UCLouvain

École polytechnique de Louvain

Machine Learning for PQUIC plugins identification

Author: **Josette AOGA**
Supervisor: **Ramin SADRE**
Readers: **Siegfried NIJSSEN, François MICHEL**
Academic year 2021–2022
Master [120] in Cybersecurity

Abstract

Network visibility is an important aspect for a network administrator and also for a network operator. Being able to identify properly the different protocols or applications that are used in a network traffic, can be helpful not only for network management but also for the detection of intrusion. Many researches have been performed to classify efficiently the network protocols. Some are based on the protocol port number located in the protocol header and others on the protocol fingerprint in the payload. Both techniques have shown inaccurate results. The most common approach used nowadays is Machine learning. The works performed using this technique, for the protocol classification, show accurate results till now.

In this thesis, we used machine learning algorithms to identify extensions of a protocol in a network. We have worked on the Pluginizing QUIC (PQUIC) framework, where we identified different plugins enabled. We based the classification on two different datasets. We built two datasets. The first is based on network statistics aggregated per flow. The second dataset is built considering the logs of QUIC protocol during the communication between the client and server. We performed several machine learning algorithms on both datasets separately. The learned models obtained interesting accuracies. Nevertheless, one of the plugins is difficult to distinguish when instances without any plugin enabled are considered in the datasets.

Acknowledgements

At this point, I would like to thank my thesis supervisor, Professor Ramin Sadre, for having accepted to guide me throughout this thesis. His valuable advice, suggestions, support and especially his patience during my thesis allowed me to improve the quality of this thesis.

I would also like to thank Professor Siegfried Nijssen and T.A François Michel for taking the time to read my thesis.

Finally, I am grateful to my beloved family for their unconditional support during my master's course despite the distance.

Contents

Chapter 1

Introduction

1.1 Context

The identification of protocols in network traffics is very important for network administrators and system developers. Indeed, a network administrator is willing to know the incoming and outgoing data of his network and also be able to manage the network traffic. Identifying protocols will help in the detection of intrusions, to ensure a high network performance in terms of latency for instance, and provide different Quality of Service(QoS). However, it is not easy to identify the protocols in a network. First, because the majority of the network traffics is encrypted. Encryption is crucial since it is used to ensure the confidentiality of users' data online to avoid network attacks [1]. Thus, no classification can be made by inspecting the payload of the packets. Second, since the end hosts could modify various parameters in the protocols, it is impossible to use the port number of the protocols.

Therefore, network protocols like QUIC, which encrypt almost all parts of the packets and allow developers to modify and add new features to its implementation, increase the challenge because new protocols doing specific things will come alive. Indeed, the authors of [2] leverage the flexibility of QUIC to design a pluginized QUIC(PQUIC) framework to enable the engineers to add their own extensions. These extensions are called plugins. This constitutes a big challenge for the administrators, who will have to identify all those new protocol plugins used by different network applications.

1.2 Goal

In this thesis, our goal is then to focus on the identification of those plugins in a network. Since the traffic is encrypted, we could not use any literature techniques

which focus on the header or on the content of the packets. The best solution to adequately identify them is to use Machine Learning techniques. Many researchers have used machine learning techniques for this purpose [3, 4, 5, 6].

For the sake of simplicity, we have focused on three(03) basic plugins created by the authors of PQUIC:

- The plugin of Forward Erasure Correction
- The plugin of Monitoring
- The plugin of Multipath

The objective is to build a machine learning model based on the collected data to be able to classify even in encrypted network the different plugins. Specifically, we have first set up the work environment to collect QUIC traffic when injecting each plugin. We then analyzed the data collected to extract features which will be useful for the identification. Then we trained the model on the collected data that will help to identify these plugins in a real environment.

1.3 Structure of the thesis

The thesis is organized in three(03) chapters

In the chapter 2, we will present the different methods that are already in used to identify protocols and to distinguish protocols version. We will also explain in more details the machine learning approach. Next we will present the Pluginizing QUIC framework along with an explanation of how QUIC protocol works and the different plugins that we consider in our work.

On Chapter 3, we will focus on the construction of the dataset by first explaining how we captured the packets, second how we selected and extracted features, third we will explain how we build the ground of truth.

Chapter 4 is shared in two parts, the first part will present the implementation of the algorithms as well as results and observations after using the dataset containing network features. In the second part, we will present the result about another dataset built now based on the logs of QUIC protocol during communication.

Chapter 2

State of the art and background knowledge

2.1 The different methods of Protocol Identification

Many researches have being done for the classification of the network traffic. These researches lead to the different techniques of classification that exist nowadays. First, researchers tried a classification technique based on the protocols' port, next they focused on Payload Based Technique and also on machine learning techniques. The following sections will present those techniques as well as their limitations.

2.1.1 Port-based Techniques

This technique is nearly the first technique exploited to classify network traffic. It is based on the well-known ports number of the protocols. In another word, to identify an application in a network traffic, we have to simply verify their port number on the packet header. Indeed, the Internet Assigned Number Authority (IANA) [7] assigned to each protocol a fixed number that is known by the whole community and is used to ensure connectivity services. For instance as it is shown in the Table 2.1, the port number 53(DNS) is used to resolve the domain name in IP address on the internet, for web applications, the port 80(HTTP) is used and to ensure the security of sensible data being transferred, the port 443(HTTPS) is used. The port number can be easily accessed since it is on the packet header and are usually not affected by encryption, thus allowing a fast flow classification. It also eases the packet filtering made by firewall and ACL(Access List).[8] [3]

However, this technique is actually rejected as it provides improper interpretation of the results. In [9] and [10], the classification results are between 50% and 70% of

Applications	Assigned ports
SSH(Secure Shell)	22
DNS(Domain Name System)	53
IMAP4 (Internet Message Access Protocol 4)	143
HTTP(HyperText Transfer Protocol	80
Kerberos(Authenticating agent)	88
HTTPS (HTTP over Tls/ssl)	443
POPS (Post Office Protocol 3)	110

Table 2.1: IANA assigned port numbers

accuracy, while in [11] the result is under 20% of accuracy. This inefficiency is due to the advents of peer to peer(p2p) protocols [1] which use dynamic port number and also the tunneling techniques [2]. The port numbers changed continually. The other thing is that with this random port number, applications can not be detected anymore, and then they can go through the firewall without being blocked [12].

2.1.2 Payload-based Techniques

The payload based technique also called Deep packet inspection(DPI) or pattern matching, is the first solution to the port based technique. On the port based method, we face the problem of the dynamic port. To solve this problem, researchers focused on the content of the packets. A packet contains a header, a payload, and a tailler. The header most of the time contains the necessary information to ensure connectivity, like the ports numbers, the IP addresses, etc. The payload contains the data to be delivered, it is the only part of the packet received by the destination or the source.

To perform the payload analysis, the content of the packets are examined to find a specific string which is the signature of the protocols in the network. This will help to extract the exact pattern which distinguish the application from one another. The Table 2.2 shows an example of strings of some p2p applications identified by Karagiannis et al. in [13]. Also, Sen et al. in [14] proposed an efficient technique based on application signatures to detect p2p applications. Indeed, the authors collected signatures from the analysis of documentation and from network trace inspection. With these signatures, they set up a filter online to track efficiently p2p traffic. In addition, M.Roughan et al. in [15] used application signatures to develop a framework for the measurement based on classification of traffic for QoS(Quality

[1]Peer to Peer is used to share file more easily on the internet between devices as it uses distributed architectures where all peers are equals.

[2]Tunneling consist of encapsulate or wrap packets in another packet so that it can bypass rules

of Service). [4] [1] [16]

P2P Protocol	String	Trans. Protocol
Edonkey 2000	Oxe319010000	TCP/UDP
	0xc53f010000	
BitTorrent	“0x13Bit”	TCP
Direct Connect	“$MyN”,”$Dir”	TCP
	“$SR”	UDP
Fast track	“Get /.hash”	TCP
	0x270000002980	UDP

Table 2.2: String at beginning of the P2P’s payload by Karagiannis

Though the approach is widely used, it presents some limitations. Firstly, the different solution can only detect known signatures. So it works only when the signature of the application is available, which means that new protocols could not be identified. The other limitation is the fact to check inside packet content without people authorization, caused a privacy’s issue, and Allman and Paxon in [17] presented several issues about the legality of this method. This technique also consumes a lot of resources. The big issue of this approach is the fact that the content of packets is encrypted. Considering this last limitation, thinking about using DPI is no longer feasible since the majority of the network traffics today are encrypted.[4]

2.1.3 Machine learning Techniques

Machine learning has been used to solve complex problems in many fields like health, finance, marketing, cybersecurity. Its techniques and algorithms are reputed to give efficient results and prediction. It has also been used in networking [18] [19] as the identification of protocols has become very difficult. With machine learning, there is no detection based on specific element in network packets. This technique uses statistical information from the network traffic. It consists of identifying network features computed based on statistics of traffic derived from monitored packets or flow. In another word, each flow is described by a set of statistical features with a value computed based on this information. Data are then collected based on the features to form a data set which will be used to perform the learning and make the prediction. Each line of this data set is one instance, it can contain a label or not depending on the type of learning that will be performed.

As it is shown on the figure 2.1, the original data collected is shared in two parts, the training data and the test data. The training data is most of the time more than the test set. Now the training set is used for the learning and then the

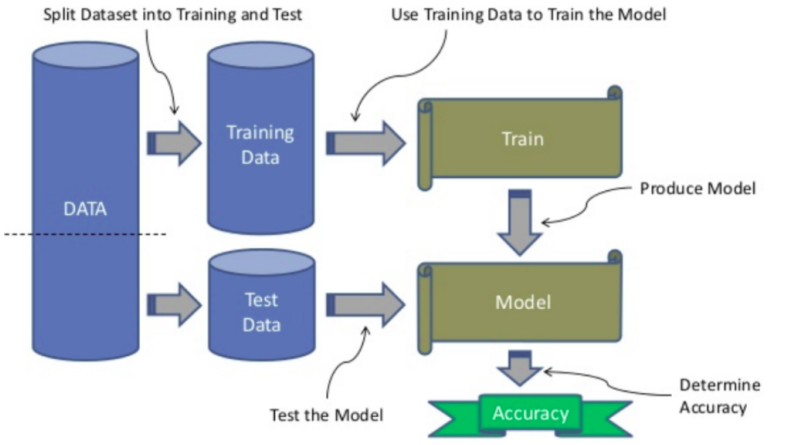

Figure 2.1: Machine learning cycle. *Source: medium.com*

model resulted from this is run over the test data to make prediction based on the accuracy. The accuracy help confirm the efficiency of the model and also to see if the features used better characterized the problem.

Analysis We have presented the most used techniques to identify protocols in a network traffic. The port based is very simple to use, and it identifies clearly the protocol since it is based on fix known ports. But it can no longer be used in today network because of dynamic port number. One solution to this technique is the Payload based approach, where the identification is made based on protocol signatures localized on the packet content. This again is no more efficient when the packet content is encrypted, so nothing can be accessed without the key of decryption. The researchers focused on machine learning technique to identify the protocols which till now give good accuracies. In this thesis, we work on an encrypted network, so obviously the technique who fit the most is the one on machine learning. In the next section, we will present it in more detail.

2.2 Machine learning approach

We can distinguish many learning types in machine learning. We present here the supervised learning and unsupervised learning that are the most known and used.

2.2.1 Supervised learning

In this learning, we have clear knowledge of what should be the output values of our samples. Its goal is to learn a function that, given a sample of data and the desired

results, best approximates the relationship between input and output observable in the data. In another word, the goal is to have a training set containing instances(X) and for each instance we map a label(Y) which identify this instance.

The learning is performed on the training to then obtained the model. As it is show on Figure 2.2, the data and the response are known and are passed to the algorithms in input and a model is built based on that. Next unseen data are passed as input to the model and based on that a prediction is made(Figure 2.3). [3]

Figure 2.2: Training step

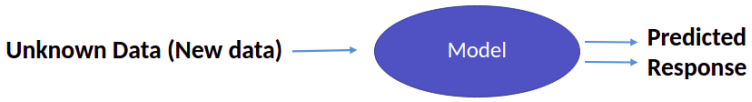

Figure 2.3: Prediction step

Mathematically, we have a training set containing $(x_1, y_1), (x_2, y_2), (x_3, y_3)...(x_m, y_m)$ where x_i is the feature vector of the $i^(th)$ line and y_i the target(label) of this instance. [3].

In supervised learning, two techniques are the most used:(1)the classification where y takes a finite number as value and (2)the regression where y takes continue values. In the followings, we describe them in detail.

2.2.1.1 Classification

In classification, each instance belongs to a category of element. In Figure 2.4 for example we have two classes: class of "cats" and class of "non cats". To write this in order that the machine understands, these values are assigned numerical values, as 1 for "cats" and 0 for "non cats". Classification method is used in many fields nowadays like in finance and banking for credit card fraud detection, in IT security for the detection of unwanted email "spam" and "not spam". In marketing for text sentiment analysis and also in health field, to predict whether a patient has a particular disease or not.

Concretely, in network applications, as it is shown on Figure 2.5 a classification process is performed in five steps.

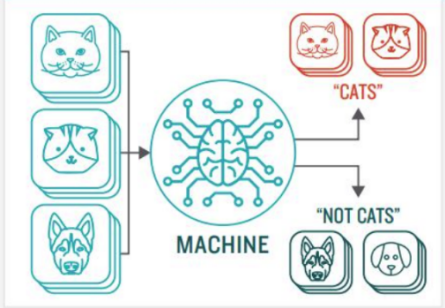

Figure 2.4: Classification example. *Source: medium.com*

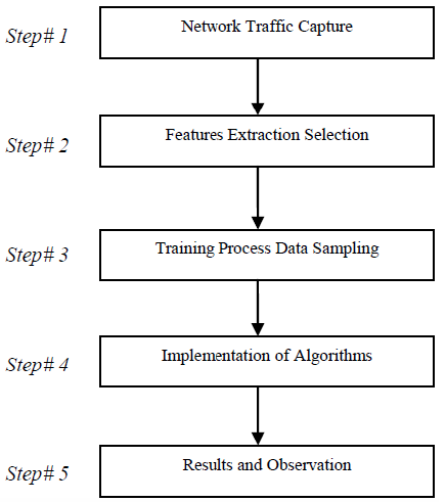

Figure 2.5: Network traffic classification steps. *Source: [3]*

Step 1 In the first step, network packets are captured in real time using network monitoring tools. Attention must be paid to isolate the relevant traffic and to capture the data from the relevant point in the network architecture.

Step 2 At this stage, data are processed and analyzed to extract meaningful features that characterize the most the data. Features like the length and the size of the packets, the flow duration, the inter-arrival time can be considered.

Step 3 Next, the data are subjected to a sampling process to get the training set that represent the most accurately the data. Once we separate the training set from the test set, it will be used for the learning process.

Step 4 The following step is then the step of the implementation. Here, machine learning algorithms are run over the training set and the test set to make a prediction

at the end. Many machine learning algorithms have been used on network traffic classification like: Decision tree, random forest, support vector machine [3, 6, 12, 5], but one can also use boosting algorithms like Adaptive boosting [20] and gradient boosting to get better prediction. We will present them in the following

Decision Tree Decision tree algorithm belongs to the category of supervised learning algorithms, it can be used to predict a continued value (regression) or a category (classification). As its name suggests, this algorithm is based on the construction of a tree, which makes the method quite simple to explain and easy to interpret. It has a root node, internal nodes that represent the conditions on the attributes, branches that represent the results of the conditions and leaves that represent the labels. It has been tested on many problems, but in some situations, its results are biased because of its high predisposition to over fitting [3]. [21]

Random Forest Random forest is an algorithm of supervised machine learning techniques that is used for the classification and regression. It helps to reduce the over fitting of the dataset and by the way increase the precision. It is a collection of a large number of individual decision trees that function as an ensemble. The random forest uses the bagging[4] method to generate the required prediction, what makes it different from the classical decision tree. The decision tree forest algorithm performs training on multiple decision trees trained on slightly different subsets of data. So Each of them works like a normal decision tree and makes a prediction. It deduces the result based on the predictions of all decision trees. [22]

Indeed, to perform the classification, the dataset(training data) will be divided into subsets. Each subset will contain observation and features, set randomly during the splitting step. The new datasets are given to every decision tree in the forest. Each tree will output a decision, the prediction will be the average or mean of the result from the different trees of the forest. This increase considerably the precision of the outcome.

Adaptive Boosting Adaptive boosting as it name imply is a boosting algorithm. Boosting as the bagging is an ensemble technique that consist of assigning higher weights to instances with high misclassification rate and lower weights to instances with low classification rate. The weighted data will go through the classification process again and apply the analytical method to the re-weighted data. The goal is to reassemble weak learners classifier, train and correct them gradually so that they classify with a minor error. Its major difference with bagging

[3]When the model learned not only the data of training but also the noise that are information which doesn't help in the problem

[4]The fact to use different samples of the training data rather than just one sample

method is that the classifiers are dependent from the previous one, so they can not work simultaneously. Adaboost specifically works with decision trees classifiers. In particular, it only creates trees with a node with two leaves, called Stumps. Stumps are considered as weak learners and constitute the base of the learning. The error of the first stump influences how other stumps are made. This improved the accuracy of the model and also the fact that the instances weight are updated after each iteration. To make the prediction, each weak learner decision is weighted according to the score they received. The strong learner's prediction will be the average of all the weak learner's responses. [23]

Gradient Boosting Gradient boosting(GBoost) is also a boosting algorithm as AdaBoost. They have similar process. In another word, it is a set of weak learners created successively to form a strong learner and also each weak learner is trained to correct the errors of the previous weak learners. GBoost particularity is that, each weak learner prediction is important, they all have the same weight in the final prediction. The particularity of Gradient Boosting is that it tries to predict at each step not the data itself but the residues. Residues is the gap between the prediction of the algorithm being created and the reality. Base on last predictions, new residuals are computed and new weak leaner train to predict these residues. The prediction of the new is often slightly better than the previous ones. To get the final prediction, each weak learner is queried, and all responses are summed up. The sum is the final prediction. [24]

Step 5 Finally, the step of the results and the observations. This step present the results of the algorithms about the classification: The accuracy for the training and test data that will be used to qualify the performance of the model.

Shafiq et al. [3] went through those steps to classify some network protocols. First they collected packets with wireshark[5] and they used NetMate(Network Measurement and Accounting meter)[6] to extract meaningful characteristics from the packets. After this they run four machine learning algorithms on the dataset to identify protocols like DNS, P2P,FTP, WWW and telnet applications. Decision tree was the algorithm which gives good accuracy

Moreover, Manzoor et al. in [6], have worked on the detection of http1 and http2 in network traffic. Their objective was to characterize HTTP1.1 and HTTP2.0 flow in a network traffic by using machine learning classification technique. They used a data set of network flow that they got from two different sources. The first one is on a campus network and the second one on an ISP network. Both from June to

[5]It is a network monitoring tool that is used to capture and analyze network traffic

[6]A network measurement tool. Can be also used to capture packets

December 2016. For the features extraction step, they used Tstat(TCP Statistic and Analysis Tool). For the labeling, each instance of the dataset is labeled based on the NPN(Next Protocol Negotiation) and ALPN(Application-Layer Protocol Negotiation) messages exchange during the TLS handshake.

Auld et al. in [25] worked on data takes for two days and eight months apart on a single website. They succeed to classify the different flows based on statistics derived from packets header without port or host information by using a Bayesian trained neural network. They got 99% for the flow of two days and 95% for the other.

2.2.1.2 Regression

The Regression is the other technique of supervised learning where the training is performed on continue values of target. The purpose of the regression is to estimate an output (numerical) value from the values of a set of input characteristics. For example, estimating the price of a house based on its area, number of floors, location, etc. Thus, the problem amounts to estimating a computational function from training data.

The objective is to find a so-called prediction function or a cost function that describes the relationship between x and y such that the equation $y = f(x)$ is fit where y is the target variable, x the explanatory variable and $f(x)$ the prediction. So from a known value of x, we can give a prediction of the values of y. We didn't provide more details for this method, as the majority of the works on protocols identification are classification tasks.

2.2.2 Unsupervised learning

Unlike the supervised learning, in unsupervised learning there is no ground truth, i.e. no labels are assigned to the data in order to correct it when it makes an error. It consists of training the model on data without labels. The machine explores the data without any indices and look for eventual patterns, similar properties that it will use to classify the data. It ingests vast amounts of data, and uses algorithms to extract relevant features required to label, sort and classify data in real time without human intervention. Instead of automated the decision and predictions, this technique will help humans identify the different patterns and relations that they can not find by themselves. It is most of the time used in the domain of cybersecurity. Among its algorithms we have the clustering, the association and dimensionality reduction. We will present the clustering method since it used in some papers for the protocol identification, and we will present dimensionality reduction, since we used one of its algorithms in our experiment.

2.2.2.1 Clustering

The clustering is the most used technique in unsupervised learning, it consists of grouping data by similar patterns. Each group must contain similar data, and different data must be in distinct groups. We distinguish three main algorithms of clustering:

Hierarchical clustering It consists in building a sequence of nested clusters. A tree hierarchy can represent these relationships between clusters of different levels. We distinguish bottom-up algorithms that build classes by successive accumulation of objects two by two. And the descending algorithms carry out progressive dichotomies of the set of objects. The lower you are in the tree, the more similar the observations are. This method can be applied when we do not know in advance the number k of groups in which the observations will be distributed.

Non-Hierarchical clustering The goal of this method is that the number of clusters to constitute is known in advance. For a given distance measure of the individuals and a known number of classes k, one can easily imagine a simple and optimal classification solution: enumerate all conceivable grouping possibilities and keep the best one. However, this solution is not applicable in practice because the number of possible combinations of groupings becomes quickly enormous. But approximate solutions can be obtained thanks to heuristics and several algorithms of this kind. Their fundamental principle is contained in the method of mobile centers. The most popular algorithm from this family is the classical k-means algorithms that, given an integer k and some points, will divide the points in k homogeneous and compact clusters.

Researchers have also tried clustering in networking. Bernaille et al. worked in [26] on the identification of applications by using only the first few packets of the flow. They proposed a model which first based on the behavior of the application, clustered the TCP flows, and secondly those clusters are used to classify the applications in an online area. The first five packets gives up to 80% of accuracy. Erman et al. presented, in [27], how they used clustering algorithms like k-means to classify network traffic. They find that for large value of K the model over learned, and then it is more subject to over-fitting.

2.2.2.2 Dimensional reduction

Dimensional reduction allows, by reducing the number of variables in the training data, to better visualize the data to perform more reliable comparisons and analysis. There are two main methods to reduce dimension. The first one consists on selecting the most interesting variables and transporting them in a smaller space,

which involves limiting the number of features to be processed. The second method is used for feature extraction, i.e. it creates more relevant variables. This operation is done beforehand by analyzing the data in the high-dimensional space. TSNE(t-distributed stochastic neighbor embedding) and PCA(Principal Component Analysis) are most used algorithms in this technique.

2.3 Presentation of Pluginizing QUIC(PQUIC)

After the presentation of the different network traffic classification techniques that have been tested so far and the most efficient among them, we will describe in the next lines the different plugins of PQUIC that we will have to identify in the network traffic.

In order to better understand how PQUIC works, we first present the QUIC protocol, on which it is based.

2.3.1 Presentation of QUIC

Among the transport layer's protocols, the main ones are TCP(Transmission Control Protocol) and UDP(User Datagram Protocol). Many applications rely on TCP to communicate, especially when it comes to ensure reliability of the packets. In addition to this, there is the challenge of data security which is managed at the transport level by adding to TCP, the TLS(Transport Layer Security) protocol that encrypts the exchanged messages. Ensuring these functionalities slows down the communication and increase considerably the time of latency for HTTP traffic. In the followings, we will present some TCP limitations.

Handshaking delay TCP realized the connectivity between the client and the server in one round trip at least. Moreover, if the data has additional requirements, such as encryption, TLS will add two more round trips. This type of operation requires establishing a new connection (a handshake) each time, so there will often be several round trips of requests and responses until the connection is established. Due to the inherent latency of long distance communications, this can add significant overhead to the overall transmission. [28]

Head of lines blocking delay Another limitation of communication over TCP is the problem of head-of-line blocking at the application layer with the HTTP/1(Hypertext Transfer Protocol) protocol. It occurs when a set of packets (and thus requests) is blocked by the first packet (and therefore the first request). This problem was handled in HTTP version 2 (HTTP2), which corrected the problem by implementing multiplexing on TCP, the purpose of which was to be

able to reuse established server connections for multiple client connections. This solution created another problem at the transport level. TCP buffers all subsequent packets when a packet is lost until it is successfully retransmitted. As a result, some HTTP connections, such as video streaming, have experienced reduced performance with HTTP/2. [28]

Due to the important limitations of transport layer protocols, QUIC protocol was created by Google in 2013 [28]. The primary goal of QUIC was to improve the performance of connection-oriented web applications that currently use TCP by using multiplexed connections and reducing latency during connection and transport in particular. QUIC is built on the HTTP/2 flow model and integrates it into the transport layer, so that a single connection can progress on a flow even if packets containing data from other flows are lost, thus mitigating the problem of head-of-line blocking in the transport as well as in the application layer. [28]

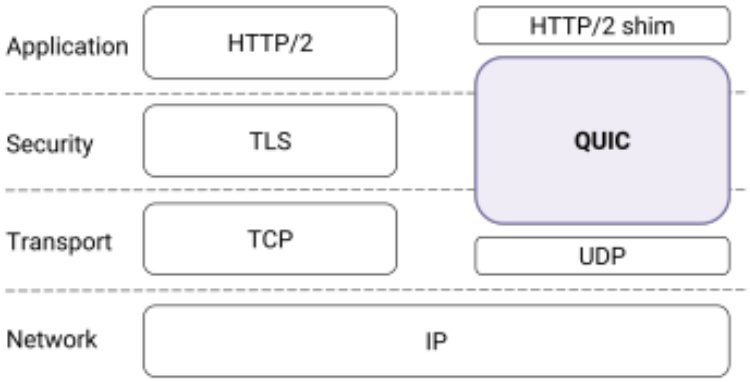

Figure 2.6: QUIC in HTTPS stack. *Source: [28]*

To establish a secure HTTP connection over TCP, we need six steps. On the other hand, with QUIC, it is done in just two steps. This is also shown in the picture 2.6 where we can see that QUIC integrates TLS by default. It is built over UDP (User Datagram Protocol) rather than TCP and incorporates encryption by default. As UDP is not reliable, each QUIC flow is monitored independently and lost data is retransmitted at the QUIC level and not at the UDP level. In other words, if an error occurs in one flow, the protocol stack can continue to serve other flows independently. This greatly improves performance on error-prone links because, in most cases, a significant amount of data can be received before TCP notices that a packet is missing or broken. This data will then be blocked or even discarded while the error is corrected, whereas, with QUIC, this data is processed while the single multiplexed stream is repaired.

QUIC ensures many functions, we present two of them in the following.

2.3.1.1 Multiplexing

The TCP protocol uses the TCP ports and IP addresses of connected nodes to identify a connection. For this reason, a client can't communicate with the server on multiple ports with a single connection. Indeed, when a TCP packet is lost, no flow on the HTTP2 connection can move forward until the packet is retransmitted to the other party. QUIC has handled the situation differently. Lost packets only affect their specific flow. Therefore, a QUIC connection is not necessarily tied to a specific port (in this case, a UDP port), IP address or endpoint. Each flow frame can be immediately distributed to that flow as it arrives, so that lossless flows can continue to be reassembled and progress through the application.

Overall, the idea is to leave a connection to the server open for a period of time rather than closing it immediately so that the same connection can be reused in case the client makes a new request to the server. Multiplexing allows multiple requests on the same connection and reduces latency. [28]

2.3.1.2 Forward Error Correction

Lost packets are not a big problem when transporting data via QUIC. Thanks to a simple XOR-based error correction system, it is unnecessary to resubmit the corresponding data. The data can be built up at any time using FEC (Forward Error Correction) packages, which are backups of the original packages for a data group.

However, error correction does not work if several packages in a dataset are missing. The approach used XOR-based FEC, which can only recover one package. This means that if two or more packages are lost, the FEC package becomes useless. Eventually, support for XOR-based FEC was removed from QUIC in early 2016. [28]

Now that we have presented QUIC and its advantages, we will present next, PQUIC and its plugins.

2.3.2 Presentation of PQUIC Plugins

PQUIC is built on a simple implementation of QUIC protocol, which is `picoquic` [29]. On the opposite of the majority of protocols which allow only a small external API to modify or add anything, PQUIC provides an API which is flexible enough to integrate into QUIC the different plugins.

Concretely, the idea is to make the QUIC protocol dynamically extensible by including an eBPF VM [30] inside PQUIC. An eBPF VM is a lightweight virtual machine that was added to linux kernel in 2014 [2]. It implemented a very simple processors which allows running byte code which is compiled into native code.

The eBPF VM is very restrictive since it contained a verifier which control many specific elements before running the code. The eBPF version inside PQUIC is a light version of the initial one with monitoring capabilities. This is to allow the extension of QUIC by adding new stuffs. Also, PQUIC based on the flexibility of QUIC packet format, add a lot of extensions. Moreover, as QUIC's packets are encrypted what provides a strong layer of security and its multi stream capabilities, PQUIC takes those advantages to allow clients and servers to exchange byte code over QUIC.

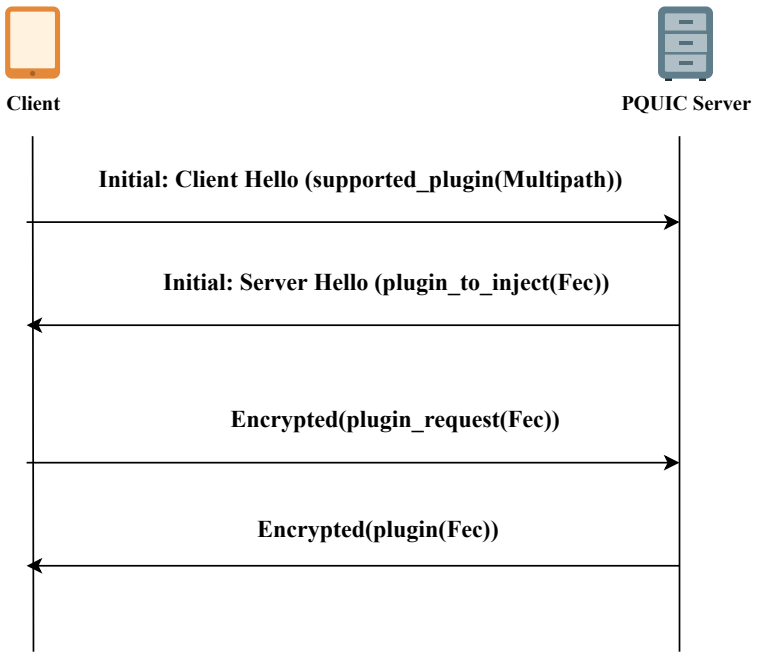

Figure 2.7: PQUIC's first connection steps

The Figure 2.7 describe how the plugins are exchanged between the client and the server during the first connection. The first element is the "Client Hello" initially sent by the client, where it exposes the different eBPF bytecode(plugins) that it supports. Second, the server answer with a "server hello" where it decides to inject a new plugin inside the client. The client answer by sending a "Plugin_resquest" with the plugin proposed by the server since it doesn't have it, and then the server inject the plugin. Next time (Fig 2.8) for their communication, the client present all the plugins it supports in the "Client Hello", the server chooses one on the "Server Hello" and then they start exchanging streams of this plugin. Notice that a part from the initial messages, the other messages are all encrypted for both types of connection. There are different plugins, which can be proposed, we will present three of them: plugin of monitoring, multi path and Forward Erasure Correction.

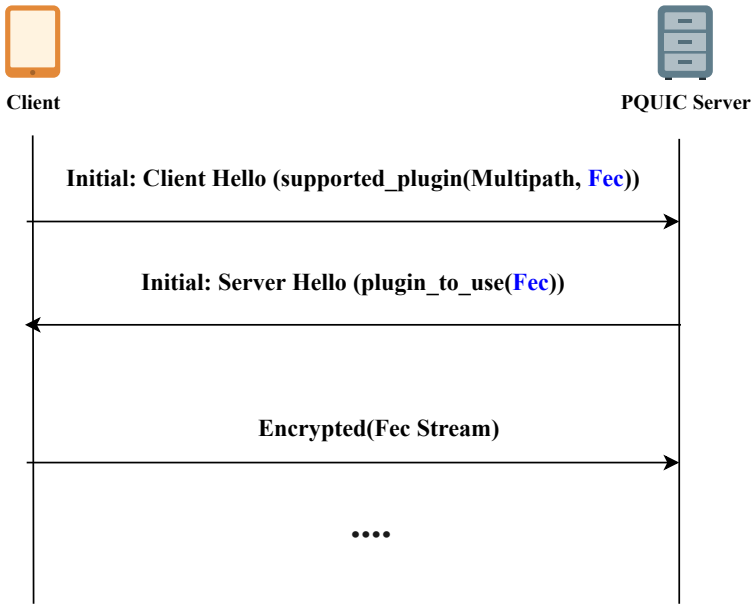

Figure 2.8: PQUIC's next connection steps

Monitoring This plugin is created to provide insight statistics about the PQUIC traffic. QUIC encrypts the whole packet and only a slight part of the header is in clear, this makes it difficult to collect information about the losses and retransmissions on transited flows.[31]

Despite the presence of the Spin Bit [32] in the packet header of QUIC, that allows to measure round-trip time(RTT) of the connection, there is no help in detecting losses and retransmissions. For this reason, the monitoring plugin has been created to provide more useful data that will help network administrators in troubleshooting. Indeed, functions have been added in PQUIC to collect data on the number of bytes sent and received, the number of packets sent and received, changes in the RTT, changes in the congestion window. These features are collected on the one hand during the handshake between the client and the server, and on the other hand they are updated once the connection is established. These different data collected at different steps of the connection are then conveyed over a UDP connection to a local daemon.[2]

The plugin of monitoring is implemented on 500 lines of C code and 86 kilobytes of byte code are injected by the server during the first PQUIC connection(Fig2.7).[2]

Multipath The plugin is an eBPF extension built on top of `picoquic`, similar to the implementation of multipath QUIC extension[33]. Indeed, on the opposite of the other transport layer protocol like TCP, UDP for instance, QUIC connection

doesn't match on the 4-tuple, the IP and the port source, the IP and the port destination. QUIC uses instead connection IDs included in packets' header, which is used to identify the streams [34]. The IDs allow QUIC to support multiple paths, even in a single connection [35].

Nevertheless, the implementation of multipath on network protocols is a big challenge, as shown for TCP in [36]. The PQUIC multipath plugin is based on the implementation of the QUIC multipath plugin and offers a better way to do it. This plugin uses path IDs and hosts addresses for connectivity between hosts so that it will associate each pair of host addresses (IP source, IP destination) with a path ID during the connection step. During the communication between the hosts, packets are scheduled based on the round-robin [7] scheduling technique on the available route, and acknowledgments are sent on a new ACK frame. The packets sent follow another scheduling algorithm set up on the shortest round trip time path [36].

The plugin counts 2600 lines of C code and 138 kilobytes of bytes code are injected, which is higher than the one of monitoring.

Forward Erasure Correction(FEC) Ensuring the reliability of packets is very important and involves a lot of research. In some cases, packets are simply retransmitted when a loss occurs. Others proposed a redundancy solution so that the receiver can recover the losses without any retransmission from the sender: [37, 38, 39]. Obviously, this is very expensive since the sender has to do a lot of computation to calculate the parity code or more complex codes so that the receiver can recover the missing packets from the redundant data sent by the sender

In order to reduce this limit of computation, the FEC plugin proposed by PQUIC's authors provides an efficient and light way of the technique. They added a Repair Symbol(RS) code inside each packet that contains a QUIC frame that will be used to recover it when it is lost. To obtain the RS code, they provided two functions implementing different Erasure Correcting Code(ECC). The former is similar to the simple XOR code implemented by Google in QUIC [40]. It performed this simple XOR between all packets and obtained a XOR RS code. But, it can only restore the lost of one packet at once. The second, used a Random linear Code(RLC)[41] methods to compute the RS code. With this technique, lost can be recuperated from more than one packet, what makes it more efficient than the first implementation.

The plugin is also written in C and contains 238 kilobytes of byte code that can be automatically downloaded from the server during the injection.

[7] scheduling algorithm assigning to each process the same time

Security aspects PQUIC's authors also worked on the security and the safety concerns, since when downloading data from an online server, it is important to trust the server to ensure what you are installing on your system doesn't contain any malware. For this reason, many safeguards have been put in place, we have the followings between them

- The eBPF virtual machine validates some of the codes that has been pushed what lead to a separation of the plugins from each other and from PQUIC
- Plugins are certified, so it is possible to verify that the plugin is certified by a trusted authority. Also, it is possible to use validation tools to verify that eBPF code terminate correctly. In another, it will verify that it doesn't contain infinite loops that would break the existing connection. For this, an automated termination checker was used to verify the termination. [2]. Moreover, the plugins can be validated by verifying the cryptographic certificates attached to the plugins.

Chapter 3

Dataset Building

In order to learn machine learning models able to identify PQUIC plugins in a network, we need a dataset for the training process. Unfortunately, there are no available datasets for PQUIC plugins. In this chapter, we present how we build the datasets we used to train the different machine learning models in order to identify PQUIC plugins. The dataset building task is split into two parts. Indeed, we built two different types of datasets, each one allowing us to accomplish a specific task. In the section 3.1 we describe the network architecture we created to gather the data. The section 3.3 presents how we built the first dataset based on the network. The section 3.4 is dedicated to dataset building for the text classification task.

3.1 Network architecture

The principle generally used to build dataset for network identification is to initiate some communications in different condition and gather features describing the communications. The collected data are then used to perform the classification task. Unfortunately, there is no well-known accessible web server using PQUIC as communication protocol to facilitate the data gathering as it could be for HTTP. For this reason, we built a specific environment to collect the data for our task. The Figure 3.1 describes the architecture needed to collect features about PQUIC plugins.

It can be noticed on the Figure a server on the internet and a client on a Local Area network(LAN) for the communication. The server is hosted on the Amazon cloud (AWS) and we set up on it the necessary for PQUIC protocol communication. For this, `picoquicdemo` tool is used. Indeed, `picoquicdemo` is an executable provided by Picoquic[1] project which is a minimalist implementation of the QUIC protocol standardized by the IETF. However, this version of `picoquicdemo` does not

[1] `https://github.com/private-octopus/picoquic`

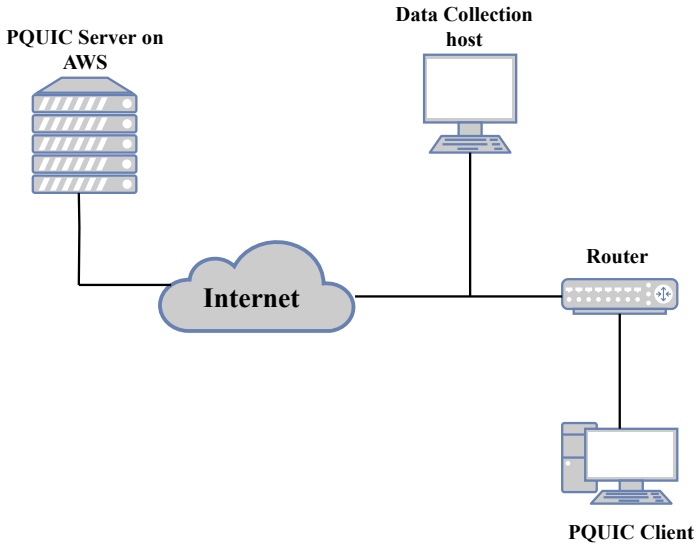

Figure 3.1: Network architecture

support protocol extension through plugins. `picoquicdemo` tool we used is the one provided by PQUIC project and available at `https://github.com/p-quic/pquic`. It is based on the original Picoquic and can act as both the PQUIC client and server. `picoquicdemo` is thus deployed on the AWS server and the client host.

There is also, in the architecture representation, a data collection host locate between the router and Internet. This machine can thus access to any traffic exchanged between the client and the server online. This allows to gather packets exchanged by the client and the server while they communicate using PQUIC protocol and different plugins.

For the sake of simplicity, in our study environment, we do not set up the data collection host where it should be. Instead, we collected the different packets directly from the client machine as we have access to it.

3.2 Data capture environment

For the data capturing, we ran `picoquicdemo` command on our server (Intel Xeon E5-2676 CPU, 1 GB of memory, running Ubuntu 20.04.2). When the server is in listening mode, we run `picoquicdemo` command in client mode on our client host (Intel Core™ i5-8265U CPU, 20 GB of memory, running Ubuntu 20.04.2). For the different instances, the client iteratively connects to the server with a specific plugin and the data exchanged is captured. During the connection, only the initial messages and the Handshakes are made, and the connection closed, so the data

collected are only these types of information. and send the first request to server. A limit value chosen between 10 and 100 kilobytes has been randomly set to collect several kinds of data. In order to have the same value choose for each plugin, a seed has been used in order to generate the same sequence of values. The script used to initiate the communications and capture the data is a python3 script executed by Python 3.8.10 interpreter.

Note that all data are collected by night and 3000 flows have been collected per plugin. The whole dataset contained thus 12000 instances.

3.3 Network features based datasets

3.3.1 Network traffic capture

In terms of network traffic classification, two data levels, both based on packets, are mainly considered for the instances: (1) packet-based level and (2) flow-based level. All datasets building process start by packets gathering, then different measures are extracted from the packets. For packets-based level datasets, each data point represents a packet, while measures are aggregated by a tuple of 5 elements for flow-based level datasets. The different attributes used to aggregate packets into flows are:

1. source IP
2. destination IP
3. source port
4. destination port
5. transport protocol

Both data levels are used in the literature, with a bit preference for flow-based datasets. In our work, we used flow-based dataset as recent studies in the literature prefer using this data level, and it provided better results. We used TShark[2] to collect packets during the client-server communication because QUIC is a UDP-based protocol and `tcpdump`[3] only works for TCP protocols. To transform collected packets into flows and extract relevant features, we tested two tools to compare the relevance of the features returned. They are presented in the following paragraphs.

[2]Wireshark cli tool

[3]`https://www.tcpdump.org`

Tshark It is Wireshark command line interface. TShark is thus a network protocol analyzer. It is able to capture packets data from a live network. It can also help read packets from files offline and bring out a summary of a conversation between a source and a server in terms of flow described by features in the Table 3.1. TShark provides 8 features after packets aggregation into flow, but some of them are irrelevant for our study. The exchange start time is irrelevant to classify flows. The total numbers of packets and bytes exchanged can be computed from the values found from the client and the server. This reduces the number of relevant features to five (05).

Features	
Client	**Server**
1. Number of packets from client	3. Number of packets from client
2. Number of bytes from client	4. Number of bytes from client
5. Total number of packets exchanged	
6. Total number of bytes exchanged	
7. Exchange start time	
8. Exchange duration	

Table 3.1: Flow based features provided by TShark

Tstat TCP Statistic Analysis Tool. Tstat is a Linux system tool which provides flow-based features from a traffic, both on the network and transport level. It can capture packets exchanges between hosts from a live network or use an existing *pcap* file. Contrary to what its name can inspire, Tstat can extract features from TCP and UDP packets. However, it provides much more features for TCP than UDP. Tstat provides more than a hundred features for TCP. The different features extracted by Tstat for UDP protocol are presented in Table 3.2

Based on both Table 3.1 and Table 3.2, it is clear Tstat provides more features than TShark even if some Tstat features are also irrelevant. Moreover, all TShark features are included in Tstat features. For this reason, we then used Tstat to build our dataset.

3.3.2 Features extraction and cleaning

.

As explained previously, to build the dataset, we initiate communications between the client and the server and capture the packets into a pcap file. As we do not have concrete applications using PQUIC, only connection communication

Features	
Client	**Server**
1. Client IP addresses	10. Server IP addresses
2. Client UDP port	11. Server UDP port
3. Client first packet absolute time	12. Server first packet absolute time
4. Client packets completion time	13. Server packets completion time
5. Number of bytes transmitted from client	14. Number of bytes transmitted from server
6. Number of all packets from client	15. Number of all packets from server
7. Check if client IP is internal	16. Check if server IP is internal
8. Check if client IP is anonymized	17. Check if server IP is anonymized
9. Protocol used over UDP from client	18. Protocol used over UDP from server
19. Fully Qualified DDNH	

Table 3.2: Flow based features provided by Tstat

exchanged by `picoquicdemo` client-server are used. To isolate the relevant communication among all that happening on hosts, filters are used to constrain TShark to only consider UDP traffic on port 4443 (QUIC port) involving IP addresses of client and server. The *pcap* file is then passed to Tstat to extract the features. Tstat produces different log files depending on the transport protocol, containing a summary of the communications aggregated by flows. The one related to UDP is used. As a flow is aggregated by IP/port of source and destination, a communication during a client and a server will result in one flow which will represent one instance in the dataset. We repeat this operation several times to build the dataset. UDP log file produced by Tstat contains nineteen (19) measures shown in the Table 3.2. The features 1 to 18 are two groups of 9 features. The first group is about the message sent from client to the server (C2S) and the second one is related to the messages sent from the server to the client (S2C).

Among features produced by Tstat, some of them have empty values while others are useless for our study. For instance, the IP address and the port number of client/server are not useful since port numbers are most of the time dynamic and PQUIC plugins are not associated to port number, so this cannot help to identify them. After removing useless features, we added some new ones.

Average and ratio values We computed some additional values that we added to the features we kept. The average client bytes per packet is computed by dividing the number of bytes sent by client by the number of packets sent by client. This value is also computed for the server side. We computed and added the ratio between sent and received bytes. The ratio is also computed for the number of packets.

Features	
Client	**Server**
1. Client packets completion time	6. Server packets completion time
2. Average client bytes	7. Average server bytes
3. Number of bytes transmitted from client	9. Number of bytes transmitted from server
4. Variance of client packets size	9. Variance of server packets size
5. Number of all packets from client	10. Number of all packets from server
11. Variance of all sent packets 12. Ratio between number of sent and received packets 13. Ratio between sent and received bytes	

Table 3.3: Features retained for network dataset

Variance computation The variance of packets size are not provided by Tstat and can not be computed by any Tstat measures. This is the reason why we do not use Tstat directly with live. The fact to save packets firstly into *pcap* file allow to compute the variance in a dedicated process. We wrote a python script using Scapy library. Scapy allows to read, send, capture and decode packets. It allows us to write a code able to read *pcap* file, get the number of bytes of different packets and compute their variance. By doing so, we computed and added to the dataset the variance packet's size sent by the client firstly and the one sent by the server secondly. Finally, we computed the variance based on client and server packets. The features that we gathered at the end of the process are depicted in Table 3.3

The Table 3.4, shows a summary of the statistics computed from the data. It contains the average of the size of packets and the average of the number of packets sent by the client/server per flow.

Plugins	**Avg. client data amount per flow (bytes)**	**Avg. client packets number per flow**	**Avg. server data amount per flow (bytes)**	**Avg. server packets number per flow**
FEC	6039.66	47.12	92053.11	619.69
Monitoring	5096.87	19.35	58900.58	46.84
Multipath	6117.22	26.2	60072.37	50.51
No plugin	3507.4	18.29	60829.28	47.8

Table 3.4: Brief summary of network statistics based dataset

3.4 Logs based dataset

In the previous section, we presented how the dataset based on network features is built. In this section, we present how we built another dataset for a text classification task.

In terms of data traffic classification, the payload is the crucial element that could contain informative to discriminative different types of traffic. The problem is that, basically, the payload of QUIC is encrypted. However, there is a possibility to get some information from the logs of QUIC outputted during the communication between the client and the server. Fortunately, `picoquicdemo` allows to output all that happens during the communication. It is possible to run `picoquidemo` command with the option `-l`. This option allows passing a file path in which all that happens during the communication is written.

As the log of QUIC can be obtained easily with `picoquicdemo` and the trace can be printed into a file, we consider in this section the fact to build a dataset containing for each flow the trace of packets exchanged. We notice also that for each communication, in addition to text printed by the option `-l` of `picoquicdemo`, there is some text printed into the console. We thus concatenate, for each flow, the text printed into the console to the text printing in the file passed as parameter to the command. As we didn't use the data collection host, for sake of simplicity, we collected these data directly from the client host. In order to stay aligned with the network statistics based collected data, the text was collected along with the network statistic features. So for each connection between the client and server, and for each plugin, we computed the values of the features based on the statistics of the network and the message exchanges between them. To perform text classification task, the texts cannot be used directly. They have to be processed to be tuned into numeric data.

3.4.1 Text pre-processing

The text pre-processing consists into tuned text into numeric data matrix. Different techniques exist to accomplish this task. Among then, we have *bag of words* and *word embedding*. In this document, only *bag of words* technique will be presented. Indeed, we tested *word embedding* technique through `word2vec`, but it leads very poor models. This can be explained by the fact that *word embedding* requires a lot of text for training process. We used existing models based on existing pre-trained embeddings like *wikipedia* and *twitter*, but our task is in a specific domain and those pre-trained embeddings does not improve the results. The following paragraphs show how we used *bag of words* technique to convert collected texts into numeric data needed for machine learning algorithms.

Tokenization All text pre-processing steps start by tokenization. The principle is to split each text into different words which will represent features for the dataset. Tokenization step often goes along with stopwords removal. The idea is to remove the words very frequent in a language in the aim to respect grammatical rules

but are not important to understand a sentence. There are language dependents and are punctuation, article, preposition, etc. After this process, it is important to identify words that have the same meaning. It is noticed that in most of the language, there are multiple words related to the same topic but with different spellings depending on their type (verb, adverb, adjective, preterit, etc.). However, they share a same root. For this reason, it is important to turn each word into its root in order to identify words having the same meaning and avoid duplicated words. Two techniques are often used to accomplish this task: *lemmatization* and *stemming*. They work differently and depending on the dataset, one or the other can be interesting.

Lemmatization Lemmatization algorithms analyze words based on dictionary and part-of-speech tagging in order to transform a word to its root while keeping that the word outputted is valid. The words *"better"*, *"best"* and *"good"* will all result for instance into the word *"good"*. On the other hand, the conjugated word *"stripes"* will result into *"stripe"* while the noun *"stripes"* will result into *"strip"*.

Stemming It returns the stem of a word, which does not need to be identical to the morphological root of the word. In text classification tasks, it is usually sufficient that related words map to the same stem, even if the stem is not in itself a valid word. Unlike lemmatization, the stemming of conjugated verb of noun *"stripes"* will result into the same word *"strip"*. On the other hand, the words *"better"*, *"best"* and *"good"* will stay unchanged after stemming.

In our work, the stemming technique is preferred as it is one of the one which produces better result. This is surely due to the fact that the texts outputted by `picoquidemo` are not intended to have a grammatical interest, and also because they contain very few valid words. Instead, the texts contain mainly specific IDs and hexadecimal addresses, as it can be noticed in Figure 3.2.

To perform tokenization and stemming, several tools propose different implementation. In our study, we used the *NLTK* library with its modules *RegexpTokenizer* for tokenizing, *PorterStemmer* for stemming and *WordNetLemmatizer* for lemmatization.

After tokenizing and stemming the collected text, they have to be tuned into features matrix. For this, we used the bag of words concepts. The idea is to compute a matrix based on the whole words in the different texts. Each word will represent a feature, while each row will be a flow. There is also some technique named *n-grams* which, in addition to each word, will add a feature column for each n-tuple of consecutive words. We do not use this technique in our work because the number of words we got after tokenization and stemming is huge and *n-grams* technique will be computationally challenging for the machine learning algorithms.

```
4add9e8717a6dbf: Sending packet type: 2 (initial), S0, Version ff00001d,
4add9e8717a6dbf:     <04add9e8717a6dbf>, <93f50a32bce82325>, Seq: 0, pl: 1186
4add9e8717a6dbf:     Prepared 1186 bytes
4add9e8717a6dbf:     Crypto HS frame, offset 0, length 235: 010000e703032ee0...
4add9e8717a6dbf:     padding, 947 bytes

4add9e8717a6dbf: Sending 1232 bytes to 0:0:0:0:0:0:0:1 at T=0.026237 (5d2e063266919)
Select returns 1232, from length 28, after 16603 (delta_t was 999830)
4add9e8717a6dbf: Receiving 1232 bytes from 0:0:0:0:0:0:0:1 at T=0.042865 (5d2e06326aa0d)
4add9e8717a6dbf: Receiving packet type: 2 (initial), S0, Version ff00001d,
4add9e8717a6dbf:     <93f50a32bce82325>, <0ac22f1b453192f8>, Seq: 0, pl: 133
4add9e8717a6dbf:     Decrypted 133 bytes
4add9e8717a6dbf:     ACK (nb=0), 0
4add9e8717a6dbf:     Crypto HS frame, offset 0, length 123: 0200007703036864...

4add9e8717a6dbf: Receiving packet type: 4 (handshake), S0, Version ff00001d,
4add9e8717a6dbf:     <93f50a32bce82325>, <0ac22f1b453192f8>, Seq: 0, pl: 1008
4add9e8717a6dbf:     Decrypted 1008 bytes
4add9e8717a6dbf:     Crypto HS frame, offset 0, length 1004: 08000060005e0010...
```

Figure 3.2: Example of text outputted by `picoquicdemo`

In fact, we get 3990030 words after tokenization and stemming.

In order to compute the values to fill in each cell of the features' matrix, many techniques are possible. The simplest one, the frequency. Each cell is filled-in by count the occurrence of the corresponding word in the related flow. Unfortunately, this technique has the disadvantage to highlight some too many frequent words which in final does not allow to discriminative the data. For this reason, we use another technique named Term Frequency-Inverse Document Frequency (TF-IDF). TD-IDF allows to set for each word value, its occurrence in each flow while reducing the value of words that are too frequent in the whole dataset to be informative. Given a flow f_i in a dataset F, the TD-IDF of a word w is expressed as:

$$tf\_idf(w, f_i) = tf(w, f_i) \cdot \log \frac{\mid F \mid}{\mid \{f_j : w \in f_j\} \mid} \tag{3.1}$$

where $tf(w, f_i)$ represents the occurrence of the word w in the flow f_i. The second term is the logarithm of the proportion of flows containing the word w among the whole flows. It is the term which allows decreasing the value of too frequent words like words *"sending"* and *"receiving"* which is present several times in the texts as it can be seen on Figure 3.2.

3.5 Ground Truth

Once we got all features for both datasets, they will constitute the columns of the dataset. It is then important to associate a label to each flow in the datasets. This process is a mandatory step before applying a machine learning classification algorithm. The labelisation in our context, consist in setting a specific value to each

flow, based on the plugin activated during the communication.When we initiate each communication (a single connection in our case), we inject a unique plugin, and we attach a class to the collected data.

In this work, we have 4 classes corresponding to the four plugins we used in our study:

- class 0: Traffic with plugin *Forwarding Erasure Correction* (FEC) activated
- class 1: Traffic with plugin *Monitoring* activated
- class 2: Traffic with plugin *Multipath* activated
- class 3: Traffic without any plugin

We built balanced datasets with same number of instance of each plugin because some machine learning algorithms like decision trees are sensible to imbalanced data. The code to build the dataset is publicly available at `https://github.com/JosetteAoga/PQUIC_Classification`.

Chapter 4

Experiments

This chapter depicts the experiments conducted on the collected data. Indeed, we applied several machine learning algorithms on our datasets in order to associate or not a plugin to data flows. Then, the results of algorithms are assessed. It is important to remind that we do not perform multilabel classification. Each flow is associated to exactly one plugin. The chapter is split into two (02) parts. In the first stage, we present results on network statistics based datasets. On the other hand, we will present results of different algorithms on the text based dataset.

4.1 Part I: Network statistics based dataset

In this section, we use the dataset collected using network statistics provided by Tstat.

4.1.1 Interpretable model

In order to understand the different elements which most accurately describe the presence of a plugin in a data flow, we learned, first, an interpretable machine learning model on the network statistics based dataset. The interpretable machine learning model, that has been used here, is the decision tree. We learned a decision tree model on the dataset while keeping the tree shallow, in order to make it interpretable. We considered a decision tree of maximum depth equal to 3. Different implementations of decision tree algorithm exist. In our work, we used CART implementation provided by the library Scikit-learn [1]. This implementation uses *index of gini* as criterion to select locally the best feature to split on. We performed a 5-folds cross validation and one of the tree learnt is shown in Figure 4.1. Its accuracies are presented in Table 4.1 and the confusion matrix is in Table 4.2.

[1] `https://scikit-learn.org/`

Note that the results are similar on all folds. At each node of the tree, there are different values showing specific things. The first element represents the feature with its threshold, on which the tree is split. The second element shows the value of the *index of gini* at the current node. The third one represents the total number of instances which fall in the node. The fourth, is the number of instances per class and the last element on the node is the majority class value based on the number of instances. On the leafs, there is no feature to split on; the class value at this stage represents the prediction of the leaf.

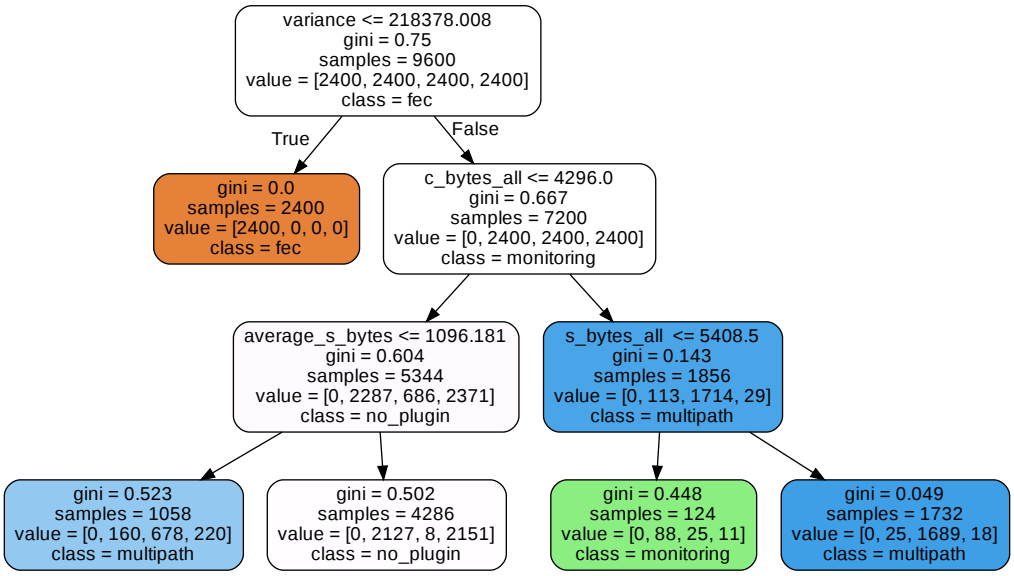

Figure 4.1: Tree with max depth 3

Algorithm	**Train score**	**Test score**
Decision Tree (max depth = 3)	72.97%	72.96%

Table 4.1: Tree max depth 3 Results

It can be noticed that *Forward Erasure Correction(FEC)* plugin is correctly identified by the tree. Only the *variance* is needed to correctly classify *FEC* flows. The tree shows that the variance of packets size exchanged in presence of *FEC* plugins is lower than others, while Table 3.4 depicts that the number of packets as well as the amount of data exchanged for *FEC* plugins are the highest. To confirm the prediction of the tree, we analyze the variance of packets size per flow

	Fec	Monitoring	Multipath	No Plugin
Fec	**100%**	0%	0%	0%
Monitoring	0%	**3.6%**	7.6%	88.8%
Multipath	0%	1.3%	**98.4%**	0.33%
No Plugin	0%	0.4%	9.73%	**89.86%**

Table 4.2: Confusion matrix at max depth 3

in case of each plugin and Figure 4.2 shows clearly that in addition to the fact that *FEC* plugin leads to a high number of packets exchanged per flow, the dispersion between packets size is lower than in the case of other plugins. In fact, the variance of packets size exchanged in case of *FEC* flows is very different from the others. This confirms why this feature is sufficient to correctly classify *FEC* plugins.

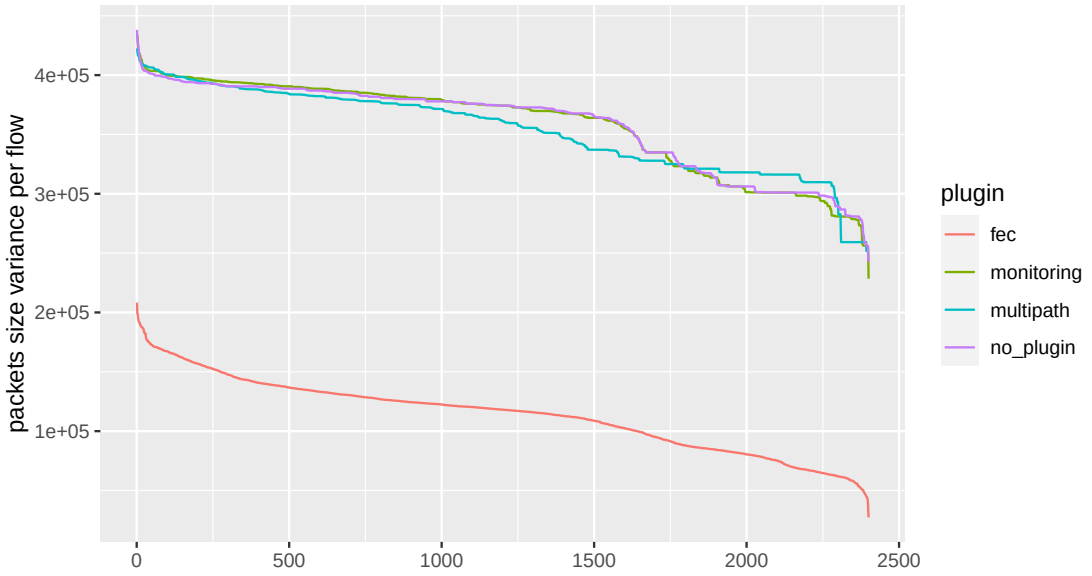

Figure 4.2: Packets size variance per flow grouped by plugin

Almost all data flow with *Multipath* plugin are correctly classified based on the amount of data sent by the server and the client. However, it is important to notice that the classifier struggles to separate *Monitoring* flows from flows without plugins. There is almost the same distribution of these two classes in the different leafs. When looking at the confusion matrix in Table 4.2, it can be seen that almost 90% of the *Monitoring* flows have been classified as without plugin. Moreover, the variance distribution shown on Figure 4.2 for these two classes are almost merged. Our intuition is that *Monitoring* plugin does not really impact the packets in the

traffic. As, it only computes statistics about QUIC protocol, we suppose that this does not really influence the nature of original packets. To assess this intuition, we reduced the dimensionality of data to 2D thanks to TSNE and plot the result in Figure 4.3. In fact, TSNE(t-distributed stochastic neighbor embedding) is a dimensionality reduction algorithm that can reduce the number of features of a dataset while preserving the distance between instances.

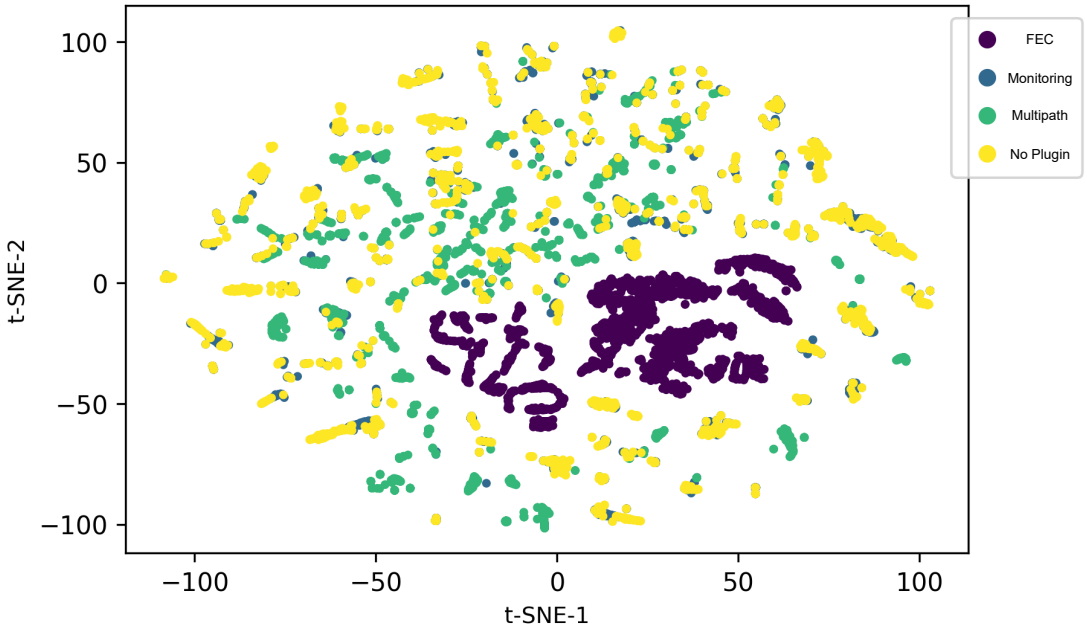

Figure 4.3: Dataset representation in 2D with TSNE

The representation of the dataset obtained by using TSNE contains the same number of data points for each plugin, especially 2400. However, when a look is taken at the Figure 4.3, three (03) colors are apparent at first glance. These are data points of plugins *forward erasure correction, multipath* and the data points without plugin. It is difficult to notice the *monitoring* plugin as it is almost merged with the data points without plugin depicted as *no plugin* on the figure. This explains why our interpretable decision tree model struggles to classify these two classes. It is also important to notice on the figure that one class stands out from the rest. It is the plugin *forward erasure correction* that the interpretable decision tree managed easily to classify. Our intuition to explain this representation is the following. In fact, as shown in Figure 3.4, the number of packets exchanged between the client and the server in case of *forward erasure correction* plugin is particularly high compared to the others. It is to be believed that this has resulted in a packet's size distribution different from the others, as we have seen for the

variance.

The interpretable decision tree provides some useful information about the classification of the different PQUIC plugins. However, the performance of the model is quite low in context of real-world applications. For this reason, we tried different machine learning algorithms in order to increase the generalization on data at the expense of the interpretability. In the following, we present the results when we tried learning some complex machine learning models on data.

4.1.2 Complex models

To improve the classification of PQUIC plugins over than the result of interpretable decision tree, we tried different algorithms producing complex machine models. These are deep decision trees, random forest, adaptive boosting (Adaboost), gradient boosting.

The performance of the learned models depends strongly on the training data as well as the test data. To eliminate the impact of the selected training and test data, we used a K-fold cross validation technique. With this technique, the whole dataset is split into K equal size parts. Successively, K-1 consecutive parts are used as training data while the last one is used as test data. This allows to use each part of the data as training and test data. Then the average of different results is computed to evaluate the model. In our work we used a 5-fold so, the dataset is split in 5 parts. Note that we created stratified folds. This implies that the class distribution of the whole dataset is preserved in each fold. The different training data used are perfectly balanced, as we have the same number of flows per plugin in the dataset. This is preferable for some algorithms.

The different algorithms we used have several hyperparameters which impact their results, and it is not always easy to find the best value for these hyperparameters. We thus used a hyperparameter tuning technique to find the most suitable value for the hyperparameters during the cross validation process. The goal of our study being to find the best model to predict plugins enabled during a network communication, we did not look for a suitable fold-based hyperparameters. Instead, we tried to find the best parameter for any fold. This will be considered as able to produce a good result on future data, as it does not depend on a specific fold. To accomplish this task, we use a grid search as hyperparameter tuning technique. The idea is to provide different potential values for the hyperparameters. Then the different combinations of hyperparameters values are used during the cross validation process. The one that produces the best average result is then selected as the best hyperparameters combination, and its result is the one we present in this document. All the experiments conducted in our study are based on implementation of algorithms provided by the Scikit-learn v1.0.1 library in python code (version 3.8.8). Experiments were performed on a server with an Intel Xeon

Platinum 8160 CPU, 320 GB of memory, running Linux Rocky 8.4.

In the following paragraphs, we will present the different algorithms we used and their results.

Decision Tree classifier As for the interpretable decision tree previously presented, we used CART implementation of decision trees. Two hyperparameters have been tuned: (1) the maximum depth and (2) the minimum number of samples per leaf. We tested values from 5 to 15 for the maximum depth, while the minimum number of samples per leaf we tested are 1, 5 and 10. The best hyperparameters selected by our grid search are a maximum depth equal to 15 and a minimum number of samples per leaf equal to 1. We computed the average of training accuracies and test accuracies outputted by each fold and use them as the training and the test accuracy of the model.

Algorithm	Hyperparameters	Training Accuracy	Test Accuracy
Decision Tree	max depth: 15 min samples per leaf: 1	92.93%	86.32%

Table 4.3: Scores of Decision Tree

As it can be seen in Table 4.3, decision tree provides 92.93% of training accuracy and 86.32% for test accuracy. The grid search selects the highest value of the depth. This shows how complex are the data. The confusion matrix in Table 4.4 shows in details how the classifications have been performed. Remind that the experiments have been performed in a context of 5-fold cross validation. Thus, $\frac{1}{5}$ of data is used as test for each fold. This corresponds to 2400 flows per fold. As the fold are stratified, this leads to 600 flows per plugin in each fold. In Table 4.4, we show the average predictions in each cell of the matrix to provide insight of what happened in each fold.

	Fec	Monitoring	Multipath	No Plugin
Fec	**99.97%**	0.03%	0%	0%
Monitoring	0.03%	**71.4%**	0.6%	28%
Multipath	0%	0.57%	**99.03%**	0.4%
No Plugin	0.03%	25.17%	0.23%	**74.57%**

Table 4.4: Confusion matrix of Decision Tree

In the confusion matrix, it can be seen that almost all the 600 flows of *forward erasure correction* plugin are correctly classified. In second position, more than 99% of the instances of the *multipath* plugin are correctly classified. About *monitoring*

plugin, 71.4% of instances are correctly classified over the 600 instances. Among the 28.6% of incorrectly classified, 28% are considered to have no plugin. For the flows without any plugin, 74.57% of instances over the 600 are correctly classified and almost the remaining are classified to be *monitoring* plugin. These results have the same trending what we have already seen with the interpretable tree. *Forward erasure correction* and *multipath* plugins are almost correctly classified. In the other hand, *monitoring* plugin and flows without plugin are confused. This can be seen in red in Table 4.4.

Random Forest classifier We also tried multiple values of hyperparameters when running the random forest algorithm. The hyperparameters tuned are: (1) the number of trees to learn in the forest are tuned between values 50, 100 and 250; (2) the maximum depth of decision trees are tested in the range from 1 to 4; (3) the minimum number of samples per leaf are tuned between the values 1, 5 and 10. The best combination of hyperparameters selected is a maximum depth equal to 4, a minimum number of samples per leaf equal to 1 and 250 trees. This is presented in Table 4.5.

Algorithm	**Parameters**	**Training Accuracy**	**Test Accuracy**
Random forest	max depth: 4 min samples per leaf: 1 trees number: 250	79.03%	78.83%

Table 4.5: Score of Random Forest

These hyperparameters produced a model with a training accuracy of 79% and 78.8% for the test accuracy. This result is detailed by the confusion matrix in Table 4.6.

	Fec	**Monitoring**	**Multipath**	**No Plugin**
Fec	**99.97%**	0%	0.03%	0%
Monitoring	0%	**28.37%**	1.93%	69.7%
Multipath	0%	0.03%	**99.93%**	0.03%
No Plugin	0%	11.9%	1.6%	**86.5%**

Table 4.6: Confusion matrix of Random Forest

The *forward erasure correction* and *multipath* flows are correctly classified. The flows without plugin are classified with 86.5% of accuracy. The *monitoring* plugin in contrary is classified with high rate of error. In fact, 69.7% of its instances have been classified as without plugin and only 28.37% are correctly classified. In

summary, apart from *monitoring* plugin classified with very high error, the other plugins instances present a quite high accuracy rate.

Adaptive Boosting classifier(Adaboost) Two hyperparameters have been tuned for Adaboost: (1) the number of trees to learn in the ensemble are tuned between values 50, 100 and 250; (2) the base estimator used is tuned. In our study, we used a CART decision tree with different maximum depth ranging from 1 to 4.

Algorithm	Parameters	Training Accuracy	Test Accuracy
Adaptive boosting	learner: CART depth 4 trees number: 100	78.08%	77.48%

Table 4.7: Score of AdaBoost

The hyperparameter selected by the grid search are 100 decision trees of maximum depth equal to 4. The training score of adaboost is 78%, while the test score is 77% (Table 4.7). This result is detailed in the confusion matrix (Table 4.8) where it can be seen that as for previous algorithms, the *forward erasure correction* and *multipath* instances are correctly predicted.

	Fec	Monitoring	Multipath	No Plugin
Fec	**99.9%**	0.03%	%0	0%
Monitoring	0.03%	**35.9%**	0.2%	63.8%
Multipath	0.03%	0.4%	**99.5%**	0%
No Plugin	0.06%	30.9%	0.13%	**68.8%**

Table 4.8: Confusion matrix of Adaboost

Almost 70% of flows without plugin are correctly classified, while the remaining is classified as *monitoring*. *Monitoring* plugins are the ones most incorrectly classified. Only 35.9% of these plugins are correctly classified. The remaining majority is predicted as having no plugin.

Gradient Boosting classifier Two parameters have been tuned for gradient boosting classifier: (1) the number of trees with the values 50, 100, and 250; (2) the maximum depth of the trees we tested are from 1 to 4.

For the classification, an accuracy of 92% for the training and 88% for the test were registered with 250 decision trees and a maximum depth of 3 selected by the grid search(Table 4.9). These percentages are quite interesting.

Table 4.10 shows the details of the classification. As for previous methods, *forward erasure correction* and *multipath* plugins are correctly classified. The

Algorithm	Parameters	Training Accuracy	Test Accuracy
Gradient Boosting	max depth: 3 trees number: 250	92.86%	88.66%

Table 4.9: Score of Gradient Boosting

	Fec	Monitoring	Multipath	No Plugin
Fec	**99.9%**	0.03%	0%	0%
Monitoring	0.03%	**78.6%**	0%	20.8%
Multipath	0%	0.2%	**99.6%**	0.1%
No Plugin	0.03%	23.7%	0.3%	**75.9%**

Table 4.10: Confusion matrix of Gradient Boosting

monitoring flows and flows without plugin still present non-negligible errors rate since they are still confusing as shown in the matrix.

4.1.3 Results and observation

We presented in the previous section, the different algorithms that we used in this work, and we have presented their performance individually on the problem. Here we will go through them again to analyze and compare them.

As the data we are working on to perform the classification are a bit complex to be correctly classified by a decision tree classifier of maximum depth equal to 3 and this was confirmed by TSNE visualization, we tried more complex algorithms and packed the comparative results into Table 4.11. Table 4.12 shows the classification rate of each algorithm per class.

Algorithms	Parameters	Training Accuracy	Test Accuracy
Decision Tree	max depth: 15 min samples per leaf: 1	**92.93%**	86.32%
Random forest	max depth: 4 min samples per leaf: 1 trees number: 250	79.03%	78.83%
Adaptive boosting	learner: CART depth 4 trees number: 100	78.08%	77.48%
Gradient Boosting	max depth: 3 trees number: 250	92.86%	**88.66%**

Table 4.11: Training and test scores per algorithm

We first tried Decision Tree again with a maximum depth equal to 15, where the test accuracy increases up to 86% and reach the higher accuracy we got on

Algorithms	Fec	Monitoring	Multipath	No Plugin
Decision Tree	99.74%	71.4%	99.03%	74.57%
Random forest	**99.97%**	28.37%	**99.93%**	**86.5%**
Adaptive boosting	99.9%	35.9%	99.5%	68.8%
Gradient Boosting	99.9%	**78.6%**	99.6%	75.9%

Table 4.12: Classification rate of algorithms per class

training data. After this, we tried different ensemble of decision trees methods in the expectation to increase the result of decision trees. Random forest provides 78% for the test accuracy. It is more interesting than our interpretable decision tree result. However, it makes more errors than the Decision tree with maximum depth equal to 15. Note in Table 4.12 that random forest is the algorithm that better classifies all plugins, except from *monitoring*. Its drop of performance is mainly due to its inability to correctly classify *monitoring* flows that most of the time, it classifies as flows without plugin.

The test accuracy of Adaboost is almost the same as the one of random forest. Adaboost reaches an accuracy of 77.48% which is the lowest we got in our experiments. In terms of the two classes, difficult to separate, Adaboost is the one that got the bad performances. This makes it have the lowest accuracy we got, despite its good ability to classify *forward erasure correction* and *multipath* flows. The last algorithm we tested is gradient boosting.It is the algorithm that got the best performance in our experiments. In fact, it is the one that obtained the lowest misclassification error for *monitoring* flows and flows without plugin. It has almost the same classification rate of *forward erasure correction* and *multipath* plugins as the other algorithms. In fact, these two classes are correctly classified by all algorithms.

4.2 Part II: Text based dataset

In this part, we use the dataset created in section 3.4 of chapter 3 to identify once again the different plugins of PQUIC. Indeed, this dataset contains words created from the log of QUIC messages exchanged between client and server. In the following, we present the different algorithms that have been used and their results.

4.2.1 Algorithms Used

Many classification algorithms exist to perform text classification. Among them, two are often used in the literature; Naive Bayes and Support Vector Machines

(SVM). We thus decided to apply these algorithms to our dataset. In addition to them, we also applied a Decision Tree algorithm because it can provide an interpretable model. In the next few lines, we will explain a bit what Naive Bayes and SVM are because we have not yet introduced these terms in our document.

Naive bayes Naive Bayes classifiers are a set of simple probabilistic classifiers used to solve many classification and regression problems. It is most of the time used as the basis for text classification, as it is quick and easy to implement. Naive Bayes classifiers share a common assumption, stating that all features in the dataset being classified are independent of each other [42]. Naive Bayes classifiers estimate the probability of individual features values to be associated to a certain class and use these probabilities to compute the probability of an instance belonging to different classes thanks to Bayes' theorem[2]. Then the class with the higher probability is predicted. To compute the probability of a feature value to be associated to a class, different assumptions are made regarding the features' distribution. When working with discrete features, multinomial or Bernoulli distribution are the most used and with continuous features, the Gaussian distribution could be the best candidate[43].

Support Vector Machine SVM is a supervised learning algorithm used for classification most of the time, but it is also used in regression problems. In the algorithm, each data item is plotted like a point in n-dimensional space. The principle of the algorithm is to find one or several hyperplanes depending on the problem to separate data while maximizing the distance (margin) between the hyperplane(s) and the different data points. SVM algorithm works generally well in practice. It can be used to solve linear as well as non-linear data. In case of non-linear data, it uses some kernel to project data points into another dimension to facilitate the separation. Different kernels are used depending on the data. The support vectors are the data closest to the border. We called the best decision boundary hyperplane[44].

4.2.2 Results and observation

In this section, we will describe the results and remarks that we got after we have performed the classification.

The same experiments settings used for network statistic based features are also used here. A 5-fold grid search cross validation has thus been performed and the average result of the best hyperparameters are presented. However, the hyperparameters used are not the same. We reduced at most the number of

[2] `https://en.wikipedia.org/wiki/Bayes%27_theorem`

hyperparameters to reduce the time and the memory consumption of the algorithms. In fact, the dataset has more than 3 million features and this is huge, and it could be even more costly to tune different hyperparameters values. For the decision tree, the only parameter we tuned is the maximum depth that we tested for values between 3 and 7. For Naive Bayes, we used the multinomial naive bayes implementation provided by Scikit-learn. In fact, using a Gaussian distribution to compute the probability could be interested in our case because the values provided by the TF-IDF are continuous values. However, the naive Gaussian bayes implementation in Scikit-learn does not support sparse feature matrices. In fact, the feature matrix produced by TF-IDF is stored in a sparse matrix data structure, because the number of zeros in this type of matrix is huge and sparse matrices reduce the amount needed to store these matrices. We tried unsuccessfully to convert the features' matrix into a dense matrix, but it throws a memory error. For this, we used a multinomial naive bayes. We did not tune any hyperparameter for naive bayes. About SVM, we tuned some hyperparameters. The kernel is chosen between polynomial and Gaussian kernel. For the polynomial kernel, the degrees 1 and 2 are explored. The regularization parameter C is chosen between 1 and 10. The idea to provide a high regularization is to make the model prefer misclassification minimization than margin maximization. This could help in the classification of *monitoring* flows and flows without plugin. The remaining hyperparameters have been set to default values.

Algorithms	**Parameters**	**Training Accuracy**	**Test Accuracy**
Decision Tree	max depth: 3	**100%**	**99%**
Naive Bayes	**X**	**100%**	95%
SVM	C: 10 kernel: poly degree: 1	**100%**	**99%**

Table 4.13: Scores of different algorithms

Table 4.13 shows that all the algorithms have a good test accuracy. The error rate is very insignificant. In order to visualize the result and be able to interpret it better, we plot the trees found by the decision tree algorithm.

Considering the tree of two folds depicted in Figure 4.4 and Figure 4.5, we can see that the words like *multipath*, *fec* and *mpiraux* are used to classify the plugins. Looking inside the text, we noticed that those words are part of the prior messages outputted by the "picoquicdemo" command. In fact, these texts are not part of the streams exchanged between the server and the client. There are not inside the file outputted by the option `-l` of `picoquicdemo`. These words are in the text we collected from the console. The pictures 4.6, 4.7 and 4.8 show console screenshots

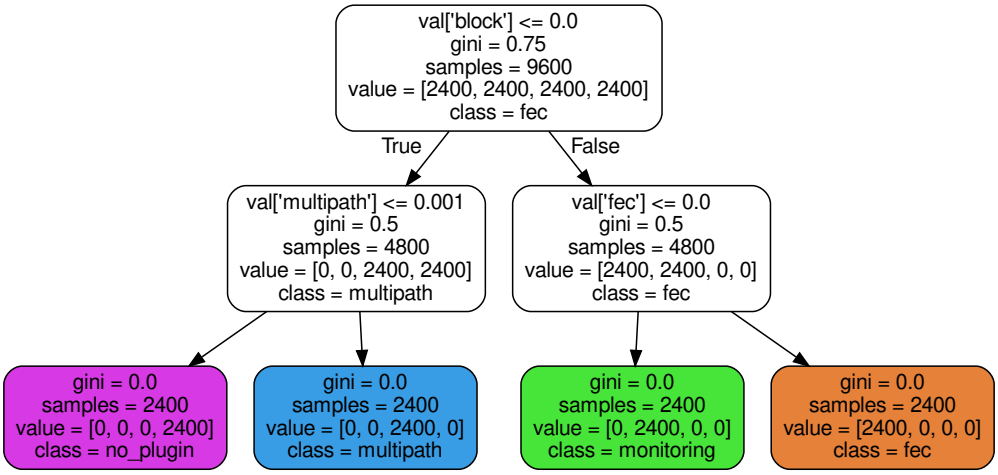

Figure 4.4: A decision tree for text based data

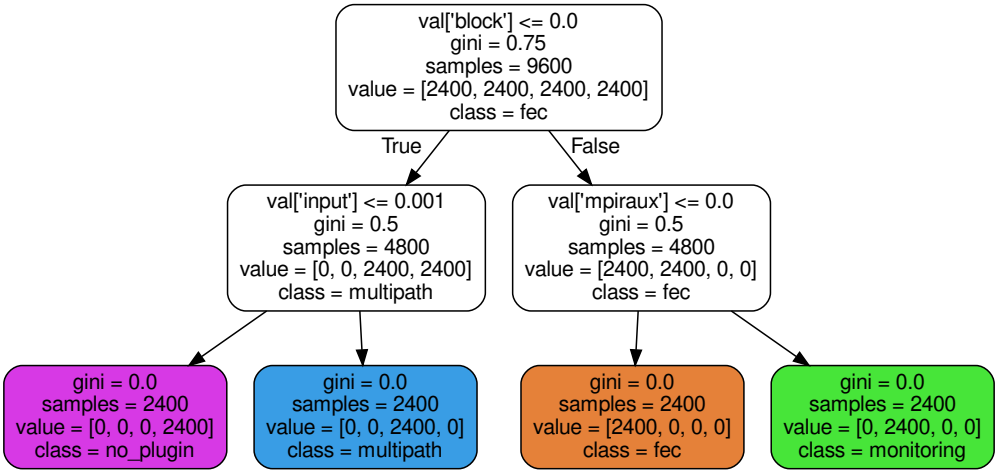

Figure 4.5: A decision tree for text based data

where those words are.

Therefore, we decided to remove the console text from our dataset. We cannot be sure that all the future plugins developed will add this kind of signature. Moreover, they are not part of the streams exchanged and in real-world applications, we cannot have access to these data.

```
include multipath.plugin...
create memory manager for plugin be.qdeconinck.multipath.rtt
create dynamic memory manager
Successfully inserted local plugin plugins/multipath/multipath_rtt.plugin
```

Figure 4.6: Console print of *multipath* plugin

```
create memory manager for plugin be.mpiraux.monitoring
create fixed block size memory manager
Successfully inserted local plugin plugins/monitoring/monitoring.plugin
```

Figure 4.7: Console print of *monitoring* plugin

```
include fec_core.plugin...
include fec_framework_window.plugin...
include fec_scheme_xor.plugin...
include fec_constant_redundancy_controller.plugin...
create memory manager for plugin be.michelfra.fecxor
create fixed block size memory manager
Successfully inserted local plugin plugins/fec/fec.plugin
```

Figure 4.8: Console print of *forward erasure correction* plugin

We thus remove the console text from the collected data and re-run the pre-processing steps detailed in section 3.4. The different algorithms were run on the new dataset. The results are then different and shown in Table 4.14. Naive Bayes shows a great performance compared to Decision Tree and SVM. It reached a test accuracy of 95.24%. It is important to note that the grid search for decision tree selected the highest depth we proposed. This gives the intuition that the tree still struggle to find few words in the network communication trace to accurately predict the different plugins. We do not show the trees learned due to the highest depth.

Algorithms	**Parameters**	**Training Accuracy**	**Test Accuracy**
Decision Tree	max depth: 7	88.9%	87.1%
Naive Bayes	**X**	**100%**	**95.24%**
SVM	C: 10 kernel: poly degree: 1	**100%**	86.16%

Table 4.14: Text classification algorithms' result without header

However, when looking at the confusion matrix in Table 4.15, it appears that the plugins *forward erasure correction* and *multipath* are correctly classified. This can be confirmed by the trees learned, which in the first nodes separate these

plugins from the others. Due to the size of the tree, we show in Figure 4.9 the first depths to show how these two classes are classified.

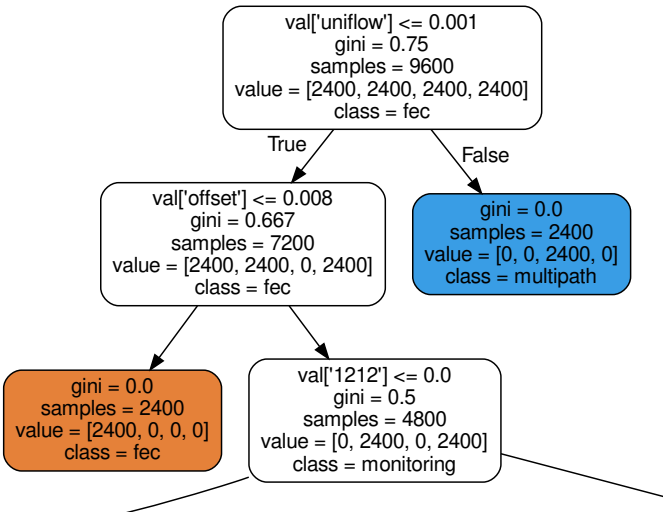

Figure 4.9: Two first depths of decision tree

It can be notice that the words *uniflow* is decisive to predict *multipath* plugins. We looked for this word in the texts collected and show in Figure 4.10 where it is found in *multipath* packets communication.

```
1f04b65537fbadc4: Sending packet type: 6 (1rtt protected phi0), S0,
1f04b65537fbadc4:     <d7d864a0401d5d2b>, Seq: 1 (1)
1f04b65537fbadc4:     Prepared 1065 bytes
1f04b65537fbadc4:     MP NEW CONNECTION ID for Uniflow 0x01 CID: 0xbd07b235346abda1, 66609fe7d1ef0ce818635dce7e4aa9e4
1f04b65537fbadc4:     ADD ADDRESS with ID 0x01 Address: 2a02:a03f:c0bf:f600:4d02:5746:a8ca:1c93
1f04b65537fbadc4:     MP ACK for path 0x00 (nb=0), 0
1f04b65537fbadc4:     padding, 1005 bytes
```

Figure 4.10: *Multipath* text portion containing the word *Uniflow*

The word *offset* appears frequently in the trace of communications. However, it appears that under a certain threshold, it indicates the presence of *forward erasure correction* plugin. *Monitoring* flows and flows without plugin are still difficult to separate, and the part of the tree used to classify them is not easy to interpret. It's based mainly on some substrings, surely found from QUIC connection ID.

Support Vector Machine algorithm has almost the same behavior as Decision Tree. The confusion matrix in Figure 4.16 can confirm that.

The trending is a bit different with Naive Bayes. Indeed, it provides the best performance of our study. The confusion matrix in Figure 4.17 shows that in addition to *forward erasure correction* and *multipath*, Naive bayes managed to correctly classify almost all *monitoring* flows. Moreover, it reached an interesting accuracy of 87.13% for flows without plugin.

	Fec	Monitoring	Multipath	No Plugin
Fec	**99.97%**	0%	0%	0.03%
Monitoring	0.03%	**66.2%**	0%	33.77%
Multipath	0%	0%	**100%**	0%
No Plugin	0.03%	17.57%	0%	**82.4%**

Table 4.15: Decision Tree: Matrix of Confusion

	Fec	Monitoring	Multipath	No Plugin
Fec	99.93%	0.07%	0%	0%
Monitoring	0%	59.03%	0%	40.97%
Multipath	0%	0%	100%	0%
No Plugin	0%	37.07%	0%	62.93%

Table 4.16: Support Vector Machine: Matrix of Confusion

	Fec	Monitoring	Multipath	No Plugin
Fec	99.9%	0.1%	0%	0%
Monitoring	1.97%	99.93%	0.03%	4.07%
Multipath	0%	0%	100%	0%
No Plugin	0.57%	12.3%	0%	87.13%

Table 4.17: Naive Bayes: Matrix of Confusion

4.3 Ideal context for PQUIC plugins identification

Based on the previous sections, it can be concluded that the flows are not too difficult to identify. However, *monitoring* plugins are confused with flows without plugin. As explained before, this is probably due to the fact that *monitoring* plugin just collect statistics about QUIC protocol. This could explain why it almost merges with data flows without plugin. Unfortunately, this causes a drop of performance of different classifier. It seems that an interpretable decision tree could correctly classify the plugin while giving a good understanding of plugins behavior if *monitoring* flows and flows without plugin were not similar. For this reason, we created a new dataset by removing the data flows without plugins and learned a decision tree with a maximum depth equal to 4. The result was clear. The classifier reaches more than 99% of accuracy on test data even though we performed a 5-fold cross validation.

Table 4.18 shows the overall training and test accuracies. Table 4.19 shows the

Algorithm	Train score	Test score
Decision Tree (max depth = 4)	99.44%	99.34%

Table 4.18: Tree max depth 4 for only plugin flows

average prediction percentage per plugin, and Figure 4.11 shows the tree outputted on one of the five folds. It can be noticed that the different plugins can be classified using almost only packets sizes. Only one feature in the tree is not related to packets sizes.

	Fec	Monitoring	Multipath
Fec	**99.97%**	0.03%	0%
Monitoring	0.03%	**98.7%**	1.27%
Multipath	0%	0.63%	**99.37%**

Table 4.19: Confusion matrix for dataset containing only plugin flows

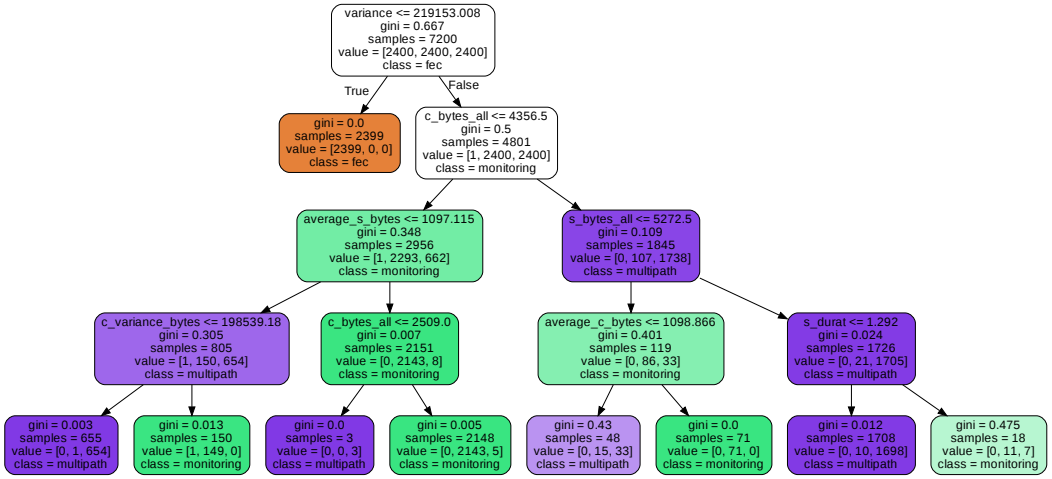

Figure 4.11: Interpretable decision tree for only plugin data flows

Chapter 5

Conclusion

In this thesis, we have described the process that we used to identify some PQUIC's plugins. Indeed, PQUIC is a framework that is built to enable the addition of new extensions(plugins) to QUIC protocol. Specifically, we tried to identify three plugins: the plugin of Forwarding Erasure Correction(FEC), the plugin of Monitoring and the plugin of Multipath.

First, we studied the different techniques that have been already used in network protocol identification. The first is the traditional port-based technique, which consist of identify protocol by their port number. This technique has been totally abandoned since the arrival of dynamic port protocols and has been replaced by the payload-based technique. The latter focused on protocol signatures localized on the packet content. This one has also been rejected because most of the network traffic used HTTPS, so the content of packets are encrypted. The other technique is the machine learning approach and till now gives good accuracies. It proves to be a solution to the two other techniques. This was the technique that we used in our experimentation process, since we are working on encrypted data.

To identify the plugins, we built two datasets: a dataset based on network statistics on QUIC traffic and a dataset based on the log of `picoquicdemo` command during the exchanged between the client and the server. After, a quick test and visualization on the dataset of network, we noticed that two classes were very closed and by the way the different classification algorithms that were used didn't succeed to really identify them. Gradient boosting algorithm is the one which present the best performance on this dataset. Not only it got the best test accuracy, but also it classifies with the lowest error rate each plugin. The other algorithms also succeeded to classify the plugins of *forward erasure correction* and *multipath* but present many errors for *monitoring* flows and no plugin flows. Specifically, random forest classify the majority of the *monitoring* flows like no plugin flows.

The experiment, based on the text dataset, shows that almost all algorithms tried in this part got accuracies over 85%, but they make also errors on the

classification of the *monitoring* flows and no plugin flows. Globally, for the whole experiment, all algorithms classified with high precision the plugins of *forward erasure correction* and *multipath* flows. Only the naive Bayes algorithm succeeded to classify the *monitoring* flows at 99%. The fact that the algorithms misclassified some plugins is not due to any complexity, but simply because the *monitoring* plugin didn't perform in our experiment specific thing to make it distinct from when there are no plugins. We have also shown this by performing the classification only on the three plugins, what gives good accuracies.

A future work will then be to work with more interesting data, where each plugin will perform more than just a connection step. It could also be interesting to consider a multilabel classification where instead of assigning to each flow exactly one plugin, many plugins can be assigned to one flow.

Bibliography

[1] Zigang Cao, Gang Xiong, Yong Zhao, Zhenzhen Li, and Li Guo. A survey on encrypted traffic classification. In *International Conference on Applications and Techniques in Information Security*, pages 73–81. Springer, 2014.

[2] Quentin De Coninck, François Michel, Maxime Piraux, Florentin Rochet, Thomas Given-Wilson, Axel Legay, Olivier Pereira, and Olivier Bonaventure. Pluginizing quic. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 59–74. 2019.

[3] Muhammad Shafiq, Xiangzhan Yu, Asif Ali Laghari, Lu Yao, Nabin Kumar Karn, and Foudil Abdessamia. Network traffic classification techniques and comparative analysis using machine learning algorithms. In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pages 2451–2455. IEEE, 2016.

[4] Talieh Seyed Tabatabaei, Mostafa Adel, Fakhri Karray, and Mohamed Kamel. Machine learning-based classification of encrypted internet traffic. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 578–592. Springer, 2012.

[5] Yohei Okada, Shingo Ata, Nobuyuki Nakamura, Yoshihiro Nakahira, and Ikuo Oka. Comparisons of machine learning algorithms for application identification of encrypted traffic. In *2011 10th International Conference on Machine Learning and Applications and Workshops*, volume 2, pages 358–361. IEEE, 2011.

[6] Jawad Manzoor, Idilio Drago, and Ramin Sadre. How http/2 is changing web traffic and how to detect it. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–9. IEEE, 2017.

[7] Internet Assigned Numbers Authority (IANA). `http://www.iana.org/assignments/port-numbers`, 2021. [Online; accessed 7-July-2021].

[8] Michael Finsterbusch, Chris Richter, Eduardo Rocha, Jean-Alexander Muller, and Klaus Hanssgen. A survey of payload-based traffic classification approaches. *IEEE Communications Surveys & Tutorials*, 16(2):1135–1156, 2013.

[9] Andrew W Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *International Workshop on Passive and Active Network Measurement*, pages 41–54. Springer, 2005.

[10] Andrew W Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–60, 2005.

[11] Valentın Carela Espanol and Pere Barlet Ros. Identification of network applications based on machine learning techniques.

[12] Pere Barlet-Ros, Valentín Carela-Español, Eva Codina, and Josep Solé-Pareta. Identification of network applications based on machine learning techniques. In *TNC 2008: Proc. of the Terena Networking Conference*, 2008.

[13] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and KC Claffy. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, 2004.

[14] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *Proceedings of the 13th international conference on World Wide Web*, pages 512–521, 2004.

[15] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 135–148, 2004.

[16] Fivos Constantinou and Panayiotis Mavrommatis. Identifying known and unknown peer-to-peer traffic. In *Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*, pages 93–102. IEEE, 2006.

[17] Mark Alllman and Vern Paxson. Issues and etiquette concerning use of shared measurement data. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 135–140, 2007.

[18] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials*, 10(4):56–76, 2008.

[19] Arthur Callado, Carlos Kamienski, Géza Szabó, Balázs Péter Gero, Judith Kelner, Stênio Fernandes, and Djamel Sadok. A survey on internet traffic identification. *IEEE communications surveys & tutorials*, 11(3):37–52, 2009.

[20] P Natesan, P Balasubramanie, and G Gowrison. Performance comparison of adaboost based weak classifiers in network intrusion detection. *Journal of Information Systems and Communication*, 3(1):295, 2012.

[21] Michel Lutz and Eric Biernat. *Data Science: fondamentaux et études de cas: Machine Learning avec Python et R.* Editions Eyrolles, 2015.

[22] Mahesh Pal. Random forest classifier for remote sensing classification. *International journal of remote sensing*, 26(1):217–222, 2005.

[23] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[24] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

[25] Tom Auld, Andrew W Moore, and Stephen F Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on neural networks*, 18(1):223–239, 2007.

[26] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

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

[28] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 183–196, 2017.

[29] Christian Huitema. picoquic. `https://github.com/private-octopus/picoquic`, 2018.

[30] Matt Fleming. A thorough introduction to ebpf. linux weekly news (december 2017), 2017.

[31] Piet De Vaere, Tobias Bühler, Mirja Kühlewind, and Brian Trammell. Three bits suffice: Explicit support for passive measurement of internet latency in quic and tcp. In *Proceedings of the Internet Measurement Conference 2018*, pages 22–28, 2018.

[32] Brian Trammell and Mirja Kuehlewind. The quic latency spin bit. *draft-ietf-quic-spin-exp-01 (work in progress)*, 2018.

[33] Alan Ford, Costin Raiciu, Mark Handley, Olivier Bonaventure, et al. Tcp extensions for multipath operation with multiple addresses. 2013.

[34] Jana Iyengar and Martin Thomson. Quic: A udp-based multiplexed and secure transport. *Internet Engineering Task Force, Internet-Draft draftietf-quic-transport-17*, 2018.

[35] Quentin De Coninck and Olivier Bonaventure. Multipath quic: Design and evaluation. In *Proceedings of the 13th international conference on emerging networking experiments and technologies*, pages 160–166, 2017.

[36] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? designing and implementing a deployable multipath {TCP}. In *9th {USENIX} symposium on networked systems design and implementation ({NSDI} 12)*, pages 399–412, 2012.

[37] Georg Carle and Ernst W Biersack. Survey of error recovery techniques for ip-based audio-visual multicast applications. *IEEE Network*, 11(6):24–36, 1997.

[38] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM SIGCOMM computer communication review*, 27(2):24–36, 1997.

[39] Jay Kumar Sundararajan, Devavrat Shah, Muriel Médard, Szymon Jakubczak, Michael Mitzenmacher, and Joao Barros. Network coding meets tcp: Theory and implementation. *Proceedings of the IEEE*, 99(3):490–512, 2011.

[40] R Hamilton, J Iyengar, I Swett, and A Wilk. Quic: A udp-based secure and reliable transport for http/2 draft-tsvwg-quic-protocol-02. *Network Working Group*, 2016.

[41] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. Network coding: an instant primer. *ACM SIGCOMM Computer Communication Review*, 36(1):63–68, 2006.

[42] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[43] Shuo Xu. Bayesian naïve bayes classifiers to text classification. *Journal of Information Science*, 44(1):48–59, 2018.

[44] David Meyer and FH Technikum Wien. Support vector machines. *The Interface to libsvm in package e1071*, 28, 2015.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**