

# POLITECNICO DI TORINO

Department of Electronics and Telecommunications

**Master's degree program in  
Communication and Computer  
Network Engineering**

Master Thesis

**Characterization and Data Analysis of Cloud  
Gaming Platforms**



**Supervisors:**

prof. MARTINO TREVISAN  
prof. PAOLO GARZA  
prof. MATTEO MAURIZIO MUNAFO  
Dena MARKUDOVA

**Candidate:**

Gabriel Grillo Caballero

**December 2021**



# Acknowledgments

First of all, I would like to thank all of the wonderful people who have helped me to get to this point.

Thanks to Dena Markudova and Gianluca Perna for all the help and guidance provided during these months and for treating me like a friend.

Thanks to my supervisor Martino Trevisan for the support and for giving me the opportunity of working on this project.

To everyone involved on the Project of Smartdata@Polito.

A mi increíble familia, que a pesar de la distancia me han brindado mas ayuda de la que podría pedir, sobre todo mis padres, que han sido una parte fundamental en el éxito que he tenido. A mi primo Enzo que me acogió aquí en Italia como un hermano, a mis tíos, al resto de mis primos, muchas gracias.

# Abstract

Gaming has evolved as one of the most used and profitable online business worldwide. Until recently, the players ran their games locally, on their PCs, laptops, phones or tablets; however, recently some companies like Google and NVIDIA have launched a new model called cloud or browser gaming. The idea is to run the game in the company servers and stream the video and audio flows to the user's device.

Little to no information has been published about how they are implementing this new service, so this thesis is aimed at doing a traffic characterization of cloud gaming, by analyzing the behavior of the most important networking metrics. Moreover, with the help of machine learning algorithms, we want to predict in which state is the game at any second, which can be useful in the future for providing classes of service to a QoE mechanism. The thesis has been organized as follows, initially, the focus was on data collection, the more data the better to perform a correct assessment of the characteristics of the service, furthermore, it is necessary for building the datasets used later on for feeding the machine learning algorithms. The next step was developing python script that allowed the extraction of the information from the Wireshark captures and the WebRTC logs, which were the tools used for obtaining the captures; this information was grouped and labeled to perform the characterization and based on the patterns observed and conclusions drawn, the features for the dataset were extracted. Two models were created, with two and three classes respectively, corresponding to different states identified inside the games. Finally, supervised classification machine learning algorithms were used to learn from the data and infer the previously mentioned states, experiments with different dataset were ran and a process of feature selection was performed on the biggest dataset to limit the noisy feature and try to achieve the best results possible.

# Contents

1. Introduction.....	1
1.1 Networking principles .....	1
1.2 Gaming Industry .....	2
1.2.1 Gaming sales post-COVID-19.....	3
1.2.2 Gaming during lockdown .....	4
1.2.3 Cloud gaming.....	5
1.3 Problem formulation.....	6
1.4 Literature Review .....	6
2. Background.....	8
2.1 RTP.....	8
2.2 DTLS .....	11
3. Methodology.....	13
3.1 Machine Learning.....	13
3.2 Supervised Learning .....	14
3.3 Unsupervised Machine Learning.....	16
3.4 Supervised Machine Learning Algorithms.....	17
3.4.1 Decision Tree.....	17
3.4.2 K-nearest neighbors .....	18
3.4.3 Random Forest.....	20
3.5 Feature Selection .....	20
3.6 Performance Measures .....	21
3.6.1 Accuracy .....	22

3.6.2 Precision.....	22
3.6.3 Recall .....	23
3.6.4 F1 score.....	23
4. Dataset .....	24
4.1 Data Collection.....	24
4.2 Data Characterization .....	27
4.2.1 General Information.....	27
4.2.2 Bitrate distribution analysis .....	31
4.2.3 Packet length distribution analysis .....	35
4.2.4 Video Frames per second distribution analysis .....	36
4.2.5 DTLS packets distribution analysis .....	40
4.2.6 Packets Inter-arrival distribution analysis.....	43
4.2.7 Packets per second received .....	45
4.2.8 Stadia vs GeForce comparison on Destiny 2 performance.....	46
5. Game Stage Classifications .....	49
5.1 Dataset construction .....	49
5.2 Three classes classification.....	52
5.2.1 Random Forest Results .....	53
5.2.2 Decision Tree .....	54
5.2.3 K-nearest neighbor.....	54
5.2.4 Results discussion .....	56
5.3 Two classes classification .....	56
5.3.1 Random Forest Results .....	56
5.3.2 Decision Tree Results .....	57
5.3.3 K-nearest neighbor.....	58
5.3.4 Results discussion .....	59
5.4 Retina dataset .....	59
5.4.1 Random Forest Results with 3 classes.....	60
5.4.2 Decision Tree Results with 3 classes.....	61
5.4.4 K-nearest neighbor Results with 3 classes.....	62
5.4.6 Random Forest Results with 2 classes.....	63

5.4.7 Decision Tree Results with 2 classes .....	64
5.4.9 K-nearest neighbor Results with 2 classes.....	65
5.4.10 Retina dataset result discussion .....	66
5.4.11 Feature Selection.....	66

# List of Figures

Figure 1: Video gaming revenue comparison with other media (Clement, 2021).....	3
Figure 2: Number of gamers worldwide (Clement, 2021).....	4
Figure 3 : Video game market revenue (Clement, 2021).....	5
Figure 4 : RTP header fields (Schulzrinne H. C., 2003).....	9
Figure 5 : RTP header fields from a captured packet .....	11
Figure 6 : Example of a DTLS packet .....	12
Figure 7 : Classification vs Regression.....	15
Figure 8 : Classification tree example .....	17
Figure 9 : K-nearest neighbor example.....	19
Figure 10 : Random Forest example.....	20
Figure 11 : Tshark script used.....	25
Figure 12: Testing environment.....	25
Figure 13: Stadia free to play games .....	26
Figure 14: GeForce now free to play games.....	26
Figure 15 : Wireshark GUI.....	27
Figure 16 : Example of the megabits per second behavior Stadia .....	32
Figure 17 : Example of different bitrate vs time plots from Stadia captures .....	33
Figure 18: GeForce Now bitrates vs time plots .....	34
Figure 19 : Bitrate CDF comparison plot .....	35
Figure 20 : Packet length CDF graphs.....	36
Figure 21 : RTP marker for frames.....	37
Figure 22 : FPS CDR graphs .....	38
Figure 23: Fps graphs from Stadia.....	39
Figure 24: Fps graphs from GeForce Now .....	39
Figure 25 : DTLS packets CDF distributions .....	41



Figure 26: DTLS packets sent vs time Stadia examples.....	42
Figure 27 : DTLS packets sent vs time GeForce examples .....	43
Figure 28: Packets inter-arrivals CD .....	44
Figure 29 : Packets inter-arrival CDF clipped .....	45
Figure 30 : Packet received CDF .....	46
Figure 31 : Dataset samples .....	50
Figure 32: Dataset samples continuation .....	50
Figure 33: Dataset cross-correlation matrix.....	52
Figure 34 : Retina Dataset columns .....	60
Figure 35: F1 scores versus number of features selected .....	67

# List of Tables

Table 1: Confusion Matrix example .....	21
Table 2 : Captures general information .....	28
Table 3: Per platform information .....	28
Table 4: Stadia games information .....	29
Table 5 : GeForce game information .....	30
Table 6: DTLS packets statistics .....	41
Table 7: Packets inter-arrivals statistics.....	44
Table 8: Destiny 2 general characteristics .....	46
Table 9 : Dataset number of classes per samples.....	49
Table 10: Random Forest classification report, 3 classes .....	53
Table 11: Random forest classifier confusion matrix, 3 classes .....	53
Table 12: Decision tree classification report, 3 classes .....	54
Table 13: Decision tree classifier confusion matrix, 3 classes .....	54
Table 14: K-nearest neighbor classification report, 3 classes .....	55
Table 15: K-nearest neighbor classifier confusion matrix, 3 classes .....	55
Table 16 : Random Forest classification report with two classes .....	56
Table 17 : Random Forest confusion matrix with two classes .....	57
Table 18: Decision Tree classification report with two classes .....	57
Table 19 : Decision Tree confusion matrix with two classes .....	58
Table 20: K-nearest neighbors classification report with two classes .....	58
Table 21 : K-nearest neighbors confusion matrix with two classes.....	58
Table and 22: Random Forest classification report, Retina dataset and 3 classes .....	61
Table 23: Random Forest classifier confusion matrix retina dataset,3 classes .....	61
Table 24: Decision Tree classification report, Retina dataset and 3 classes.....	62
Table 25: Decision Tree classifier confusion matrix Retina dataset and 3 classes....	62

Table 26 : K-nearest neighbor classification report, Retina dataset and 3 classes.....	62
Table 27: K-nearest neighbor confusion matrix Retina dataset and 3 classes .....	63
Table 28: Random Forest classification report, Retina dataset and 2 classes.....	63
Table 29: Random Forest classifier confusion matrix retina dataset,2 classes .....	64
Table 30: Decision Tree classification report, Retina dataset and 3 classes.....	64
Table 31: Decision Tree classifier confusion matrix Retina dataset and 3 classes ....	64
Table 32 : K-nearest neighbor classification report, Retina dataset and 3 classes.....	65
Table 33: K-nearest neighbor confusion matrix Retina dataset and 3 classes .....	65
Table 34: Random Forest, 2 classes, 6 features classification report .....	67
Table 35: Random Forest, 2 classes, 6 features confusion matrix.....	67



# Chapter 1

## Introduction

### 1.1 Networking principles

The Internet nowadays has become an indispensable tool for us to carry out our daily activities, it reaches every single sector of modern-day life. Due to the huge diversity of services and apps that are available online today the traffic generated by each application can have very distinct characteristics that differentiate them from each other. All this traffic usually traverses a sequence of network nodes to reach its destination, but each service has specific requirements in order to perform well, for example, voice traffic requires a fixed bitrate, low latency and low jitter and can accept some packet loss, however for file downloading delays and jitter are not a problem but requires a low or negligible lost probability. When there is congestion on the network, the resources are scarce so some packets need to be dropped, then it is valuable to have a policy that establishes a priority, which flows will suffer losses and will not have a significant impact on their performance, since the transport layer will retransmit them later and which are of the upmost importance like real-time applications for video and audio streaming, where packets cannot be retransmitted because a delayed packet is deemed as a lost packet. Hence, classes of service are defined to give priority to the flows on the routers queues to ensure an acceptable Quality of Service (QoS). QoS is the term used usually to describe the performance of an IT system. There are several metrics used to describe it such as speed, expressed on bitrate or throughput; delays, loss probability, error probability and many others. With the emergence of real-time network applications like video-streaming or gaming another concept has come out to take into account the considerations and sensation of the users called Quality of Experience (QoE) which allows Service Providers to have some feedback on the quality that their users subjectively

perceive their product. With this thesis a QoE solution will not be provided, but a method to identify different QoE classes on cloud gaming will be developed.

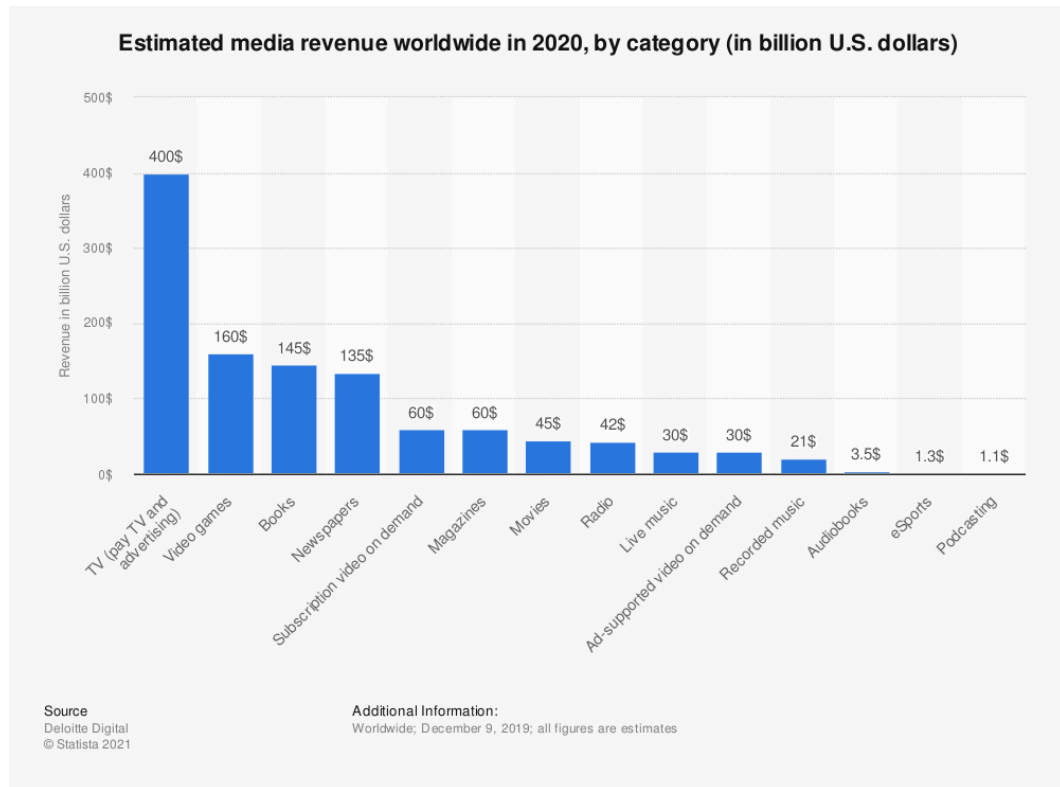
When the internet was created, it was not envisioned to carry real-time traffic which is the most common type of traffic on today's networks, many protocols have been developed to meet the demands today's internet users, one of the most important ones for video and voice traffic is the Real-Time Transport Protocol. RTP was created to solve a very important problem, TCP cannot be used to transport real-time traffic because it uses a sliding window for flow and congestion control, which destroys the performance of live multimedia due to the retransmissions and shrinking of the window, that being the case, the only other option was using UDP. While UDP does fulfill the requirement of being not connection oriented and best effort delivery since, it is not enough to guarantee the correct reconstruction of streams on the receiver end because it does not offer mechanisms that allow packet loss detection or ordered packet delivery. In further chapters RTP will be described in more details. It was a key protocol for the development of this study.

Recently, machine learning has proved its potential for improving network's management and optimization. The ability to perform complex mathematical computations to data that has increased in volume and variety is very powerful, it is used for recognizing patterns, built models, make predictions and take decisions based without much human intervention. On this paper machine learning techniques will be exploited for distinguishing the state of the user game during a browser gaming session.

## **1.2 Gaming Industry**

Video games occupy nowadays a very important place on the entertainment industry worldwide. The global gaming market has grown from 52.8 billion of dollars on 2012 to an estimate of 138.4 billion on 2021, and it is expected to keep growing in the following years.

Figure 1 breaks the revenue worldwide among the media sources.



**Figure 1: Video gaming revenue comparison with other media (Clement, 2021)**

As it can be observed, gaming is now the second highest revenue among all media sources, only behind TV (pay TV and advertising) and ahead of traditional media like books and newspapers, which demonstrates the importance it has on today's market.

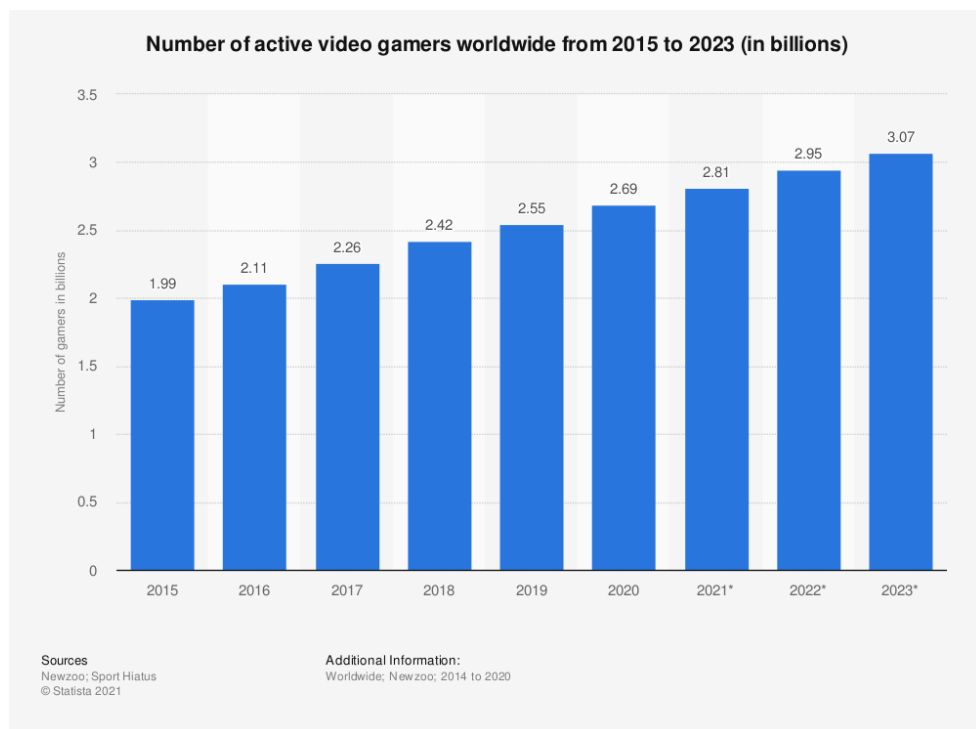
### 1.2.1 Gaming sales post-COVID-19

Looking back at gamer spending in 2020, worldwide digital gaming spending on in-game content and paid downloads has increased by 12 percent and 21 percent respectively, highlighting the growth of digital revenues. (Clement, 2021)

This development is not only due to COVID-19 – the industry has continuously making inroads to live service revenues and in-game monetization. However, during the last year, gamers have used these services at an unprecedented volume and the exceptional situation has led many holdouts to finally embrace digital purchases. (Clement, 2021)

### 1.2.2 Gaming during lockdown

As of June 2020, time spent video gaming during the COVID-19 pandemic increased by double digits in all regions, with Latin American gamers increasing their time spent on video games by 52 percent. Asia-Pacific was ranked second in terms of increased user engagement with a 42 percent increase of gaming time. Multiplayer games proved especially popular during COVID-19. A survey of European gaming audiences found that playing video games during lockdown made players feel less isolated and happier overall. Especially online multiplayer players felt positive about their gaming experiences during lockdown periods. (Clement, 2021)



**Figure 2: Number of gamers worldwide (Clement, 2021)**

According to Figure 2 the number of gamers has and will continue to increase worldwide, as of today 36.25% of the population plays online, which is distinctive indicator of the relevance of this sector on today's world.



Gaming can be divided into different sectors, which are the different devices or platforms which are used by the users to play. On Figure 2 the revenue of the main sectors is displayed.

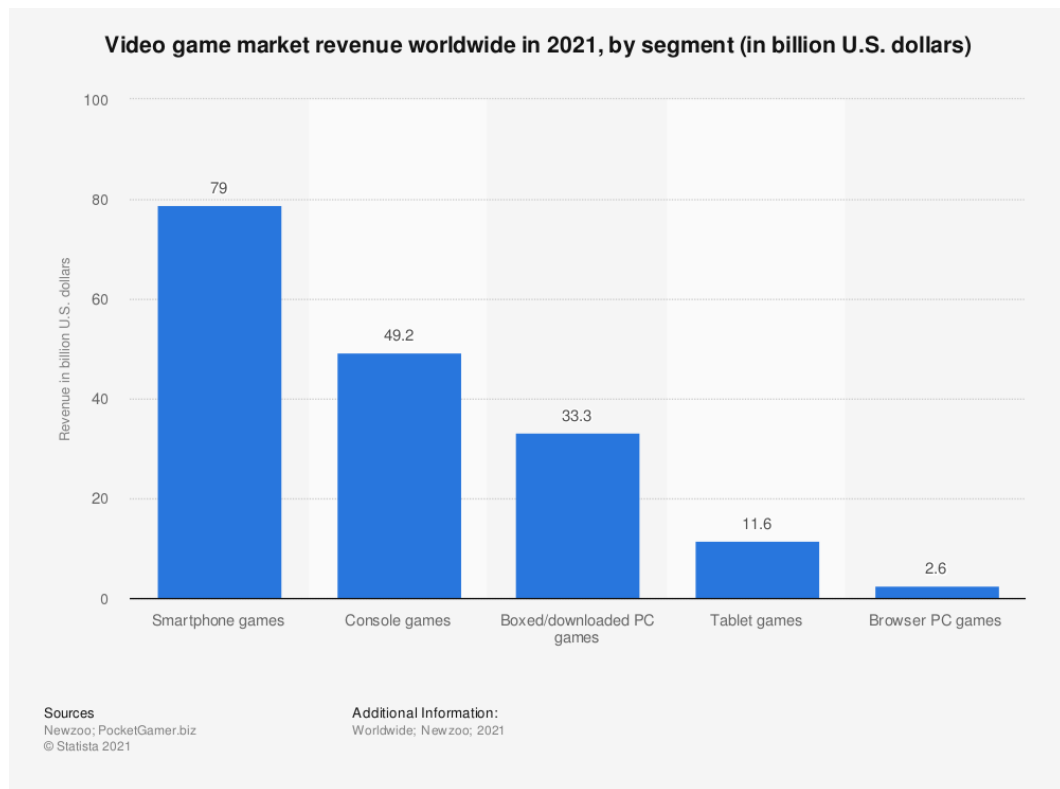


Figure 3 : Video game market revenue (Clement, 2021)

According to the data of Statista Smartphone gaming brings the highest revenue, more than the next two sectors combined, meanwhile browser PC games represent the list revenue at this moment, but that could change in the next years, browser gaming is almost brand new compared with the other segments of the industry and has the potential of being the next big market on the future.

### 1.2.3 Cloud gaming

Cloud gaming is a new class of services that promises to revolutionize the videogame market. It allows the user to play a videogame with basic equipment while using a remote server for the actual execution. The multimedia content is

streamed through the network from the server to the user. This service requires low latency and a large bandwidth to work properly with low response time and high-definition video. Three among the leading tech companies, (Google and NVIDIA) entered this market with their own products, and others, like Microsoft and Amazon, are planning to launch their own platforms in the near future. However, these companies released so far little information about their cloud gaming operation and how they utilize the network. (Domenico, Perna, Trevisan, Vassio, & Giordano, 2021)

The main advantage of this platforms is that the user doesn't need to have a state-of-the-art hardware to play brand new games which demand it, but instead, just need a good internet connection and of course being subscribed to the service.

### **1.3 Problem formulation**

Since cloud gaming has been developed recently as a model, there is not much information or studies regarding its implementation by the different vendors. Our first goal with this thesis is to perform a data collection process from Stadia and GeForce Now, using Wireshark to sniff the flow exchange between us, the client, and the servers, WebRTC which provides extra information about the traffic like resolution, codecs used, etc. and the software FlashBack Express to capture the screen where the games are being played.

The next goal after obtaining the data will be to do a traffic characterization using a python script to obtain all of the features considered important for this project and plot the results obtained.

Finally, train a machine learning model that can be implemented on the Service Providers nodes able to predict the state of the game at any moment with the intent of having a way to apply QoE mechanism on the future that can improve the user satisfaction, attract new customers and, therefore increase their revenue.

### **1.4 Literature Review**

The literature review showed that many studies have been done on classification of RTP traffic for video streaming, and there a few papers about cloud gaming in general, architecture, performance, quality evaluation, latency analysis, etc. In this thesis, we characterize cloud gaming traffic and go one step forward, developing a classifier that distinguishes game phases, for easier QoE algorithm adoption.

Some papers study the employment of protocols by RTC applications, such as online meetings and cloud gaming:

- (Domenico, Perna, Trevisan, Vassio, & Giordano, 2021) Which lays out a study of the employed protocols by the cloud gaming services and the workload they impose on the network.
- (Carrascosa & Bellalta, 2020) It provides a deep understanding of Stadia traffic characteristics by identifying the different protocols involved for both signaling and video/audio contents.
- (Suznjetic, Slivar, & Skorin-Kapov, 2016) Which research's GeForce Now adaptation mechanisms when facing variable network conditions.

On the other hand, books useful to understand and deepen the knowledge on the RTP protocol are:

- (Perkins, 2003)
- (Schulzrinne, 2003)

Understanding RTP was vital for the development of this thesis, both of the cited documents give a very extended explanation on the functionalities of the protocol.

For the last part of the thesis machine learning plays an important role, publications that were helpful for the development of the work were:

- (Perna, Markudova, Trevisan, & Garza, Online Classification of RTC Traffic, 2021). It proposes a machine learning-based application, to classify, in real-time, the media streams generated by RTC applications.
- (Bonaccorso, 2017). It explains how the different machine learning algorithms work, their purpose, best case of usage and examples.

Information gathered from these papers was very helpful for the development of this thesis, unlike those books and papers, our objective is to train a classification algorithm that is able to distinguish between states of the cloud gaming sessions that can be helpful for the implementation of future QoE mechanism.

## Chapter 2

# Background

### 2.1 RTP

Real Time Protocol (RTP) is, as its name suggests, the protocol standardly used for real-time audio and video communications over the internet; originally proposed by the Audio-Video Transport Working Group of the Internet Engineering Task Force (IETF) on RFC 1889 which been made obsolete by RFC3550. RTP allows to add timestamps, for synchronization purposes; sequence numbers, for packet loss detection and reordering; and source identifiers to the packets, and it is able to identify the kind of information being transported.

RTP provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services. RTP does not address resource reservation and does not guarantee quality-of-service for real-time services. The data transport is augmented by a control protocol (RTCP) to allow monitoring of the data delivery in a manner scalable to large multicast networks, and to provide minimal control and identification functionality. RTP and RTCP are designed to be independent of the underlying transport and network layers. (Schulzrinne H. C., 2003)

RTP is defined consisting of two closely linked parts:

- the real-time transport protocol (RTP), to carry data that has real-time properties.
- the RTP control protocol (RTCP), to monitor the quality of service and to convey information about the participants in an on-going session. The latter aspect of RTCP may be sufficient for "loosely controlled" sessions, i.e., where there is no explicit membership control and set-up, but it is not

necessarily intended to support all of an application's control communication requirements. This functionality may be fully or partially subsumed by a separate session control protocol, which is beyond the scope of this document. (Schulzrinne, Casner, Frederick, & Jacobson, 2003)

The RTP header is shown in Figure 4.

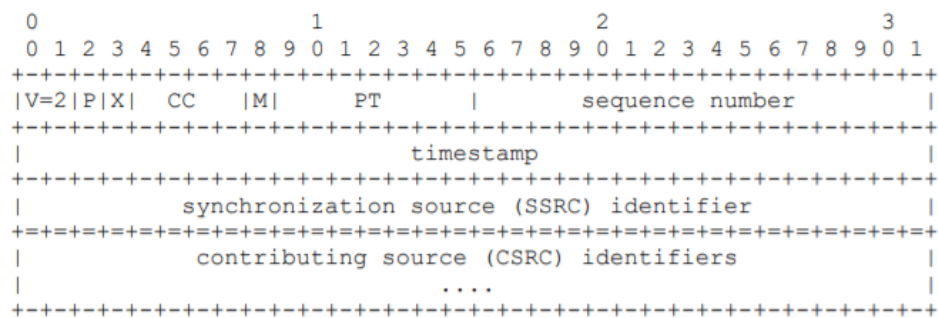


Figure 4 : RTP header fields (Schulzrinne H. C., 2003)

The header fields have the following meaning:

- version (V): 2 bits; the version currently use is number 2.
- padding (P): 1 bit If the padding bit is set, the packet contains one or more additional padding octets at the end which are not part of the payload. The last octet of the padding contains a count of how many padding octets should be ignored, including itself. Padding may be needed by some encryption algorithms with fixed block sizes or for carrying several RTP packets in a lower-layer protocol data unit.
- extension (X): 1 bit If the extension bit is set, the fixed header MUST be followed by exactly one header extension.
- CSRC count (CC): 4 bits The CSRC count contains the number of CSRC I dentifiers that follow the fixed header.
- marker (M): 1 bit The interpretation of the marker is defined by a profile. It is intended to allow significant events such as frame boundaries to be marked in the packet stream. A profile MAY define additional marker bits or specify that there is no marker bit by changing the number of bits in the payload type field.

- **payload type (PT): 7 bits** This field identifies the format of the RTP payload and determines its interpretation by the application. A profile may specify a default static mapping of payload type codes to payload formats. Additional payload type codes may be defined dynamically through non-RTP means. A receiver must ignore packets with payload types that it does not understand
- **sequence number: 16 bits** The sequence number increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. The initial value of the sequence number should be random (unpredictable) to make known-plaintext attacks on encryption more difficult, even if the source itself does not encrypt, because the packets may flow through a translator that does.
- **timestamp: 32 bits** The timestamp reflects the sampling instant of the first octet in the RTP data packet. The sampling instant must be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations
- **SSRC: 32 bits** The SSRC field identifies the synchronization source. This identifier should be chosen randomly, with the intent that no two synchronization sources within the same RTP session will have the same SSRC identifier.
- **CSRC list: 0 to 15 items, 32 bits each** The CSRC list identifies the contributing sources for the payload contained in this packet. The number of identifiers is given by the CC field. If there are more than 15 contributing sources, only 15 can be identified. CSRC identifiers are inserted by mixers, using the SSRC identifiers of contributing sources. (Schulzrinne, Casner, Frederick, & Jacobson, 2003)

The RTP header fields and their values can be examined using Wireshark, as it can be seen on Figure 5.

No.	Time	Actual Time	Source	Destination	Protocol	Length	Source Port	Destination Port	Payload type	Synchronization Source identifier	Sequence number	Info
117201	121.499770	Apr 19, 2021 10:29:54.172226000	192.168.152.87	136.115.130.173	DTLSv1	203	55459	44700				Applicat
117202	121.499850	Apr 19, 2021 10:29:54.172306000	192.168.152.87	136.115.130.173	DTLSv1	131	55459	44700				Applicat
117203	121.500726	Apr 19, 2021 10:29:54.173182000	136.115.130.173	192.168.152.87	RTP	359	44700	55459	111 0xfc4beb6d (4232833901)		12063	PT-Dynam
117204	121.501310	Apr 19, 2021 10:29:54.173766000	192.168.152.87	136.115.130.173	DTLSv1	175	55459	44700				Applicat
117205	121.502582	Apr 19, 2021 10:29:54.175018000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57eabe9f (1474956959)		39469	PT-Dynam
117206	121.503091	Apr 19, 2021 10:29:54.175547000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57eabe9f (1474956959)		39470	PT-Dynam
117207	121.503091	Apr 19, 2021 10:29:54.175547000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57eabe9f (1474956959)		39471	PT-Dynam
117208	121.503091	Apr 19, 2021 10:29:54.175547000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57eabe9f (1474956959)		39472	PT-Dynam

```

> Frame 117209: 1222 bytes on wire (9776 bits), 1222 bytes captured (9776 bits)
> Ethernet II, Src: Sophos_Ec:84:c7 (00:1a:8c:84:c7), Dst: LcFOHefc_30:ad:4d (e8:6a:64:30:ad:4d)
> Internet Protocol Version 4, Src: 136.115.130.173, Dst: 192.168.152.87
> User Datagram Protocol, Src Port: 44700, Dst Port: 55459
  > Real-Time Transport Protocol
    0.. .... = Version: RFC 1889 Version (2)
    0.. .... = Padding: False
    .... .... = Extension: True
    .... 0000 = Contributing source identifiers count: 0
    0.. .... = Marker: False
    Payload type: DynamicRTP-Type-96 (96)
    Sequence number: 39473
    Timestamp: 1096062586
    Synchronization Source identifier: 0x57eabe9f (1474956959)
    Defined by profile: Unknown (0xbede)
    Extension length: 1
    Header extensions
    Payload: 0002a648aed5b320f53fd3202d3c789fc0f033de1a49d56513a8b97c9117528bdf6995..
  
```

Figure 5 : RTP header fields from a captured packet

## 2.2 DTLS

It is of the utmost importance to have secure web sessions, the information transmitted through a network today can be very personal and delicate. Therefore, security protocols play a decisive role within the protocol stack, they can provide single or mutual authentication, integrity and privacy.

TLS is without a doubt the most widely deployed security protocol for networking. It runs between the application and transport layer and provides a transparent connection-oriented secure channel. However, this last characteristic of being connection oriented, using TCP on the transport layer, it does not allow to be used for securing unreliable datagram traffic. Since a larger number of protocols that have surged lately have been designed to use UDP they cannot be secured with TLS.

The unreliability of datagram traffic creates two different problems for TLS:

- TLS does not allow the decryption of individual records independently; the message integrity checks fail if a packet gets lost
- If a TLS handshake packet is lost the process breaks

Another problem is that in TLS's traffic encryption layer (called the TLS Record Layer), records are not independent. There are two kinds of inter-record dependency:

- Cryptographic context (stream cipher key stream) is retained between records.
- Anti-replay and message reordering protection are provided by a MAC that includes a sequence number, but the sequence numbers are implicit in the records.

DTLS solves the first problem by banning stream ciphers and solves the second problem by adding explicit sequence numbers. (Rescorla & Modadugu, 2012)

DTLS was invented to provide security to this kind of applications, media streaming, VoIP and online gaming. The purpose of DTLS is to make the least number of changes possible to TLS to provide datagram transport service.

No.	Time	Arrival Time	Source	Destination	Protocol	Length	Source Port	Destination Port	Payload type	Synchronization Source identifier	Sequence number	Info
117207	121.503091	Apr 19, 2021 10:29:54.175547000 M.	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	1474956959	39471	PT-Dynam
117208	121.503091	Apr 19, 2021 10:29:54.175547000 M.	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	1474956959	39472	PT-Dynam
117209	121.503091	Apr 19, 2021 10:29:54.175547000 M.	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	1474956959	39473	PT-Dynam
117210	121.503091	Apr 19, 2021 10:29:54.175547000 M.	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	1474956959	39474	PT-Dynam
117211	121.503091	Apr 19, 2021 10:29:54.175547000 M.	136.115.130.173	192.168.152.87	RTP	77	44700	55459	96 0x57ea0e9f (1474956959)	1474956959	39475	PT-Dynam
117212	121.503620	Apr 19, 2021 10:29:54.176876000 M.	136.115.130.173	192.168.152.87	DTLSv1.2	107	44700	55459				Applicat
117213	121.504086	Apr 19, 2021 10:29:54.176542000 M.	192.168.152.87	136.115.130.173	RTCP	172	55459	44700				Generic
117214	121.505382	Apr 19, 2021 10:29:54.177838000 M.	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	1474956959	39476	PT-Dynam

Frame 117212: 107 bytes on wire (856 bits), 107 bytes captured (856 bits)
> Ethernet II, Src: Sophos_6c:8d:c7 (00:1a:8c:6c:8d:c7), Dst: LCF0eFe_30:ad:4d (e8:6a:64:30:ad:4d)
> Internet Protocol Version 4, Src: 136.115.130.173, Dst: 192.168.152.87
> User Datagram Protocol, Src Port: 44700, Dst Port: 55459
> Datagram Transport Layer Security
<ul style="list-style-type: none"> <li>DTLSv1.2 Record Layer: Application Data Protocol: Application Data           <ul style="list-style-type: none"> <li>Content Type: Application Data (23)</li> <li>Version: DTLS 1.2 (0xfeed)</li> <li>Epoch: 1</li> <li>Sequence Number: 7025</li> <li>Length: 52</li> <li>Encrypted Application Data: 0001000000001b710b6e725d49924c9896e97c3922338a829af11a25e34e475e2cc09fb...</li> </ul> </li> </ul>

Figure 6 : Example of a DTLS packet



## Chapter 3

# Methodology

### 3.1 Machine Learning

Machine learning can be defined as the study of computer algorithms that can improve automatically through experience. Applications range from data mining programs that discover general rules in large data sets, to information filtering systems that automatically learn users' preferences. It has surged as one of the most critical and successful artificial intelligence branches.

Techniques based on Machine Learning have been successfully applied in a large number of fields such as finance, medicine, pattern recognition, engineering, etc. The ability of machine learning algorithms to learn from current context and generalize into unseen tasks would allow improvements in the efficacy of many tasks.

*“A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .” (Mitchell, 1997)*

In our case, we could define  $E$  as the amount of data collected from the games,  $T$  the classification of when the user is playing or not and  $P$  the percentage of predictions that were correct.

The main goal of machine learning is to create, improve and tune mathematical models that can be continuously trained, with data from a certain environment to predict and make decisions trying to determine the correct

probability distributions and uses them to compute the action that is most likely to be successful at a certain task.

The machine learning process developed on the thesis can be summarized on 4 steps:

- Data collection
- Data preparation
- Machine learning modelling
- Features engineering

Based on their desired outcome machine learning algorithms can be classified into groups:

- Supervised learning: is an approach characterized by the use of labeled datasets. These datasets are designed to train algorithms into classifying data or predicting outcomes accurately.
- Unsupervised learning: designed to analyze and cluster unlabeled data sets. These algorithms discover hidden patterns in data without the need for human intervention.
- Semi-supervised learning - combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- Reinforcement learning: the algorithm learns a policy of how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback that guides the learning algorithm.

## 3.2 Supervised Learning

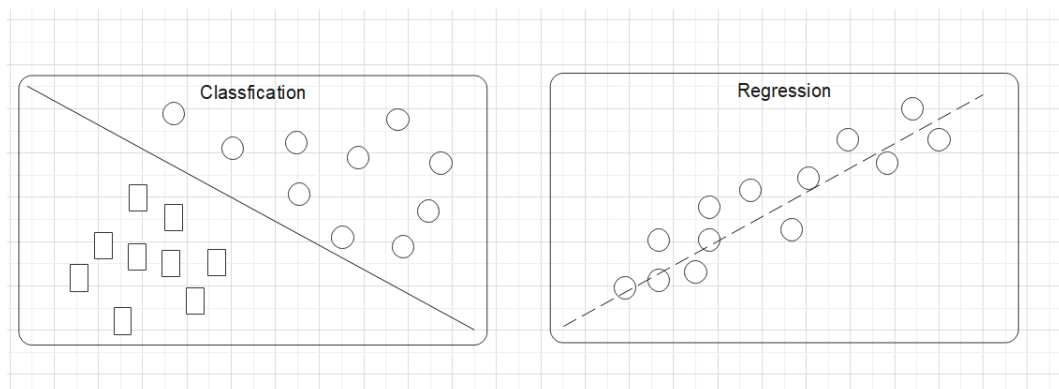
Supervised learning can be defined as the machine learning task of learning a function that maps an input to an output based on example input-output pairs (Stuart J. Russell, 2010). It uses labeled datasets to train algorithms that give either a discrete number of outputs, called categories and therefore the process is named classification; or continuous values output, then the process is named regression.

The learning process in a supervised machine learning model consists in two parts: training and testing. During the training process, samples in training data are taken as input in which features are learned by learning algorithm or learner and build the learning model. In the testing process, learning model uses the execution engine to make the prediction for the test or production data. Tagged data is the output of learning model which gives the final prediction or classified data. (Nasteski, 2017)

Commonly used algorithms in supervised learning are naive bayes, logistic regression, support vector machines, random forests, and neural networks. In both regression and classification, the objective is to infer specific relationships in the input data that allow us to effectively predict the output data. The correct output is the one registered on the input data, so the correctness of the model also depends on whether the data labels correspond to the truth real-world situation; noisy, data labels will decrease the accuracy of the model.

As stated, supervised machine learning is divided into:

- **Classification:** its algorithms are used to try to accurately assign data into specific classes. It learns from the dataset characteristics and attempts to draw some conclusions on how those entities should be labeled. Common classification algorithms are linear classifiers, decision trees, k-nearest neighbor, support vector machines (SVM) and random forest.
- **Regression:** unlike classification the dataset is not divided into targeted classes, the objective is to make real-values predictions base on the input dataset, examples where it is use are for predicting temperature, sales, revenue, etc. Linear regression, logistical regression, support vector machine, Multivariate Regression and polynomial regression are popular regression algorithms.



**Figure 7 : Classification vs Regression**

Figure 7 is a graphical representation of the differences on the classification and regression problems, on the left we can observe how squares and circular samples are classified on different classes and on the right, we can see how the dash line tries to infer the path of the samples.

### 3.3 Unsupervised Machine Learning

Unlike supervised learning, unsupervised machine learning algorithm are not provided with any labels or scores on the input dataset. Their purpose is to discover hidden patterns in the dataset without human guide. They group information according to similarities and differences despite no categories being provided to them. The two classic examples of unsupervised learning are clustering and dimensionality reduction.

Common unsupervised learning applications are:

- Object segmentation
- Similarity detection
- Automatic labeling

Clustering is the most typical implementation of unsupervised learning and can be defined as a volume of high-density points separated from other clusters by a relatively low-density volume. Common clustering approaches are:

- Centroid-based Clustering: organizes the data into non-hierarchical clusters, k-means is the most widely-used centroid-based clustering algorithm
- Density-based Clustering: connects areas of high density into clusters, which allows for arbitrary-shaped distributions if dense areas can be connected.
- Distribution-based Clustering: assumes data is composed of distributions and tries to grouped them consequently
- Hierarchical Clustering : creates a tree of clusters, any number of clusters can be chosen by cutting the tree at the right level.

## 3.4 Supervised Machine Learning Algorithms

### 3.4.1 Decision Tree

Decision trees, as its name suggests, use a tree-like model of decision and their possible outcome or classes. They can be used for both classification, either binary, where labels are  $[1, -1]$  and multiclass, where labels are  $[0, \dots, k-1]$ ; and regression, they can be represented as a flowchart-like structure in which each node represents a test on an attribute, each branch, the result of the node test and each leaf the output value.

On Figure 8 an example of a classification tree is demonstrated, it can be observed that we start from a root node, where the whole training set is considered; then depending on the value of a variable  $X$  it continues through a branch, after that another test is performed in another node, depending on a  $Y$  variable, 4 of those branches lead to a leaf, each leaf represents a class; one of the nodes has a third test for the data and then it splits into 2 more leaves. This is a very simple example; in practice the tests are not that clear so the tree, depending on the algorithm used, usually adopts the strategy of selecting the best attribute for the split locally at each step.

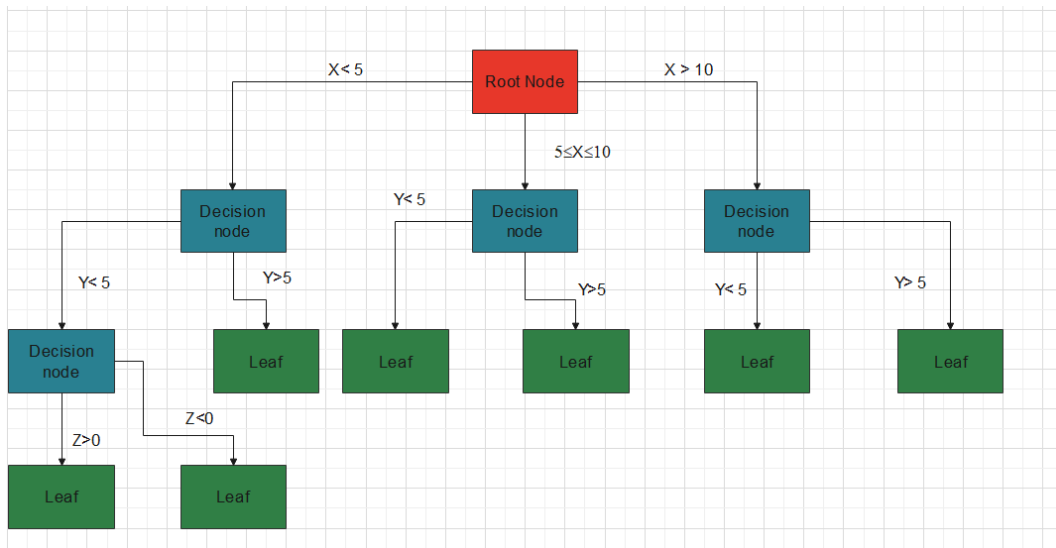


Figure 8 : Classification tree example

Several tree algorithms have been developed, such as Ide, C4.5, C5 and CART (Classification and Regression Trees). Scikit-learn, a very popular python machine learning library and the one used on this thesis, uses CART, which

constructs binary trees using the feature and threshold that yield the largest information gain at each node.

In the decision tree, the nodes are split into sub-nodes based on a threshold value of an attribute. The CART algorithm does that by searching for the best homogeneity for the sub-nodes, with the help of the Gini Index criteria.

$$GI = \sum_{i=0}^c p_i(1 - p_i)$$

Where  $c$  is the total number of classes and  $p_i$  the probability of class  $i$ .

The root node is taken as the training set and is split into two by considering the best attribute and threshold value. Further, the subsets are also split using the same logic. This continues till the last pure sub-set is found in the tree or the maximum number of leaves possible in that growing tree. This is also known as Tree Pruning.

As it works its way down the tree with the training data, the splitting method must know when to stop splitting. The most frequent halting method is to utilize a minimum amount of training data allocated to each leaf node. If the count is less than a certain threshold, the split is rejected and the node is considered the last leaf node.

### 3.4.2 K-nearest neighbors

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these. K nearest neighbors is a simple algorithm compared with the others you can find in the machine learning field, that can be used to solve classification and regression problems. It works by finding the distance between a sample and the other data points, selecting a specified number of neighbors  $K$ , closest to the sample and then votes for the most frequent label if it is used for classification or averages the labels if it is used for regression. Therefore, there are two very important parameters to select in order for the algorithm to work correctly, one is the value of  $K$  and the other one is how the computation of the distance will be computed. The distance can, in general, be any metric measure, standard Euclidean distance is the most common choice. If the value of  $K$  can lead to underfitting or overfitting issues if not correctly selected. Figure 9 shows a representation of how the K-nearest neighbors classifier works.

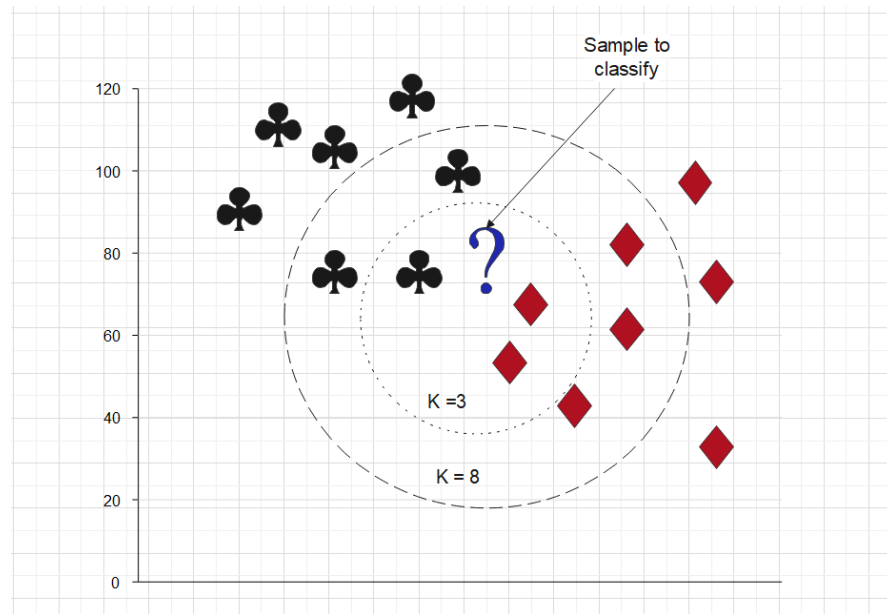


Figure 9 : K-nearest neighbor example

Most common metrics for the distance are:

- Euclidean Distance:

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

- Minkowski Distance

$$d(x, y) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

- Cosine distance:

$$\cos \theta = \frac{a \cdot b}{||a|| \cdot ||b||}$$

- Manhattan Distance:

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

### 3.4.3 Random Forest

Random forests are a supervised learning algorithm use for classification and regression that employs the ensemble learning method constructing a number of classification trees, which can be specified by the parameter *nestimators*, trained separately. For classification problems, the output of the random forest is selected by majority voting, in other words, the class selected by most trees; for regression problems, the output is the average of the predictions of all the individual trees. The forest generated by the algorithm is trained trough bagging or bootstrap aggregating which allows to reduce the variance and helps avoid overfitting.

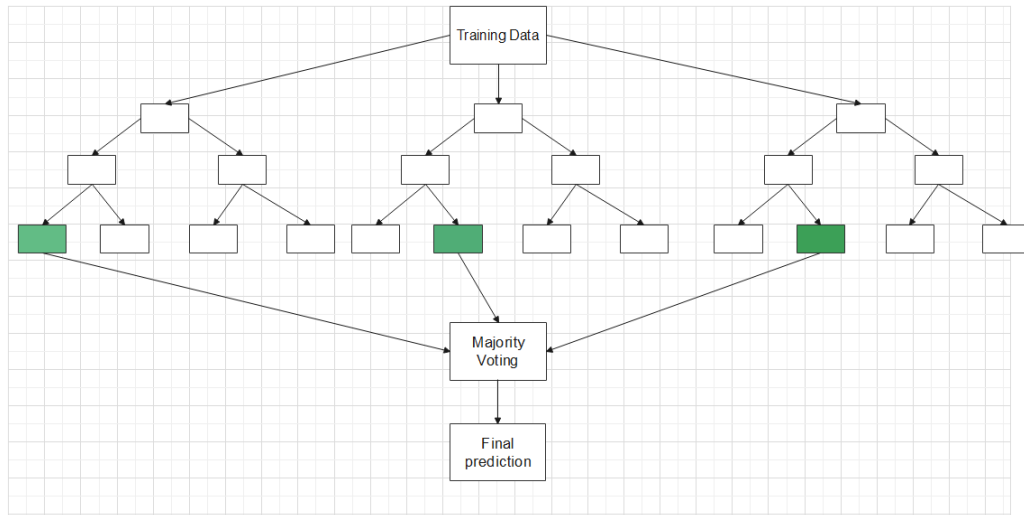


Figure 10 : Random Forest example

## 3.5 Feature Selection

Feature selection or dimensionality reduction, as its name indicates, is the process of reducing the number of input fields that are going to be fed to the machine learning models, some of these input variables do not contribute helpful information for the model and end up just as noise, so removing them boosts their performance. It reduces the overfitting, because with less redundant data the lower



the probability of taking decisions influenced by noise, it also lowers the training times since there are fewer datapoints and therefore lower computational complexity.

There are three main classes of feature selection used:

- Filter method: statistical approach, based on general features such as correlation with the variable to predict, only the variables that pass the filter are fed to the machine learning algorithm
- Wrapper method: Evaluate subsets of variables, which allow to identify relationship between them, they add or remove variables at every step to try to find the set that maximizes the machine learning algorithm performance.
- Embedded method: Tries to combine the advantages of the two previous methods

### 3.6 Performance Measures

There are several measures to evaluate the performance of the ML model predictions. On this paper we focus on classification problems, so the measure will be explained for a classification purpose. The most commonly used ones are:

- Accuracy
- Precision
- Recall
- F1 score

Before going into detail for each one of them, there are four terms that need to be explained. Taking as an example a classifier with only two classes a Confusion Matrix can be created like the one seen on Table 1.

**Table 1: Confusion Matrix example**

Actual Class	Predicted Class	
	0	1
0	True Negative	False Positive
1	False Negative	True Positive

True Positives (TP): Values correctly predictive as Class 1.

True Negatives (TF): Values correctly predicted as Class 0.

False Positives (FP): Values which were predicted as Class one but were actually from Class 0.

False Negatives (FN): Values which were predicted as Class 0 but were actually from Class 1.

With these four parameters in mind, now we can explain the rest of the measurements scores.

### 3.6.1 Accuracy

It is the most intuitive of the four measures, it is computed as the ratio of correctly predicted observations to the total of the samples.

$$Acc = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

It is a good measure for dataset that have equiprobable classes. The problem with accuracy is that for datasets who have very unbalanced probabilities, for example, if 90% of the samples are class 1, the algorithm may simply classify all samples as class 1, which is quite bad, it did not learn anything from the data, just that class 1 is the great majority, however this algorithm would have a 90% accuracy, which in theory is quite good.

### 3.6.2 Precision

The precision is defined as the ratio between the correctly predicted observations of a class to the total of predicted observations from that class. In other words, how many of the predictions for a specific class were correct.

$$Prec = \frac{TP}{(TP + FP)}$$

### 3.6.3 Recall

Recall is the ratio between the correctly predicted observations from a class to all of the samples on one class. It gives information of how good is the model on labeling an specific class.

$$Rec = TP / (TP + FN)$$

### 3.6.4 F1 score

The F1 score takes into consideration the precision and the recall, it is computed as the harmonic mean of them. It ranges from 0 to 1, being 1 the perfect score, and it gives much clearer measure of the model's accuracy.

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

# Chapter 4

## Dataset

### 4.1 Data Collection

The first step required for the development of the paper was to do a data collection from both platforms Stadia and GeForce Now. It was done as it follows.

Using a laptop and a monitor connected via HDMI, connected to internet through an Ethernet cable using the network from OGR, no other windows applications were opened to try to have the least amount less undesired traffic as possible, of course, there may be some process running on background from the Windows Operating System that can generate some traffic, but this will be filtered using tshark filters later.

Wireshark is one of the most used network protocol analyzers. It allows the user to do live captures of packets on a selected network adapter or read and analyze packets from a previously saved captured file. It provides many functionalities as:

- Decoding of hundreds of protocols, with more being added constantly
- Multi-platform: runs on most operating systems
- Powerful display filters and coloring rules to the packet lists for intuitive analysis
- Decryption support for many protocols, including IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, and WPA/WPA2

- Captured network data can be browsed via a GUI, or via the TTY-mode Tshark utility

```
tshark -i 8 -w D:\example.pcap -F pcap -p

PAUSE
```

Figure 11 : Tshark script used

Just before launching the game a Tshark script (Figure 11) to capture traffic, a program named Flashback Express to capture the screen were ran and a Chrome window with WebRTC internals was open to use the WebRTC tool functionalities. Then the game was played accessing it through another Chrome tab, for between 4 to 15 minutes, the next step was exiting and saving the pcap from tshark, the JSON log from WebRTC and the screen capture, they were saved on folders depending on the platforms with the following format:

X\_YY\_Z

X: number of the capture

YY: Abbreviation of name of the Game

Z: 1 or 2; 1 if it was captured from the loading phase, 2 if it was captured starting from a pause, no loading phase.

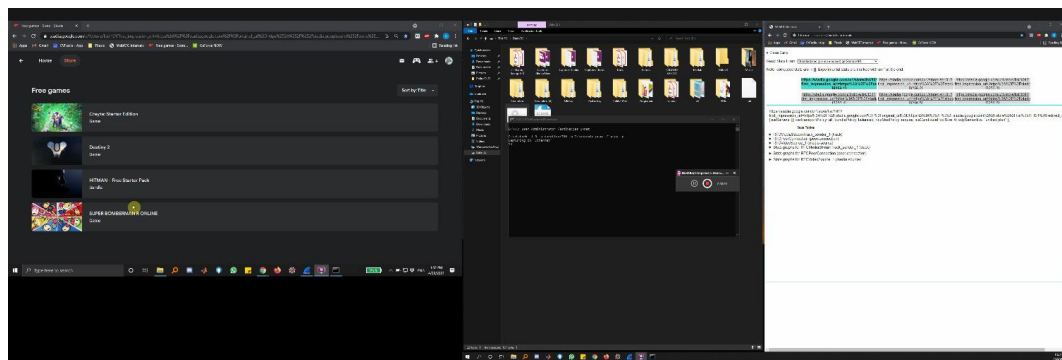


Figure 12: Testing environment

The games played on Stadia are shown on Figure 13, and were the 4 that are being offered for free at the moment, Crayta, Destiny 2. Hitman, Super Bomberman R Online.

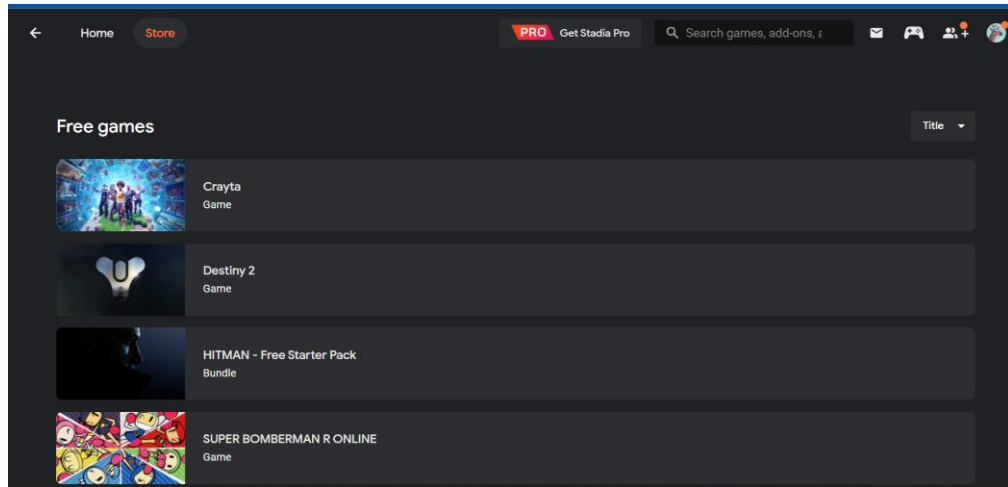


Figure 13: Stadia free to play games

On the other hand, GeForce Now allows you to use personal accounts from other gaming platforms such as Steam so you can play free games that those offer, for that reason they offer a total of 94 free to play games, I used my personal Steam account and played the two games I had access, which are Destiny 2 and Dota 2.

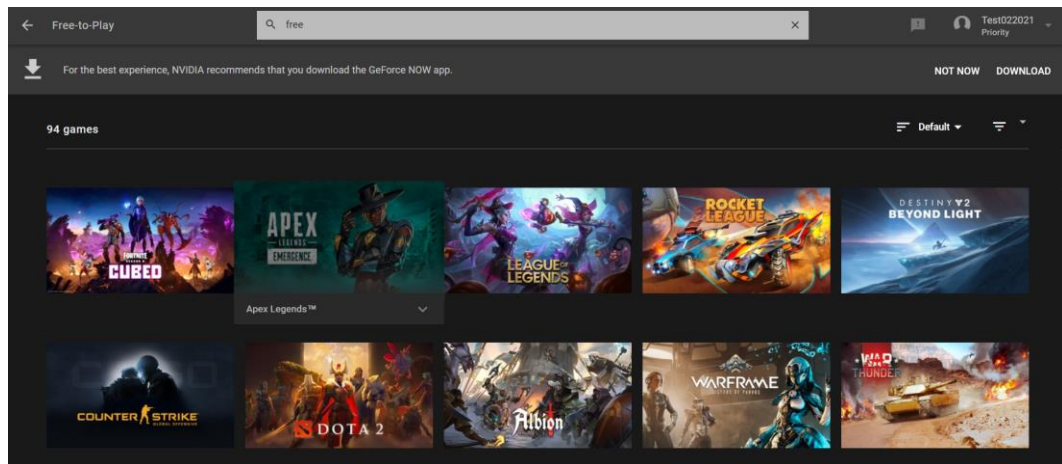
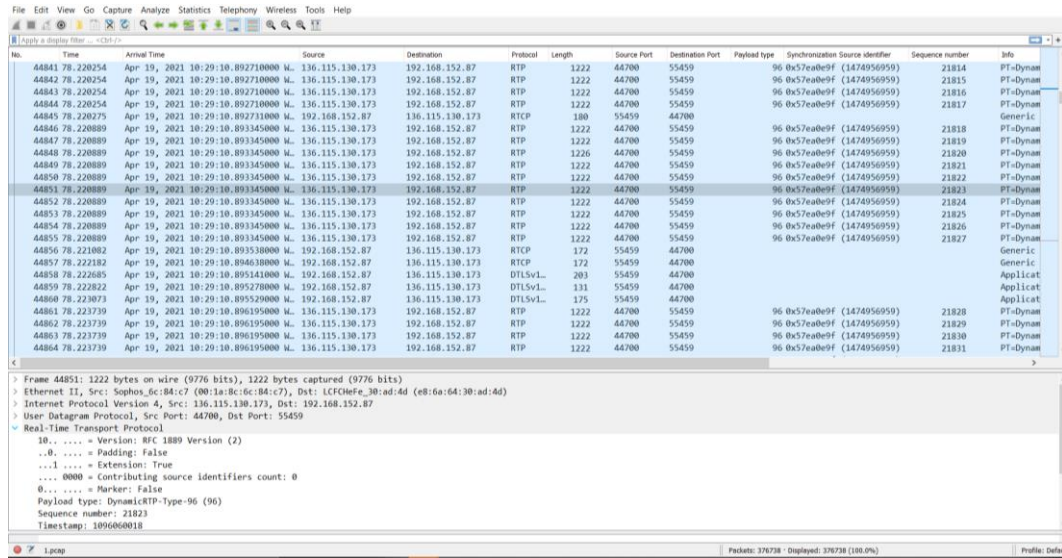


Figure 14: GeForce now free to play games

On Figure 15 can be observed an RTP flow carrying the game video stream and some DTLS packets carrying the user's keyboard commands to the cloud server when the game is running.



The screenshot displays the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons for file operations, capture control, and analysis. The main window is divided into three panes: a packet list on the left, a packet details pane in the middle, and a packet bytes pane on the right.

The packet list pane shows a table of captured packets. The columns are: No., Time, Arrival Time, Source, Destination, Protocol, Length, Source Port, Destination Port, Payload type, Synchronization Source identifier, Sequence number, and Info. The packets are filtered by 'Apply a display filter: <Ctrl>F'. The selected packet is 48851, which is an RTP packet of type 96 (96) from source 136.115.130.173 to destination 192.168.152.87. The details pane shows the selected packet's structure, including Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Real-time Transport Protocol (RTP) details.

No.	Time	Arrival Time	Source	Destination	Protocol	Length	Source Port	Destination Port	Payload type	Synchronization Source identifier	Sequence number	Info
48841	78.220254	Apr 19, 2021 10:29:10.892710000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21814	21814	PT-Dynan
48842	78.220254	Apr 19, 2021 10:29:10.892710000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21815	21815	PT-Dynan
48843	78.220254	Apr 19, 2021 10:29:10.892710000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21816	21816	PT-Dynan
48844	78.220254	Apr 19, 2021 10:29:10.892710000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21817	21817	PT-Dynan
48845	78.220275	Apr 19, 2021 10:29:10.892731000	136.115.130.173	192.168.152.87	RTP	180	55459	44700	96 0x57ea0e9f (1474956959)	21818	21818	Generic
48846	78.220889	Apr 19, 2021 10:29:10.893345000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21819	21819	PT-Dynan
48847	78.220889	Apr 19, 2021 10:29:10.893345000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21820	21820	PT-Dynan
48848	78.220889	Apr 19, 2021 10:29:10.893345000	136.115.130.173	192.168.152.87	RTP	1226	44700	55459	96 0x57ea0e9f (1474956959)	21821	21821	PT-Dynan
48849	78.220889	Apr 19, 2021 10:29:10.893345000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21822	21822	PT-Dynan
48850	78.220889	Apr 19, 2021 10:29:10.893345000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21823	21823	PT-Dynan
48851	78.220889	Apr 19, 2021 10:29:10.893345000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21824	21824	PT-Dynan
48852	78.220889	Apr 19, 2021 10:29:10.893345000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21825	21825	PT-Dynan
48853	78.220889	Apr 19, 2021 10:29:10.893345000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21826	21826	PT-Dynan
48854	78.220889	Apr 19, 2021 10:29:10.893345000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21827	21827	PT-Dynan
48855	78.221082	Apr 19, 2021 10:29:10.893380000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21828	21828	PT-Dynan
48856	78.221082	Apr 19, 2021 10:29:10.893380000	136.115.130.173	192.168.152.87	RTP	172	55459	44700	96 0x57ea0e9f (1474956959)	21829	21829	PT-Dynan
48857	78.222182	Apr 19, 2021 10:29:10.894638000	136.115.130.173	192.168.152.87	RTP	172	55459	44700	96 0x57ea0e9f (1474956959)	21830	21830	PT-Dynan
48858	78.222685	Apr 19, 2021 10:29:10.895141000	136.115.130.173	192.168.152.87	RTP	203	55459	44700	96 0x57ea0e9f (1474956959)	21831	21831	PT-Dynan
48859	78.222822	Apr 19, 2021 10:29:10.895278000	136.115.130.173	192.168.152.87	RTP	131	55459	44700	96 0x57ea0e9f (1474956959)	21832	21832	PT-Dynan
48860	78.223073	Apr 19, 2021 10:29:10.895295000	136.115.130.173	192.168.152.87	RTP	175	55459	44700	96 0x57ea0e9f (1474956959)	21833	21833	PT-Dynan
48861	78.223739	Apr 19, 2021 10:29:10.896195000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21834	21834	PT-Dynan
48862	78.223739	Apr 19, 2021 10:29:10.896195000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21835	21835	PT-Dynan
48863	78.223739	Apr 19, 2021 10:29:10.896195000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21836	21836	PT-Dynan
48864	78.223739	Apr 19, 2021 10:29:10.896195000	136.115.130.173	192.168.152.87	RTP	1222	44700	55459	96 0x57ea0e9f (1474956959)	21837	21837	PT-Dynan

The details pane for the selected packet (48851) shows the following structure:

- Frame 48851: 1222 bytes on wire (9776 bits), 1222 bytes captured (9776 bits) on interface 0
- Ethernet II, Src: Sophos, Dst: 136.115.130.173, Dst: 192.168.152.87
- Internet Protocol Version 4, Src: 136.115.130.173, Dst: 192.168.152.87
- User Datagram Protocol, Src Port: 44700, Dst Port: 55459
- Real-time Transport Protocol
  - Version: RFC 1889 Version (2)
  - Contributing source identifiers count: 0
  - Marker: False
  - Sequence number: 21823
  - Timestamp: 1096060018

Figure 15 : Wireshark GUI

## 4.2 Data Characterization

### 4.2.1 General Information

This section is dedicated to the analysis of the data collected, taking a deeper look into the characteristics of the traffic, which will also allow us to distinguish features for the composition of a dataset to feed to the machine learning algorithms.

A total of 196 captures were made between both platforms of traffic, summing up to 48797 seconds or 13 hours 33 minutes and 17 seconds, the average time played per capture was 4.15 minutes. 100 of the captures belong to GeForce Now with a total of 26138 seconds and the other 96 were from Stadia games for a total of 22479 seconds.

**Table 2 : Captures general information**

<b>Platform</b>	<b>Total Captures</b>	<b>Time(s)</b>
<i>Stadia</i>	96	22479 or 6 hours, 14 minutes and 39 seconds
<i>GeForce Now</i>	100	26138 or 7 hours 15 minutes and 38 seconds
<i>Total</i>	196	48797 or 13 hours 33 minutes and 17 seconds

In the following table, the average information from the Stadia and GeForce Now captures is gathered

**Table 3: Per platform information**

	<b>Stadia</b>	<b>GeForce Now</b>
<b>Average Bits per second</b>	11.33Mbps	7.65Mbps
<b>Maximum bitrate</b>	29.3Mbps	24.44Mbps
<b>Average packets per second received</b>	1215.68	869.64
<b>Average frames per second</b>	59.72	59.86
<b>Average packets inter arrivals</b>	3.42ms	4.57ms
<b>Average packet length</b>	1003 bytes	994 bytes
<b>Average packets per second sent</b>	109.05	75.14
<b>Resolution</b>	1920x180 and 1280x720	1366x768 and 1280x720
<b>Games played</b>	4	2



The following tables show the general statistics about each one of the games played in each platform.

**Table 4: Stadia games information**

<i>Stadia</i>	<b>Destiny 2</b>	<b>Crayta</b>	<b>Hitman</b>	<b>Bomberman Online</b>
<i>Average Bits per second</i>	15.7 Mbps	9.04 Mbps	7.13 Mbps	6.4 Mbps
<i>Maximum bitrate</i>	29.3Mbps	28.3Mbps	28 Mbps	27.85 Mbps
<i>Average packets per second received</i>	1674.82	975.18	775.67	704.58
<i>Average frames per second</i>	59.78	58.2	59.9	59.8
<i>Average packets inter arrivals</i>	3.89ms	1.76ms	3.484ms	3.09ms
<i>Average packet length</i>	994.79 bytes	1123.34 bytes	992 bytes	998.42 bytes
<i>Average packets per second sent</i>	112.65	113.71	122.1	82.05
<i>Resolution</i>	1920x1080 and 1280x720	1920x1080	1920x1080	1920x1080
<i>Captures</i>	45	10	25	16
<i>Seconds</i>	11039 seconds	1404 seconds	5608	4428

Table 5 : GeForce game information

<b>GeForce Now</b>	<b>Destiny 2</b>	<b>Dota 2</b>
<i>Average Bits per second</i>	9.25 Mbps	4.95 Mbps
<i>Maximum bitrate</i>	24.4Mbps	13.5Mbps
<i>Average packets per second received</i>	1033.1	593.1
<i>Average frames per second</i>	59.86	59.86
<i>Average packets inter arrivals</i>	4.36ms	4.89
<i>Average packet length</i>	989.31 bytes	1003.06 bytes
<i>Average packets per second sent</i>	56	107
<i>Resolution</i>	1366x768 and 1280x720	1366x768 and 1280x720
<i>Captures</i>	60	40
<i>Seconds</i>	16541	9777

In the following subchapters a deeper analysis is performed to characterize the data. A series of graphs were plotted that represent the Cumulative Distribution Function (CDF) of each of the networking measurements listed below.

The CDF is defined as the probability that a random variable  $X$  will take a value less than or equal to a certain point  $x$ . The CDF is always a non-decreasing function, in other words if  $y \geq x$ , then  $FX(y) \geq FX(x)$ . Finally, the CDF approaches 1 as  $x$  becomes large.

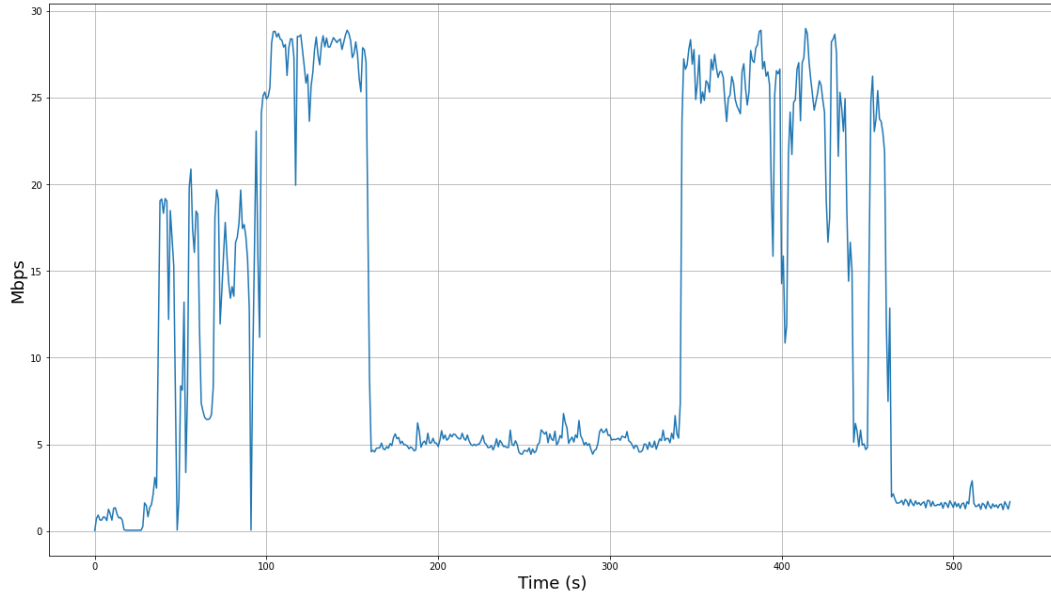
$$FX(x) = P(X \leq x) \quad \forall x \in \mathbb{R}$$

$$\lim_{x \rightarrow \infty} FX(x) = 1$$

### 4.2.2 Bitrate distribution analysis

Firstly, we want to take a look at the bit rate behavior. It is one of the most important parameters in networking, it has a great impact on the quality of the audio and the video streams. Depending on the video resolution it is expected to go up, the higher the resolution.

As an example, we can see the video stream bitrate from one of the captures made while playing Destiny 2 on Stadia.



**Figure 16 : Example of the megabits per second behavior Stadia**

The graphs on Figure 16 depicts how on the beginning of the connection, which correspond to the loading stage of the game, the bit rate is very bursty, with a high variance, and with several troughs. The minimums are because while loading the game there are several times where the screen goes black for a couple of seconds, therefore there is no need to transmit information. After that initial period, the bitrate increases significantly and is more stable around 28Mbps, still with some peaks and troughs but this is normal for video traffic, on this phase the game is being played normally. Then, around the 180th second the bitrate drastically and stables again at around 5 Mbps, this corresponds to the period where the game was paused. At the second 340 the game was resumed so the bitrate increases once again, at second 400 a small pause of around 2 seconds was done leading to a local minimum, and finally after a few more seconds of play the game was quitted, resulting on a stable bitrate of around 3Mbps.

This behavior was observed on most of the captures, more examples of it can be seen on Figure 17.

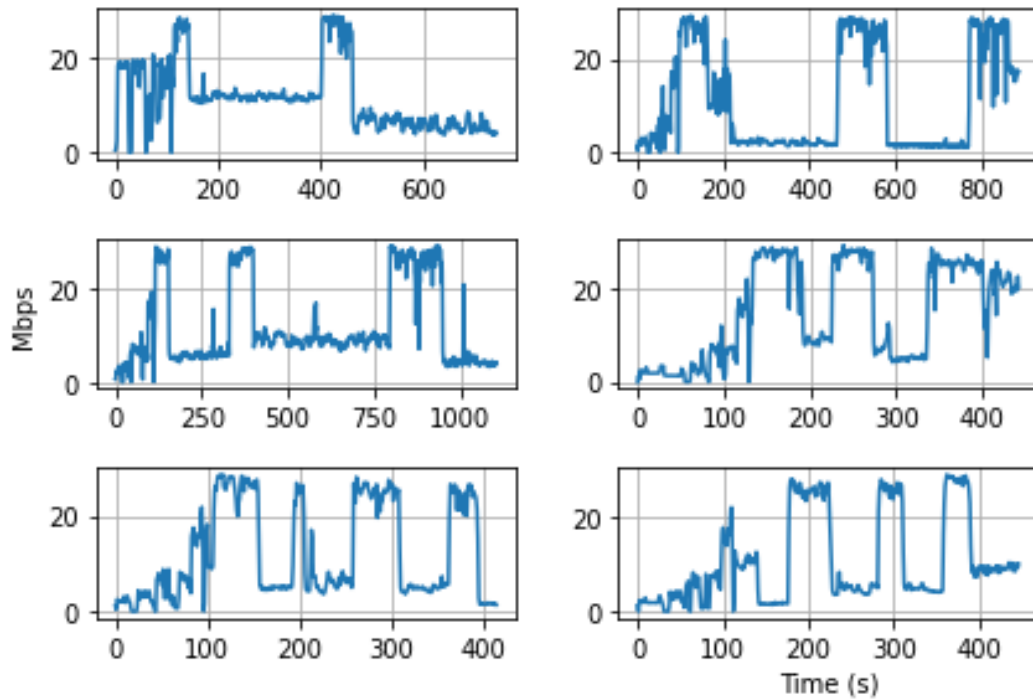


Figure 17 : Example of different bitrate vs time plots from Stadia captures

The same trend was observed on the captures from GeForce Now but with lower maximum bitrate, which was expected since the resolution was lower than the majority of the captures of Stadia. Figure 18 shows some of the GeForce Now captures bitrate plots, it is noticeable that the distance between the peaks and the lows on these graphs are shorter, this is because the maximum bitrate here is shorter, but the overall behavior of burst with several close to zero lows during the loading phase, burst with peaks during the game and lower constant rate during pauses is accurate.

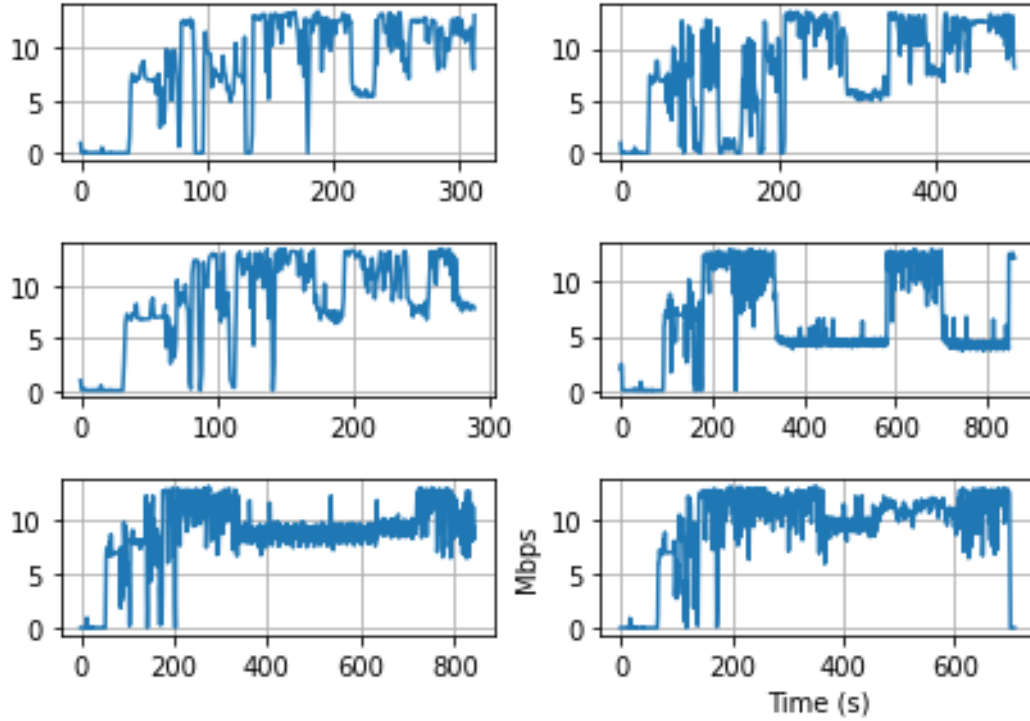


Figure 18: GeForce Now bitrates vs time plots

After collecting the total bits per second from each of the captures, they were split depending on the platform they were from and then their respective CDF were plotted. As we can see on Figure 19, the first notable difference is the range on the x axis, as previously mentioned Stadia has usually higher resolution and therefore higher bitrates. On the Stadia function we can observe that it surges more sharply on the extremes, which was expected, a large number of low values on the bitrate at the values between 0 and 5 Mbps are from the startup phase of the game and the pauses, while the surge of values at over 23 Mbps are those of the game running. On the other hand, from the GeForce Now graph we can take that very rarely the bitrate surpasses 20 Mbps and the rest of the values are more evenly distributed between 0 and 15 Mbps.

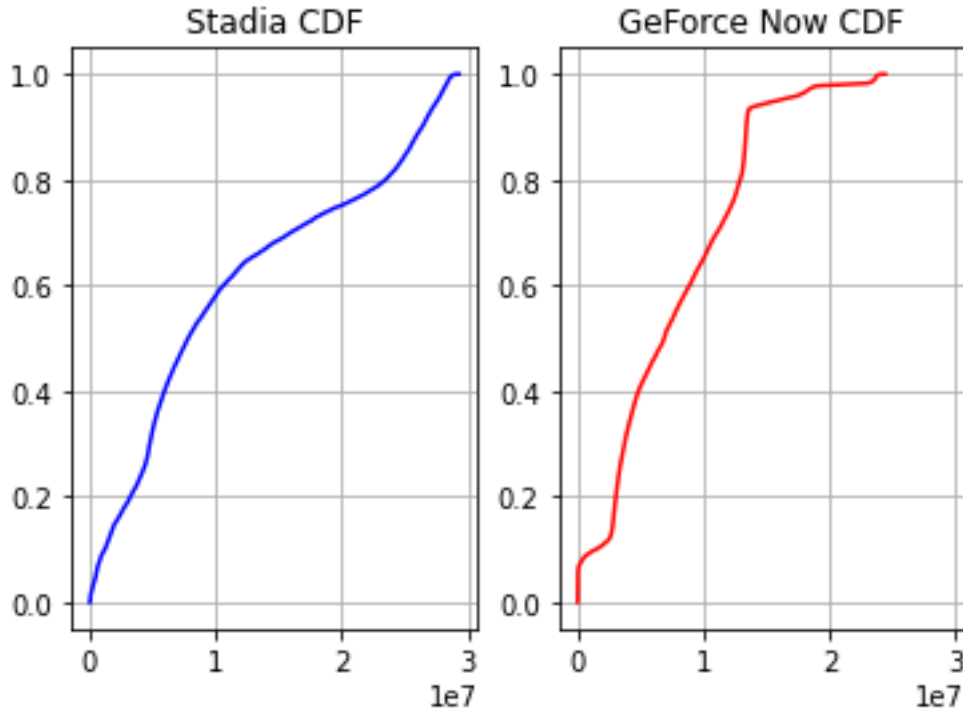


Figure 19 : Bitrate CDF comparison plot

### 4.2.3 Packet length distribution analysis

Next step we take a look into the distribution of the packet length. On Figure 20 we can see the CDF of the packets from both platforms and they are quite similar. Both have a small percentage, around 18% of samples which weighed less than 300 bytes for Stadia and 250 bytes for GeForce Now. The function then grows almost insignificantly, especially the GeForce Now one that almost seems to be parallel to the x axis; this means that almost no packets captured had lengths between 300 and 1200 for Stadia and between 250 and 1100 for GeForce Now. This is not very surprising, usually internet packets travel either with small payloads, usually for control and signaling purposes or with the maximum possible payload to try to achieve the best possible efficiency. The abrupt increase in the final parts of the graphs indicate that the majority of the packets were carrying over 1200 bytes on Stadia and 1100 bytes on GeForce Now.

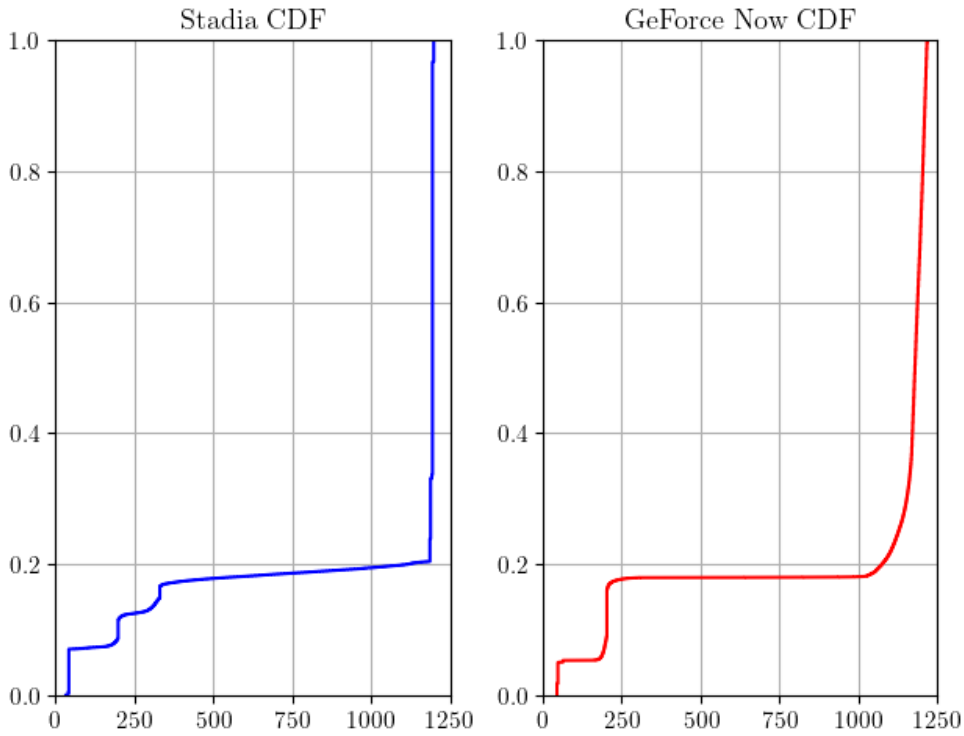


Figure 20 : Packet length CDF graphs

#### 4.2.4 Video Frames per second distribution analysis

The video frames refer to the number of individual images that are shown per unit of time, typically per second, on a display, it is measured in frames per second (fps) or Hertz (Hz). They are standardized by the Society of Motion Picture and Television Editors (SMPT). The higher the frame rate, the better quality of the video.

The frames are differentiated on the captured traffic by means of a marker field on the RTP header, highlighted in Figure 21. The use of this bit may vary from application to application, depending on the RTP profile being used, but in this case, it signals the beginning of a new video frame.



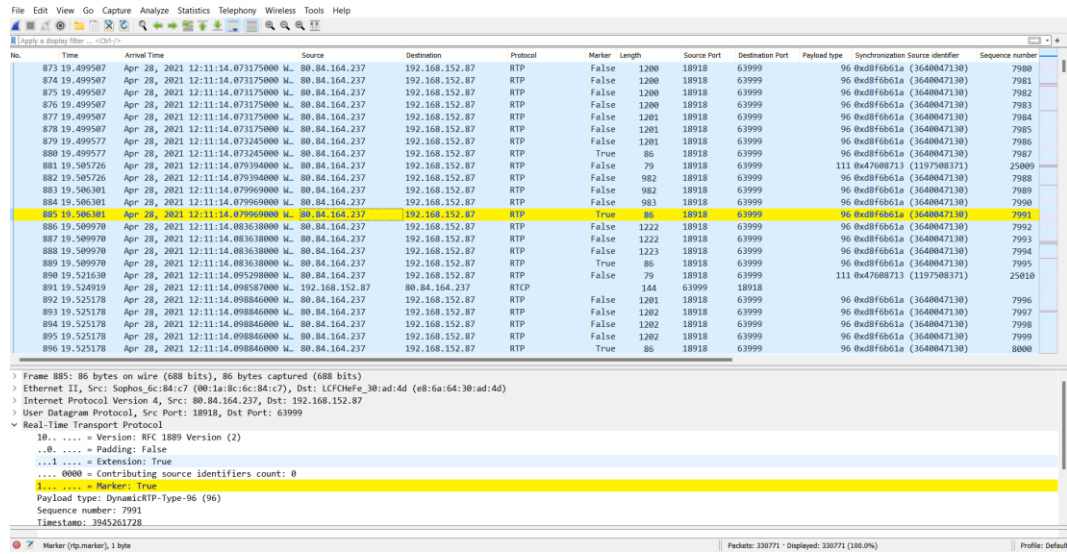


Figure 21 : RTP marker for frames

The behavior of the fps was quite constant on all of the captures. The mean captures fps is approximately 60 in both platforms, which is the standard fps for High Definition (HD) video, with very low standard deviation, 2.64 Stadia and 1 GeForce Now. Therefore, the CDF graph on Figure 22 reflects these statistics, the probability of having a fps value less than 57 is practically zero and it explodes when it gets closer to 60. Figures 22 and 23 have some examples of the fps behavior vs time while playing,

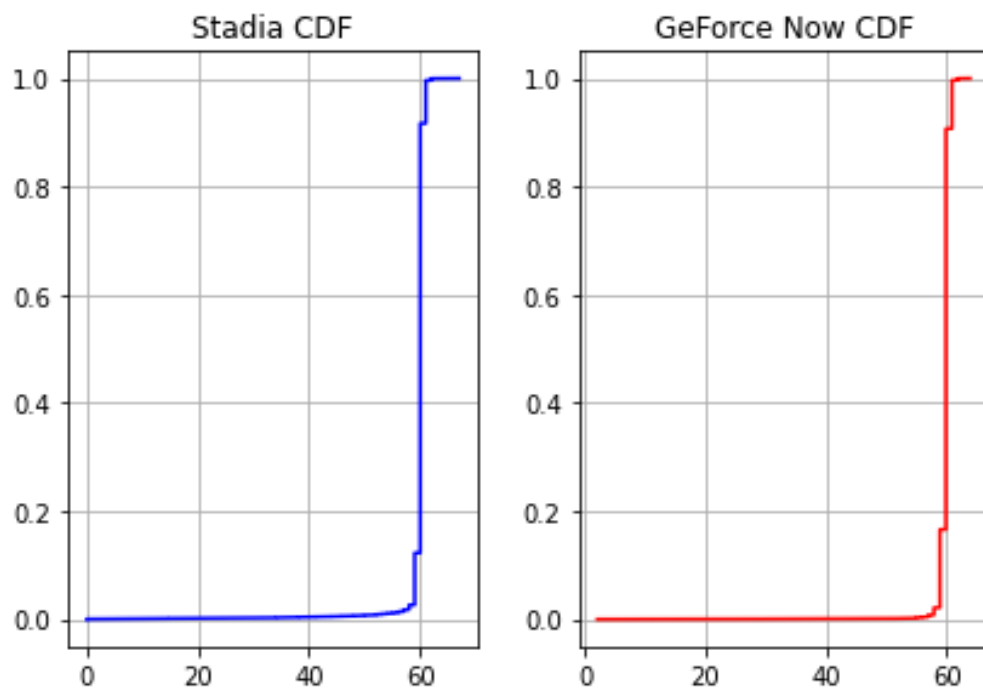


Figure 22 : FPS CDR graphs

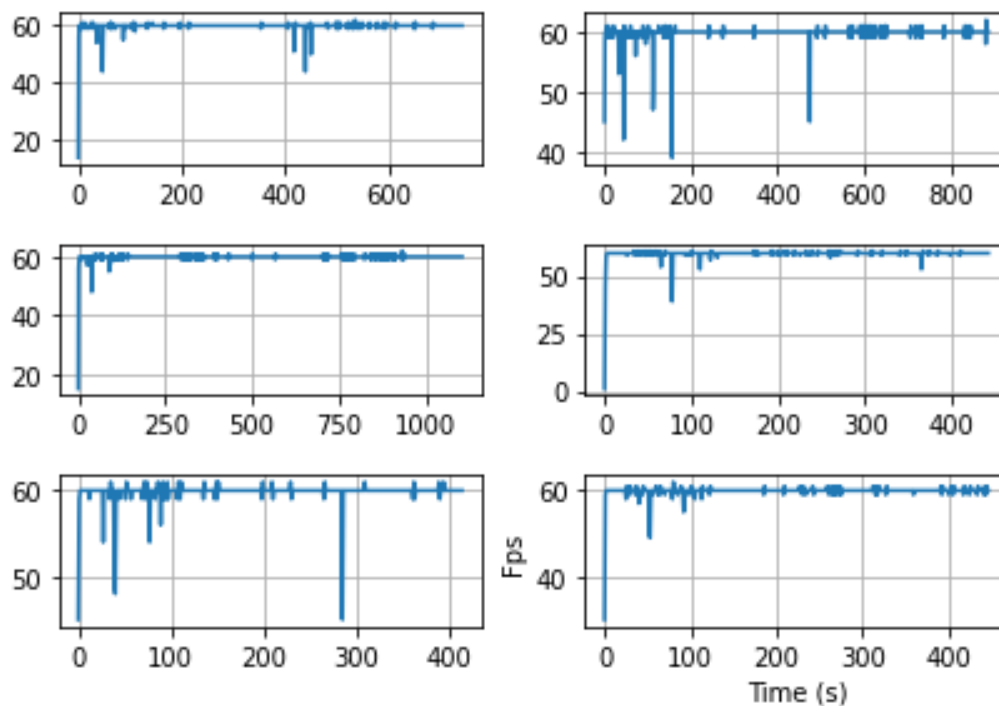


Figure 23: Fps graphs from Stadia

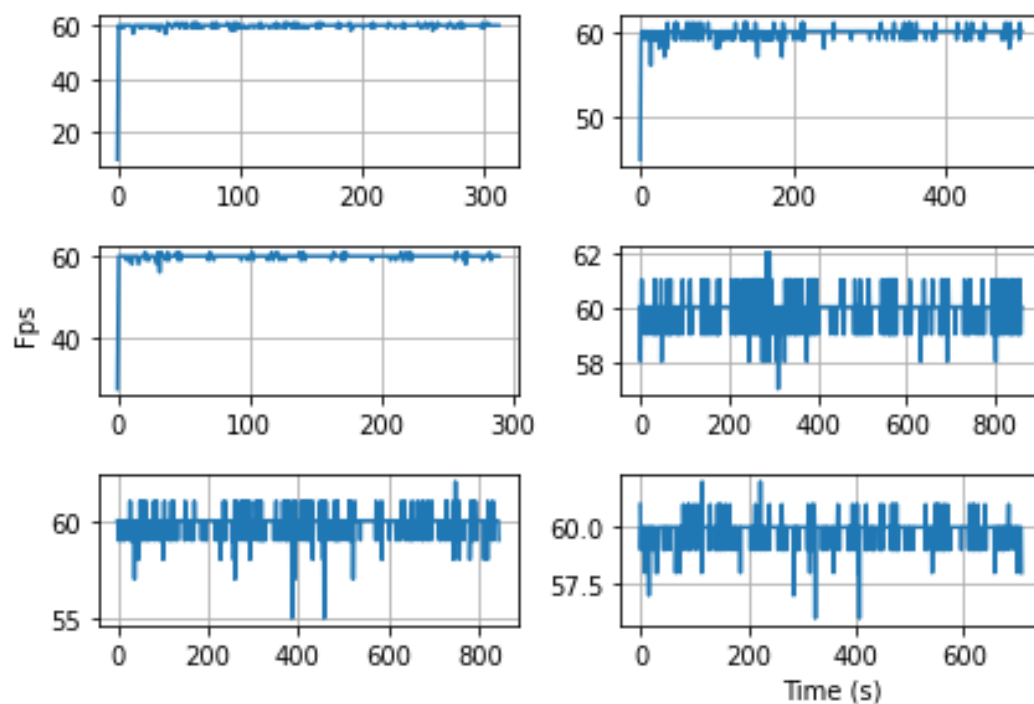


Figure 24: Fps graphs from GeForce Now

Given these characteristics we can conclude that the fps is not a good feature to feed to a machine learning algorithm because it does not provide any kind of information about the state of the game, it is always constant no matter what.

#### **4.2.5 DTLS packets distribution analysis**

As explained in chapter 2.2, DTLS is a protocol design to provide the security functionalities of TLS to applications working with unreliable traffic. It is used by Stadia and GeForce Now to exchange information securely, and its main usage is to transmit the user's keyboard and mouse movement commands to the server.

Figure 25 shows the CDF of the DTLS packets. We can observe that the functions are not very similar as on previous cases. The one from Stadia has a very small probability of having a sample lesser than 80 DTLS packets sent, then abruptly increases reaching almost 100% probability of having less than 200 packets, but there are still a low number of packets that can reach until over 500 packets. On the other hand, GeForce Now has almost a 0.38 probability of having 0 or a very small number of packets sent, then the graph increases, with a lower sharpness as the Stadia one, reaching also almost 1% probability at 200 packets. Table 5 shows the statistical differences between the behavior of DTLS packets.

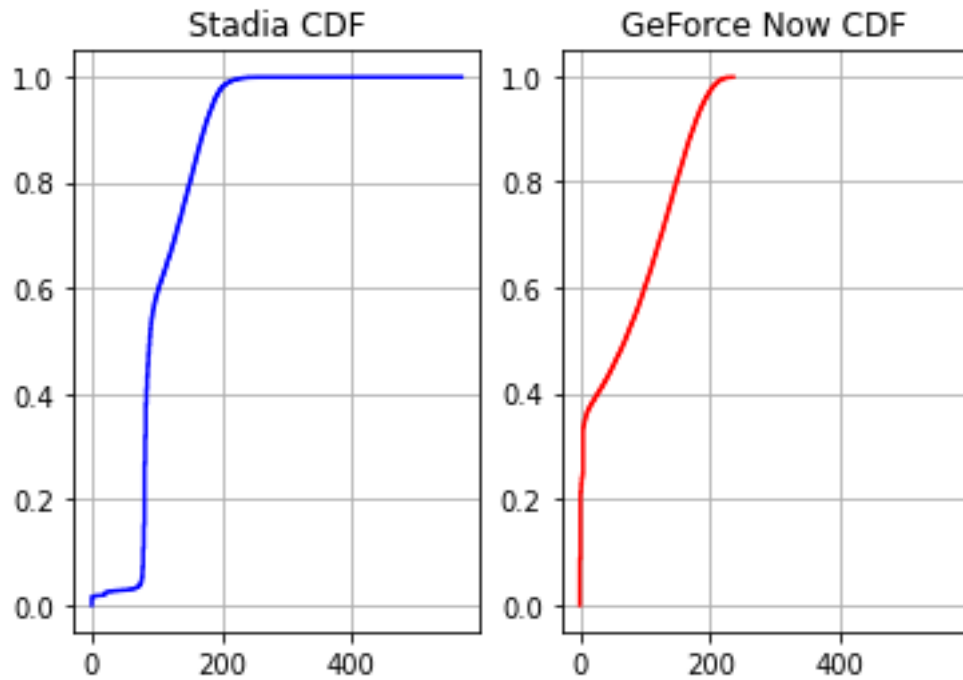


Figure 25 : DTLS packets CDF distributions

Table 6: DTLS packets statistics

Platform	Mean	Standard dev.	Min	Max
<b>Stadia</b>	109.05	41.86	0	569
<b>GeForce Now</b>	75.14	69.31	0	236

The difference in behavior can also be seen on Figures 26 and 27. The Stadia plots start from 0 and then increase with time to either stabilize around 90 or have a more bursty behavior around 200. On the other side, GeForce Now plots, do have a similar behavior on the bursty periods reaching also values around 100, the big discrepancy is that in the other periods almost no packets are sent, they values stay constantly at around 0.

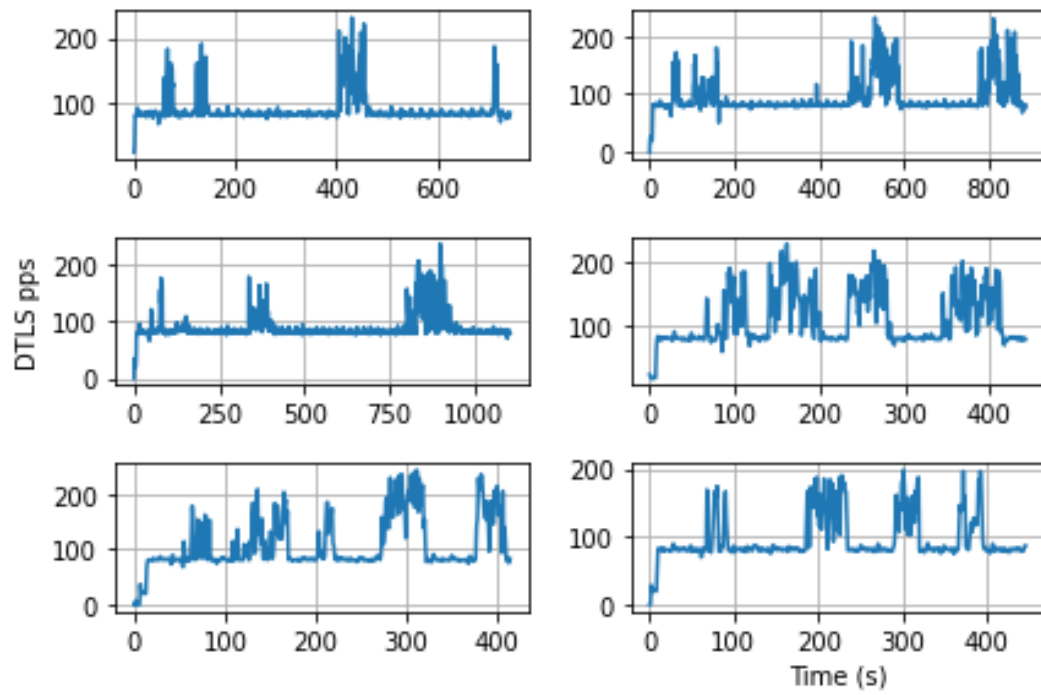


Figure 26: DTLS packets sent vs time Stadia examples

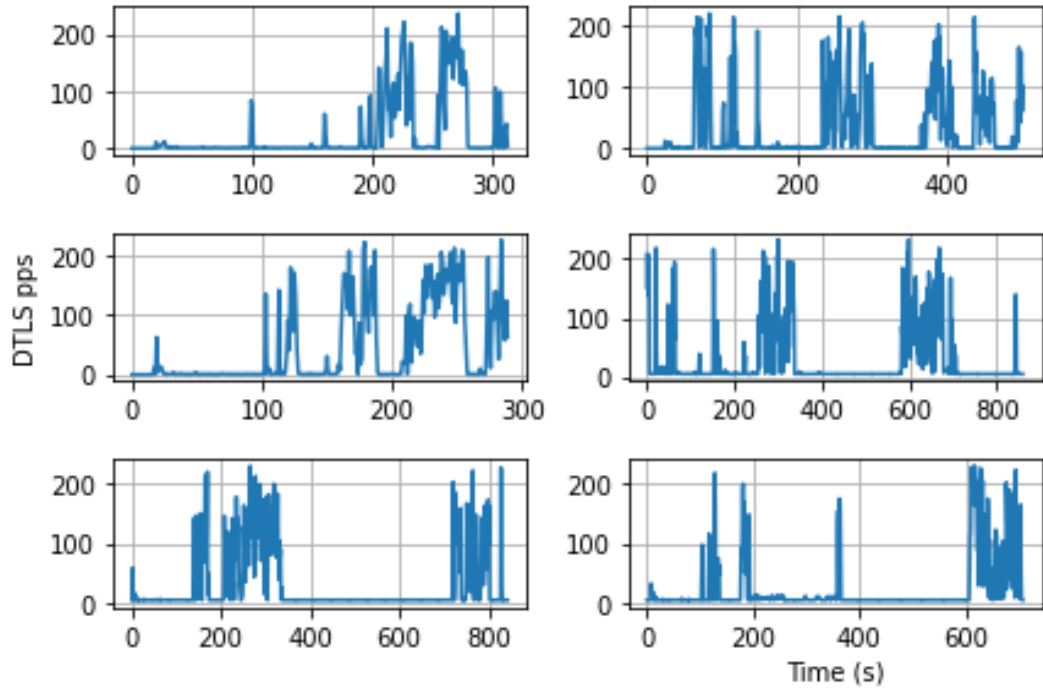


Figure 27 : DTLs packets sent vs time GeForce examples

The periods where the bursty behavior is seen, correspond on their majority to either the game's loading phase or playing; the constant periods are usually the times where the game was paused. Therefore, the DTLs behavior will be use as part of the machine learning dataset since is a good indicator of the state of the game.

#### 4.2.6 Packets Inter-arrival distribution analysis

The next aspect to be analyzed is the packets inter-arrival times. On Figure 28 the CDF functions were plotted, both shoot up rapidly; just a very small number of samples are above the 30ms mark. The maximum value for Stadia inter-arrivals was of 760ms and for GeForce Now 906ms. It was expected that the Stadia packets had lower inter-arrivals given that the bitrate was on average almost 4Mbps higher, consequently, the packets had to arrive at a quicker rate and with higher payloads.

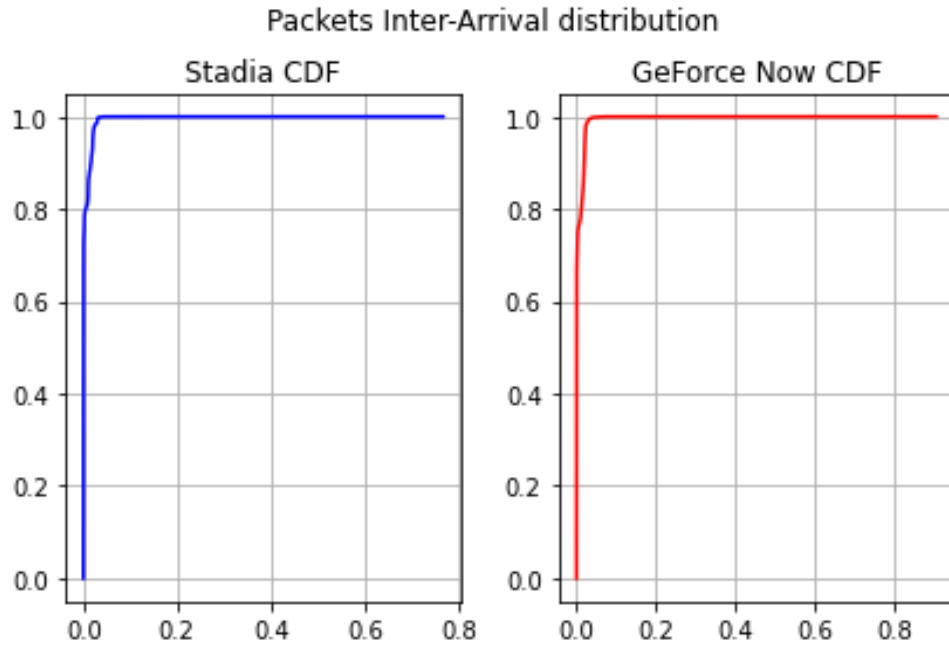


Figure 28: Packets inter-arrivals CD

Table 7: Packets inter-arrivals statistics

Platform	Mean	Standard dev.	Min	Max
<b>Stadia</b>	3.42ms	7.11ms	0	760ms
<b>GeForce Now</b>	4.57ms	8.3ms	0	830ms

Due to the huge difference between the maximum values and the mean of the densities, we cannot observe with details the growth of the CDF on Figure 28, So another plot was done, taking into consideration the samples with up to 30ms values. The samples clipped represent a 0.72% for Stadia and 0.84% for GeForce Now. The new functions on Figure 29, show with more details the inter-arrival distribution. Most of the values are very close to zero in both cases, over 70% Stadia and over 60% GeForce, the rest of the points are located below the 30ms established mark.



Stadia packets show two small peaks on that interval, on 10 and 20 milliseconds, while the GeForce packets are more equally distributed on that segment with a small peak on the 21 milliseconds value.

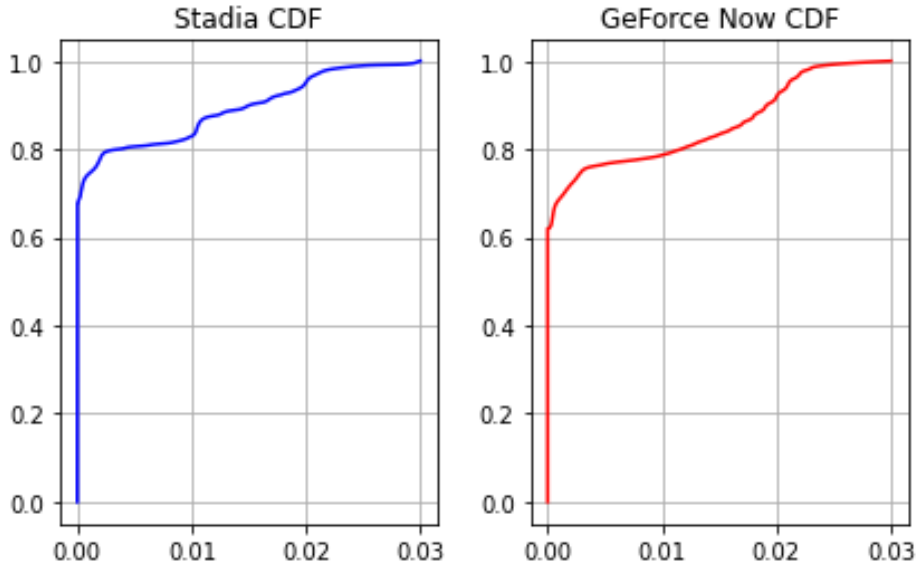


Figure 29 : Packets inter-arrival CDF clipped

#### 4.2.7 Packets per second received

The final characteristic taken into consideration is the distribution of the packets received from the video stream flow. On Figure 30 the CDFs were plotted, it is noticeable that the functions are very similar to those of Figure 19, which makes sense because there are very closely related variables, the bitrate depends on the number of packets per seconds and the length of those packets, as explained on Chapter 4.2.3 the packet lengths don't vary much, it can be considered almost as a constant, therefore the dependence between the pps and bps is direct.

The mean of the packets per second from Stadia was 1215.68 and 869.64 from GeForce Now, with a standard deviation of 965.66 and 540.15 respectively. The variables values range between 0 and 3097 on the graph on the left and from 12 to 2605 on the right, which has the particularity that over 93% of the time the packets received were less than 1500.

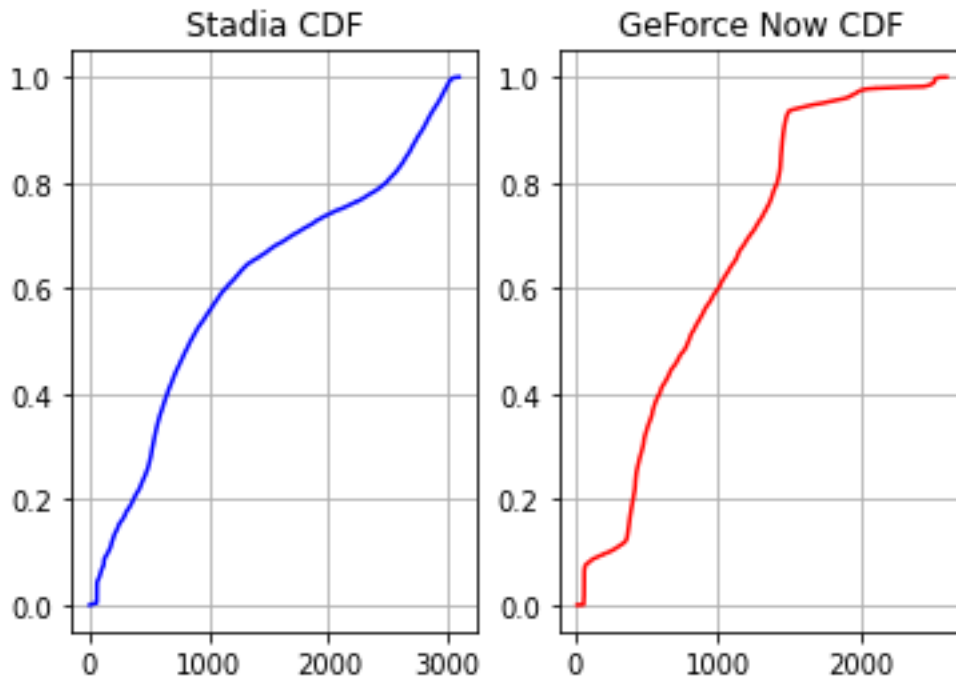


Figure 30 : Packet received CDF

#### 4.2.8 Stadia vs GeForce comparison on Destiny 2 performance

As showed previously on Tables 2 and 3, the network parameters vary from game to game, which is not surprising, there are many types of games with very different network demands. But, since there was a game that was available to play on both platforms, Destiny 2, it is interesting to analyze how it performed on each of them. That was the reason why it was also the game from which the greatest number of captures were taken. We can see a direct comparison of some general parameters on Table 7.

Table 8: Destiny 2 general characteristics

Destiny 2 GeForce Now

Destiny 2 Stadia

<i>Average Bits per second</i>	9.25 Mbps	15.7 Mbps
<i>Maximum bitrate</i>	24.4Mbps	29.3Mbps
<i>Bps standard deviation</i>	5.46 Mb	10.215 Mb
<i>Average packets per second received</i>	1033.1	1674.82
<i>Maximum packets per second received</i>	2605	3097
<i>Packets per second received standard deviation</i>	566.82	1067.99
<i>Average frames per second</i>	59.86	59.78
<i>Average packets inter arrivals</i>	4.36ms	3.89ms
<i>Maximum packets inter arrivals</i>	762ms	767 ms
<i>Packets inter arrivals standard deviation</i>	7.93ms	8.19ms
<i>Average packet length</i>	989.31 bytes	994.79 bytes
<i>Maximum packet length</i>	1218 bytes	1198 bytes

<i>Packet length standard deviation</i>	405.21	397.87
<i>Average packets per second sent</i>	56	112.65
<i>Packets per second sent maximum</i>	236	244
<i>Packets per second sent standard deviation</i>	64.14	43.465
<i>Resolution</i>	1366x768 and 1280x720	1920x1080 and 1280x720
<i>Captures</i>	60	45
<i>Seconds</i>	16541	11039

From the data gathered on Table 7, we can see the difference between the traffic from the platforms, Stadia has an average bitrate that is on average 6Mbps higher than GeForce Now, which makes sense because most of the captured data from Stadia had a video resolution of 1920x1080, 43 out of the 45 captures, and only 2 had 1280x720, while the GeForce Now captures 24 were transmitted with 1280x720 and 36 with 1366x768 pixels.

The average frames per second are very similar, always approximately 60 like seen on Chapter 4.2.4. The average packet length is also approximately the same in average, maximum and standard deviation. The interarrivals times are less for Stadia as expected, there is a big difference between the packets per second received so the interarrivals had to behave on this way. Lastly, the packets per second sent, DTLS packets, were significantly less on the GeForce Now captures, with a very high standard variation as depicted on Figure 25.

## Chapter 5

# Game Stage Classifications

### 5.1 Dataset construction

After the data characterization, the next step on the thesis was the creation of a dataset with all of the previous features to train a Machine Learning algorithm able to predict the different stages in the game. Because it was not possible to automatize the building of the target, not all the captures were taken, there were 33 captures selected from both GeForce and Stadia, they sum to a total of 12297 samples. Table 9 groups the number of samples per class, the 0+2 class refers to the case in which those classes were merged.

**Table 9 : Dataset number of classes per samples**

<i>Classes</i>	<i>Samples</i>
<i>0</i>	2837
<i>1</i>	4972
<i>2</i>	4488
<i>0+2</i>	7325
<i>Total</i>	12297

The features used for the first two experiments were a total of 21, each of which will be explained next and are shown on Figure 31.

	target	pps	larr_mean	larr_std	larr_max	pkt_len_mean	pkt_len_std	pkt_len_min	pkt_len_max	dtls	bps	filename	bps_previous_1s	bps_previous_2s
0	0.0	159	0.006220	0.009983	0.030160	968.018868	403.912306	83	1198	26.0	153915	Test_D2_1_1_	0.0	0.0
1	0.0	99	0.010050	0.009260	0.046525	645.767677	507.493640	101	1198	22.0	63931	Test_D2_1_1_	153915.0	0.0
2	0.0	300	0.003293	0.006648	0.037319	1061.183333	280.660073	124	1198	20.0	318355	Test_D2_1_1_	63931.0	217846.0
3	0.0	266	0.003712	0.006979	0.035942	1039.255639	320.049683	68	1198	21.0	276442	Test_D2_1_1_	318355.0	382286.0
4	0.0	234	0.004226	0.007257	0.022970	1024.350427	308.313702	136	1198	20.0	239698	Test_D2_1_1_	276442.0	594797.0

Figure 31 : Dataset samples

bps_previous_3s	inter_arrival_count_p1s	inter_arrival_count_p2s	inter_arrival_count_p3s	dtls_p1s	dtls_p2s	dtls_p3s
0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
0.0	0.006220	0.000000	0.000000	0.000000	26.0	0.0
0.0	0.010050	0.006220	0.000000	0.000000	22.0	26.0
536201.0	0.003293	0.010050	0.006220	0.006220	20.0	22.0
658728.0	0.003712	0.003293	0.010050	0.010050	21.0	20.0

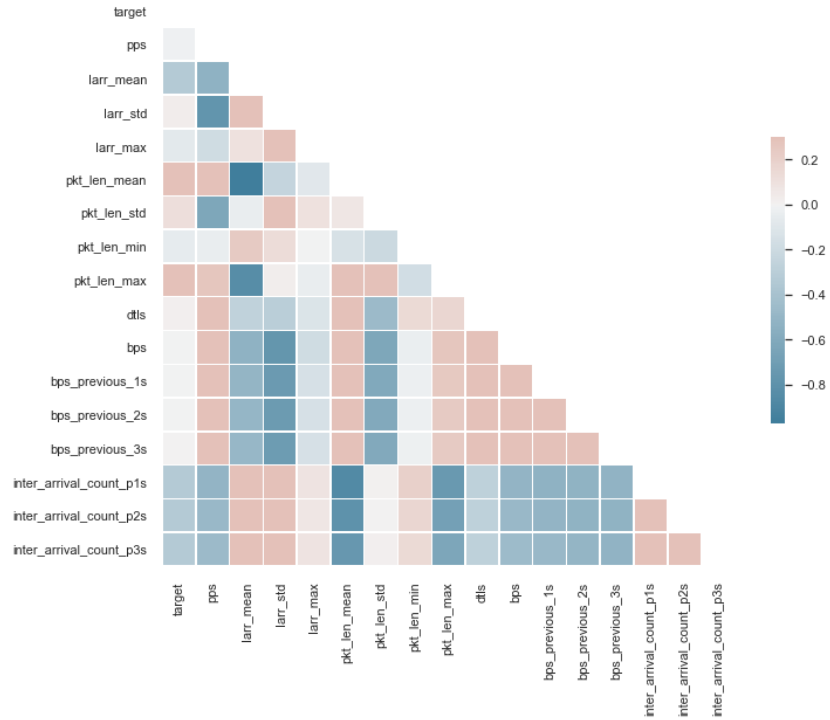
Figure 32: Dataset samples continuation

Dataset fields:

1. Target: As its name indicates, this field contains the different in game stages which need to be inferred by the machine learning classifiers using the information from the rest of the fields.
2. Pps: Packets per second received from the game video stream, collected from the Wireshark captures. It was selected because it gives an idea of the amount of traffic that was received on any particular second, which should be very useful for predicting the classes.
3. larr\_mean: Packet's inter-arrival mean per second; it expresses the average of the difference between the arrival times of the packets on each second. This category is very important from a networking point of view, it shows on average how stable or bursty was the packet arrival which could be a good distinction between classes.
4. larr\_std: Packet's inter-arrival standard deviation, similar to the previous one, but on this case is the standard deviation what was taken, which gives an expression of the amount of variation of the set of inter-arrivals.
5. larr\_max: Maximum value of the packet's inter-arrivals on each second.
6. Pkt\_len\_mean: Mean of the UDP length of the packets, it is a key factor it varies depending on whether the packets were control or user data traffic, and if it was user data traffic in our case for RTP it shows the amount of bytes carried at any given time.

7. Pkt\_len\_std: Packet length standard deviation, more statistical measures of the samples.
8. Pkt\_len\_min: Minimum packet length of the samples
9. Pkt\_len\_max: Maximum packet length of the samples
10. Dtls: Number of DTLS packets sent, provides details about the user's keyboard and mouse command interacting with the game. It is a major factor for this analysis, while the user is playing, he would be need to be sending commands all of the time, while on the other phases of the game the number of commands should be much lower, as seen on Figures 26 and 27.
11. Bps: Bytes per second received, as shown on Figures 16,17 and 18 there is a big difference on the bitrate depending on the state of the game.
12. Filename: Is used only for splitting the dataset into training and validation, then is dropped.
13. Bps\_previous\_1s: Bytes per second received on the previous second.
14. Bps\_previous\_2s: Bytes per second received on the previous 2 seconds added up.
15. Bps\_previous\_3s: Bytes per second received on the previous 3 seconds added up.
16. Inter-arrival\_count\_p1s: Inter-arrival mean of the packets on the previous second.
17. Inter-arrival\_count\_p2s: Inter-arrival mean of the packets two seconds before.
18. Inter-arrival\_count\_p3s: Inter-arrival mean of the packets two three before.
19. Dtls\_p1s: DTLS packets sent on the previous second.
20. Dtls\_p2s: DTLS packets sent on the previous two seconds.
21. Dtls\_p3s: DTLS packets sent on the previous three seconds.

The fields between 13 and 21 were added considering the behavior observed on Chapter 4.2, it was concluded that information about the previous seconds would be key to improve the predictions of the classifiers.



**Figure 33: Dataset cross-correlation matrix**

On Figure 33, the cross-correlation matrix of the dataset was plotted. The column of interest for us is the first one, that shows the correlation between the target column and the rest of the fields. Surprisingly, fields like the ones related with the bps and the DTLS packets do not show the high correlation that was expected. On the other hand, we can see that the fields related with the inter-arrival times and the packets lengths have a higher correlation as predicted.

## 5.2 Three classes classification

The first experiment performed was taking into consideration the three distinctive game stages as classes, loading up phase, gaming and pause.



### 5.2.1 Random Forest Results

The first classifier used was a random forest, also from the sklearn library, the classification report can be seen on Table 10 and the confusion matrix on Table 11.

**Table 10: Random Forest classification report, 3 classes**

<i>Classes</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>	<i>Support</i>
0	0.49	0.74	0.59	1276
1	0.87	0.75	0.80	3237
2	0.53	0.45	0.49	1440
<i>Accuracy</i>		0.689		5953

**Table 11: Random Forest classifier confusion matrix, 3 classes**

Actual Class	Predicted Class		
	0	1	2
0	1023	97	156
1	380	2426	431
2	508	278	654

As we can see the results from the random forest classifier were not very good. class 0, had an acceptable recall of 80%, but a very low precision of only 54%, the classifier is not distinguishing correctly between class 0 and 2, as we can see on the confusion matrix on Table 11, over 500 values of class 2 were predicted as class 0, which results on the low precision of class 0, on the low recall of class 2 and therefore in a low f1 score for both classes. Finally, class 1 was predicted much better than the others with a 0.80 f1 thanks to a high precision of 0.87, and an acceptable recall of 0.75.

### 5.2.2 Decision Tree

The second classifier used was decision tree. The classification report can be seen on Table 12 and the confusion matrix on Table 13.

**Table 12: Decision tree classification report, 3 classes**

<i>Classes</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>	<i>Support</i>
0	0.53	0.80	0.64	1276
1	0.87	0.72	0.79	3237
2	0.48	0.45	0.46	1440
<i>Accuracy</i>		0.67		5953

**Table 13: Decision tree classifier confusion matrix, 3 classes**

Actual Class	Predicted Class		
	0	1	2
0	1016	97	163
1	352	2344	541
2	532	262	646

The Decision Tree performed very similar to the Random Forest, it also predicts over 37% of class 2 samples as class 0, leading to a very low precision of both and very low recall of the former. Also, here class 1 was predicted with much higher efficiency 0.79 ifl score is a good result, again very high precision and lower recall.

### 5.2.3 K-nearest neighbor

The first classifier used was K-nearest neighbor. The classification report can be seen on Table 14 and the confusion matrix on Table 15.

**Table 14: K-nearest neighbor classification report, 3 classes**

Classes	Precision	Recall	F1-score	Support
0	0.56	0.70	0.62	1276
1	0.86	0.81	0.83	3237
2	0.54	0.49	0.52	1440
Accuracy	0.70			5953

**Table 15: K-nearest neighbor classifier confusion matrix, 3 classes**

Actual Class	Predicted Class		
	0	1	2
0	892	105	279
1	296	2607	334
2	392	336	712

The final classifier, K-nearest neighbors obtained the best results out of the three on classes one and two, class two improvement was very low, and it was still not well predicted, on the contrary class 1 showed an enhancement, the recall value was 0.81, only classifier one who managed to surpass the 0.8 mark for this value, added to a 0.86 precision, the f1 score rose to 0.83. On the other hand, class 0 disimproved with respect to the previous results, the recall declined 0.1, more of its samples were predicted as class 2.

### 5.2.4 Results discussion

The models trained with the three classifiers did not perform as well as expected, except for class 1. The bad results can be for several reasons. First of all, there is the problem of defining the exact timing of the transition between classes, since the modularity of the dataset was defined on a second-to-second basis, the exact second on which the transition occurs can be hard to predict, depending on which millisecond the change happened; furthermore, the traffic characteristics may not abruptly change, but take a couple of seconds to adjust which difficult the prediction of those particular sample. Secondly, the target column was made using the video recordings manually, there is a possibility of errors made during this process, this aspect also affected the size of the dataset, having more samples could improve greatly the accuracy of the estimations, that way the estimators would train with more samples of classes 0 and 2 and they would probably be more effective distinguishing between them. Finally, expanding the fields of the dataset could also be an interesting approach.

## 5.3 Two classes classification

Due to the problems seen on Chapter 5.2 with classes 0 and 2, the next experiment was done merging the two of them. Being QoE the final goal of the classification, the most important result is having a model that can tell if the users are gaming or not, which is the key moment to provide a higher QoE.

### 5.3.1 Random Forest Results

The same experiments were ran as on Chapter 5.2 but this time with only two classes, gaming and not gaming, starting with the Random Forest Classifier the results can be seen on Tables 16 and 17.

**Table 16 : Random Forest classification report with two classes**

<i>Classes</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>	<i>Support</i>
<i>0</i>	0.75	0.86	0.80	2716
<i>1</i>	0.87	0.76	0.81	3237
<i>Accuracy</i>		0.808		5953

**Table 17 : Random Forest confusion matrix with two classes**

Actual Class	Predicted Class	
	0	1
0	2340	376
1	763	2474

Compared with the results of Chapter 5.3.1, there is a big improvement in the accuracy of the classifier, specially, class 0, which now is the result of merging the previous classes 0 and 2, obtained a 0.80 f1-score, with 0.75 precision and 0.86 recall. Class one behaved similarly, which was expected because no changes were apply with respect to last chapter's experiment. Consequently, the overall accuracy of the classifier also increased significantly, from 0.689 to 0.808.

### 5.3.2 Decision Tree Results

The results of the Decision Tree classifier can be seen on Tables 18 and 19.

**Table 18: Decision Tree classification report with two classes**

<i>Classes</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>	<i>Support</i>
<i>0</i>	0.74	0.85	0.79	2716
<i>1</i>	0.86	0.74	0.80	3237
<i>Accuracy</i>		0.79		5953

**Table 19 : Decision Tree confusion matrix with two classes**

Actual Class	Predicted Class	
	0	1
0	2156	295
1	418	752

Like on the previous classifier, the decision tree predictions improved for class 0, reaching a 0.79 f1 score, class 1 also increased in 0.01, hence the accuracy grew to 0.79. However, these outcomes are slightly inferior to the ones obtained with the random forest classifier.

### 5.3.3 K-nearest neighbor

. The results of the K-nearest neighbor classifier can be seen on Tables 20 and 21.

**Table 20: K-nearest neighbors classification report with two classes**

<i>Classes</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>	<i>Support</i>
0	0.77	0.86	0.81	2716
1	0.87	0.78	0.82	3237

**Table 21 : K-nearest neighbors confusion matrix with two classes**

Actual Class	Predicted Class	
	0	1
0	2210	241
1	414	756

Finally, the classifier who performed the best out of the three was the K-nearest neighbors, with a f1-scores of 0.81 and 0.82 for classes 0 and 1 respectively. Like in the previous two algorithms, the union of classes 0 and 1 improved in all aspects the performance of the new class 0 and in the overall accuracy of the model.

### 5.3.4 Results discussion

The results of the three classifiers using two classes showed an improvement on class 0, by merging the previous former classes 0 and 2, the new class 0 was able to get a bigger amount of samples which allowed the classifiers to infer much better the new class, the scores for class 1 remained very similar to the ones of Chapter 5.2, slightly over 0.8 with the best result being obtained by the K-nearest neighbor classifier. Most of the limitations mentioned in Chapter 5.2.4 remain, problems with the timing and possible human errors are possible, increasing the dataset would probably provide better results as well.

## 5.4 Retina dataset

In order to try to improve the results obtained so far, it was thought to add more fields to the dataset that have not been considered on the previous experiments and may help the classification algorithm to predict the classes with higher efficiency. For that reason, the next experiments were done using a dataset derived from the open-source software Retina.

Retina is an open-source command-line tool that produces rich and complex statistics from real-time communication (RTC) traffic. Starting from raw packet captures, it creates summaries of observed streams with flexible statistics and tracks the evolution of the stream over time. Retina is modular and highly configurable, providing the ability to configure output statistics, temporal resolution as well as many other parameters. Furthermore, if the packet captures are accompanied by application logs, it can reconcile the data and enrich its output with application and QoE- related statistics. (Perna, et al.)

Retina helps troubleshoot RTC applications and enables the use of Machine Learning models for traffic classification and Quality of Experience assessment. We believe Retina can be extremely useful for researchers studying RTC traffic and network professionals interested in effective traffic analysis. (Perna, et al.)

Retina is an easy-to-use command-line tool that extracts advanced network statistics for RTC sessions found in packet captures. It goes deeper than general tools in understanding RTC traffic. Starting from a capture, Retina searches for

RTC traffic, identifies streams and outputs more than 130 statistics on packet characteristics. (Perna, et al.)

The software can be found on the following GitHub link: <https://github.com/GianlucaPoliTo/Retina>.

The same captures selected in the Chapters 5.2 and 5.3 were used as the input of Retina algorithm, which produced a dataset from which 84 fields were selected, moreover, the information regarding the DTLS packets sent and obviously the target column, On Figure 34, the list of the columns of the dataset can be seen.

```
In [21]: 1 dataset.columns

Out[21]: Index(['interarrival_std', 'interarrival_mean', 'interarrival_min',
'interarrival_max', 'interarrival_count', 'interarrival_kurtosis',
'interarrival_skew', 'interarrival_moment3', 'interarrival_moment4',
'interarrival_max_min_diff', 'interarrival_max_min_R',
'interarrival_min_max_R', 'interarrival_len_unique_percent',
'interarrival_max_value_count_percent', 'kbps', 'len_udp_std',
'len_udp_mean', 'len_udp_min', 'len_udp_max', 'num_packets',
'len_udp_kurtosis', 'len_udp_skew', 'len_udp_moment3',
'len_udp_moment4', 'len_udp_max_min_diff', 'len_udp_max_min_R',
'len_udp_min_max_R', 'len_udp_len_unique_percent',
'len_udp_max_value_count_percent', 'interlength_udp_std',
'interlength_udp_mean', 'interlength_udp_min', 'interlength_udp_max',
'interlength_udp_count', 'interlength_udp_kurtosis',
'interlength_udp_skew', 'interlength_udp_moment3',
'interlength_udp_moment4', 'interlength_udp_max_min_diff',
'interlength_udp_max_min_R', 'interlength_udp_min_max_R',
'interlength_udp_len_unique_percent',
'interlength_udp_max_value_count_percent',
'rtp_inter_timestamp_num_zeros', 'rtp_inter_timestamp_std',
'rtp_inter_timestamp_mean', 'rtp_interarrival_min',
'rtp_interarrival_max', 'rtp_interarrival_count',
'rtp_interarrival_kurtosis', 'rtp_interarrival_skew',
'rtp_interarrival_moment3', 'rtp_interarrival_moment4',
'rtp_interarrival_max_min_diff', 'rtp_interarrival_max_min_R',
'rtp_interarrival_min_max_R', 'rtp_interarrival_len_unique_percent',
'rtp_interarrival_max_value_count_percent', 'rtp_marker_sum_check',
'rtp_seq_num_packet_loss', 'inter_time_sequence_std',
'inter_time_sequence_mean', 'inter_time_sequence_min',
'inter_time_sequence_max', 'inter_time_sequence_count',
'inter_time_sequence_kurtosis', 'inter_time_sequence_skew',
'inter_time_sequence_moment3', 'inter_time_sequence_moment4',
'inter_time_sequence_max_min_diff', 'inter_time_sequence_max_min_R',
'inter_time_sequence_min_max_R',
'inter_time_sequence_len_unique_percent',
'inter_time_sequence_max_value_count_percent', 'label', 'pcap',
'target', 'dtls', 'kbps_previous_1s', 'kbps_previous_2s',
'kbps_previous_3s', 'kbps_previous_4s', 'kbps_previous_5s',
'inter_arrival_count_p1s', 'inter_arrival_count_p2s',
'inter_arrival_count_p3s'],
dtype='object')
```

Figure 34 : Retina Dataset columns

As shown on Figure 33, the Retina datasets takes gives us a deeper statistical characterization of the captures derived from the general traffic features analyzed on Chapter 4, such as inter-arrivals, UDP packets length and bitrate.

#### 5.4.1 Random Forest Results with 3 classes

Tables 22 and 23 gather the results obtained using the random forest classifier with the Retina database and 3 classes target.



**Table and 22: Random Forest classification report, Retina dataset and 3 classes**

<i>Classes</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>	<i>Support</i>
0	0.48	0.77	0.59	1190
1	0.87	0.70	0.77	3186
2	0.51	0.47	0.49	1430
<i>Accuracy</i>		0.657		5806

**Table 23: Random Forest classifier confusion matrix retina dataset,3 classes**

Actual Class	Predicted Class		
	0	1	2
0	921	87	182
1	505	2226	455
2	503	259	668

The results obtained with this database were slightly inferior from the ones obtained on Chapter 5.2.1. Once again, classes 0 and 2 have an f1 score lower than 60, with a high recall on class 0 but extremely low precision and the other way around for class 2. Class 1 was the one predicted with a better overall performance with a 0.77 f1 score.

### 5.4.2 Decision Tree Results with 3 classes

Tables 24 and 25 collect the results obtained using the decision tree classifier with the Retina database and 3 classes target.

**Table 24: Decision Tree classification report, Retina dataset and 3 classes**

<i>Classes</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>	<i>Support</i>
0	0.47	0.74	0.58	1190
1	0.83	0.67	0.74	3186
2	0.39	0.38	0.38	1430
<i>Accuracy</i>		0.611		5806

**Table 25: Decision Tree classifier confusion matrix Retina dataset and 3 classes**

Actual Class	Predicted Class		
	0	1	2
0	876	113	201
1	398	2132	656
2	574	314	542

The decision tree output shows the same behavior as the previous classifier on the 3 classes, however it performed worse than the random forest with the same dataset, and worse than the decision tree with the previous dataset.

#### 5.4.4 K-nearest neighbor Results with 3 classes

Tables 26 and 27 show the results obtained when the k-nearest neighbor classifier with the Retina database and 3 classes target.

**Table 26 : K-nearest neighbor classification report, Retina dataset and 3 classes**

<i>Classes</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>	<i>Support</i>
0	0.44	0.44	0.44	1190

<i>1</i>	0.84	0.67	0.74	3186
<i>2</i>	0.41	0.60	0.49	1430
<i>Accuracy</i>	0.602			5806

**Table 27: K-nearest neighbor confusion matrix Retina dataset and 3 classes**

Actual Class	Predicted Class		
	0	1	2
0	519	137	534
1	360	2122	704
2	304	267	859

This classifier got the worse outcomes of the three, the accuracy was just 0.602, classes 0 and 2 f1 scores were below 0.5, the performance plummeted compared with the outcomes of the same classifier with the previous dataset.

#### 5.4.6 Random Forest Results with 2 classes

Like it was done in Chapter 5.3, the experiments will be repeated with a dataset that has classes 0 and 2 merge. The results were gathered on Tables 28 and 29.

**Table 28: Random Forest classification report, Retina dataset and 2 classes**

<i>Classes</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>	<i>Support</i>
<i>0</i>	0.72	0.87	0.79	2620
<i>1</i>	0.87	0.73	0.79	3186
<i>Accuracy</i>	0.792			5806

**Table 29: Random Forest classifier confusion matrix retina dataset, 2 classes**

Actual Class	Predicted Class	
	0	1
0	2282	338
1	869	2317

The results obtained with the merging of classes 0 and 2, like with the previous dataset, significantly improved with respect to the 3 classes experiment. Both classes obtained with this model 0.79 f1 score, and a 0.792 accuracy, which increased in 12%, however, with respect to the previous dataset, it decreased in 2%.

#### 5.4.7 Decision Tree Results with 2 classes

Tables 30 and 31 collect the results obtained using the decision tree classifier with the Retina database and 3 classes target.

**Table 30: Decision Tree classification report, Retina dataset and 3 classes**

<i>0</i>	0.68	0.86	0.76	2620
<i>1</i>	0.85	0.66	0.74	3186
<i>Accuracy</i>		0.75		5806

**Table 31: Decision Tree classifier confusion matrix Retina dataset and 3 classes**

Actual Class	Predicted Class	
	0	1

0	2260	360
1	1088	2098

The same behavior as on the Random Forest was observed, significant improvement with 2 classes but lower f1 scores and accuracy than the outcomes of the previous dataset. Despite the outcomes being quite similar, the Random Forest outperformed the Decision Tree in both classes.

### 5.4.9 K-nearest neighbor Results with 2 classes

Finally, the Retina dataset, with two classes on the target feature was used to train the K-nearest neighbor model and the results are shown on Tables 32 and 33.

**Table 32 : K-nearest neighbor classification report, Retina dataset and 3 classes**

0	0.66	0.92	0.77	2620
1	0.90	0.61	0.72	3186
Accuracy		0.747		5806

**Table 33: K-nearest neighbor confusion matrix Retina dataset and 3 classes**

Actual Class	Predicted Class	
	0	1
0	2405	215
1	1253	1933

The results of this classifier were resembling the ones from the Decision Tree and lesser than the ones from the Random Forest. It reached a very high precision on class one but low recall and conversely for class zero.

#### 5.4.10 Retina dataset result discussion

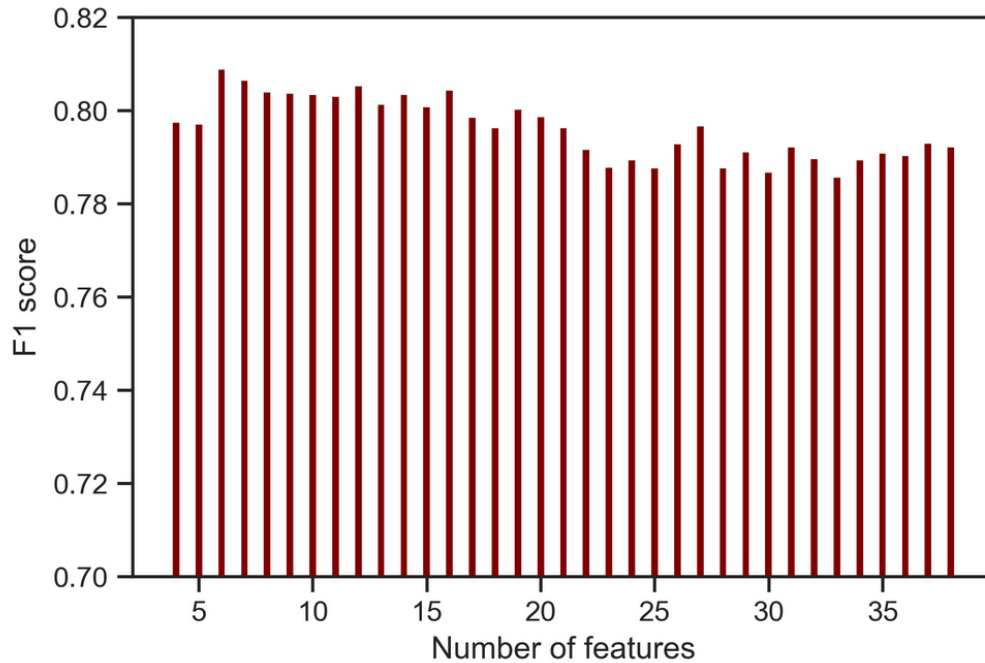
With this dataset we intended to provide more information(features), to the classifiers to see if we were able to rise the performance of the predictions. Unfortunately, the great number of features did not prove to be very useful, but, on the contrary, decreased the accuracy and f1 scores of the outcomes with both versions of the target feature. The most noticeable change was that the K-nearest neighbor classifier went from being the best with the first dataset to being the worse with the Retina dataset, this algorithm handles better lower dimensional datasets due to its simplicity relative to the others, and, unsurprisingly, the Random Forest produced the best results.

We can conclude from the previous experiments that on this case is better to use lower dimensional datasets, selecting the fields which seem more indicatives of the performance of the games as done on Chapter 5.1. To improve the outcomes is necessary to increase the samples used to feed the classification algorithms and to define a way of automatically create the target column from the video recordings.

#### 5.4.11 Feature Selection

The final experiment done was to do a feature selection procedure with the Retina dataset, with 2 classes on the target column, using the Recursive Features Selection(RFS) algorithm from the sklearn library, and as the model estimator we chose the Random Forest because it was the one who performed the best with the Retina dataset, the features\_to\_select parameter was set to one so on the output we could get an array with the ranking of importance of the columns inferred by the algorithm, the final parameter was the step, that corresponds to the number of features to be removed on each iteration, it was also set to 1, so we could have the highest precision as possible.

On Figure 35 we can observe the f1 scores of the Random Forest classifier, using a different number of features, starting by the ones if highest importance according to the RFS. The classifier fed with 6 features was the one that performed the best, reaching 0.8089.



**Figure 35: F1 scores versus number of features selected**

Using the previous results, we trained the Random Forest classifier once more, using the 6 most important features of the Retina dataset, with 2 classes on the target feature. The results of the predictions are shown on Tables 34 and 35.

**Table 34: Random Forest, 2 classes, 6 features classification report**

<i>0</i>	<i>0.75</i>	<i>0.87</i>	<i>0.81</i>	<i>2620</i>
<i>1</i>	0.88	0.76	0.82	3186
<i>Accuracy</i>	0.813			5806

**Table 35: Random Forest, 2 classes, 6 features confusion matrix**

Actual Class	Predicted Class	
	0	1

0	2201	329
1	755	2431

The outcomes acquired using the six most important features of the Retina dataset are almost identical to the ones obtained when using the first dataset. We were unable to obtain higher f1 scores than 0.82 for the gaming sessions and 0.81 for the not gaming periods. We can conclude that there is no need of increasing the number of features of the dataset for this task, the ones derived initially are good enough and produced the best results possible with the amount of data available.



# Conclusions

In conclusion, this thesis was done with two main objectives, first characterizing the network traffic of a relatively new service that, in our opinion, has the potential of revolutionizing the online gaming industry, one of the most profitable industries nowadays, that is cloud gaming. We selected the two biggest platforms to our knowledge, Stadia from Google and GeForce Now from NVIDIA and performed a data collection process, ending up with 196 captures, which were composed by the Wireshark pcap, the WebRTC JSON log and the video capture of the screen.

Then, after gathering all of the data, we analyzed the behavior of the most significant networking features for the video traffic, those being, packets per second received, frames per second, bits per second, packet's inter-arrival distribution, packet's length distribution, and packet's sent distribution, in our case DTLS packets. After analyzing the behaviors, we concluded that the characteristics of the traffic changed significantly depending on the stage on which the user was on the game. Taking that into consideration, we started with our second objective, which was creating a mechanism for providing different QoE classes using machine learning algorithms.

The machine learning algorithms chosen were Random Forest, Decision Tree and K-nearest neighbor. We used two different datasets to try to obtain the best results possible, one was created by us, using the information extracted for the data characterization, and the other one was obtained by using the open-source software Retina, and adding information to it. The last step of the dataset creation was the target column which was developed manually by watching the screen recordings of the games.

Using the three algorithms with both datasets we concluded that the best options to predict the classes was using only two classes, one for when the user is playing and the other one for the times in which he is not, pauses or loading up stage of the game, with the dataset created by us during this thesis and using the either the Random Forest or the K-nearest neighbor classifier, both of them scored 0.87 precision, 0.78 recall, 0.82 f1 score and an accuracy of 0.81 and using the Retina dataset with the 6 most significant features given as an output of the Recursive

Feature selection algorithm also produces an f1score of 0.81 for class 1 with a Random Forest classifier.

For future work, is necessary to develop a mechanism for creating the target column using the video recordings that way a bigger dataset can be created for fitting the algorithms with more samples and, on this way, the accuracy and f1-scores can rise even more.

# References

- Bonaccorso, G. (2017). *Machine learning algorithms*. Packt Publishing Ltd. .
- Bunton, D. (2002). Generic moves in PhD thesis introductions. In J. Flowerdew, *Academic discourse* (pp. 57-75). London: Pearson Education Limited.
- Carrascosa, M., & Bellalta, B. (2020). Cloud-gaming: Analysis of Google stadia traffic.
- Clement, J. (2021, June 4). *Statista*. Retrieved from Statista: <https://www.statista.com/topics/8016/covid-19-impact-on-the-gaming-industry-worldwide/#dossierKeyfigures>
- Domenico, A. D., Perna, G., Trevisan, M., Vassio, L., & Giordano, D. (2021). *A network analysis on cloud gaming: Stadia, GeForce Now and PSNow*.
- Kwan, B. S. (2009). Reading in preparation for writing a PhD thesis: Case studies of experiences. *Journal of English for Academic Purposes*, pages 180-191.
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.
- Nasteski, V. (2017). *An overview of the supervised machine learning methods*. Horizons.
- Perkins, C. (2003). *RTP: Audio and Video for the Internet*. Addison-Wesley Professional.
- Perna, G., Markudova, D., Trevisan, M., & Garza, P. (2021). *Online Classification of RTC Traffic*.
- Perna, G., Markudova, D., Trevisan, M., Garza, P., Meo, M., & Munafo, M. M. (n.d.). *Retina: An Open-Source Tool For Flexible Analysis of RTC Traffic*.
- Rescorla, E., & Modadugu, N. (2012, January). *datatracker.ietf.org*. Retrieved from ietf.org: <https://datatracker.ietf.org/doc/html/rfc6347>

Russell, S. J., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Schulzrinne, H. C. (2003). *RTP: A transport protocol for real-time applications*.

Schulzrinne, H., Casner, S., Frederick, R., & Jacobson, V. (2003, July). "*RTP: A Transport Protocol for Real-Time Applications*". Retrieved from tools.ietf.org: <<https://www.rfc-editor.org/info/rfc3550>>.

Suznjevic, M., Slivar, I., & Skorin-Kapov, L. (2016). . Analysis and qoe evaluation of cloud gaming service adaptation under different network conditions: The case of nvidia geforce now. *Eighth International Conference on Quality of Multimedia*.