Figure 4.20 shows the setup of the Facebook monitoring graph. As we can see, we selected InfluxDB as the data source and defined an InfluxQL query that gets the test accuracy values. The "value" represents the name of the InfluxDB field (from the "Facebook" measurement) that stores the accuracy result of each test.

Every time a test is executed and a new point of data is stored in our database, Grafana automatically updates the charts with the new value. This way we have our dashboard updated in real time.

4.7.3 Weekly Email Reporting

One of the features of the system is to send automatic emails to all the team members with the weekly test report. The email is sent every Monday at 9 am to inform about the result of last Saturday's tests. The email always has the Grafana dashboard URL link in

Figure 4.20: Facebook graph configuration on Grafana

the email body to easily access the dashboard for all the team members. All applications that have achieved a classification accuracy below 80% are identified in the body of the email. If no test has obtained an accuracy lower than 80%, it is reported on the email body that all tests ran "OK".

To implement this routine we developed a script in Robot Framework with a Python function and scheduled it with Cron to run every Monday at 9 am.

To send the email we use the Linux Mail User Agent program mailx. This console application allows sending emails with the use of shell commands, which is very suitable for our system since it is intended that every task is performed without the need for user interaction.

In Python, we developed a function that queries the InfluxDB database to check the results of Saturday's tests and write and return the body of the email to the RF script. It uses the Requests Python package to send a GET request to the InfluxDB API /query endpoint with the InfluxQL query as a parameter.

Figure 4.21 gives an example of the RF script implemented to check the database and send the report email. As we can see, we can easily use Python functions as RF keywords, we just need to import the Python function as a Library. (The sender@email.com and receiver@email.com must have the correct email addresses, in this picture we use these addresses for illustrative purposes only).

```
*** Settings ***
Documentation  Weekly Email Reporting

Library  influxdb_routine.py
Library  OperatingSystem

*** Test Cases ***
Get influxDB data
    ${email_body}=  influxdb_routine.get_influxdb_data
    Run  echo "${email_body}" | mailx -s "Mobile Apps DPI Accuracy Tests" -r sender@email.com receiver@email.com
```

Figure 4.21: Robot Framework script to check database and send email report

5

Autonomous Monitoring System: Core DPI Classification Results

In the previous chapter, we have explained how we developed and implemented the autonomous monitoring system for the core network traffic classification. In this chapter, we will show the built monitoring dashboard and analyze the obtained results for the core mobile applications classification.

After that, we will present the biggest challenges for the DPI classification that we have detected during the project development.

5.1 Core Network DPI Accuracy: Grafana Dashboard

In this section, we start by presenting the monitoring dashboard with all the applications, and then we show the individual graph for each of them so that we can have a simpler and more effective analysis of the DPI behavior for each of the applications.

As mentioned before, we have built a Grafana Dashboard that displays the accuracy results obtained from the application tests. Figure 5.1 shows the dashboard with the core network DPI classification for the five applications we monitored.

Figure 5.1: Grafana Dashboard

5.1.1 Results of DPI Classification Accuracy

The following figures show the accuracy results of the core DPI classification for each of the applications. They are an expansion of figure 5.1.

Apart from WhatsApp, the graphs correspond to a monitoring period of more than five months.

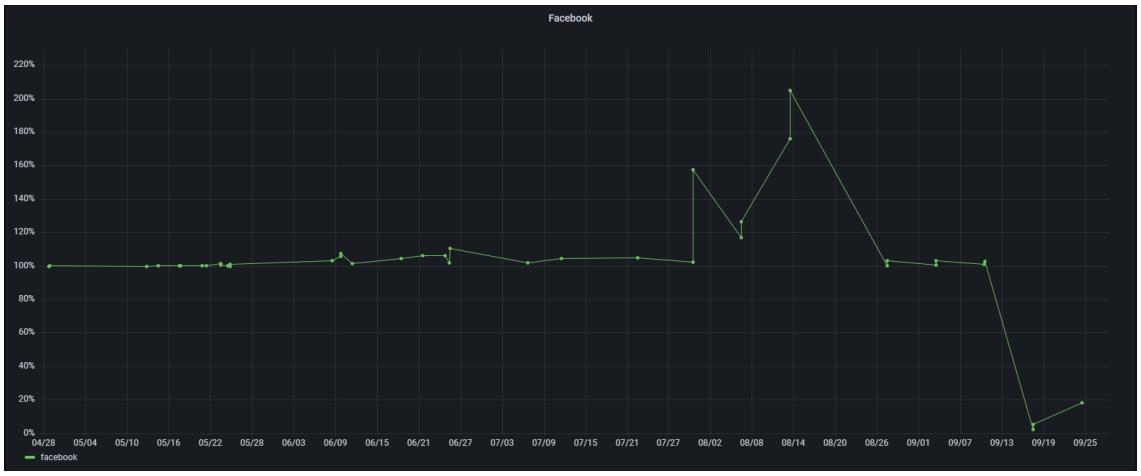


Figure 5.2: DPI Facebook Accuracy

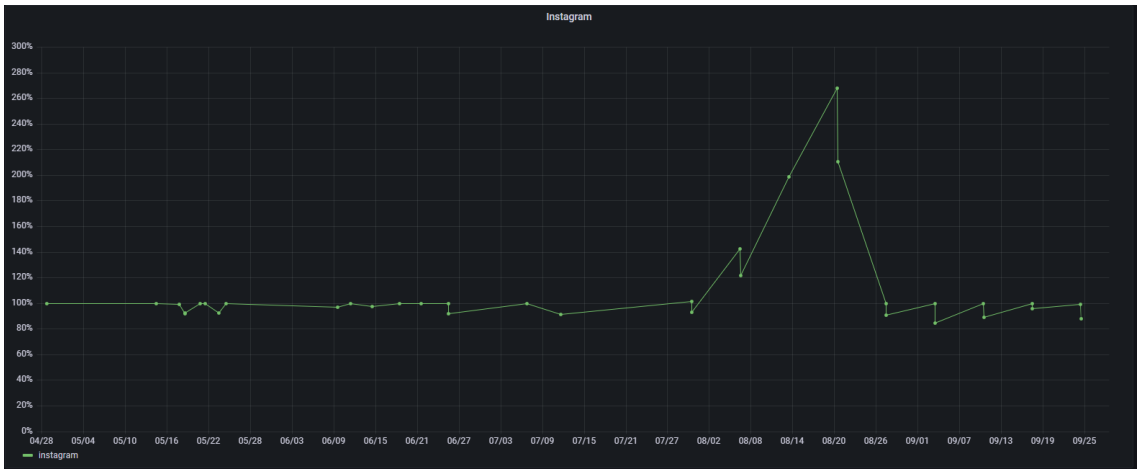


Figure 5.3: DPI Instagram Accuracy

5.1.1.1 Analysis of the Results

Being this project a monitoring and automation work, we do not have expected results in terms of the quality of the classification. In terms of the system capabilities, we can conclude that we have achieved the objective of having a fully autonomous system capable to monitor the core network traffic classification for some of the most relevant mobile applications.

Evaluating the core network traffic classification, in contrast to what we could expect after all the state-of-the-art studies about the limitations of DPI techniques for traffic

Figure 5.4: DPI TikTok accuracy

Figure 5.5: DPI Youtube Accuracy

classification, due to the complexity that the encryption of the new web protocols brings to network communications, the results were impressive.

As we can see from these graphs (figure 5.2 to figure 5.6), the core DPI has achieved excellent results for all applications.

There are some big deviations over a period of time for almost all applications that are not directly related to the DPI performance. Due to the fact of the laboratory network is used for various tests, some configurations may have changed. One of the aspects that conducted to these errors was the occurrence of some problems in terminating the user session on the core network. This way, some traffic was calculated in excess since it was accumulated due to the difficulties in terminating the user session.

5.2 Challenges for the DPI classification

When we look at the results obtained from the DPI, we can think that maybe DPI techniques are effective even with the emergence of encryption of the network data. However,

Figure 5.6: DPI WhatsApp Accuracy

we have detected some challenges that may put DPI in danger.

5.2.1 Maintenance of DPI rules definition

As we have explored in section 3.4, the DPI is based on rule matching. Due to constant application updates, and the introduction of new web protocols, among other situations, it is necessary to keep the rules updated in order to keep up with application changes.

This introduces a necessary effort for a team. In order to keep the user experience of their clients, they are subject to the changes of these applications.

5.2.2 Encrypted DNS

We have already presented the topic of encrypted DNS in the state-of-the-art section 2.2.3. We observed the increasing availability of encrypted DNS services on the network and verified that they are a major obstacle for DPI.

By default, the APN used for the tests in the laboratory network does not have encrypted DNS configured. During the project development, we performed some experiences of classification using a DNS resolver that uses encryption (DoH/DoT) and DPI has revealed many difficulties in classifying the traffic correctly.

5.2.3 TLS 1.3 + pre-shared keys

Since almost everything is encrypted in today's network communications, the DPI uses the information of the Server Name Indication (SNI) as one of the most reliable for the inspection of the packets. When using TLS 1.3, the DPI can no longer access this information. It gets even more complicated when combined with DoH/DoT.

6

Implementation of a Data Collection System

It is true that we have verified pretty good results from the DPI classification, however, the difficulties do exist and are increasingly difficult to overcome. As explored In the state-of-the-art section, the most recent techniques for network traffic classification lead to the use of Machine Learning and Deep Learning algorithms to address these difficulties.

As it was referred to in the state-of-the-art section 2.3, is necessary to have a big amount of data, in order to build a Machine or Deep Learning model for network traffic classification. One of the major difficulties today is not only producing large and representative datasets but also there is a lack of these datasets available.

With our work, we are able to extend the functionalities of the system to a data collection mechanism in parallel with the already implemented functions of the system. Since we are able to generate real data in an autonomous way, this data represents a big value for the construction of strong datasets and for the building and training of ML/DL models.

In this section, we will show how we can collect data with our work and how it can help in the implementation of ML or DL models for traffic classification.

6.1 Data Collection: Architecture

The architecture used for the data collection is illustrated in figure 6.1. The network used is the same laboratory network mentioned before in which the project was developed.

As we can see in figure 6.1, traffic is copied when it arrives at a router that is configured for port mirroring. A port mirroring instruction tells the router/switch to send a copy of traffic received in a specific port to another specific port.

Therefore, a replica of the traffic goes to a Linux machine that has Wireshark and tcpdump tools running.

Using our system, we can use the Robot Framework to orchestrate the traffic capture with the execution of the tests. This way we can collect reliable data from real devices.

With this feature, not only we are able to collect data, but also, if the test achieves low results of accuracy from the DPI, we can check the captures to try to understand what is

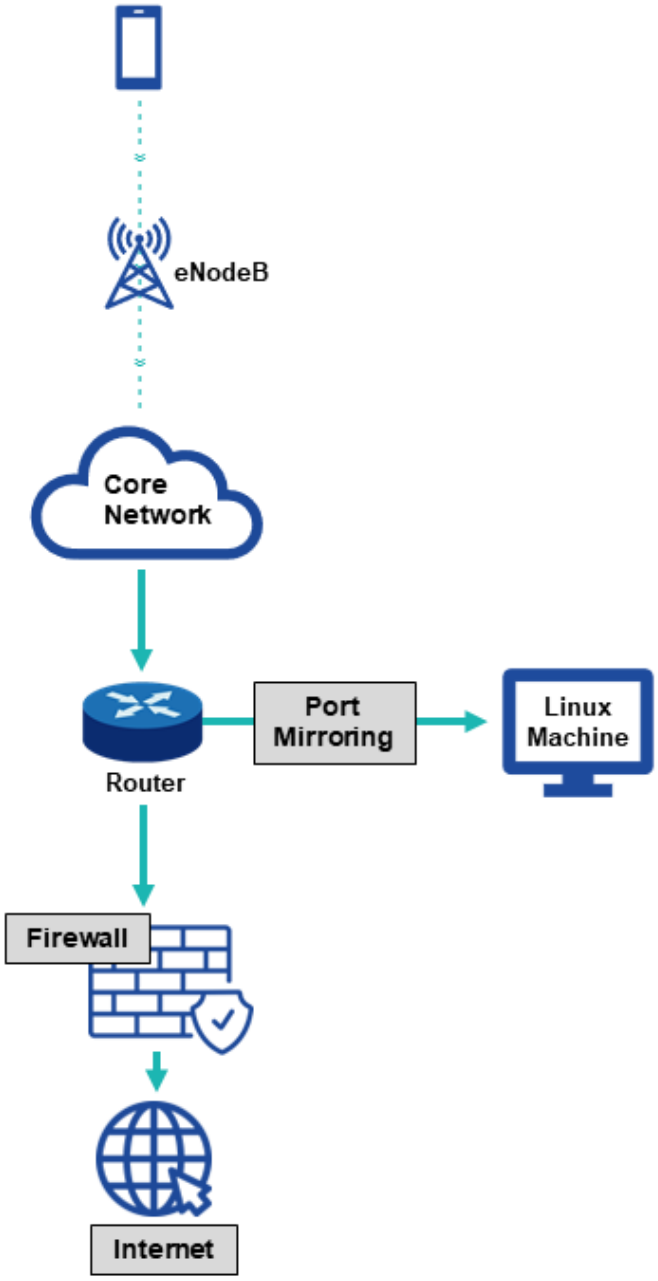


Figure 6.1: Data Collection Architecture

the reason and try to improve and keep updated the rule definitions for the DPI.

6.2 Data Collection: How it works

We used our Robot Framework script tests to prove that it can be a great way to capture traffic from real devices in a real network (in this case in an ISP) and autonomously. The task of collecting data is often very time-consuming and exhausting when manually

performed.

As we can see in figure 6.1 the traffic is copied to a Linux machine, so we used the tcpdump tool on that remote machine. This tool is a command-line utility that allows the capture and analysis of network traffic.

Once again, we used the SSHLibrary for Robot Framework to access the remote machine and execute the tcpdump commands to perform the capture of the network packets from the applications we are monitoring with our system. We save the traffic captures in *pcap* or *cap* files. These are the most common files for storing network packets data.

Figure 6.2 demonstrates how our system can be adapted to perform the data collection task.

6.3 Data Processing

For the implementation of Machine Learning or Deep Learning models for traffic classification, besides the data collection, is necessary to process and prepare the data to implement the algorithms.

The main tasks that need to be executed on the raw network captures before the training phase of the models are the following:

- Group the packets by traffic flows (IPSrc, IPDst, PortSrc, PortDst, Protocol)
- Labeling of the data according to the applications under test

Once we are collecting the data at the core level, the network packets are encapsulated in a GTP tunnel, as explained in section 3.2.3.2. Therefore, it is necessary to decapsulate the packets in order to get reliable data captures. In this work, we explored the GTP decapsulation and the identification of traffic flows steps of the data processing phase. Below we will talk about the tools used for data processing.

6.3.1 Tools used for Data Processing

For the processing of the data, we did some experiences using some open source tools and some python libraries for the manipulation of pcap files.

6.3.1.1 TraceWrangler – Packet Capture Toolkit

TraceWrangler [29] is an open source tool that allows manipulating the network capture files. We used this tool to decapsulate the GTP tunnel from the packets. It generates a new pcap file without the headers from the GTP tunneling from the original trace file.

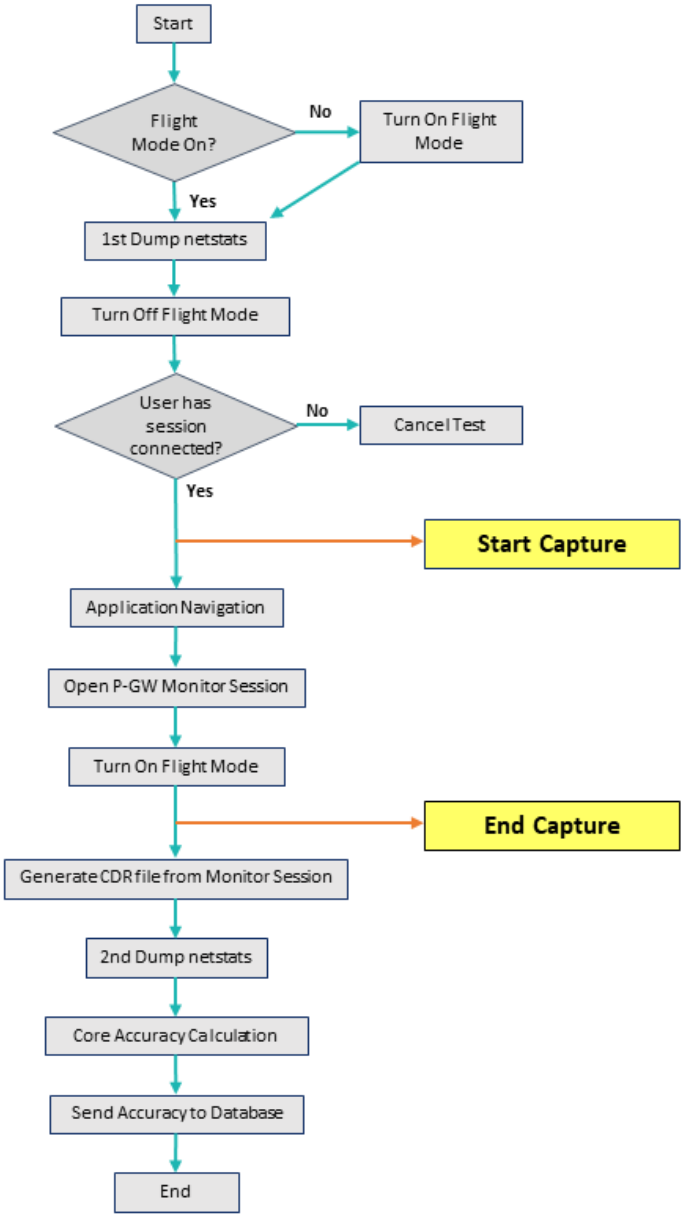


Figure 6.2: Data Collection Procedure

6.3.1.2 SplitCap

SplitCap [38] is an open source tool that as its name refers, allows splitting the capture files into other files based on some criteria. We used this tool to split the capture file into multiple files grouped by flows.

This tool is executed via shell commands which allow being able to use this tool autonomously through scripts.

6.3.1.3 PyShark

PyShark [19] is an open library for Python that allows manipulating network capture files. The manipulation that we performed using the previous tools can also be carried out by the use of this library.

An advantage of using this library is that it allows us to create scripts to perform tasks autonomously, thus continuing in the automation paradigm of this work.

This approach is more low level due to the fact that we are writing code to manipulate the network packets, which gives us more freedom to manipulate the packets according to our requirements. We managed to decapsulate the GTP headers from the packets in a programmatic way.

6.4 Analysis of the network captures: Contribution to improving the quality of DPI

In section 5.2 we presented some of the challenges that we verified with the most impact on the DPI accuracy. With this data collection mechanism, we could inspect in more detail the traces from our devices and analyze when the results were not good.

With the analysis of the network captures, we were able to improve the classification for some of the applications under test, by updating the rules defined on the DPI.

These popular applications are constantly changing and updating. These changes need to be monitored. We give more monitoring and maintenance power to the system, with this feature allowing us to record network captures from the usage of the applications at the same time that we verify the accuracy of the core classification.

7

Conclusion and Future Work

7.1 Conclusion

The objective of this work was to build an autonomous system capable of monitoring the core network traffic classification for some mobile applications of interest in an ISP and study the use of Machine Learning and/or Deep Learning algorithms to overcome the difficulties that exist in the traditional network traffic classification approaches.

The core classification is carried out by Deep Packet Inspection (DPI) techniques. Therefore, for the monitoring purpose, it was necessary to retrieve information that could verify the core classification (DPI). We built a system based on testing with real devices in order to control the accuracy of the core classification.

The solution that we took to calculate the accuracy of the core, was to compare the DPI classification with the network statistics from the devices under testing when using the applications.

The need for automation these days is increasingly important, and when it comes to monitoring tasks it is a great asset. The automation was intended since the beginning of the project, so, this work was also an automation project. All the features of the system were developed to operate autonomously without human interaction.

The use of Robot Framework proved to be very strong to automate tasks when it is necessary to access remote machines and execute commands, scripts, among other tasks, and also to automate mobile navigation through applications integrated with Appuium. Which were the main tasks developed for this work.

The easy integration between Robot Framework and Jenkins was also very suitable for the implementation of the system, once it helps with the scheduling of the tests, automatic test execution, and the reporting information about the execution of the tests. The system was configured to run the tests every week.

The use of the InfluxDB and Grafana tools also proved to be good options for building a monitoring dashboard. This dashboard displays graphs for each application under test with the DPI accuracy results for each of them.

In summary, the built and configured system is operational and being used as an asset

in this ISP.

This work also serves as a starting point for the implementation of network traffic classifiers based on Machine Learning and Deep Learning algorithms. Besides the study that was conducted in the state-of-the-art section, we implemented a data collection mechanism integrated with the autonomous monitoring system.

7.2 Future Work

With automation becoming more and more important and essential, this work leaves useful tools to evolve towards task automation. There is an endless line of work that can be explored.

Regarding the system operation and the implementation of ML/DL techniques, we refer below to the most relevant aspects to work on in future work.

7.2.1 Test the classification on a larger number of devices

One aspect that can be relevant to improve the quality of the system is to increase the number of devices we used for the monitoring tests. Since we only used android devices, it would be relevant to also monitor the accuracy of the core classification with iOS devices. By having more and different test devices, the testing environment is closer to the real world, allowing a more trustable monitorization of the accuracy of the DPI.

7.2.2 Test the classification with a larger number of mobile applications

For this work, we only monitored the core classification for a few mobile applications. Even though these applications are the most relevant, it would be good if the monitorization could be expanded to a larger number of applications.

7.2.3 Continue with the ML/DL implementation techniques

The future of DPI is getting more and more complicated. It is necessary to start implementing new approaches.

In this work, we made a study about the use of ML/DL techniques for network traffic classification and implemented a data collection mechanism. This is a starting point for the implementation of classifiers using AI algorithms that aim to address the difficulties present in DPI techniques.

For future work, there is the whole implementation phase of the ML/DL models. For the data collection and building of datasets, we leave a very strong approach in order to optimize these tasks with automation.

Bibliography

[1] *3rd Generation Partnership Project (3GPP)*. URL: https://www.3gpp.org/ (cit. on pp. 23, 24).

[2] I. Akbari et al. "A Look Behind the Curtain: Traffic Classification in an Increasingly Encrypted Web". In: *Abstract Proceedings of the 2021 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '21. Virtual Event, China: Association for Computing Machinery, 2021, pp. 23–24. ISBN: 9781450380720. DOI: 10.1145/3410220.3453921. URL: https://doi.org/10.1145/3410220.3453921 (cit. on p. 13).

[3] P. Amaral et al. "Application Aware SDN Architecture using Semi-supervised Traffic Classification". In: *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2018, pp. 1–6. DOI: 10.1109/NFV-SDN.2018.8725753 (cit. on p. 5).

[4] P. Amaral et al. "Machine Learning in Software Defined Networks: Data collection and traffic classification". In: *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. 2016, pp. 1–5. DOI: 10.1109/ICNP.2016.7785327 (cit. on p. 4).

[5] *Appium*. URL: https://appium.io/ (cit. on p. 19).

[6] *AppiumLibrary*. URL: https://serhatbolsu.github.io/robotframework-appiumlibrary/AppiumLibrary.html (cit. on p. 49).

[7] R. Boutaba et al. "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities". In: *Journal of Internet Services and Applications* 9 (1 2018-12). ISSN: 18690238. DOI: 10.1186/s13174-018-0087-2 (cit. on p. 13).

[8] Chriswernst. *Chriswernst/dbscan-python: This is an example of how DBSCAN (density based spatial clustering of applications with noise) can be implemented using python and its libraries numpy, Matplotlib, opencv, and scikit-learn*. URL: https://github.com/chriswernst/dbscan-python (cit. on p. 10).

[9] Y. Cui et al. "Innovating transport with QUIC: Design approaches and research challenges". In: *IEEE Internet Computing* 21 (2 2017-03), pp. 72–76. ISSN: 10897801. DOI: 10.1109/MIC.2017.44 (cit. on p. 13).

[10] J. Dai et al. "An analysis of Network Traffic Identification based on Decision Tree". In: *2021 International Conference on Artificial Intelligence and Electromechanical Automation (AIEA)*. 2021, pp. 308–311. DOI: 10.1109/AIEA53260.2021.00072 (cit. on p. 8).

[11] *DNS over TLS vs. DNS over HTTPS | Secure DNS*. URL: https://www.cloudflare.com/learning/dns/dns-over-tls/ (cit. on p. 15).

[12] S. Dong, D. Zhou, and W. Ding. "The Study of Network Traffic Identification Based on Machine Learning Algorithm". In: *2012 Fourth International Conference on Computational Intelligence and Communication Networks*. 2012, pp. 205–208. DOI: 10.1109/CICN.2012.211 (cit. on p. 10).

[13] J. Erman, M. Arlitt, and A. Mahanti. "Traffic Classification Using Clustering Algorithms". In: *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*. MineNet '06. Pisa, Italy: Association for Computing Machinery, 2006, pp. 281–286. ISBN: 159593569X. DOI: 10.1145/1162678.1162679. URL: https://doi.org/10.1145/1162678.1162679 (cit. on pp. 9, 10).

[14] Z. Fan and R. Liu. "Investigation of machine learning based network traffic classification". In: *2017 International Symposium on Wireless Communication Systems (ISWCS)*. 2017, pp. 1–6. DOI: 10.1109/ISWCS.2017.8108090 (cit. on p. 7).

[15] M. Finsterbusch et al. "A Survey of Payload-Based Traffic Classification Approaches". In: *IEEE Communications Surveys Tutorials* 16.2 (2014), pp. 1135–1156. DOI: 10.1109/SURV.2013.100613.00161 (cit. on p. 4).

[16] A. Ghedini. *Encrypt it or lose it: how encrypted SNI works*. URL: https://blog.cloudflare.com/encrypted-sni/ (cit. on p. 15).

[17] *Introduction to HTTP/2 nbsp;|nbsp; web fundamentals nbsp;|nbsp; google developers*. URL: https://developers.google.com/web/fundamentals/performance/http2/#um_resumo_da_hist%5C%C3%5C%B3ria_do_sdpy_e_do_http2 (cit. on p. 13).

[18] *K means clustering simplified in Python: K means algorithm*. 2021-04. URL: https://www.analyticsvidhya.com/blog/2021/04/k-means-clustering-simplified-in-python/ (cit. on p. 9).

[19] KimiNewt. *Kiminewt/pyshark: Python wrapper for tshark, allowing python packet parsing using Wireshark dissectors*. URL: https://github.com/KimiNewt/pyshark (cit. on p. 67).

[20] L. Kong, G. Huang, and K. Wu. "Identification of Abnormal Network Traffic Using Support Vector Machine". In: *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. 2017, pp. 288–292. DOI: 10.1109/PDCAT.2017.00054 (cit. on p. 7).

[21] P. Laukkanen. "Data-Driven and Keyword-Driven Test Automation Frameworks". In: (2006). URL: http://eliga.fi/writings.html (cit. on p. 20).

[22] Y. Liu, W. Li, and Y. Li. "Network Traffic Classification Using K-means Clustering". In: *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*. 2007, pp. 360–365. DOI: 10.1109/IMSCCS.2007.52 (cit. on p. 9).

[23] *Long Short-Term Memory (LSTM) networks*. URL: https://www.mathworks.com/discovery/lstm.html (cit. on p. 12).

[24] M. Lopez-Martin et al. "Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things". In: *IEEE Access* 5 (2017), pp. 18042–18050. DOI: 10.1109/ACCESS.2017.2747560 (cit. on p. 12).

[25] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[26] V. M and S. Manmadhan. "Self Learning Network Traffic Classification". In: *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. 2015, pp. 1–5. DOI: 10.1109/ICIIECS.2015.7193038 (cit. on p. 9).

[27] J. Manzoor, I. Drago, and R. Sadre. "How HTTP/2 is changing web traffic and how to detect it". In: 2017-06, pp. 1–9. DOI: 10.23919/TMA.2017.8002899 (cit. on pp. 4, 13, 17).

[28] A. Munther et al. "Network traffic classification — A comparative study of two common decision tree methods: C4.5 and Random forest". In: *2014 2nd International Conference on Electronic Design (ICED)*. 2014, pp. 210–214. DOI: 10.1109/ICED.2014.7015800 (cit. on p. 8).

[29] *Packet capture toolkit*. URL: https://www.tracewrangler.com/ (cit. on p. 65).

[30] C. Patton. *Good-bye ESNI, hello ECH!* URL: https://blog.cloudflare.com/encrypted-client-hello/ (cit. on p. 16).

[31] Rahul et al. "IP Traffic Classification of 4G Network using Machine Learning Techniques". In: *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*. 2021, pp. 127–132. DOI: 10.1109/ICCMC51019.2021.9418397 (cit. on p. 4).

[32] S. Rezaei, B. Kroencke, and X. Liu. "Large-Scale Mobile App Identification Using Deep Learning". In: *IEEE Access* 8 (2020), pp. 348–362. DOI: 10.1109/ACCESS.2019.2962018 (cit. on p. 12).

[33] S. Rezaei and X. Liu. "Deep Learning for Encrypted Traffic Classification: An Overview". In: *IEEE Communications Magazine* 57.5 (2019), pp. 76–81. DOI: 10.1109/MCOM.2019.1800819 (cit. on pp. 4, 12).

[34] S. Rezaei and X. Liu. "How to Achieve High Classification Accuracy with Just a Few Labels: A Semi-supervised Approach Using Sampled Packets". In: *CoRR* abs/1812.09761 (2018). arXiv: 1812.09761. URL: http://arxiv.org/abs/1812.09761 (cit. on p. 12).

[35] *Robot framework*. URL: https://robotframework.org/ (cit. on p. 20).

[36] D. Shamsimukhametov et al. "Is Encrypted ClientHello a Challenge for Traffic Classification?" In: *IEEE Access* 10 (2022), pp. 77883–77897. DOI: 10.1109/ACCESS.2022.3191431 (cit. on p. 17).

[37] H. Singh. "Performance Analysis of Unsupervised Machine Learning Techniques for Network Traffic Classification". In: *2015 Fifth International Conference on Advanced Computing Communication Technologies*. 2015, pp. 401–404. DOI: 10.1109/ACCT.2015.54 (cit. on p. 9).

[38] *SplitCap - A fast PCAP file splitter*. URL: https://www.netresec.com/?page=SplitCap (cit. on p. 66).

[39] *SSHLibrary*. URL: http://robotframework.org/SSHLibrary/SSHLibrary.html (cit. on p. 37).

[40] *Support Vector Machine (SVM) algorithm - javatpoint*. URL: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm (cit. on p. 7).

[41] V. Tong et al. "A Novel QUIC Traffic Classifier Based on Convolutional Neural Networks". In: *2018 IEEE Global Communications Conference (GLOBECOM)*. 2018, pp. 1–6. DOI: 10.1109/GLOCOM.2018.8647128 (cit. on p. 17).

[42] U. Trivedi and M. Patel. "A fully automated deep packet inspection verification system with machine learning". In: *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. 2016, pp. 1–6. DOI: 10.1109/ANTS.2016.7947802 (cit. on p. 19).

[43] P. Yadav. *Decision tree in machine learning*. 2019-09. URL: https://towardsdatascience.com/decision-tree-in-machine-learning-e380942a4c96 (cit. on p. 8).

[44] J. Yang, J. Narantuya, and H. Lim. "Bayesian Neural Network Based Encrypted Traffic Classification using Initial Handshake Packets". In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks – Supplemental Volume (DSN-S)*. 2019, pp. 19–20. DOI: 10.1109/DSN-S.2019.00015 (cit. on p. 18).

[45] A. Yu. *How to teach a computer to see with Convolutional Neural Networks*. 2018-11. URL: https://towardsdatascience.com/how-to-teach-a-computer-to-see-with-convolutional-neural-networks-96c120827cd1 (cit. on p. 11).