POLITECNICO DI TORINO

Department of Electronics and Telecommunications

Master's Degree in ICT for Smart Societies

Master's Thesis

Machine learning for predicting losses in the networks

Supervisors:
Prof. Michela MEO
Dena MARKUDOVA
Gianluca PERNA

Candidate:
Tailai SONG

2022

Abstract

In modern computer and telecommunication networks, traffics are transmitted based on packet-mode protocols. Due to the dynamic as well as the complicated properties of networking and the spread of broadband and internet access, one of the major issues is packet loss. Some of the protocols, like Transmission Control Protocol (TCP), can tackle this problem and recover losses, but for Real-time Transport Protocol (RTP), packet delivery cannot be guaranteed, which will significantly affect the Quality of Experience (QoE) for services. Therefore, on one hand, a preventive solution that can forecast the packet loss is needed to help build an adaptive mechanism and improve the QoE, and on the other hand, studies of packet loss can help to understand the network status, e.g., congestion.

In this thesis, we will focus on RTP-based Real-time Communication (RTC) services, and comprehensive analyses for a case study based on two RTC applications will be conducted to understand the specific properties of packet loss. On top of that, several machine learning approaches will be developed to predict losses based on historical statistics of packets.

In particular, an improvement to the command-line tool *Retina* will be made to derive the exact number of packet losses, and then, the dataset about statistics of RTP packets will be analyzed, processed, and utilized to feed the machine learning models. Finally, in-depth studies will be performed as well.

Acknowledgements

I want to start by expressing my sincere gratitude to Dena Markudova and Gianluca Perna for their genuine help and guidance throughout the development of my thesis. This has not only helped me to improve my skills, but has also been enlightening for me in the long run.

Second, I want to express my deepest appreciation and greatest respect to Professor Michela Meo for all of her advice and support, as well as for giving me such a wonderful opportunity. Being able to work under your supervision is truly an honor and a delightful experience.

Also, many thanks to all the professional and intelligent people (Prof. Martino Trevisan, Prof. Paolo Garza, Prof. Maurizio Matteo Munafò) in the project for their suggestions and information.

Finally, I want to thank my family for all of their support over the years, both financially and emotionally.

Table of Contents

List of Tables

List of Figures

Acronyms

CDF
Cumulative Distribution Dunction

DNN
Deep Neural Network

DT
Decision Tree

ECDF
Empirical Distribution Dunction

GBDT
Gradient Boosting Decision Tree

IP
Internet Protocol

kbps
Kilobit per second

LSTM
Long short-term memory

MAE
Mean Absolute Error

ms
Millisecond

NAT

Network Address Translation

NN

Neural Network

pcap

Packet capture

QoE

Quality of Experience

RF

Random forest

RFE

Recursive Feature Elimination

RTC

Real-time Communication

RTCP

RTP Control Protocol

RTP

Real-time Transport Protocol

RNN

Recurrent Neural Network

s

Second

SMOTE

Synthetic Minority Oversampling Technique

SSRC

Synchronization source

Tbps

Terabytes Per Second

TCP

Transmission Control Protocol

UDP

User Datagram Protocol

XGBoost

eXtreme Gradient Boosting

Chapter 1

Introduction

1.1 Motivation

In the modern lives of human beings, telecommunications and internet communications are the most indispensable ways to exchange information. It is without doubt that people can barely carry out daily activities without them. The volume of traffic produced by both humans and machines reaches an incredibly vast number as more sophisticated technologies are introduced into the communication field and people's needs continue to rise. According to the Global Internet Map 2021[1], the global traffic on international links in 2021 reaches around 280 terabytes per second (Tbps) at peak hour and around 180 Tbps on an average scale, while the numbers in 2020 are only 190 and 110 Tbps. Moreover, peak international internet traffic increased at a compound annual rate of 30% between 2016 and 2020. Meanwhile, based on Global Mobile Data Traffic[2], traffic generated by mobile devices, one of the most significant and popular methods, will reach 77.5 exabytes per month in the world in 2022, increasing 36% from 2021 and about 7 times higher than in 2017. Consequently, while the enormous quantity and explosive growth signify the prosperity of contemporary society and increase industry profits as well as national revenues, they also have a tendency to bring about more issues, such as slow and unreliable content delivery, lowering the QoE for all types of communication services, among which, RTC applications face a lot of challenges that require the service provider to exert more effort.

Nowadays, RTC applications reach every corner of modern-day life, and RTP is one of the most prevalent network protocols that plays a crucial role in delivering

real-time content for RTC. From a commercial and leisure standpoint, RTP services, such as streaming media and video teleconference, are essential tools for today's society since they enable connections among people for work, entertainment, or personal errands from remote. In particular, when it comes to the great pandemic of COVID-19, the increasing demands for telecommuting due to lockdown result in a huge boost for RTC applications. The traffic volume is increased by 15-20% almost within a week, and web conferencing applications show a dramatic increase of more than 200% during business hours[3]. Moreover, the rise of COVID-19 caused a tremendous global spike in the video conferencing market, amounting to 5.77 billion in 2020[4], which substantially facilitates the growth of business applications. For example, *Zoom* expects to boost revenue by 200% and profits by 300% in 2020, and predicts that it will nearly double its revenue to more than $1.78 billion in 2021 and, as a result, triple its stock price from about $69 to $208[5]. On top of that, there was already a huge trend before the strike of the pandemic: in 2019, out of more than 1300 business professionals, 55% of respondents agree that companies that use video conferencing are more collaborative; usage of video communication at work has increased for 48% of respondents as compared to two years ago; and 80% of business professionals use video conferencing for meetings, while 78% use it to facilitate team meetings[6]. Not surprisingly, the market will continue to grow from USD 6.87 billion in 2020 and reach a staggering amount of USD 14.58 billion by 2029[4]. Additionally, video conferencing is also an energy-friendly and economically viable way to participate in a meeting by avoiding business travel, which helps reduce energy consumption and waste production and eventually cut nearly 5.5 million metric tons of CO_2 emissions by 2020[7]. Thus, it is significantly important for scientists and engineers to devote more resources to studying RTC traffic in order to improve the QoE perceived by users and guarantee a higher level of service.

It is undeniable and quite natural that users tend to expect the best quality in telecommunications services, and thus, in order to attract more customers and gain higher subscriber loyalty, the service providers ought to be aware of the users' QoE and work to make improvements. The International Telecommunication Union states that QoE is *"The degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user's personality and current state"*[8]. The QoE can be affected by human influence factors, system influence factors, and context influence factors[9], and out of all the factors, the most relevant and objective are probably those related to the network performance.

For RTC applications based on RTP, one of the most common and vital problems

affecting QoE is packet loss. RTP also relies on packet switching, like other telecommunications protocols do, to aggregate data into packets that can be sent over networks. However, unlike TCP, which has the function to detect packet losses and recover them by retransmission, RTP applications suffer from the phenomenon of packet loss, resulting in penalties in QoE for users. In this context, it is exactly due to the inherent difficulties of preventing and recovering losses, we need to come up with reliable and effective solutions that can detect the network circumstances and predict the potential losses. As a result, the system could monitor the network performance and adapt itself to changing configurations by taking precautions to avoid prospective packet losses or referring to a proper recovery strategy depending on the information retrieved from RTP traces, and the QoE for users can finally be maximized.

1.2 Objective

In recent years, when it comes to prediction problems, machine learning techniques have been proven to be effective. The objective of this thesis, which focuses on RTP traces, is to propose various machine learning approaches and then highlight the one that performs the best to predict the existence of packet loss in the near-term future based on the past statistics about the aggregations of RTP packets for a certain duration.

In general, this thesis is an elaboration of the command-line tool *Retina*[10]. Particularly, the main datasets are derived according to the packet captures obtained from two RTC applications during multiple real video conferences.

Moreover, we will adhere the traditional pipeline of developing machine learning models to enhance the ability for data processing, data analysis, model tuning, etc., and consolidate the knowledge of machine learning and networking. On top of that, we will outline the difficulties and discuss the constraints of the problem, and finally propose solutions for model optimization. Additionally, extra data gathered from some common RTC applications, e.g. *Google Meet*, will be analysed as well to deepen the understanding of packet loss across multiple domains and to check the versatility of algorithms.

Finally, the latent as well as ultimate goal is to demonstrate the feasibility of forecasting losses and the potentiality of implementing the algorithm into real-case scenarios to effectively and reliably reduce the packet loss and, as a result, improve QoE for users.

1.3 Literature review

Numerous works related to RTC applications address topics like classification of RTP flows, network management, and so on. Meanwhile, attention was also drawn to the analysis and prediction of packet loss. This section will focus on the current state of the art and go over several related works produced by other professionals.

In 2004, Lopamudra Roychoudhuri and Ehab S. Al-Shaer[11] focused on the prediction of packet loss in real-time audio streams based on the available bandwidth, delay variation, and trend, enabling proactive error recovery for real-time audio over the Internet. They have identified the delay at the capacity saturation point of a path as the loss threshold delay, after which packet loss is more likely, and then tracked the trends of the delay as an indication of congestion causing packet loss. In particular, they have formalized a delay-loss correlation model to define a loss predictor approach and evaluated the performance through a simulation.

After two years, Fernando Silveira Filho and Edmundo de Souza e Silva[12] have evaluated different hidden Markov chain based models as predictors of short-term loss statistics and proposed an adaptive algorithm to estimate near future losses based on recent measurements and compare the accuracy of different underlying models. They have developed a novel recursive algorithm that evaluates the distribution of the number of packets lost in a time window in the future given the recent history, and lastly, they have proposed a hierarchical hidden Markov model (HMM) that has lower computational complexity.

Hooman Homayounfard[13] has addressed in his PhD thesis the issue of defining a Data Mining (DM) model for packet switching in the communications network. The main idea behind his work and related research projects is that time-series of network performance parameters, with periodical patterns, can be used as anomaly and failure detectors in the network. His project finds frequent patterns in delay and jitter time-series to be used in real-time packet-loss predictions. He has proposed two models for approximation of delay and jitter time-series and prediction of packet-loss time-series — namely the Historical Symbolic Delay Approximation Model (HDAX) and the Data Mining Model for Smart Routing in Communications Networks (NARGES) using two kinds of datasets: (i) the Distributed Internet Traffic Generator (D-ITG) and (ii) the OPNET Modeller (OPNET) datasets.

Instead of predicting the packet loss, Manish P Ganvir and Dr. S.S.Salankar[14] refereed to the forecasting of packet loss rate by proposing an artificial neural network trained with Particle Swarm Optimization (PSO) as a training algorithm.

In recent years, the authors of [15] specifically focused on the video loss prediction in wireless network by obtaining a model that observes the Service and Experience Quality Metrics, such as PSNR (Peak Signal-to-Noise Ratio) and packet loss. They have outlined a methodology and mathematical model for video quality loss in the wireless network from simulated data, and the methodology is based on logarithmic and exponential mathematical functions with parameters defined by linear regression. Furthermore, Dr. Kalpana Saha and Tune Ghosh[16] have adopted machine learning techniques to predict packet losses for providing autonomous Call Admission Control (CAC). In particular, decision trees and logistic regression are used, and based on the performance derived from experiments and observations, the decision tree model gives the better result.

According to the status quo, the majority of methods for predicting packet loss are based on the end-to-end delay, which necessitates observations and data collection from both packet sender and receiver because the delay is calculated by the difference between the time taken for a packet to be transmitted across the network from source and the time taken for the packet received at destination. In contrast, the premise of this thesis is that the only information available is the statistics about packets that have been received, which means the system is observing the network condition on just one side of the communication without the need for detection on other sides. It is simply because, on the one hand, the packet loss and its causal phenomena such as congestion may only affect a few participants in RTC applications; on the other hand, the QoE refers to the individual perception of the service quality, which may differ from the application's overall performance. However, it has been widely proved[17][18][19] that delay has a strong correlation with packet loss, which means the prediction based on delay is more likely to result in better performance. In contrast, firstly, the statistics about packet aggregation may not be characteristic if we do not have a lot of losses. Secondly, the statistics from various applications may differ from each other. Thirdly, errors and uncertainties can occur during the collection, integration, and calculation of packets as well as the subsequent computation of statistics. All of these aforementioned factors could lead to more challenges for prediction.

1.4 Thesis outline

The rest of this thesis is as follows:

- **Chapter 2** introduces the general background knowledge about RTP, the information about *Retina* and overviews of machine learning approaches;

- **Chapter 3** presents the preliminary work related to *Retina*, packet loss characterization, and data preprocessing;
- **Chapter 4** formulates the problem, and in the meantime, illustrates the metrics used for model evaluation and possible challenges;
- **Chapter 5** focuses on the development of six machine learning approaches and the corresponding results;
- **Chapter 6** refers to certain preferable machine learning methods derived previously to perform in-depth analyses regarding real case scenarios, model optimization and extra data from more applications;
- **Chapter 7** draws conclusions, and presents the limitations and future works.

Chapter 2

Background

In this chapter, in order to better understand the context, we will review the background knowledge of RTP, specifically in terms of RTC and packet loss. Moreover, the introduction of *Retina* will be presented. Finally, all of the machine learning approaches that have been implemented will be briefly introduced as well.

2.1 RTP

The Real-time Transport Protocol (RTP)[20] 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. Applications typically run RTP over UDP to make use of its multiplexing and checksum services, but RTP does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees. RTP consists of two closely-linked parts: (i) the real-time transport protocol (RTP), to carry data that has real-time properties; (ii) the RTP control protocol (RTCP), to monitor the quality of service and to convey information about the participants in an on-going session. RTP is used in multiple scenarios, such as audio and video conferences. Moreover, there are several important terminologies[20] related to RTP for this thesis:

- **Synchronization source (SSRC)**: the source of a stream of RTP packets, identified by a 32-bit numeric SSRC identifier carried in the RTP header so as not to be dependent upon the network address. All packets from a synchronization source form part of the same timing and sequence number

space, so a receiver groups packets by synchronization source for playback. It is randomly chosen by the source at session startup.

- **Port** (either source port or destination port): the abstraction that transport protocols use to distinguish among multiple destinations within a given host computer. RTP depends upon the lower-layer protocol to provide some mechanism such as ports to multiplex the RTP and RTCP packets of a session.
- **Payload type**: it is a 7-bit number that identifies the format of the RTP payload and determines its interpretation by the application. It is the type of coding used in the packet payload. For example, for Jitsi Meet, "111" means audio and "114" means video.
- **Sequence number**: it is a 16-bit numeric identifier that indicates the original sending order and increments by one for each RTP packet sent. Details are in paragraph 2.1.2.

In the following, we will review several key elements that are important for this thesis.

2.1.1 RTC

Real-time communication (RTC) is used to describe all types of live telecommunication that take place with negligible latency or transmission delays. RTC services include mobile telephony, voice over IP (VoIP), teleconferencing, instant messaging, interactive gaming, etc. Nowadays[21], for RTC applications, RTP is still most widely adopted as the fundamental protocol. Additionally, there are multiple other protocols and solutions cooperating with RTP to support RTC applications, among which, WebRTC[1] is one of the most important and popular tools. WebRTC[22] is a set of high-level and standardized APIs used in browsers and mobile applications for video and audio communication. It represents the standard way for RTC applications to run via the web and organize the use of different protocols.

In this thesis, we focus on two types of RTC applications for online meetings supporting audio, video, and screen sharing: Cisco Webex Teams[2] and Jitsi Meet[3]. Both of them use RTP for streaming multimedia content along with STUN (Session

[1]https://webrtc.org/

[2]https://www.webex.com/team-collaboration.html

[3]https://meet.jit.si/

Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT) for session establishment. In particular[21]:

- **Webex Teams** (or Webex for short) is a business-oriented service that offers paid plans for enterprises and institutions that require video call service. It is available as a standalone application for PC and mobile devices, but it can also be used through browsers that support the WebRTC standard.
- **Jitsi Meet** (or Jitsi for short) is a free-of-charge RTC application that provides a simple browser-based user interface for WebRTC-compliant browsers. It is fully open-source and it is possible to run a private Jitsi server or rely on the public service available online.

Moreover, the data[4] from both applications is collected based on 62 hours of real calls. Note that in the later stage, data from more RTC applications will be included as well, and the reasons we focus on these two applications are: i.) The ground truth for packet loss is available in a convenient way; ii.) The quantity of the collected data is abundant.

2.1.2 Packet loss

Packet loss is an undesired phenomenon of one or more packets failing to reach the destination when travelling through the network, and it can be caused by issues related to networks, hardware, software, or security reasons. In particular, it is mainly due to the following two reasons:

- **Network congestion** is the major cause of packet loss. It is an inherent phenomenon occurring in network elements when the input rate of packet flow exceeds the capacity of the network node or link. When the quantity and rate of incoming packets overwhelm a certain router in a network, the buffer tends to overflow and has no choice but to drop the packets intentionally. Packet loss can be considered as an indicator of congestion so that the study of packet loss can also help understand network congestion, and the prediction as well as the consequent reaction can be referred to as a heuristic way to tackle congestion.

[4]This is supported by the ML4QoE group from the SmartData@PoliTO center on big data and data science and Cisco Systems Inc.

- **Wireless networks** are more prone to packet loss than wired networks due to the probable effects of weak signal strength, distance, physical obstacles, and natural or human-made interference. It is an issue related to network medium because of system noise, hardware failure, software corruption and many more[23]. Moreover, packet loss problem is more complicated and difficult to handle in mobile networks and the observed loss rates are much higher than in the stationary case[24].

It is for these reasons that the difficulties in packet loss prediction are highlighted. However, although we do not distinguish the causes of packet loss in this thesis and instead focus on the characteristics of statistics of packets with or without losses for the two applications (Webex and Jitsi) we are working on, network congestion is still the major reason for packet loss because the data is collected in relatively steady and decent network conditions.

The effect of packet loss can be severe, resulting in the degeneration of QoE. In general, packet loss leads to the reduction of network throughput, corruption of data integrity, growth of latency, etc. For RTC applications, packet loss can be even worse since what we need is the fast delivery of packets rather than guaranteeing the delivery. It can result in audio inconsistency, video frame deficiencies, temporal interruptions, and other issues. A typical example can be seen in Fig2.1[25]. Therefore, these not only require a strategy for mitigation of packet loss effect but also strengthen our motivation.

On the one hand, for some protocols, it is possible to detect and recover packet loss, such as TCP. The Transmission Control Protocol is able to ensure reliable delivery of packets by identifying the packet loss based on packets without acknowledgement and consequently recovering them through retransmission. Furthermore, in order to cope with congestion, which is responsible for most of the packet losses, TCP also introduces several congestion control strategies such as slow start, congestion avoidance, fast retransmit, and fast recovery.

On the other hand, RTP does not guarantee the successful delivery of packets because it runs over UDP, which does not detect and deal with packet loss since UDP doesn't refer to retransmission, which causes delay and is not appropriate in real-time communications. For RTC applications, 0-1% of packet loss is good, 1%-2.5% is acceptable, 2.5%-5% is poor, 5%-12% is very poor, and greater than 12% is bad[26]. However, RTP enables applications to resort to some mechanisms to tackle packet loss: the indication of the number of packet loss in the RTCP receiver report, the variation of bit rate based on congestion level, etc. Moreover, some specific approaches are implemented to deal with packet loss, like Forward

Figure 2.1: An example of packet loss effect

Error Correction (FEC) for audio packet loss[11]. But this is far from adequate, simply because all the technologies have their drawbacks, and on top of that, the lost packets are no longer available, which reinforces even more our motivation to develop a preventive method informing the system and reacting to loss before it occurs.

Technically, for RTP, packet loss can be identified and packet sequence can

be restored by the receiver based on the sequence number[20] provided by the RTP header. The sequence number is a 16-bit number that is randomly chosen at session startup and increased monotonically by one for each RTP data packet sent, which means that, theoretically, a series of consecutive RTP packets ought to have a consecutive sequence number and one or more missing values of discontinuity indicate the corresponding lost packets. This is the exact logic to derive the actual number of packet losses when it comes to the work related to *Retina* described in section 3.1.

2.2 Retina

Retina[10][5] stands for Real-Time Analyzer, and it is an open-source command-line tool written exclusively in *Python* that produces rich and complex statistics from RTC traffic. Based on one or more raw packet captures of RTC traffic, *Retina* is able to derive a log of statistics and track the evolution of the stream over time for RTP flows in *csv* files. The architecture of *Retina* is in Fig.2.2. In the following, we will check out three aspects related to *Retina*.

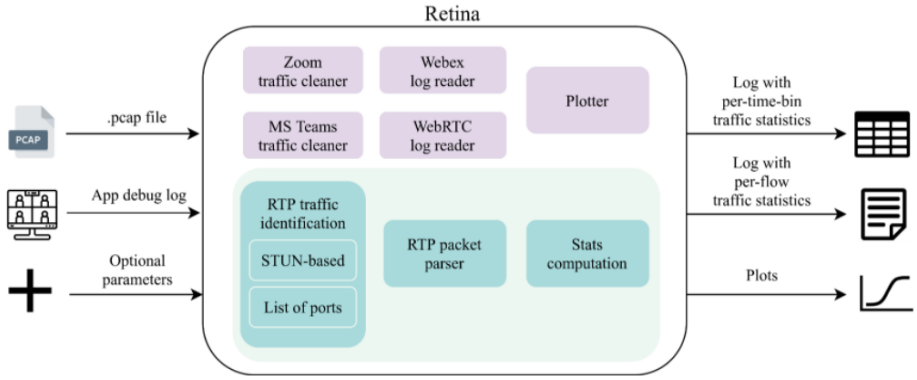


Figure 2.2: Retina architecture

2.2.1 Packet capture

Packet capture, or its abbreviation - *pcap*, is an application programming interface (API) for capturing network traffic. A *pcap* file is created by analyzer software like

[5]https://github.com/GianlucaPoliTo/Retina

Wireshark[6] to collect and store packet information with payload in an encryption format during network communication, and it can be used to analyze the network characteristics. *Retina* directly works with one or more *pcap* files to derive statistics about RTP packets. Packets will be distinguished by different flows for the entire RTP traffic and then aggregated into multiple time bins for a continuous time series. Additionally, *pcap* files are also used in conjunction with *Wireshark* during the thesis development to investigate specific RTP packet contents.

2.2.2 Outputs of Retina

Besides the plots of network traffic, the most important output of *Retina* is the data sheet in a *csv* file containing multiple types of statistics for a certain time aggregation. The data is collected for different flows[7] in the traffic following the time series, and the duration of each sample is the time aggregation (time bin)[8]. *Retina* allows users to configure the time aggregation over which the statistics are computed, and as a result, it is able to derive the overall behavior for all the packets in each time bin. In this thesis, we will always refer to the time bin of 500 milliseconds[9].

Regarding the statistics, there are 96 specific types in total and they are mainly about the following categories[10]:

- **Interarrival**: the difference between the arrival time of the current packet and the previous one. For example, the *"interarrival_mean"* is the mean value for all the interarrival time in a time bin.
- **Length UDP**: UDP length of the current packet, which is the total length

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

[7]A flow is characterized by its SSRC, IP address of source and destination, port of source and destination, and payload type. Different flows may share the same time series but with various characteristics, so that we need to deal with them differently and separately.

[8]The time aggregation or time bin is characterized by its corresponding start timestamp, which means the difference between any adjacent samples (time bins) is exactly the length (500 ms in our case) of the configured time aggregation.

[9]In this thesis, work is completed using time bins with a fixed length of 500 milliseconds because it is an appropriate duration for having an adequate number of time bins with packet loss and forming a distinct trend that can be used to characterize packet loss. However, it is also a good idea to investigate the variation of time bins in the future.

[10]Details are in https://smartdata.polito.it/rtc-classification/

of the UDP header and data. For example, *"len_udp_std"* is the standard deviation of UDP lengths of all packets in a time bin.

- **Inter-length**: the difference between the UDP length of the current packet and the previous one. For instance, *"interlength_udp_min"* is the minimum difference in UDP length between any two adjacent packets in a time bin.
- **RTP inter-timestamp**: the difference between the current packet RTP timestamp and the previous one. The RTP timestamp reflects the sampling instant of the first octet in the RTP data packet[20]. The initial value of the timestamp is generated randomly, and certain consecutive RTP packets may have equal timestamps if they are produced at the same time.
- **Inter-time sequence**: the difference between the sequence number and the RTP timestamp of the current packet. The initial value for RTP timestamp and sequence number are both generated randomly, but multiple consecutive RTP packets may have the same timestamp if they are produced at once, while the sequence number is always monotonic.

The statistics can be considered as the characteristics of all the received packets in a time bin. If the traffic during a certain time bin experiences losses in between, the missing packets will affect the overall behavior not only for the target time bin but also the neighborhood time bins, because the congestion is not an instantaneous phenomenon but an evolution in the network, which may not induce losses at the beginning but influence the traffic continuously and result in an extraordinary pattern with respect to normal conditions. In this thesis, we will refer to these statistics as features to characterize the time bin with or without loss and make predictions. The work in this thesis is totally based on the outputs of *Retina* as a data-driven problem.

2.2.3 An issue of Retina

Retina was designed specifically to output statistics for packet captures without taking into account the packet loss. Thus, the major problem of *Retina* is that it can only output an estimation of packet loss but not be able to derive the exact number of packet losses in a time aggregation. Although our prediction is only about the existence of packet loss in the near future, regardless of the exact number of losses, the actual quantity is substantially useful when it comes to the understanding of packet loss and the analysis of the traffic. Therefore, the first task of this thesis, which will be presented in the next chapter, is to improve upon the old version to obtain the exact quantity of packet losses in each time bin.

2.3 Algorithm introduction

As one branch of artificial intelligence, machine learning is the study of computer algorithms that improve automatically through experience. It serves and supports a series of applications ranging from data mining programs that discover general rules in large data sets to information filtering systems that automatically learn users' interests. In this thesis, we have developed multiple machine learning approaches to learn the underlying patterns of packet loss phenomena based on the statistics and apply the knowledge to identify and predict potential losses in the future.

In this section, six machine learning methods developed and implemented in this thesis in terms of classification and anomaly detection will be briefly introduced in order to give a fundamental insight into the methodologies for predicting packet loss.

2.3.1 Classification

Classification in machine learning is the process of using computer algorithms to recognize and categorize objects, and thus to help separate automatically a large number of targets into corresponding classes. In our case, the categories are time bins with or without packet loss, and what we need to do is to develop machine learning algorithms based on different characteristics of the two classes to derive a classifier with decent performance. In particular, we have referred to four different algorithms described in the following:

Decision tree A decision tree (DT) is a decision-making tool using a tree-like structure to go from observations of an object to finally drawing conclusions about the target value. It is a non-parametric supervised learning method for creating a model that predicts the value of a target by learning simple decision rules inferred from the data features. For classification, each node in the tree is a decision node that refers to a certain feature value as a specific threshold, for example, the one that maximizes the information gain, to split the tree until reaching child nodes in the same category. An example can be seen in Fig.2.3

A decision tree is one of the most popular and traditional machine learning algorithms for classification due to its simplicity, low cost, and good interpretability. However, it tends to create over-complex trees, leading to overfitting that does not generalize the data very well. In this thesis, we refer to the *Python* package

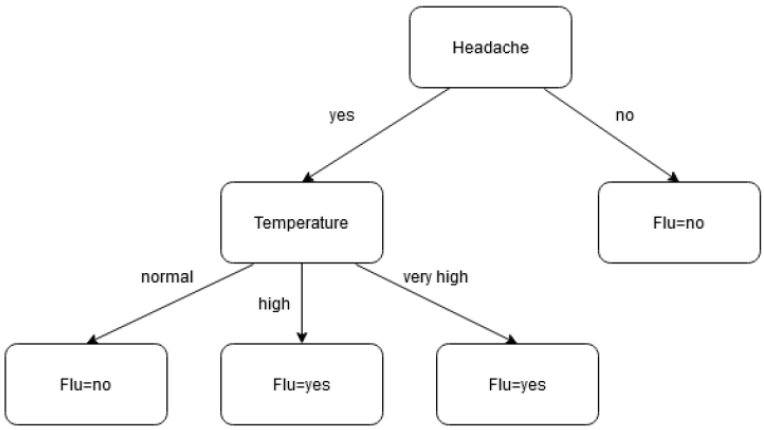


Figure 2.3: An example of decision tree

scikit-learn to implement decision trees.

Neural network A neural network (NN), also known as an artificial neural network (ANN) or deep learning, is a class of machine learning algorithms that use a network of connected units (neurons) capable of adapting to states and exchanging information to perform classification or regression. It is a concept inspired by biological neural networks in animal brains, and it tries to recognize latent relationships and patterns in data based on a process that resembles the way the human brain operates. In our case, we have developed two types of neural networks: deep neural network (DNN) and long short-term memory neural network (LSTM NN).

A DNN is a typical neural network with multiple intermediate layers other than the input layer and output layer. Fig.2.4[11] shows an example of the DNN structure. It is a feedforward network following a multi-layered perceptron (MLP) structure that retrieves data from the input layer and transmits information through the network to the output layer, and consequently, modifies the weight as well as the bias of each neuron through backpropagation based on the difference between outputs and desired values.

LSTM NN is a type of recurrent neural network (RNN) that relies on connections between nodes that form a directed or undirected graph along a temporal sequence so that it works better for dynamic behavior. LSTM outperforms RNN for inputs

[11]https://www.ibm.com/it-it/cloud/learn/neural-networks

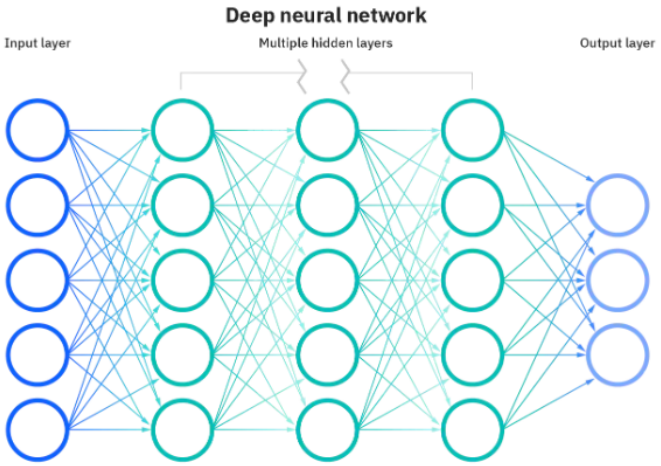


Figure 2.4: An example of DNN structure

over a long period of time by replacing RNN cells with LSTM cells (Fig.2.5[12]) including an input gate, an output gate, and a forget gate. LSTM is able to use internal states to memorize a sequence of inputs so that it does not lose information if the sequence has correlations in between. In our case, in order to make a prediction, we will refer to the statistics in the past, which are a series of values affected by packet loss. Thanks to LSTM, we are able to extract the latent trend of a series of statistics instead of treating them individually.

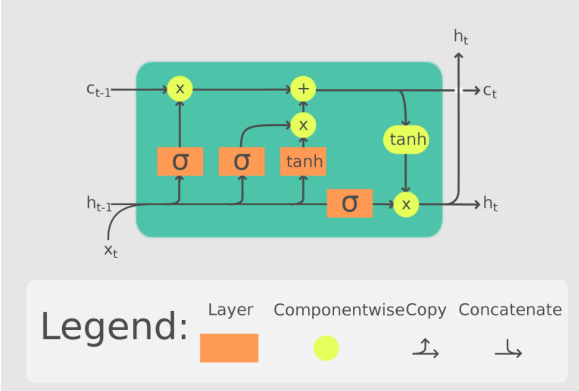


Figure 2.5: The LSTM cell

An artificial neural network typically works efficiently and versatilely. On the

[12]https://en.wikipedia.org/wiki/Long_short-term_memory

contrary, due to the lack of interpretability, it normally works as a black box. In this thesis, we refer to *TensorFlow* to implement both DNN and LSTM.

Random forest Random Forest (RF) is an ensemble supervised learning method that elaborates on decision trees. For classification, random forest constructs a number of classification trees with different selections from inputs and training paths, and refers to the majority of output categories as the predicted class. The basic mechanism is illustrated in Fig.2.6[13].

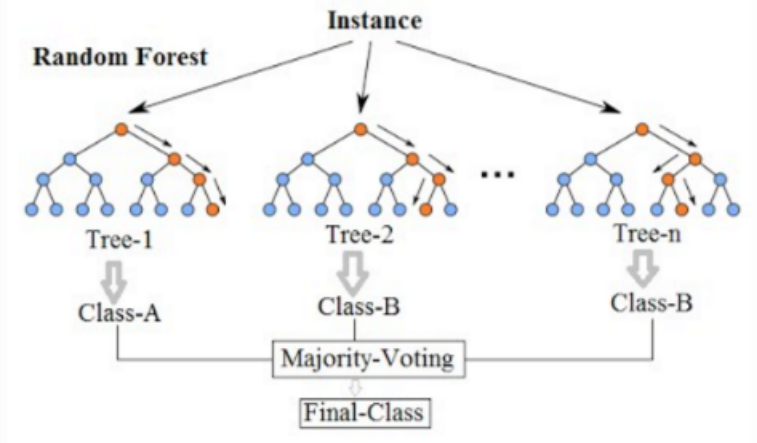


Figure 2.6: An example of how random forest works

As one of the most important machine learning algorithms, although random forest requires more computational resources and training time, it can overcome the problem of overfitting and tends to have better performance. In this thesis, we refer to *scikit-learn* to implement random forest and, at the same time, we also refer to *imbalanced-learn* to deal with imbalanced datasets.

Gradient boosting tree Gradient boosting[27] is a machine learning technique that sequentially builds relatively simple models in which each one tries to optimize the model and make a prediction based on the error generated by the previous one. Similar to random forest, gradient boosting tree or gradient boosting decision tree (GBDT) is also an ensemble machine learning method that elaborates on decision

[13]https://community.tibco.com/wiki/random-forest-template-tibco-spotfire

trees to fit a tree on the residual error of the previous tree instead of building parallel trees.

In particular, in order to implement GBDT, we will use *eXtreme Gradient Boosting (XGBoost)*[14] which still refers to the principle of gradient boosting but improves upon it by introducing regularization to avoid overfitting.

2.3.2 Anomaly detection

Anomaly detection, as known as outlier detection, is a group of machine learning algorithms that endeavor to recognize rare events or observations that do not follow normal behavior or predetermined rules and significantly deviate from the majority of the targets. Such suspicious cases may produce unusual and distinguishable patterns that can be used in machine learning approaches to detect anomalies. In this thesis, we have tried two methods in the following:

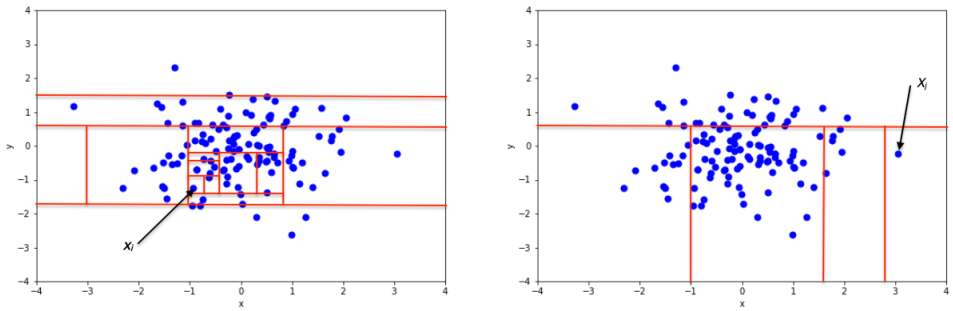
Isolation forest Isolation forest (IF) is an unsupervised anomaly detection method that tries to explicitly isolate the outliers from the dataset rather than constructing a profile of normal cases. It starts from a random value in a random feature and splits the dataset into separate portions. Such a process can be realized by a binary tree. Then, it continues repeating the partition until all the data at the node has the same values and finally outputs an anomaly score for each data point. For an outlier, it is easier to separate from the rest of the samples due to its deviation from normal cases and results in a shallow depth in the tree, while for normal instances, they are close to each other because of similar behaviors and, thus, it is harder to be isolated, leading to a deep depth in the tree. A simple example describing the isolation process is in Fig.2.7[15]. In this thesis, isolation forest is also realized by *scikit-learn*.

Autoencoder An autoencoder is a special type of artificial neural network that has the same shape of input and output layers with a bottleneck in between. Fig.2.8[16] shows an example structure. It tries to retrieve the input data, learn the latent representation, and reconstruct the input as much as possible. An

[14]https://xgboost.readthedocs.io/en/stable/

[15]https://en.wikipedia.org/wiki/Isolation_forest

[16]https://medium.com/analytics-vidhya/autoencoders-with-tensorflow-2f0a7315d161



(a) Normal data point **(b)** Anomalous data point

Figure 2.7: An example of isolating a non-anomalous and an anomalous point in a 2D Gaussian distribution

autoencoder consists of two components: the encoder and the decoder, in which the encoder tries to summarize the input and reduce the dimension to feed the bottleneck, and the decoder attempts to reproduce the input with a smaller reconstruction error. In order to make the autoencoder work for anomaly detection, we use it as an unsupervised machine learning method by only inputting normal cases. As a result, the NN can only learn the latent representation of normal cases, and with an input of an outlier, the autoencoder will reconstruct the input the way it reconstructs a normal case and therefore generate a larger error. In this thesis, we develop the autoencoder by *TensorFlow*.

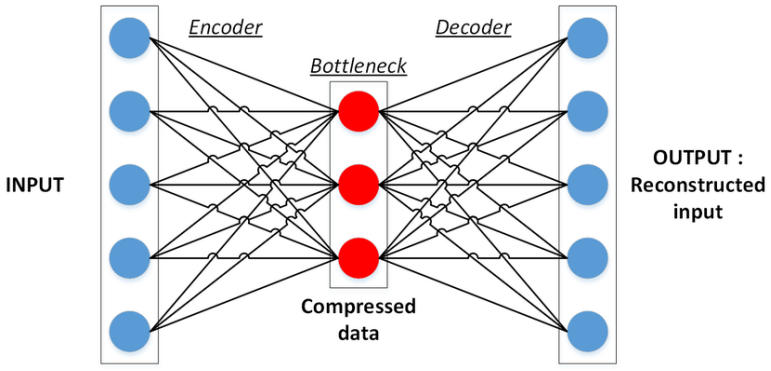


Figure 2.8: An example of autoencoder

Chapter 3

Dataset description

The objective is to predict the presence of packet loss based on statistics about previously received packets. In this chapter, we present the detailed description of the dataset that we will be working on, starting from the data source to the final clean and well-structured data sheet. In particular, all of the preliminary work will be discussed, and besides the introduction of the procedure deriving the actual number of packet losses for *Retina*, we will focus on data preprocessing and packet loss characterization, and finally be ready to formulate the problem.

3.1 Work related to Retina

In the first place, it is essential to derive the quantity of packet loss. *Retina* collects packets for each flow and aggregates them into certain time bins (500 ms in our case) to calculate statistics, and what we need is the number of packet losses in each time bin.

Based on chapter 2.1.2, losses can be detected through the investigation of sequence numbers. Therefore, the way is to gather the sequence numbers of all packets in a flow and then identify the missing values to reveal packet loss. If the difference in sequence numbers between two contiguous packets is greater than one, it means one or more sequence numbers are missing. In other words, one or more packets are lost. To sum up, the number of packet loss can be calculated as:

$$n_{loss} = seq_t - seq_{t-1} - 1,$$

in which n_{loss} is the number of packet loss between adjacent packets, seq_t is the

sequence number of the packet at current timestamp and seq_{t-1} is the sequence number of packet at previous timestamp. At last, the packet aggregation process also includes information about the number of packet losses for each time bin by summing up the number of losses between any continuously received packets in a time bin. However, there are still three concerns that may lead to mistakes that we should be aware of:

1. The RTP sequence number is generated randomly for the first packet and increased by 1 for each of the following packets sent, but it won't keep growing to infinity because the sequence number space is limited. The RTP sequence number is a 16-bit number and the maximum is $2^{16} - 1 = 65535$ while the minimum is 0. When the sequence number reaches the maximum, it will start over from 0 and this kind of mechanism can be considered a transition. If we encounter a series of sequence numbers with a transition among them in a time bin and do not take into account the problem, it will generate mistakes[1] to packet loss calculation.

2. We may have multiple duplicates of packets with the same sequence number due to: (i) a sender somehow detects that a packet is not transmitted correctly (the packet actually arrives without problems at the receiver end), and then it might retransmit the exact packet and result in duplicates at the destination; (ii) problems related to hardware, software, or networks. It is a huge problem for the calculation of the number of losses. For example, we have a series of sequence numbers of [100, 101, 102, 100, 103, 104] in which the second "100" is a duplicate. As a result, the packet loss will be [0, 0, -3, 2, 0]. Not to mention that it will lead to more problems if the duplicate arrives at a different time bin.

3. RTP does not guarantee in-order delivery. RTP packets are sent from the sender in order with incremental sequence numbers and are supposed to arrive at the receiver in the same order as well. But in reality, due to the intrinsic property of RTP and the dynamic and complicated situations of networks, a certain destination may encounter chaotic arrivals of packets with sequence numbers that are out of order. This is not a big problem for RTP since the receiver is able to restore the order based on the sequence number and "declare" the arrivals with a very short and acceptable delay.

[1]For example, we have a series of sequence numbers: [65533, 65534, 65535, 0, 1, 2]. If we do not identify the transition, the resulting packet loss will be [65534-65533-1, 65535-65534-1, 0-65535-1, 1-0-1, 2-1-1], that is [0, 0, -65536, 0, 0], which is absolutely a mistake.

However, when it comes to the identification of packet loss, this has to be taken into consideration because we are dealing with packets following a time series and the aggregations for time bins are also in sequence so that we can utilize the previous information to predict future packet loss, but the out-of-order arrivals and the corresponding out-of-order sequence numbers actually follow an in-order time series, which will lead to an incorrect calculation of packet loss. For instance, a series of RTP packets with sequence numbers from 1 to 10 are sent in order but they arrive at the destination not in the original order and result in sequence numbers of [1, 2, 3, 4, 6, 5, 7, 8, 9, 10]. If we do not restore the order, the resulting packet losses are [0, 0, 0, 1, -2, 1, 0, 0, 0], but in reality, there's no loss at all. It is not a problem if these packets are aggregated into the same time bin since the summation of the losses is still 0, but if they are clustered into two neighboring time bins[2], it will yield an incorrect calculation of packet loss in both of them. In other words, we must solve the problem since we are dealing with packet loss in different time bins and an out-of-order packet that is supposed to arrive at a certain time bin but actually arrives at the previous or following time bin will lead to incorrect calculation. Not to mention that it may also interact with the aforementioned problems and generate even more mistakes.

The problems mentioned above may affect the packet loss calculation separately or along with each other to generate even more errors, which will in turn mistakenly treat the time bin with loss as one without loss or the other way around, and for a data-driven problem, if the original labels of data are incorrect, the result will be biased, inaccurate, and even totally wrong. Thus, it is vital to solve the problems by taking into account all the aforementioned aspects, and it is also better to deal with all the sequence numbers for the entire flow and then make aggregation instead of the other way around. The flowchart of the proposed solution is in Fig.3.1. In particular:

- The block of checking the existence of transitions is simply because we need to deal with sequence numbers between two transitions, either between the start of the flow and the first transition, or between the last transition and the end of the flow, separately so that it is possible to eliminate the potential errors induced by transitions. Furthermore, out-of-order arrivals of packets

[2]For example: we have two time bins with timestamp 12:10:00:000 with sequence number [1, 2, 3, 4, 6] and 12:10:00:500 with [5, 7, 8, 9, 10]. The out-of-order packet is the one with sequence number 5, and the resulting packet losses for them are [0, 0, 0, 1] and [1, 0, 0, 0], which is incorrect and corrupts two flawless time bins.

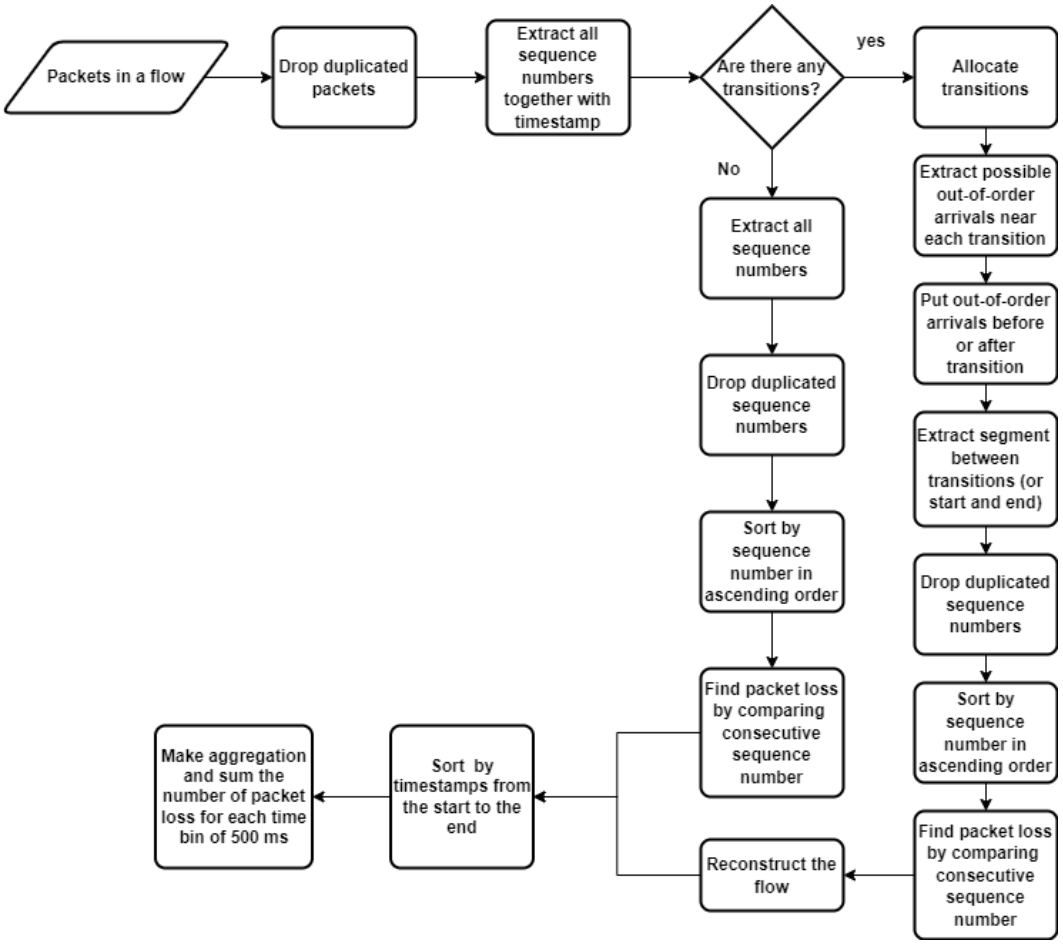


Figure 3.1: Flowchart of packet loss calculation for a flow

may also occur around the transition[3]. Therefore, we also need to extract those out-of-order near transitions and put them back in the correct position.

- The blocks for dropping duplicated sequence numbers will only eliminate the same sequence number for an individual segment between transitions (start or end of the flow), since it is possible to have the same sequence number if there's a transition. Note that the drop action will also drop the timestamp, but this is not a problem because duplicates will induce errors for the packet loss calculation, while on the other hand, they are not packet loss.
- For an individual segment, we simply sort sequence numbers in an ascending

[3]e.g., a series of sequence numbers of [65533, 0, 65534, 65535, 1, 2] or [65533, 65535, 0, 65534, 1, 2].

order because there's no transition in between. Consequently, possible out-of-order arrivals will not generate any problems after the sort, and more importantly, by considering all the sequence numbers in the entire flow, the problems related to out-of-order arrivals in another time bin will also be solved after the sort.

Additionally, *Retina* allows users to include log files (*txt* for Webex and *log* for webRTC), which are available from specific applications or browsers and have information related to RTP traffic, including the record of packet loss, which can be used as the ground truth. At the beginning, we also referred to them to evaluate the correctness of outputs, but this kind of log file is not always accessible and is not accurate, so we can only use them as a reference.

As a consequence, the first job of this thesis is done. Based on the procedure mentioned above, we are finally able to derive the actual quantity of packet loss in each time bin for all flows in RTP traffic, and make them available and put them together with statistics for the same time bins in the *Retina* output, which is the final data sheet we are working on for the prediction.

Moreover, as mentioned in section 2.1.1, we are working with two RTC applications (Webex and Jitsi) and the data is collected during dozens of hours of real calls, including 70 teleconferences with two or more participants. Note that the type of network is either WiFi or Ethernet, and mobile networks like 4G or 5G are never used. All the RTP traffic of each call is recorded and captured by *Wireshark* on one side of the communication, which means we are observing the traffic on this side. Thus, there are two types of packets: received packets and sent packets, and packets that are sent out from the end of the observation do not have losses (on this end) because they are assumed to be generated without errors and losses can only occur after being sent out. In general, the RTP traffic is divided and recorded into multiple *pcap* files (details are in Table.3.1), and by running the new version of *Retina*, 70 *csv* files are generated to be the dataset for further analysis. In particular, each file is composed of multiple rows in which each row corresponds to a time bin[4] of 500 ms and 87 columns including timestamps for time bins, statistics, number of packet losses, name of the flow and other redundant information. At this stage, we can start the actual work related to packet loss prediction. Note that in the following stages, we merge datasets from both applications all together as an integrated dataset and work with it instead of using them independently to distinguish a specific application. This is because Webex and Jitsi both refer to

[4]A time bin of 500 ms represents a data sample with the statistics calculated over 500 ms, which means in 1 second, we have 2 samples.

RTP as the base network protocol for real-time communication so that they share similar characteristics, and another reason will be illustrated in paragraph 3.2.1.

Description	Value
Number of *pcap* files	70 (Webex: 21, Jitsi: 49)
Total duration [hour]	66.51 (Webex: 20.81, Jitsi: 45.7)
Average duration [minute]	57.83 (Webex: 59.45, Jitsi: 57.13)
Maximum duration [hour]	Webex: 2.25, Jitsi: 2.11
Minimum duration [second]	Webex: 80.5, Jitsi: 103.5
Start time of data collection	2020-04-17 07:59:45
End time of data collection	2021-01-18 15:03:18
Maximum number of participants	6
Minimum number of participants	2

Table 3.1: Summary of the data collection

3.2 Data preprocessing and packet loss characterization

After the operation of *Retina* with the complete function of obtaining the number of packet losses, over all the *pcap* files, we get multiple raw datasets, which are not ready for the implementation of machine learning methods because we still need to perform data preprocessing and packet loss characterization to have an integrated and well-structured dataset. In general:

- **Data preprocessing** is a typical procedure during machine learning development or data mining to eliminate errors, remove irrelevant information, select useful segments, and so on. Normally, it is composed of data cleaning, data transformation, and data reduction. In our case, we have in raw data redundant information about packets sent from our own end, missing values due to inconsistent timestamps, different scales of values, complex situations for flows, etc., which require a series of preprocessing steps that will be discussed in detail in the following.
- **Packet loss characterization** is actually the analysis and characterization of time bins[5] with loss, and the comparison with respect to time bins without

[5]In the following, time bins with loss or without loss may also be expressed as "loss" time bins or "no-loss" time bins for simplicity. Similarly, for datasets of time bins with loss or without loss, simple expressions are "loss" dataset/data/sample/set or "no-loss" dataset/data/sample/set.

loss. It is the key process to help analyze as well as understand packet loss phenomena in the network for our particular RTC applications and gain useful and meaningful features that can describe the behaviors of packet loss and be utilized for machine learning approaches. We will conduct overall analyses for all datasets and comparable characterizations in an average manner.

Moreover, the characterization of the exact time bin with loss is not useful because we are trying to predict the presence of packet loss in a future time bin whose information is not currently available, so that we cannot correlate the knowledge of the time bin with loss with the status of the time bin in the future. In other words, what we care about is the characteristics of time bins right before the loss, which can be utilized to derive the trends and features of the RTP flow as we are approaching the loss, and it is exactly based on this tendency that we can make predictions using machine learning methods. However, the overview of time bins with loss across all the RTP traffic in terms of distribution, quantity, statistics, and so on, is significant to help understand the properties of packet loss and reveal the difficulties for prediction.

Technically, in order to forecast the potential packet loss in the future by making use of statistics in the past, we need to find a way to differentiate the behavior of statistics. On one hand, the statistics are supposed to have a steady state when all the packets arrive successfully, while on the other hand, the packet loss will have an impact because when it comes to the calculation of statistics in a time bin, a missing packet could lead to a bias. For example, it enlarges the interarrival time between two adjacent packets that ought to have a packet/packets in between and eventually influences the average interarrival time of the time bin. Moreover, as mentioned before, packet loss can be considered as an indicator of network congestion since it is the major source of loss phenomena, and congestion is not an instantaneous event but a cumulative process that also influences the packets' successful arrival and consequently affects the statistics of them. For instance, when congestion occurs, the affected router in the network starts to accumulate packets in its buffer and further induces an increasing delay for each packet that is about to arrive and waits in the buffer. This kind of situation will eventually lead to the variation of arriving packets until the congestion is solved. That is to say, the network is experiencing a progressive and continuous change as we are approaching the packet loss, which will be reflected in the changing of statistics of received packets not only near the packet loss but also from a relatively long time ago, while on the contrary, without the influence of loss, the statistics of received packets are calculated taking into account all the packets without a critical variation over time. To sum up, the characterization of packet loss is actually the process of finding the trend of the statistical

changes so that we are able to tell apart a "loss" time bin from a "no-loss" time bin and exploit certain types of statistics with an evident tendency for machine learning approaches.

Data preprocessing is blended with packet loss characterization, which helps make decisions, and the overall architecture is in a flowchart (Figure 3.2) divided into 4 major steps:

1. Process for each file;
2. Process for each flow;
3. Process after removing missing values;
4. Final dataset generation.

In the following, we will follow them step by step to check out the packet loss distribution, characterize packet loss, select features, and finally, derive the dataset that can be directly used during machine learning method development.

3.2.1 Process for each file

The first step is a process for all the *csv* files containing raw data of RTP traffic, and at this stage, a preliminary analysis of packet loss is performed as well.

Preprocessing

First of all, we need to read data from all *csv* files to generate seventy data sheets, and each of them contains a timestamp, all types of statistics, number of packet loss, flow name[6] and some redundant information for each time bin.

[6]A flow name is a combination of SSRC, IP address of source, IP address of destination, port of source, port of destination, and payload type. The column is named "flow", which is enough to distinguish flows so that we can get rid of the rest of the columns like "p_type", specifically describing each entity. For example, *('0xd38ab074', '192.168.152.62', '192.168.152.144', 51066, 58205, 111)*, "0xd38ab074" is SSRC, "192.168.152.62" and "192.168.152.144" are IP addresses, "51066" and "58205" are ports, and "111" is payload type.

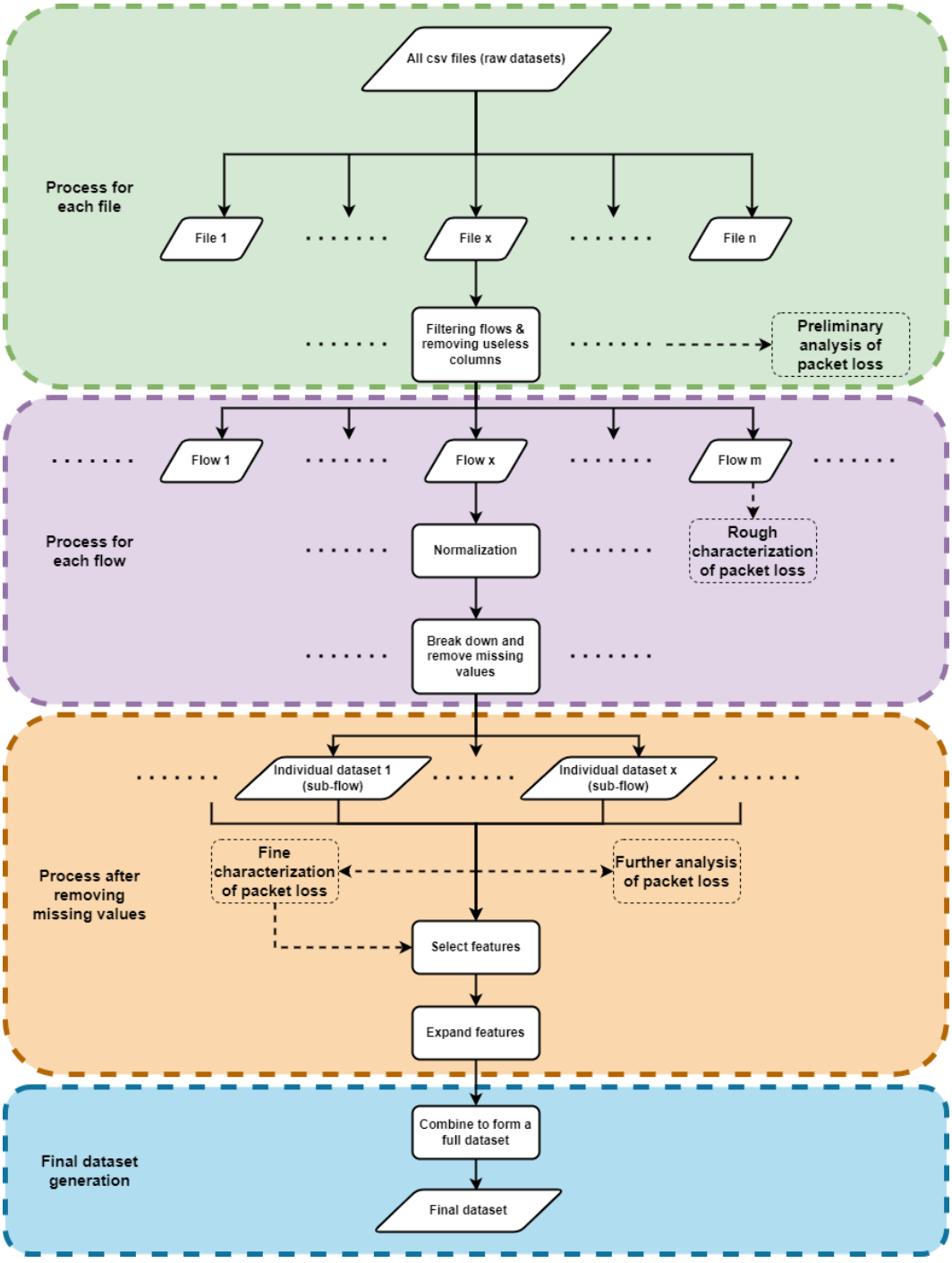


Figure 3.2: Flowchart of data preprocessing and packet loss characterization

Besides the removal of useless columns, including the name of the software, label for audio or video, IP address for source or destination, port for source or destination, and so on and so forth, the most important process is to eliminate the flow generated from the end of observation. As mentioned at the end of section 3.1, the flow with the source being on our own side does not have any losses when the traffic collection is done on our side. The packets produced by our end will be sent out and potential losses will only occur after the delivery, which cannot be detected based on our observation.

The way to tell apart the flow of our own end from multiple ends is based on the IP address of the source. If the source IP address of a flow starts with "192", it means this flow is transmitted from our end to the other end of the communication because in modern network communications, network address translation (NAT) is applied to avoid the need to assign a new IP address to every new host and solve IPv4 address exhaustion. Through NAT, an IP address will be translated into another common address when a packet is transmitted from the local network to the outside, and when a packet is received at the gateway of the local network, the IP address of the source is a translated version, which is totally different. However, in our case, there are flows with source and destination IP addresses that both begin with "192," and this is simply because the flow is a peer-to-peer communication in which both sides of the flow are in the same local network. Under this special circumstance, we also need to eliminate such flows because there is no way to know which one is generated from our end and packet loss is also unlikely to occur in a local network.

Additionally, we also remove flows with a total length (duration) smaller than or equal to 5 seconds (number of time bins smaller than or equal to 10), and the reason will be presented in the later stage.

As a consequence, by filtering the flows and removing useless information for each file, we complete the first step of data preprocessing and obtain multiple meaningful RTP flows for each file. In general, we have 1217 flows in total that are transmitted from outside for 70 RTP traces, and based on these flows, we can get an overview of packet loss.

Preliminary analysis of packet loss:

The preliminary analysis of packet loss will take into account quantity, distribution, situations for both applications, etc.

Overview of overall quantity To start with, an overview of the total quantity for all flows can help understand how RTP traffic is affected by packet loss and how many losses we have. Before that, we need to define two terms:

- A **sparse loss** means a time bin with loss and in its neighbourhood of 5 seconds (10 samples before this time bin and 10 samples after), there's no more time bin with loss.

- A **concentrated loss** means a time bin with loss and in its neighbourhood of 5 seconds, there's at least one more time bin with loss.

A summary of six types of quantities is in Table 3.2. Time bins with loss only occupy 1.52% of the total quantity, which is a very small amount with respect to time bins without loss. However, the small percentage does not indicate a good quality of communication because, on one hand, the losses are not equally distributed over all flows, and for some flows, we encounter a lot of losses compared to other flows. On the other hand, around three-fourths of all the time bins with losses are concentrated losses, which means losses tend to occur at the same time, resulting in an aggregation of losses because the causal phenomenon, congestion, induces losses at certain times and the effect will linger for a while, keeping generating losses. This kind of burst of losses significantly affects communication. Moreover, regarding the actual number of packets sent, most of them are successfully received and only 0.36% are lost, which is a good performance, but it is still an issue if the lost packets carry critical information. The analysis of overall quantity gives us an overview of the RTP traffic, but this is not enough and we need to dive into details regarding both applications and packet loss distributions.

Description	**Quantity[-]**	**Percentage[%]**
Total time bins	2137727	-
Time bins with loss	32472	1.52
Sparse loss	7374	0.34 (22.4)
Concentrated loss	25098	1.18 (77.6)
Total packets	59393645	-
Total lost packets	216735	0.36

Table 3.2: Summary of quantities for all flows

Overview of quantity for each application Although both Webex and Jitsi rely on RTP for real-time communication and share similar characteristics for statistics, it is still important and necessary to investigate the distribution of packet loss for each of the applications.

In the first place, a bar chart in Fig.3.3 shows the total quantities for "loss" and "no-loss" time bins of each application, and from it, Jitsi has more data than Webex, which coincides with the fact that Jitsi has more RTP traffic collections of 49 files out of 70, and this also holds for the quantity of time bins with or without loss for both applications. On top of that, both applications possess an awfully similar percentage of around 1.5% for time bins with loss, which indicates that the "loss" time bins are distributed identically for both applications and rarely occur with respect to "no-loss" time bins.

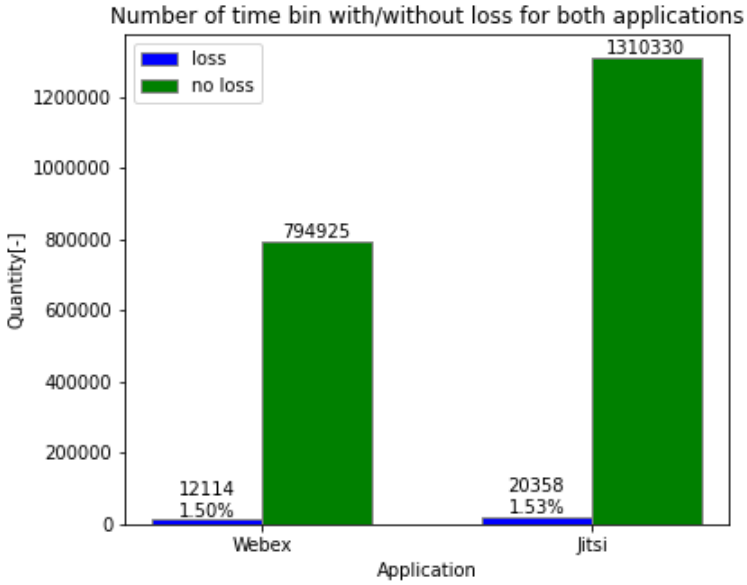
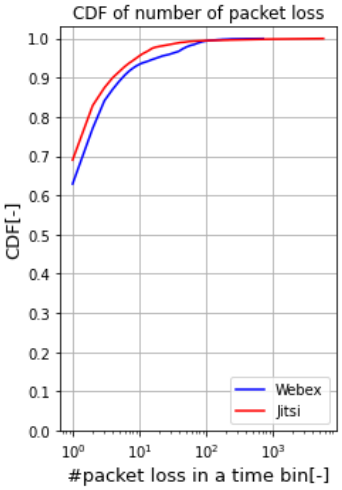


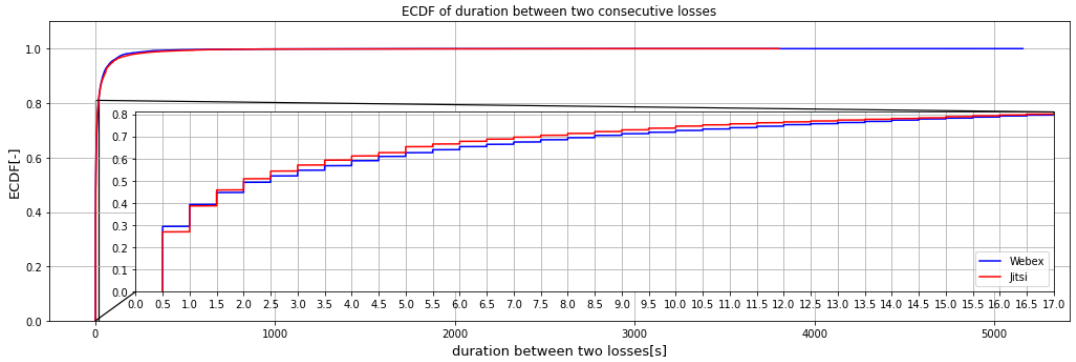
Figure 3.3: Quantities of time bins with or without loss for Webex and Jitsi

Moreover, the study of the quantity of time bins is not adequate and in order to further comprehend the packet loss phenomenon for each application, the cumulative distribution function (CDF) of the number of packet losses in a single time bin and the empirical distribution function (ECDF) of duration between any consecutive "loss" time bins[7] for both applications are presented in Fig.3.4. On one hand, based on Fig.3.4a, around 65% for Webex and 70% for Jitsi of time bins with loss only

[7]Consecutive "loss" time bins means two time bins where there's no loss at all in between.



(a) CDF of number of packet loss in a time bin



(b) ECDF of duration between consecutive "loss" time bins

Figure 3.4: Detailed information of packet loss for both applications

have 1 packet loss in 500 ms, while most of the "loss" time bins (around 95%) for both applications have a small number of packet losses less than or equal to 10, which means in most cases, regardless of the application, we have a small number of packet losses. Meanwhile, we also have very few cases of "loss" time bins with packet losses greater than one hundred, and especially for Jitsi, certain time bins even have packet losses of more than one thousand, which is possibly due to unexpected Internet disconnection, software issues or equipment failure. We do not exclude this data because: (i) they are very few and they are still losses after all; (ii) the exact cause remains unknown. On the other hand, according to Fig.3.4b, in general, around 80% of the durations between adjacent "loss" time bins are shorter than 17 s and, in some extreme cases, we have durations greater than

1000 seconds, indicating a long RTP session without any loss and a good quality of communication. In particular, zooming in the figure, around 30% of durations are 0.5 s, which means the subsequent "loss" bin out of the consecutive bins comes right after the previous one, and 60% are less than 5 s, which demonstrates that larger amounts of losses are aggregated over a time window of 5 seconds. Additionally, there are 36,309,637 packets sent and 135,930 lost, occupying 0.37% for Jitsi, while there are 23,089,095 packets sent and 81,040 lost, occupying 0.35% for Webex, which is similar to the previous percentage of 0.36% for the entire dataset.

In a word, despite the fact that Webex has relatively a few more packet losses in a time bin according to the CDF, both applications share almost the same distribution. Together with the analysis in terms of quantity, although Webex and Jitsi are two completely different applications, the circumstances of packet loss phenomenon for both of them resemble each other very much, which proves that it is reasonable to put all the datasets together.

Based on the preliminary analysis of packet loss, we gain an elementary understanding and some intuition about the packet loss phenomenon in our specific case. On top of that, each file is broken down into several RTP flows that are ready for further processing and analysis. Note that from now on, all the data processing, analyses, and the subsequent predictions are based on flows instead of the entire set of packets in a certain call. Although multiple RTP flows may stream over the same time, which means the prediction along a time series may make more sense by taking into account all the previous information about packets regardless of the source or type, it is still feasible and reasonable to deal with flow by flow because:

- The streaming of different RTP flows may overlap over a certain time duration, but the start or the end of each flow is different in most cases, which means the number of flows in a certain time bin may be different from another one, and thus the quantity of features (statistics) we can retrieve varies along the time and cannot feed a unified machine learning model with a fixed input shape.

- Another possible way to consider all the flows is to average the behaviors in a certain time bin. But, on one hand, the averaging of different flows will introduce more errors and loss of information. On the other hand, different flows have different scales of values for statistics, and in order to average them, a normalization process is necessary to reduce the scale, but the normalization of values in one time bin is not adequate since the packets from a flow in 500 ms cannot represent the overall characteristics and reflect the actual range of values of the flow.

- *Retina* calculates statistics based on each flow, and thus the combination of multiple flows will reduce the effect of packet loss over statistics with respect to an individual flow, which eventually leads to an ambiguous result.

3.2.2 Process for each flow

The second step is based on data about all the flows derived above, and at this stage, we can take a glance at the trend of statistics with respect to packet loss in flows to give a rough characterization of packet loss.

Preprocessing

The prediction is for the presence of packet loss instead of the quantity of packet loss, so first we convert the quantity of packet loss in a time bin to the existence condition of packet loss for the time bin. In other words, we create a binary label for each time bin (each data sample or each row in the data sheet) for each flow in our dataset: *0* represents time bin without loss ($number\ of\ packet\ loss\ in\ a\ time\ bin = 0$) and *1* represents time bin with loss ($number\ of\ packet\ loss\ in\ a\ time\ bin > 0$).

Secondly, a normalization process is needed due to the fact that statistics from various flows may have different numerical scales. For example, a flow of medium-quality video has a bit rate of around 2500 kbps while a flow of audio has a bit rate of around 200 kbps. Normalization is the process of modifying values on different scales to a notionally common scale. By reducing or increasing the scales of different ranges of values to a common scale, issues caused by different scales of values in flows can be eliminated without losing the original trend and distribution of statistics in our dataset. Another potential advantage is that machine learning approaches tend to work better with values of smaller scale. Specifically, for all the values of each type of statistics in each flow, we perform a min-max normalization to bring all values into the range [0,1] and the formula is:

$$x_i' = \frac{x_i - X_{min}}{X_{max} - X_{min}},$$

in which x_i' is the normalized value of a certain original statistic value x_i, and X_{max} and X_{min} represent the maximum and minimum value of a certain type of statistic X in a flow.

Finally, there's still another critical issue about missing values. In our dataset, each flow does not stream continuously along time, which means our dataset for

each flow cannot guarantee the continuity in terms of timestamp and the duration between two adjacent time bins might be larger than 500 ms. In other words, the information between these kinds of contiguous time bins is missing. Without taking care of such a problem, both analysis and prediction of packet loss will definitely lead to erroneous results because the dependency of previous statistics for prediction may include values from dozens of seconds or even several minutes ago, which are not informative and not correlated with current conditions at all[8]. In order to solve the problem, one possible solution is to replenish all the missing values by interpolation based on the position of the missing values with respect to "loss" time bins. For example, if a missing value is 3 seconds before a time bin with loss, we can fill in the void with the average value of all the time bins with a "loss" time bin in the next 3 s. However, interpolation may not be appropriate in our case because: i,) discontinuity in flows is very common in reality, which means frequent replenishment with interpolation cannot represent the actual behaviour; ii,) interpolation will introduce errors and uncertainties due to averaging; iii,) interpolation may lead to results that meet our expectations since a lot of time bins are derived based on mean values, which are smoother to eliminate extreme cases. Therefore, we decide to refer to another approach by discarding all the missing values. In order to guarantee the continuity of time bins, we only keep available time bins between two adjacent missing values[9] that have a duration greater than 5 s in between. That is to say, we only retain continuous time bins with sequential timestamps that have a whole duration larger than 5 s. A better illustration can be seen in Fig 3.5 and the reason why we refer to 5 s as a threshold is discussed

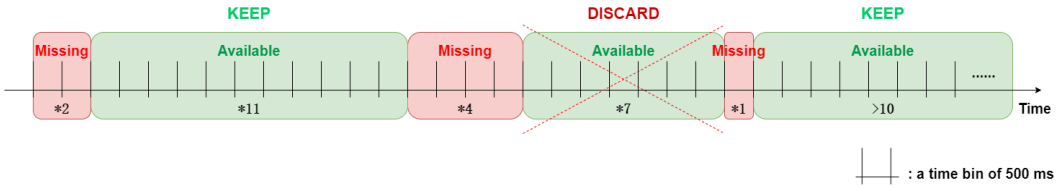


Figure 3.5: The strategy of discarding or keeping data due to missing values

in the next section. Apparently, based on the removal of improper time bins, we break down the original flow into multiple individual sub-flows with continuous

[8]For example, if we refer to the past 3 seconds (6 samples) to make a prediction and the timestamp for each time bin in ascending order is: "2022-05-17 09:09:40.500", "2022-05-17 09:09:45.500", "2022-05-17 09:09:46.000", "2022-05-17 09:09:46.500", "2022-05-17 09:09:47.000", "2022-05-17 09:09:47.500", we will encounter an unexpected duration (the information in this duration is missing) of 5 s between the first and second timestamp and include a value from 7 s ago, which is supposed to be a value from 5 s ago.

[9]Two missing time bins that should have existed and there's no more missing values in between.

time bins lasting more than 5 s (the number of time bins greater than 10 samples) as a whole, and from now on, they are ready for further analysis for consecutive time series and feature selections without worrying about the missing values.

Rough characterization of packet loss:

The information about time bins from the near future is unknown, and in order to predict the presence of packet loss, the available resources in possession are statistics for each time bin in the past, so it is necessary to ascertain the types of statistics that can be utilized to characterize the time bin with packet loss. More precisely, the characterization of packet loss is not the description of the loss itself but actually the trend analysis of statistics in a certain number of time bins right before the target time bin in the future. As a result, it allows us to distinguish the future time bin with or without loss using machine learning approaches based on different trends of statistics in the past.

Based on previous preprocessing, we derive the dataset for each flow, and in this section, we focus on the rough characterization of packet loss in terms of certain random flows and some random bins to cast a glance at some samples and obtain a fundamental understanding of statistical trending. Prior to this, the time window of study must be defined in order to determine the number of statistics in time bins for analyses and subsequent predictions. In our case, we decide to refer to a length of 5 seconds[10] because it is neither too long to include useless information from a relatively long time ago that has nothing to do with the future target time bin, nor too short to lose potentially helpful and informative values, leading to an inadequate quantity of features, not to mention that an adjustment of the time window to a shorter duration is also available if it's needed during the machine learning approach development. This is also the reason why we discard the data between adjacent missing values if the length is not greater than 5 seconds (10 samples). Additionally, with a duration less than or equal to 5 seconds, we do not have a time window of 5 seconds but even in the worst case scenario, when the length is only 5.5 seconds (11 samples), we still have at least one sample at the end that can be used since a 5 s-long time window of study before it with continuous time bins is available. Moreover, for the sake of characterization of packet loss, we also need to distinguish between different conditions of packet loss and make comparisons with respect to the behavior of time bins without loss, because the

[10]A time window of 5 seconds right before the target time bin with or without loss consists of 10 time bins (10 data samples) of 500 ms.

sparse loss and the concentrated loss both have a significant amount that cannot be ignored. In particular, we define three cases in the following:

1. **Case 1:** The five-second time window with no loss at all is located right before a time bin with loss. In other words, a "loss" time bin with no loss in the past 5 seconds. In this case, we want to know the effect of packet loss on statistics in the past 5 seconds without the interference from other losses.
2. **Case 2:** The five-second time window with at least one "loss" time bin in it is located right before the time bin with loss. In other words, a "loss" time bin with a loss or losses in the past 5 seconds. In this case, the amplification of effect due to extra losses in the past 5 seconds will be indicated.
3. **Case 3:** The five-second time window has no loss in it at all, and in its neighborhood of three seconds, there's no loss either. In this case, we want to detect the behavior of statistics without the influence of packet loss, and the introduction of a neighborhood time window without loss will eliminate the impact of a possible "loss" time bin on the head or tail of the time window of study to ensure the reliability of the evaluation of statistics.

A graphical illustration is in Fig.3.6, and the same strategy will be applied also for the fine characterization of packet loss in section 3.2.3.

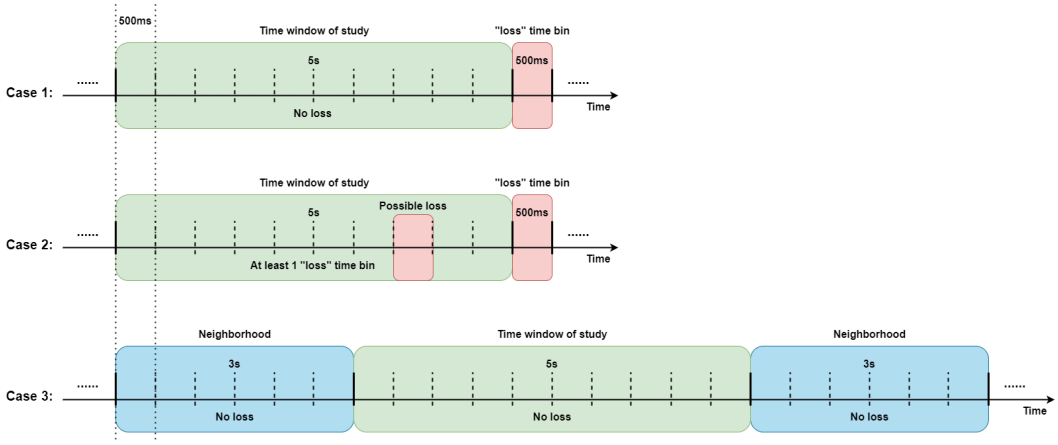


Figure 3.6: Three cases of time window of study

Overall behaviour of losses in a flow First of all, it is not harmful to check the global patterns of statistics with respect to packet loss for a flow to take a peak

at packet loss from a holistic perspective. Here, we refer to two random flows and present the behavior of bit rate, which is the number of bits that are conveyed per unit of time. As shown in Fig.3.7, the top figure indicates that the bit rate experiences some dramatic changes around packet loss while the normal cases range from 10 to 30 kbps, and from the bottom graph, the missing values discussed in section 3.2.2 are confirmed by the discontinuity at the beginning, and the changes in bit rate are irregular with unpredictable behavior. Although it provides a brief overview of the flow, the overall behavior cannot assist us in making predictions because we refer to a much smaller scale of time duration, which requires elaborate analyses focusing on the time window of study defined above. Thus, we do not perform additional comparisons for other types of statistics.

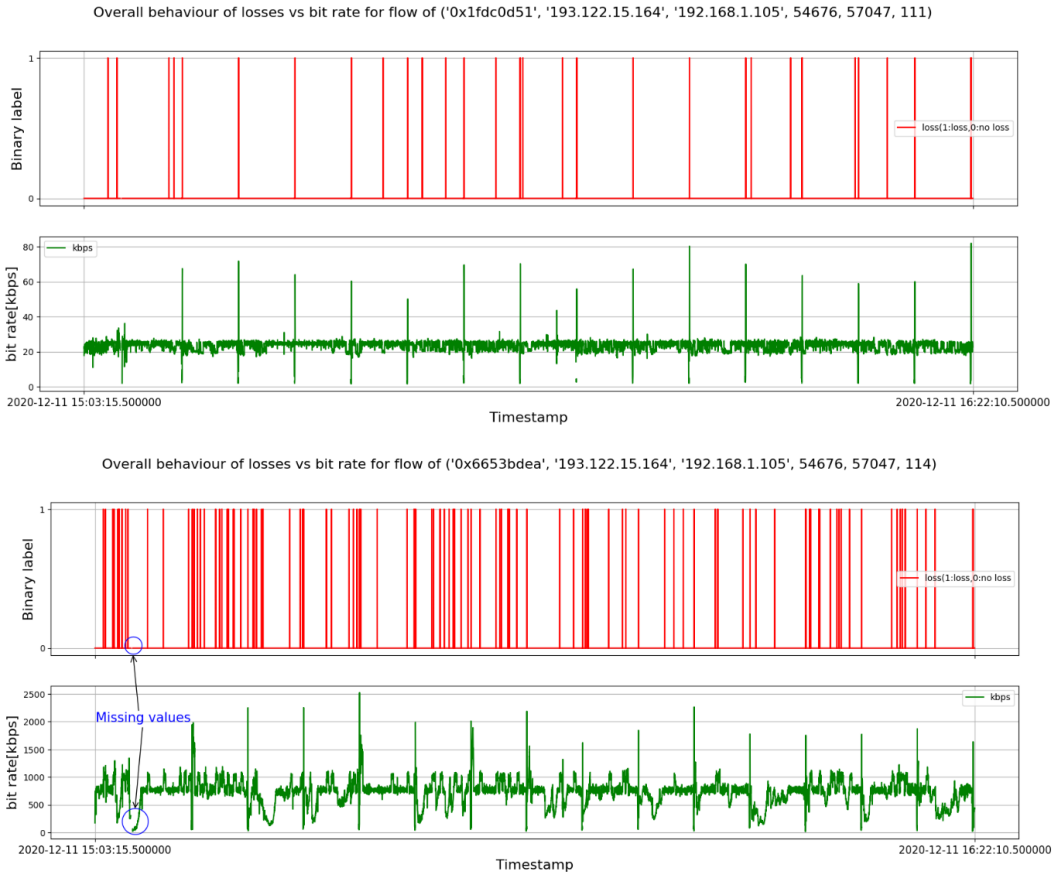


Figure 3.7: Patterns of bit rate and packet loss for two random flows

Behaviour of a sampled loss Secondly, by focusing on the time window of study, we extract a random loss to examine the patterns of certain statistics. Note

that in this paragraph, the three cases are not implemented since we focus on a random loss from a random flow regardless of the conditions, and the correlations between statistics and the "loss" time bin are not considered either, so that the categories of statistics are chosen arbitrarily to represent diverse behaviours. In Fig.3.8, the blue line in each graph shows the patterns of the mean value of interarrival time, the kurtosis of interarrival time, the bit rate, the percentage of unique UDP length, and the summation of the number of RTP markers, and the red lines indicate the packet loss condition for each time bin where the peak stands for a "loss" time bin. The mean values of interarrival time experience an irregular change at the beginning and a gentle increase as approaching the loss, whereas the kurtosis indicates a drop near the loss followed by an abrupt rise. Meanwhile, both the bit rate and percentage of unique UDP length have similar behaviour, with a valley around 1.5 seconds before the loss. Moreover, the quantity of RTP markers doesn't change with respect to the packet loss.

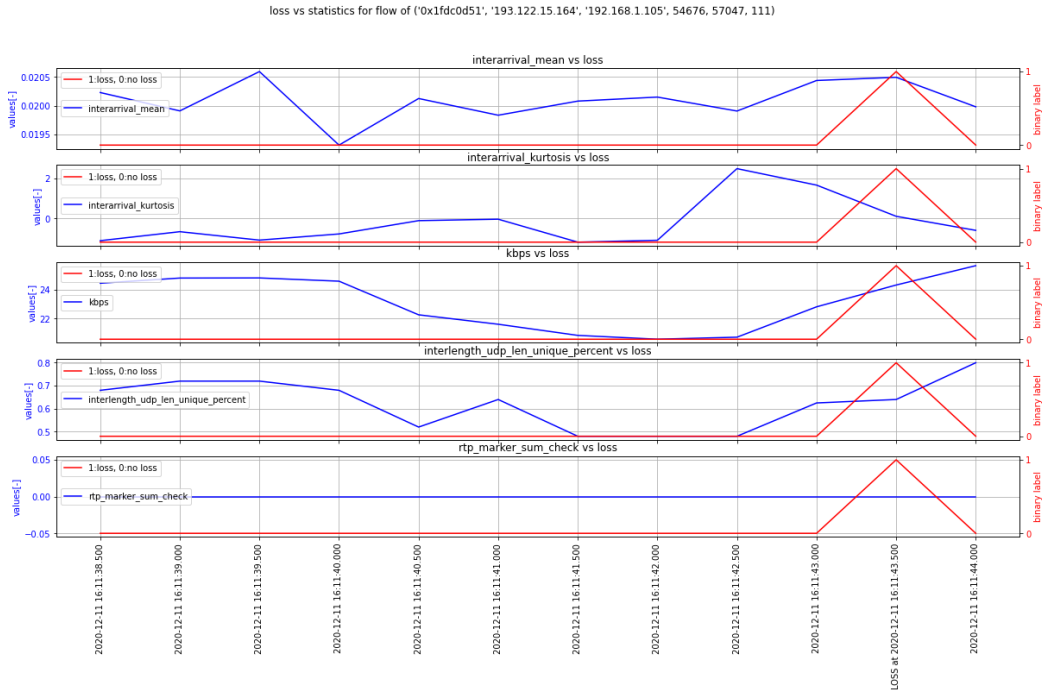


Figure 3.8: Patterns of 5 types of statistics with respect to a random loss in a random flow

Based on the aforementioned analyses, we can confirm the variation of statistics that meet our expectations and can be utilized to identify time bins with loss. Furthermore, they also give us an intuition of which statistics could be considered

features and which should be neglected. However, this is just the behavior for one "loss" time bin, which absolutely cannot represent the whole flow, so in the following, we will average the behaviors, taking into account the entire flow to grasp the overall patterns and referring to the three cases of time window to make comparisons.

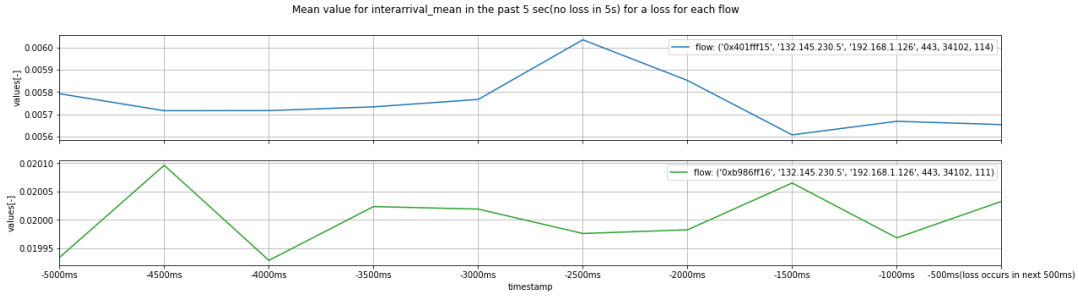
Average behaviour of losses Finally, averaging[11] all the statistics of each time bin in the target time window of study in a flow regarding three different cases can reflect the behavior of the "loss" time bin from an overall point of view to cover and represent various patterns as much as possible. In this paragraph, we look over two example statistics[12] from two random flows, distinguishing the three cases.

On the one hand, Fig.3.9 illustrates the patterns of the mean value of interarrival time in two flows for the three cases. According to Fig.3.9a and Fig.3.9b, the mean values of interarrival time experience irregular changes with distinct behaviors regardless of flows or cases. Although Fig.3.9c indicates a stationary result for both flows when there's no packet loss, which can be proved not only by the average value with tiny and negligible variations but also by the low standard deviation, while both of the other two cases show various degrees of fluctuations, the mean value of interarrival time might not be a good choice for identifying "loss" time bins and making predictions because it doesn't share a common pattern across different flows and cases, leading to equivocal results.

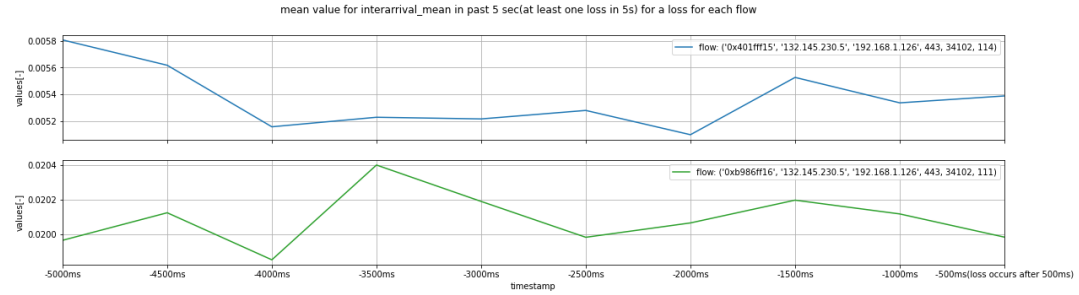
On the other hand, the same types of plots for the fourth moment of UDP inter-length are presented in Fig.3.10. Firstly, based on Fig.3.10c, the same behaviour of steady state for "no-loss" time window still holds with a rather lower standard deviation. Secondly, Fig.3.10a shows that we still cannot find anything in common with case 1. However, a similar trend of increasing in advance and decreasing afterwards is illustrated in Fig.3.10b when there are extra losses affecting the time

[11]So-called averaging means that we examine all the time bins in a flow or a sub-flow and, for a certain type of statistic, calculate the mean value of all the values in the corresponding time bins that meet certain specific conditions (the case of the time window of study and the location). After that, such a mean value of the target statistic represents the average behavior of the corresponding time bin in the corresponding time window of study. Note that, in the following fine characterization of packet loss, we will also refer to this process to comprehend the overall patterns of statistics in the time window of study by taking into account all the time bins in the entire dataset instead of an individual flow.

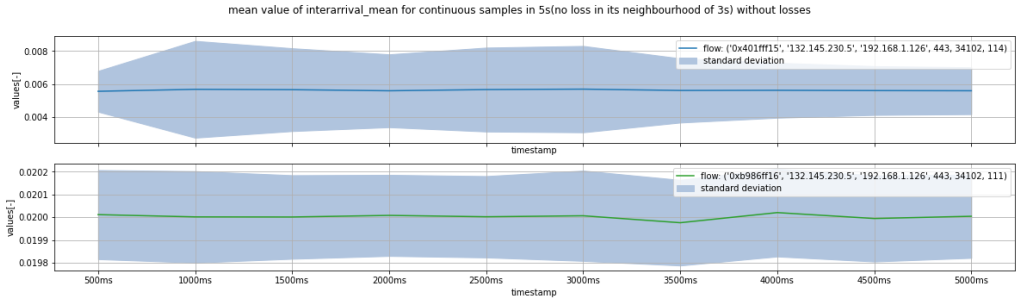
[12]We do not perform more analyses and explain the meaning of the statistics simply because the average behaviors of the entire dataset instead of certain flows are more important and meaningful, which will be performed in section 3.2.3.



(a) Case 1



(b) Case 2

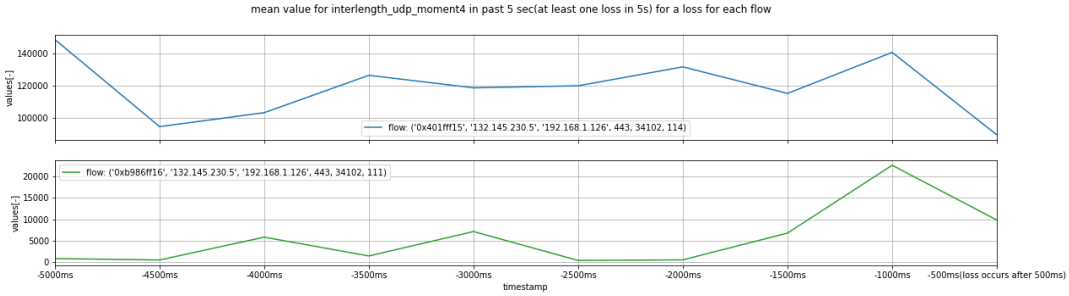


(c) Case 3 (with standard deviation)

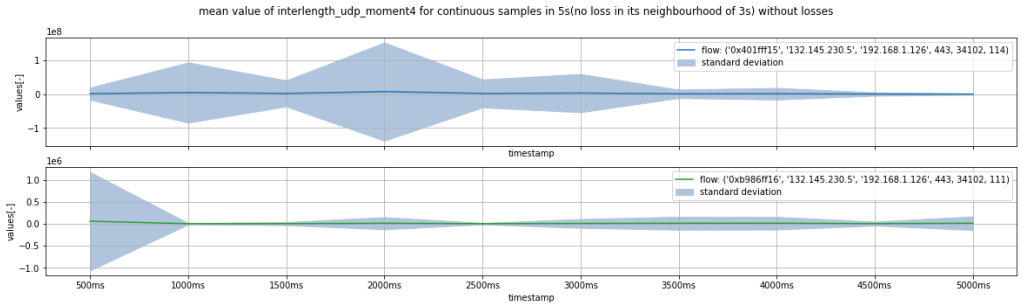
Figure 3.9: Average behaviour for the mean value of interarrival time in the time window of study for three cases

window together with the loss at the end, which means the fourth moment of UDP inter-length might be helpful to recognize the packet loss due to the identical pattern and might be possible to work as a feature for prediction since around 70% of "loss" time bins are concentrated losses with similar conditions to Case 2.

(a) Case 1



(b) Case 2



(c) Case 3 (with standard deviation)

Figure 3.10: Average behaviour for the fourth moment of UDP inter-length in the time window of study for three cases

The average behaviors for random flows are not enough because we have thousands of flows in the entire dataset and a random one is not representative and informative enough. At this stage, by removing missing values, we break down each flow into multiple individual datasets (sub-flows), which are ready for the most significant process to finally derive the useful features.

3.2.3 Process after removing missing values

The third step is the process for datasets without missing values. Through previous preprocessing, we have in our possession a great number of individual datasets with sequential time bins without discontinuity for all sub-flows. Based on the new datasets, we can perform more analyses regarding the packet loss to further comprehend its distribution and condition, and eventually be able to characterize the packet loss based on the average behavior among all individual datasets. In the end, proper feature selection and elaboration will be conducted based on the fine characterization to generate the final dataset.

Further analysis of packet loss

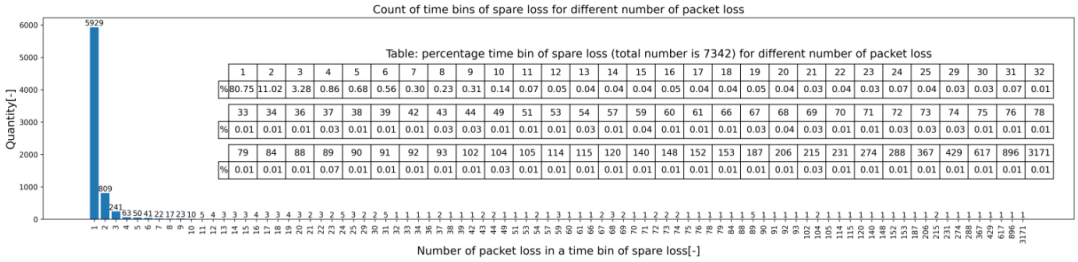
After the removal of missing values, we inevitably lose some data due to the inadequate duration of continuous time bins. As a result, we have 2098640 time bins left and 30504 of them are time bins with losses, of which 7342 (24.1%) are sparse losses and 23162 (75.9%) are concentrated losses. Additionally, a summary regarding the number of packet losses is in Table 3.3 and it almost coincides with the CDF derived in paragraph 3.2.1. Furthermore, the further analyses focus on the number of packet losses in different types of "loss" time bins and the distribution of "loss" time bins in terms of the time window of study.

Number of packet loss in a time bin[-]	**Number of corresponding time bins[-]**	**Percentage[%]**
1	21019	68.91
2	4162	13.64
3	1657	5.43
4	783	2.57
5	491	1.61
6	384	1.26
7	247	0.81
8	177	0.58
More than 8	1584	5.19

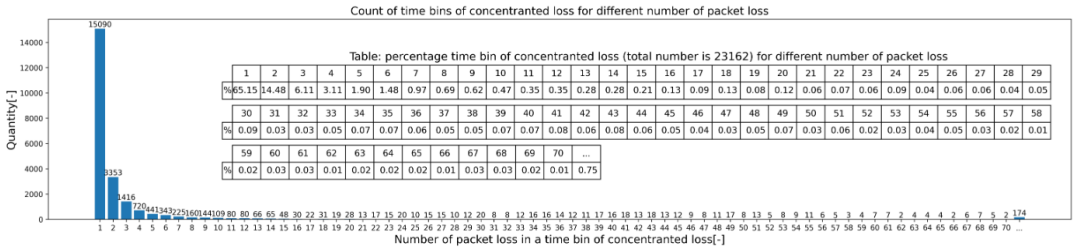
Table 3.3: Summary of number of time bins for different quantities of packet loss after removing missing values

Number of packet loss in time bins of sparse or concentrated loss For sparse and concentrated loss defined in paragraph 3.2.1, we want to know the correlation between the quantity of packet loss and the loss category. The bar chart

in Fig.3.11 show the quantity of time bins with different numbers of packet losses for sparse and concentrated losses, and the tables present the related percentage. One on hand, for both types of losses, most of the packet losses in a time bin range from one to ten, which is the same as the CDF in paragraph 3.2.1 with around 95% of packet losses in time bins being less than or equal to 10. On the other hand, according to Fig.3.11a, 80.75% of time bins in sparse loss only have 1 packet loss, and this is greater than the percentage (68.91%) of the same type of time bin in the whole dataset, which means the time bin of sparse loss tends to have a lower amount of packet loss. At the same time, according to Fig.3.11b, the number of time bins with only one loss occupies 65.15% which is less than the quantity in the whole dataset, and the percentages for time bins with more losses become larger with respect to sparse loss, which indicates that a concentrated loss time bin is inclined to encounter more losses. The reason for this could be that the concentrated loss is caused by a more severe congestion or other stronger network problems that have a larger impact and last longer to cause more packet losses, whereas the minor congestion or problem accounting for the sparse loss is less detrimental.



(a) Sparse loss

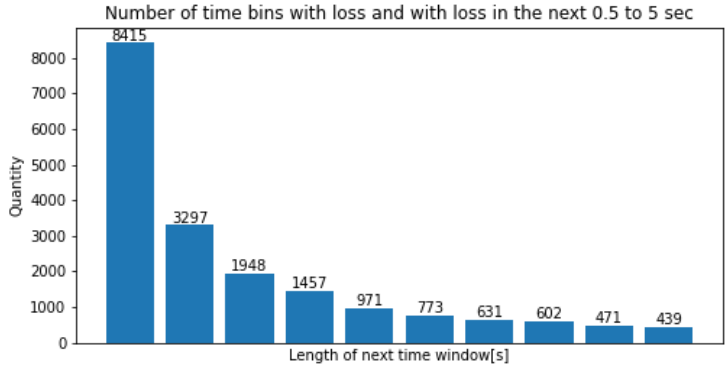


(b) Concentrated loss

Figure 3.11: Bar chart with percentage table of quantity of time bins for different types of loss and different numbers of packet loss

Distribution of time bin with loss for the time window of study For the time window of 5 seconds, we want to know the loss condition in it after

encountering a loss. Keep in mind not to confuse the definition with respect to concentrated loss, since concentrated loss needs to have at least one packet loss in its neighborhood, while in this case, the existence of previous losses is ignored. Fig.3.12 shows the number of "loss" time bins at different positions in the time window of study after any time bins with loss. Firstly, for all "loss" time bins, there is a 27.59% chance of having another "loss" time bin right after encountering a loss, which means plenty of "loss" time bins are next to each other. Secondly, as the duration between losses increases, we have fewer chances of encountering another loss. Thirdly, all of these types of "loss" time bins with loss at most in their previous 5 seconds occupy 62.3% of all the losses, which is less than the percentage of concentrated loss. Generally speaking, it is very likely to encounter another loss after a loss, which means it is also very likely to include the statistics related to another "loss" bin or even more losses in the characterization of a loss.



	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	Total
%	27.59	10.81	6.39	4.78	3.18	2.53	2.07	1.97	1.54	1.44	62.3

Figure 3.12: Number of "loss" time bins with loss in its following 0.5 to 5 seconds

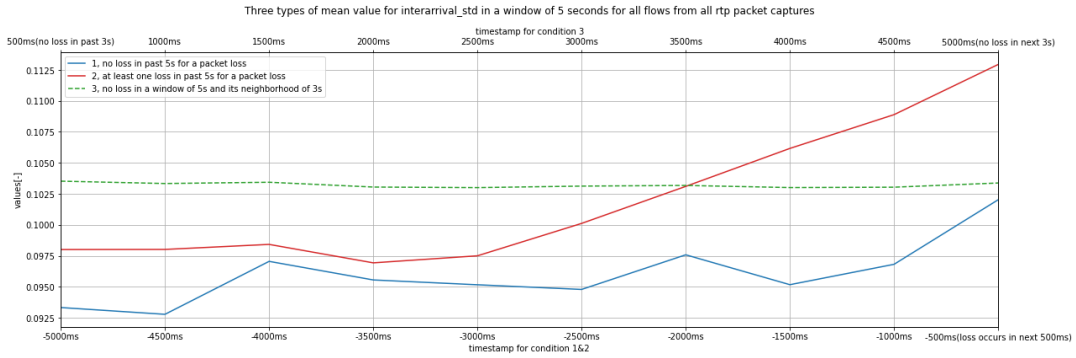
Fine characterization of packet loss

Based on all of the individual datasets without missing values, we perform a fine characterization of packet loss, taking into account the average behavior over the entire dataset, so that for each statistic in the three types of time windows of study, it allows us to capture the overall patterns which can represent the characteristics of most "loss" time bins. Although the averaging process would introduce errors and loss information to some degree, the fine characterization of packet loss is still the key procedure to determine the most appropriate and informative statistics that are strongly correlated with packet loss for prediction.

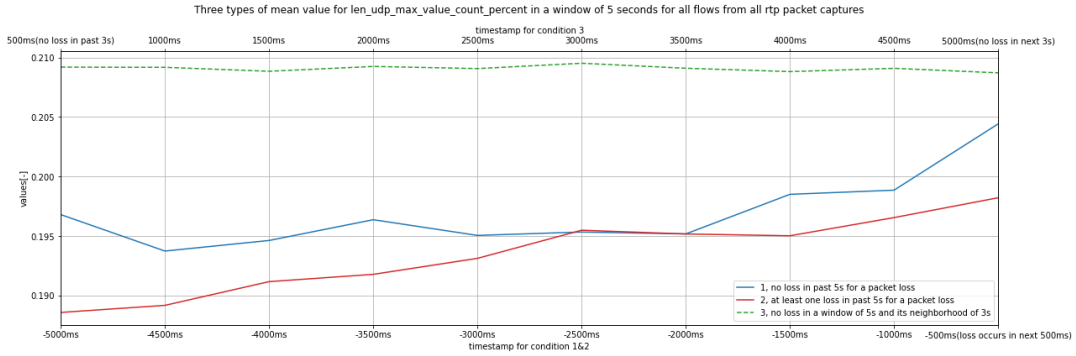
To study the correlations with packet loss comprehensively, we conducted fine characterization for each type of statistics considering different time windows of study, and we will present seven examples of statistics for different degrees of correlation in the following. Note that in each plot, the blue line represents the time window of study for case 1, the red line represents case 2, and the green line represents case 3. Furthermore, each time instance is ordered chronologically, and each value corresponding to a timestamp represents the average value across the entire dataset for the same type of time bin of 500 ms. Additionally, for cases 1 and 2, the packet loss occurs in the time bin 500 ms after the end of the time window, and for case 3, there's no loss either in the time window nor in its neighborhood of 3 seconds.

Strong positive correlation Fig.3.13 shows the average patterns of the standard deviation of interarrival time and the percentage of the quantity of maximum UDP length. Generally speaking, they both have positive correlations with packet loss as approaching "loss" time bin regardless of the window types, while for case 3, both of them present a steady state when there's no packet loss affecting the flow. Due to the strong correlations and evident differences between cases 1, 2 and 3, they can be selected as features for prediction. In particular, the standard deviation of interarrival time in a time bin indicates the degree of fluctuation of interarrival time. As we are close to the "loss" time bin, the effect of congestion or other network problems becomes severe enough to induce larger delays between affected packets with respect to other normal packets. Consequently, we will encounter certain peculiar interarrival times, which eventually lead to a larger standard deviation. Although for case 1 in Fig.3.13a, it experiences a gentle variation from the past 4000ms to 2500ms, the overall pattern still follows an ascending tendency.

Strong negative correlation As getting closer to packet loss, the minimum value of interarrival time for each time bin in the time window possesses a declining trend, which can be proved by Fig.3.14a. Such a phenomenon occurs because the interarrival time tends to be longer and longer because of the persistent effect of congestion as time goes on in cases 1 and 2, and after the normalization process, the minimum interarrival time will have an even smaller scale when we have several larger values. When we are away from packet loss, due to the relatively weak influence of loss, the interarrival time tends to have a homogeneous performance, leading to a larger value of the minimum after normalization even if the absolute value is smaller. Meanwhile, a similar decreasing pattern for the standard deviation of the difference between timestamp and sequence number is illustrated in Fig.3.14b. On top of that, both statistics follow a stable trend for case 3, so that they are



(a) Standard deviation of interarrival time



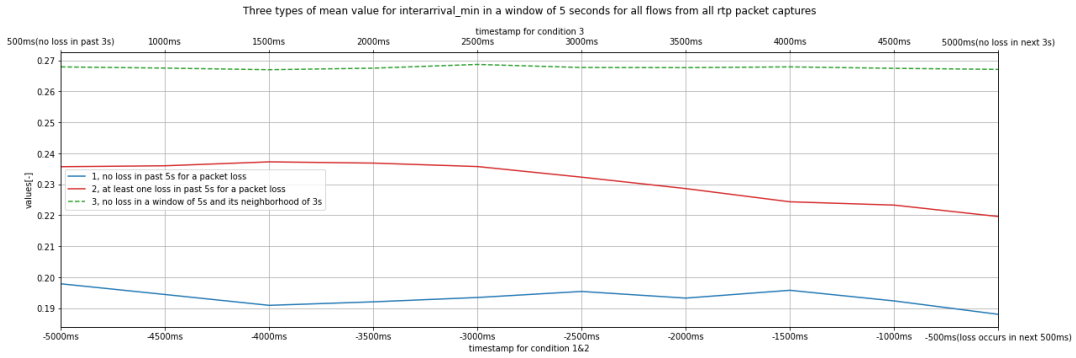
(b) Percentage of the quantity of maximum UDP length

Figure 3.13: Two examples of statistics for strong positive correlation

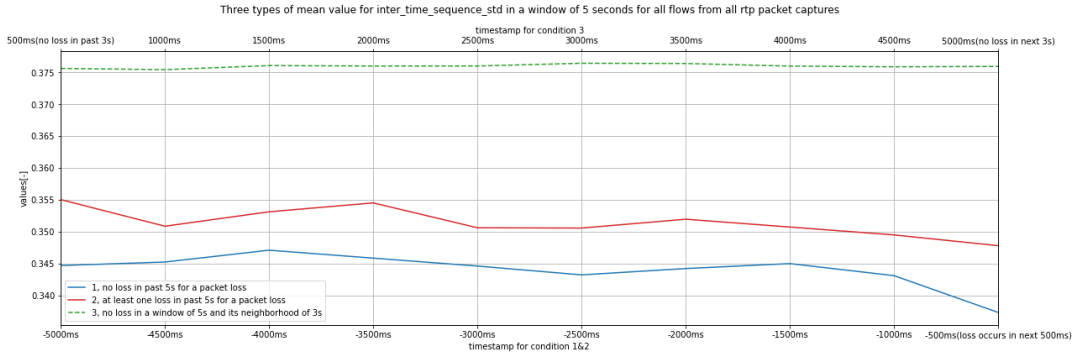
eligible for the feature selection.

Weak correlation Fig.3.15 shows the average behaviour for the maximum UDP length in each time bin. In spite of the slight rising tendency at the end of the time window for cases 1 and 2, most of the patterns share the same condition with respect to case 3, with steady behavior when there's no packet loss, which means the correlation between packet loss and maximum UDP length is not intense and the loss phenomenon can only affect the statistics recently. In this context, referring to maximum UDP length as a feature for prediction is not a wise choice, but it cannot be ignored that the statistics in the past 1000 ms and 500 ms are still informative enough to be selected separately.

Opposite correlation According to Fig.3.16, the mean value of the difference between the UDP lengths of adjacent packets in time bins follows a decreasing



(a) Minimum interarrival time



(b) Standard deviation of the difference between timestamp and sequence number

Figure 3.14: Two examples of statistics for strong negative correlation

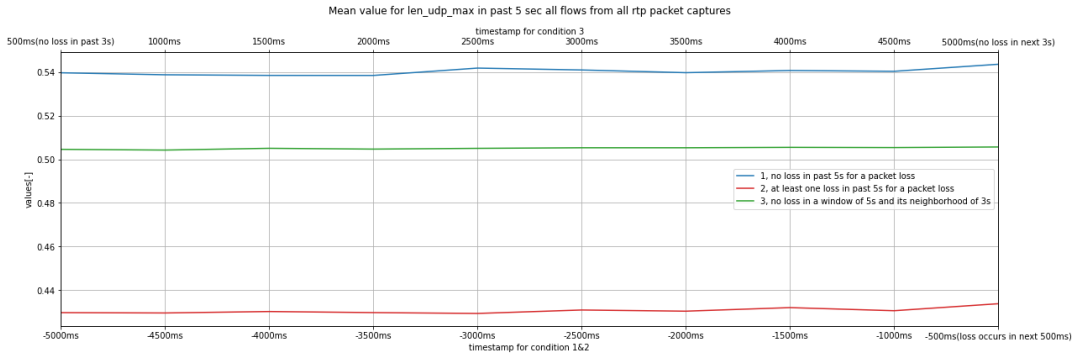


Figure 3.15: Maximum UDP length for weak correlation

trend at the end of the time window for case 1, but holds an increasing trend for case 2 when there are extra losses in the time window. Despite the fact that it has a distinct behavior with respect to a stable pattern in case 3, the mean value of the difference cannot be used as a feature for prediction because there's no way to

distinguish between cases 1 and 2.

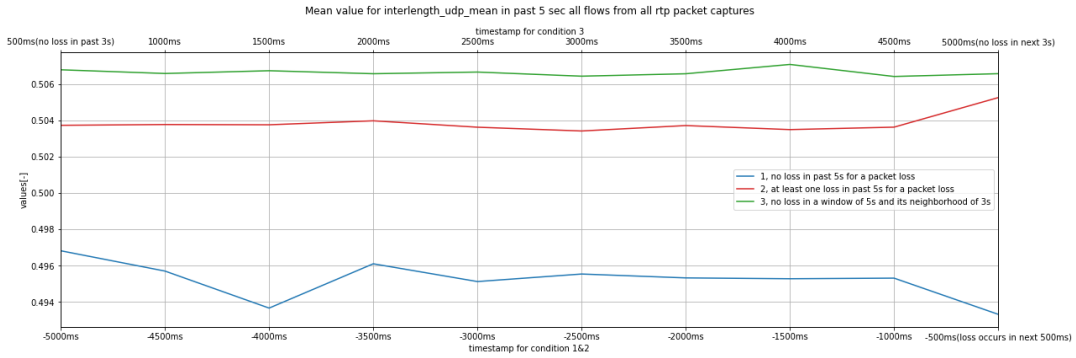


Figure 3.16: Mean value of difference between UDP length of current packet and the previous one for opposite correlations in case 1 and case 2

No correlation The *"interlength_udp_min_max_R"* represents the ratio of the minimum value for the difference between the UDP lengths of adjacent packets, and it tries to capture the importance of the minimum in a non-linear way in a time bin. According to Fig.3.17, it fluctuates a lot with irregular patterns over time for all three cases and thus has nothing to do with the packet loss. Therefore, even though the variation in case 3 is relatively small, it should be completely discarded during the feature selection for prediction.

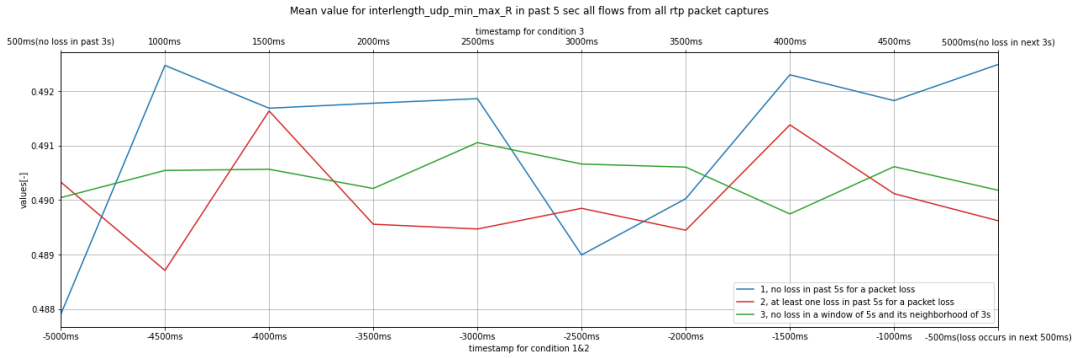


Figure 3.17: Ratio of minimum difference between UDP length of current packet and the previous one for no correlation between statistics and packet loss

Based on the results of fine characterization of packet loss, we have located 15 types of statistics that have obvious patterns, strong correlations with packet loss,

and steady state in case 3 without the impact of packet loss. As a result, we are ready to select the features and generate structured data sheets for each individual dataset in the following preprocessing.

Preprocessing

The preprocessing during the process after removing missing values deals with each individual dataset and transforms them into well-structured datasets with class labels and meaningful statistics, which will be finally integrated into a single large dataset to feed the machine learning methods.

Feature selection The first step in preprocessing is to choose columns that correspond to useful features for prediction based on a fine characterization of packet loss and to remove useless columns with features that do not correlate to packet loss or have only weak relationships.

The process for fine characterization of packet loss has derived 15 features, but there are also correlations among the features themselves, which means if we refer to all of the 15 features, we would include information that is repetitive and needless. In order to remove features that are redundant and reduce the size of the dataset for simplicity, we first perform a correlation analysis by evaluating the Pearson correlation between each possible pair of features, and then randomly eliminate one of the pairs if they have a Pearson correlation coefficient in absolute terms greater than 0.9. According to the heat map in Fig.3.18, which indicates the results between each pair of features, we confirm the internal relationships with a high Pearson coefficient between several features. For example, the *"interarrival_std"* and the *"interarrival_max_min_diff"* have strong correlation in between with a Pearson coefficient of 0.9 because they both reflect how the values of interarrival time in a time bin are close to the mean or spread out over a wider range.

Based on the correlation analysis, we eliminate four more features, leaving 11 for prediction. A summary of the feature selection process can be seen in Table.3.4. Most of the selected features are related to interarrival time, which is the time between the arrival time instances of two adjacent packets. This is because the causal phenomenon of congestion leads to a longer waiting time in the affected router and thus introduces delays for packets which are directly reflected on the interarrival time. Finally, besides the columns describing packet loss and timestamp, we only keep columns corresponding to the 11 chosen features, which means for each individual data sheet, we only have 13 columns in total.

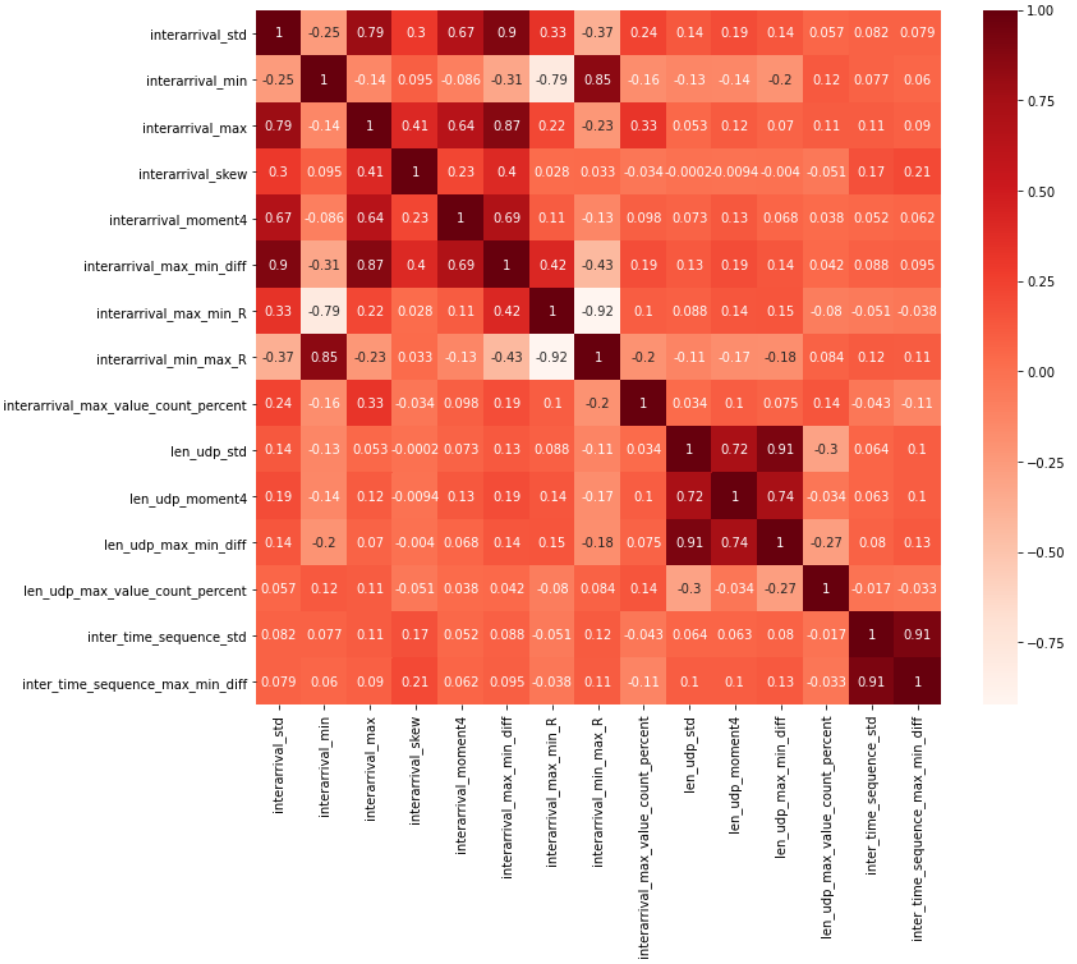


Figure 3.18: Heat map of correlations among all the selected features

Feature expansion With the feature selection process, each individual dataset has the necessary statistics for each time bin with or without loss, but what we actually need are the statistics in the time window of 5 seconds right before the target time bin. Therefore, the second step is to expand features by generating statistics for the past 5 s for each time bin in each dataset. In other words, for each time bin, we pretend it to be the target time bin in the future so that we need to include the past 10 values for each type of statistics as the actual features to be used to make a prediction, and the original 11 features are no longer useful and can be deleted since they are supposed to be unknown. Note that, for each individual dataset, we need to remove the first 10 samples since we don't have enough data for the previous 10 samples. Consequently, we generate 10 features for each of

Feature Name	Feature Description	Kept
interarrival_std	Standard deviation of interarrival time	✓
interarrival_min	Minimum value of interarrival time	✓
interarrival_max	Maximum value of interarrival time	✓
interarrival_skew	Skewness of interarrival time	✓
interarrival_moment4	4th moment of interarrival time	✓
interarrival_max_min_diff	Difference between the maximum and minimum of interarrival time	✓
interarrival_max_min_R	Ratio of maximum value for interarrival time	
interarrival_min_max_R	Ratio of minimum value for interarrival time	
interarrival_max_value_count_percent	Percentage of quantity of maximum value for interarrival time	✓
len_udp_std	Standard deviation of UDP length	
len_udp_moment4	4th Moment of UDP length	✓
len_udp_max_min_diff	Difference between the maximum and minimum of UDP length	✓
len_udp_max_value_count_percent	Percentage of quantity of maximum value for UDP length	✓
inter_time_sequence_std	Standard deviation of difference between timestamp and sequence number	✓
inter_time_sequence_max_min_diff	Difference between the maximum and minimum for the difference between timestamp and sequence number	

Table 3.4: Summary of feature selection process

the 11 statistics to eventually gain 110 features[13] for prediction. Finally, we are able to derive multiple well-structured datasets about time bins with timestamp, conditions about packet loss, and features of statistics in the past 5 seconds.

3.2.4 Final dataset generation

At this stage, we have filtered out flows generated from the observation side, removed useless columns, normalized each flow, dealt with missing values, characterized the average pattern for packet loss, and selected as well as expanded features for each individual dataset, which is ready for a combination with all the potential problems solved. Before the last step of dataset combination, we also assign a numerical ID

[13]For example, *"interarrival_std_minus_500ms"* means the standard deviation of interarrival time in the time bin from 500 ms ago.

and name it as *"flow_id"* to datasets[14] to distinguish different flows for future use.

As a result, we pool all of the individual datasets to create a large, well-structured, and clean dataset ready for machine learning method implementation and manipulation. The final dataset has 2,028,660 rows in which each row represents a time bin, and 113 columns in which we have the timestamp, the binary class label for the presence of packet loss, the flow ID for potential usages, and 110 columns as features for 11 statistics in the previous time window of study. A screen shot of the entire dataset in *Python Pandas dataframe* format is shown in Fig.3.19.

3.3 Summary

This chapter includes a complete explanation of the preliminary work associated with the dataset, starting with the generation of the raw data and concluding with the construction of the final dataset. More significantly, we obtain numerous statistics from the characterization of packet loss that could be used as features for machine learning methods.

Firstly, we improve the command-line tool *Retina* by taking into account all potential problems as well as using sequence numbers to calculate the exact number of packet losses in a time bin. As a consequence, we get 70 *csv* files with all the necessary information.

Secondly, we perform the data preprocessing together with data analysis in order to generate the meaningful, well-structured, and clean dataset needed for machine learning algorithm implementation. In particular, the procedure is divided into 4 steps:

1. The **process for each file** deals with any individual files by filtering RTP flows and eliminating useless information. Meanwhile, it also provides a general overview of the phenomenon of packet loss, taking into account quantity, distribution, etc.

2. The **process for each flow** extracts meaningful RTP flows out of each file and performs a preliminary characterization of packet loss regarding either an entire flow or a single "loss" time bin by defining the time window of study.

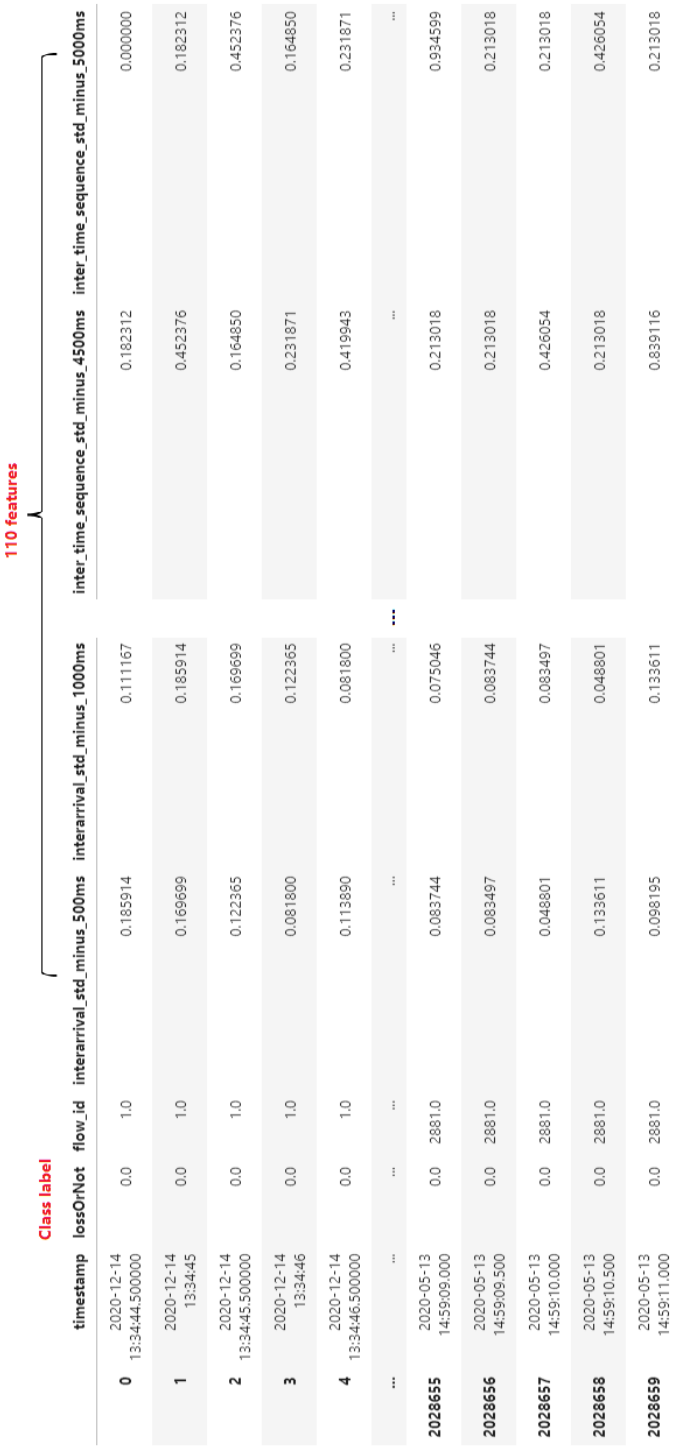
[14]Note that the flow ID is not assigned to each individual dataset but to datasets from the same flow, which means multiple individual datasets may have the same flow ID.

Afterwards, it normalizes the data in each flow and removes missing values to produce multiple sub-flows.

3. The **process after removing missing values** selects 11 types of statistics based on the comprehensive characterization of packet loss in an average manner as well as the correlation analysis and eventually derives 110 features according to the 5-s-long time window of study. In the meantime, a thorough analysis is carried out to better understand the distribution of packet loss.

4. The **final dataset generation** process combines all individual datasets to form a complete dataset including 2,028,660 time bins with class label, features, and other useful information.

Finally, we possess the final dataset to resolve the prediction problem from a data-driven point of view. Prior to that, it is necessary to formally define the problem and illustrate metrics used to assess performance and potential challenges in the next chapter.



	timestamp	lossOrNot	flow_id	interarrival_std_minus_500ms	interarrival_std_minus_1000ms	inter_time_sequence_std_minus_4500ms	inter_time_sequence_std_minus_5000ms
0	2020-12-14 13:34:44.500000	0.0	1.0	0.185914	0.111167	0.182312	0.000000
1	2020-12-14 13:34:45	0.0	1.0	0.169699	0.185914	0.452376	0.182312
2	2020-12-14 13:34:45.500000	0.0	1.0	0.122365	0.169699	0.164850	0.452376
3	2020-12-14 13:34:46	0.0	1.0	0.081800	0.122365	0.231871	0.164850
4	2020-12-14 13:34:46.500000	0.0	1.0	0.113890	0.081800	0.419943	0.231871
...	...	...	...	...	...	...	...
2028655	2020-05-13 14:59:09.000	0.0	2881.0	0.083744	0.075046	0.213018	0.934599
2028656	2020-05-13 14:59:09.500	0.0	2881.0	0.083497	0.083744	0.213018	0.213018
2028657	2020-05-13 14:59:10.000	0.0	2881.0	0.048801	0.083497	0.426054	0.213018
2028658	2020-05-13 14:59:10.500	0.0	2881.0	0.133611	0.048801	0.213018	0.426054
2028659	2020-05-13 14:59:11.000	0.0	2881.0	0.098195	0.133611	0.839116	0.213018

Figure 3.19: A screenshot of the final dataset

Chapter 4

Problem statement

With the preprocessing on datasets generated by *Retina* for RTP traffic, we obtain a final dataset that is officially ready for machine learning approach development and implementation. In this chapter, we will present the problem, metrics for the evaluation of model performance, and potential challenges we may face.

4.1 Problem formulation

The primary objective of this thesis is to predict the existence of packet loss in a time aggregation based on the statistics of previously received packets. More specifically, we are observing the RTP packet flow on one side of the communication and at any time instance, we want to predict and identify the presence condition of packet loss among all the future packets in the exact next 500-milliseconds-long time bin based on predefined and carefully chosen statistics about successfully received packets in each time bin of 500 ms (10 time bins in total) following a chronological order in the exact past 5 seconds. Mathematically speaking, the problem can be defined as:

$$\begin{aligned} Y_{t\to t+0.5} =& f(X_{1,t-0.5\to t}, ..., X_{1,t-5\to t-4.5}; ..., X_{i,t-n\times 0.5\to t-(n-1)\times 0.5}, ...; ...), \\ & i \in \mathbb{Z}, n = 1..10, X \in [0,1], Y : \begin{cases} 1, & \text{loss;} \\ 0, & \text{no loss.} \end{cases}, \end{aligned} \tag{4.1}$$

in which $Y_{t\to t+0.5}$ is the predicted binary label representing the presence of packet loss in the time bin from the current time instance to the next 0.5 s (500ms) and

$X_{i,t-n\times0.5\rightarrow t-(n-1)\times0.5}$ is the normalized i^{th} type of statistics calculated over all the received packets in the time bin of 0.5 s from the time instance of $t - n \times 0.5$ to $t - (n - 1) \times 0.5$. And we are expected to find a machine learning approach $f()$ utilizing statistics X as features to output the correct label Y for packet loss condition in the near future.

A simple conclusion that can be drawn based on the formulation and the dataset is that the features follow a discrete time sequence and the problem is a time-series based prediction problem, which means it may make sense to resort to certain machine learning methods that typically work well for time series problems, like Autoregressive Integrated Moving Average (ARIMA) or Kalman filter. However, our problem has a lot of distinctions with respect to traditional time series problems:

- In each RTP session, we have multiple flows with different durations, start times, and end times, which means at any given time instance, we may have a different number of flows, leading to different shapes of input for machine learning models. If we want to use a uniform model with a fixed size of input for all the flows in any time instance, we need to average the values across all the flows, which leads to more errors as well as bias and information loss.

- As opposed to a lot of time series problems with numerical targets like temperature forecasting, the output of our problem is a binary label, which does not have a strong statistical correlation with previous features.

- A huge advantage for solving classical time series problems is the inclusion of the latest data for updating the model and avoiding deviation from the trend. On the contrary, this kind of moving window strategy is not affordable in our problem because we are dealing with a real-time application with a time scale of hundreds of milliseconds, which is not enough for retraining a model based on the newest collected packets. In other words, the problem requires a predetermined, pre-trained, and universal machine learning model that works and responds quickly to any type of legitimate input. Not to mention that the amount of data for retraining is also not adequate.

- Unlike many traditional time series problems with strong and obvious periodicity, such as seasonality or hourly period, our problem is totally ruleless, with not only irregular packet loss but also most of the cases being lossless.

Therefore, it makes more sense to seek help from other types of machine learning methods as described in section 2.3. On one hand, our problem can be considered as a classification problem with binary class labels if we assume that the sequential

features of statistics are all independent and discrete values[1]. On the other hand, a packet loss can be considered an unexpected event, in other words, an anomaly that rarely occurs during communication. Instead of identifying both classes of "loss" and "no-loss" time bins, we can refer to anomaly detection to isolate packet loss from normal cases. In summary, instead of following the conventional analysis of the time series prediction problem, we will refer to **classification** and **anomaly detection** to identify or distinguish "loss" time bins in the future to make the prediction come true.

4.2 Metrics

A brief introduction of metrics for measuring and describing model performance will be presented in this section. For both classification or anomaly detection, the outputs of models will be either correct or incorrect binary labels for both classes. Hence, we will refer to the following commonly used metrics and terminologies for now:

1. **Confusion matrix** is a specific table in order to indicate algorithm performance by visualizing the exact quantity of correct or incorrect predictions for all classes. An example of a confusion matrix for binary classification is in Table.4.1, in which:

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

Table 4.1: Confusion matrix

- **True Positives (TP)**: correct predictions for class 1.
- **True Negatives (TN)**: correct predictions for class 0.
- **False Positives (FP)**: outputs that are classified as class 1 but actually belong to class 0.

[1]They are not exactly independent. We still rely on the trend of features, but for classification, the tendency is represented by the values of features instead of the correlations among features as well as class labels.

- **False Negatives (FN)**: outputs that are classified as class 0 but actually belong to class 1.

2. **Accuracy** is the most direct and intuitive measurement that shows the percentage of correct predictions over all samples, and it is calculated as follows:
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

3. **Precision** describes the portion of the predictions for a specific class that are correct. It is calculated as the ratio between the correctly predicted objects of a class and the total predicted objects of that class:
$$Precision = \frac{TP}{TP + FP}.$$

4. **Recall** illustrates the model's performance in a specific class and can be considered as the accuracy of that class. It is defined as the ratio between the correct predictions from a class and all the samples from that class:
$$Recall = \frac{TP}{TP + FN}.$$

5. **F1 score** evaluates the overall performance of the model, taking into account both precision and recall, and it is calculated as the harmonic mean of them:
$$F1\ score = 2 \times \frac{precision \times recall}{precision + recall}.$$

Note that, i.) for precision, recall, and F1 score, they are measurements for a specific class, and in order to evaluate the overall performance of the model considering both classes, we will also consider the macro average values; ii.) due to the intrinsic property of imbalanced quantity for both classes, some of the metrics mentioned above are not useful. We will discuss this issue in section 5.1.3.

4.3 Challenges

Before going into the details of the methodologies, we foresee some challenges and obstacles that will impede the model's performance due to inherent and substantial properties of the problem:

- According to the fine characterization in section 3.2.3, certain statistics do follow a trend as approaching the packet loss in an average manner, but due to the potential different reasons for the packet loss phenomenon and the dynamic as well as complicated network conditions, there still exist a great number of flows that do not follow the same trend or even have a trend. For example, the same rough characterization for another two flows in terms of the standard deviation of interarrival time is in Fig.4.1[2] and for cases 1 and 2, both flows follow an increasing trend at the beginning and a decreasing trend as they approach the packet loss, while the trend is in a steady state for case 3. However, they are totally different from the average behaviour of the standard deviation of interarrival time over the entire dataset presented in Fig.3.13a with an upward tendency, which means by referring to the standard deviation as a feature for prediction, the model will probably not work for these two flows.

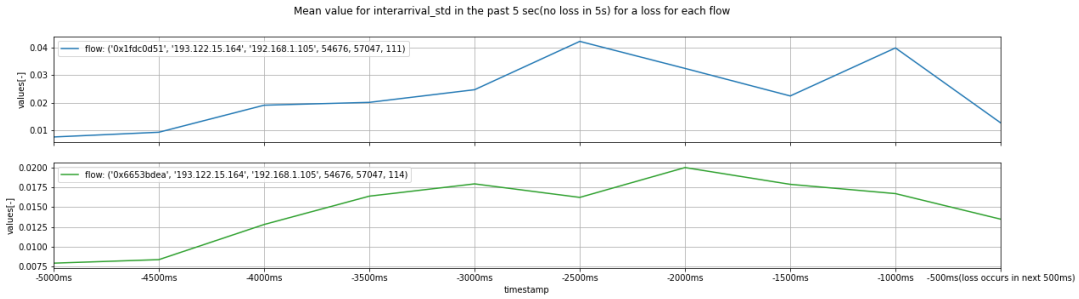
- In our problem, according to paragraph 3.2.1, most of the "loss" time bins belong to concentrated losses, which indicates that it is difficult to distinguish the "no-loss" time bins between adjacent concentrated losses since they are close to both losses and thus are easily affected by packet loss.

- Based on paragraph 3.2.1, most "loss" time bins for both applications only have 1 packet loss. On the other hand, a time bin could have hundreds or even thousands of packets, which infers that the impact of only one packet loss will be diluted, so that the trend of statistics may not be influenced a lot in some cases.

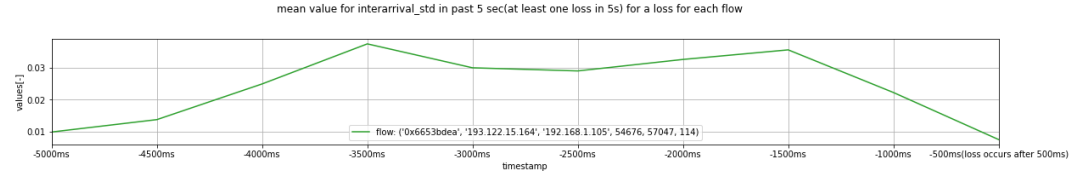
- According to Fig.3.3, both applications only have around 1.5% "loss" time bins. Our dataset is extremely imbalanced, leading to an imbalanced classification problem, which is a classification predictive modeling problem where the dataset has skewed class proportions and the distribution of samples across the classes is not equal. Normally, an imbalanced dataset would force the model to bias towards the majority class and result in poor performance for the minority class.

- Although the data collection is done with a standardized procedure to avoid external and environmental disturbances as much as possible, the RTP packets are still captured at different times on different devices with dynamic as well as volatile network conditions, which means that it is nearly impossible to

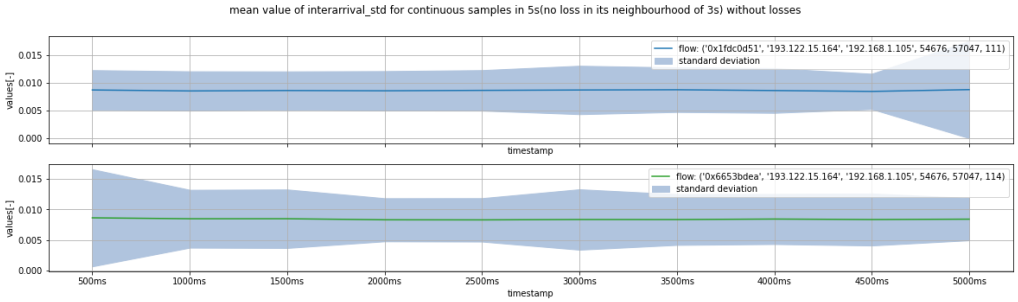
[2]For flow *"0x1fdc0d51, 193.122.15.164, 192.168.1.105, 54676, 57047, 111"*, there's no "loss" time bin for case 2.



(a) Case 1



(b) Case 2



(c) Case 3 (with standard deviation)

Figure 4.1: Average behaviour of the standard deviation of interarrival time in the three cases for another two random flows

formalize a completely stable environment and the variations among statistics could be induced by other sources instead of only packet loss.

In a word, the prediction of packet loss is a very hard problem with numerous obstacles to be tackled, in which some difficulties cannot be avoided due to the intrinsic properties of the RTP communications and the dataset, but the disturbance due to the imbalanced dataset can be somehow relieved by the sampling strategies described in the next chapter.

Chapter 5

Methodology and experimental result

In this chapter, we will demonstrate all the machine learning approaches in terms of classification and anomaly detection and their corresponding experimental results. Additionally, for certain methodologies, some specific and technical discussions will be presented as well. More importantly, in order to compare models and investigate performance while focusing on the models themselves rather than the prediction problem, we simplify the work flow without taking into account possible elaborations and optimizations. In other words, the evaluation results from the methodologies developed in this chapter are references instead of the final solid solutions so that we are able to allocate the best approach in a convenient way and perform in-depth analyses to delve into the prediction problem in the next chapter.

5.1 Decision tree

To begin with, we use the decision tree as a fundamental method as well as a reference to help better understand the prediction problem, illustrate possible outputs, and determine the configurations of the training phase and model evaluation for subsequent approaches.

5.1.1 Sampling strategy

According to Fig.5.1, "loss" time bins only occupy 1.39% of all the time bins in the final dataset, which leads to an imbalanced classification problem as described in section 4.3. The most common way to solve the problems caused by imbalanced data is to balance the majority and minority classes by artificially modifying the quantities for them to reach a comparable number. Such a process is called sampling.

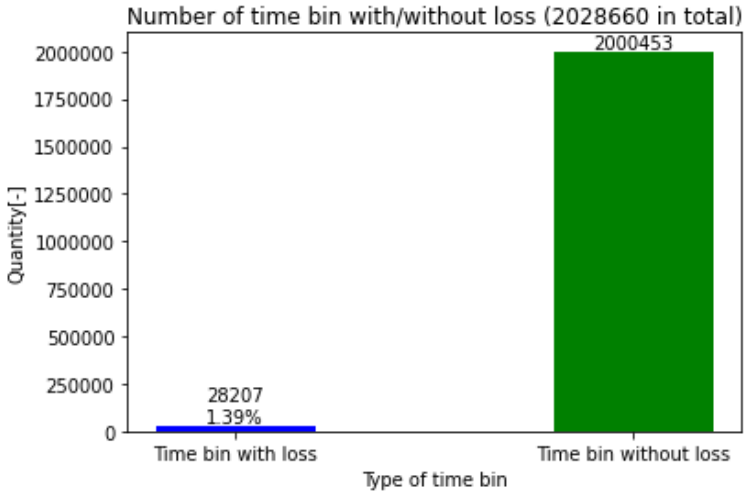


Figure 5.1: Quantity of time bins with or without loss

The imbalanced dataset is not a problem during the test phase but extremely affects the model during the training phase. Therefore, we design three extra sampling strategies for training and compared the results with the model trained on the original dataset without sampling. Note that for the model development, we randomly extract 70% of the entire dataset as the training dataset and the rest 30% as the test dataset[1], and refer to this partition as the base scenario. Fig.5.2 helps visualize the sampling strategies and quantities for both classes in each strategy. In particular:

[1]Notes: i.) The original dataset will be first split into "loss" datasets with only samples representing time bins with packet loss and "no-loss" datasets with only samples for time bins without packet loss, and we always shuffle both datasets before model development; ii.) Then, we extract 70% from either dataset and combine them afterwards for training, and regard the rest of both datasets as test datasets; iii.) As a result, the portions for training and testing will be the same for both classes.

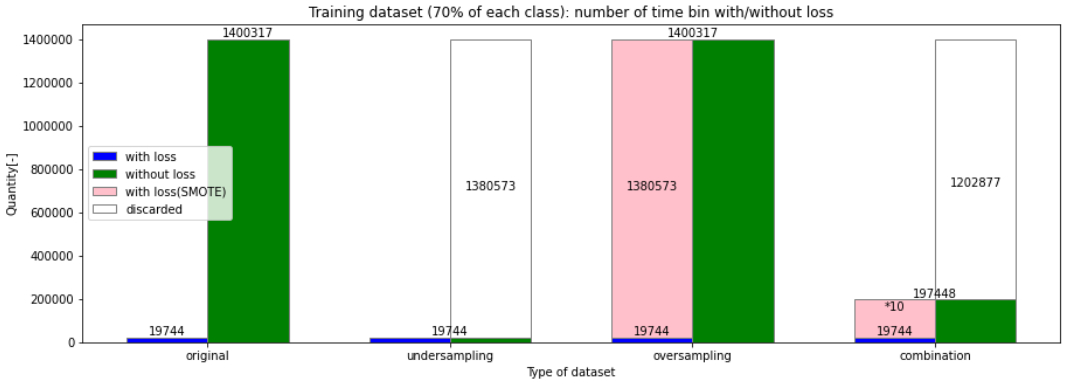


Figure 5.2: Training dataset: quantity of time bins with or without loss for sampling techniques and base scenario

- **Original** dataset refers to the imbalanced base scenario without sampling technique. The model trained on this dataset will suffer from the issues caused by imbalanced data.
- **Undersampling** means reducing the amount of "no-loss" training dataset to the same amount of "loss" training dataset as in the base scenario, with 70% of the original "loss" dataset. Consequently, we achieve a balance between both classes for training but significantly cut down the entire training dataset. The undersampling is done randomly, which means the discarded portion of the original "no-loss" training dataset is randomly chosen. Due to the vast reduction of "no-loss" training set, we may lose information about time bins without loss.
- **Oversampling** maintains the quantity of "no-loss" training dataset but augments the "loss" training dataset to the same amount by the Synthetic Minority Oversampling Technique (SMOTE)[28], which selects the nearest neighbors along the line drawn among samples in the feature space to generate new samples and is proved to be an effective method for oversampling. As a result, we artificially and intentionally generate a new training set with a very large number of samples and equality for both classes. Oversampling will increase the "loss" training set to around 70 times more than the original one, which means most of the new "loss" data generated will not be real and, thus, tend to introduce errors.
- **Combination** of undersampling and oversampling aims at reducing the quantity of "loss" training set and increasing the quantity of "no-loss" training set to the same amount at the same time. In our case, we refer to 10 times of the original "loss" training set. Note that the undersampling is also done

randomly and the oversampling is also done by SMOTE. As a result, we reach a balance not only between classes but also between sampling strategies to neither penalize the "no-loss" training set too much nor amplify the "loss" training set too much.

5.1.2 Experimental result

We have trained four independent decision trees without any specific configurations for simplicity on the four types of training datasets and tested the models on test datasets. The results in terms of metrics are summarized in Table.5.1, and the resulting confusion matrices are in Fig.5.3.

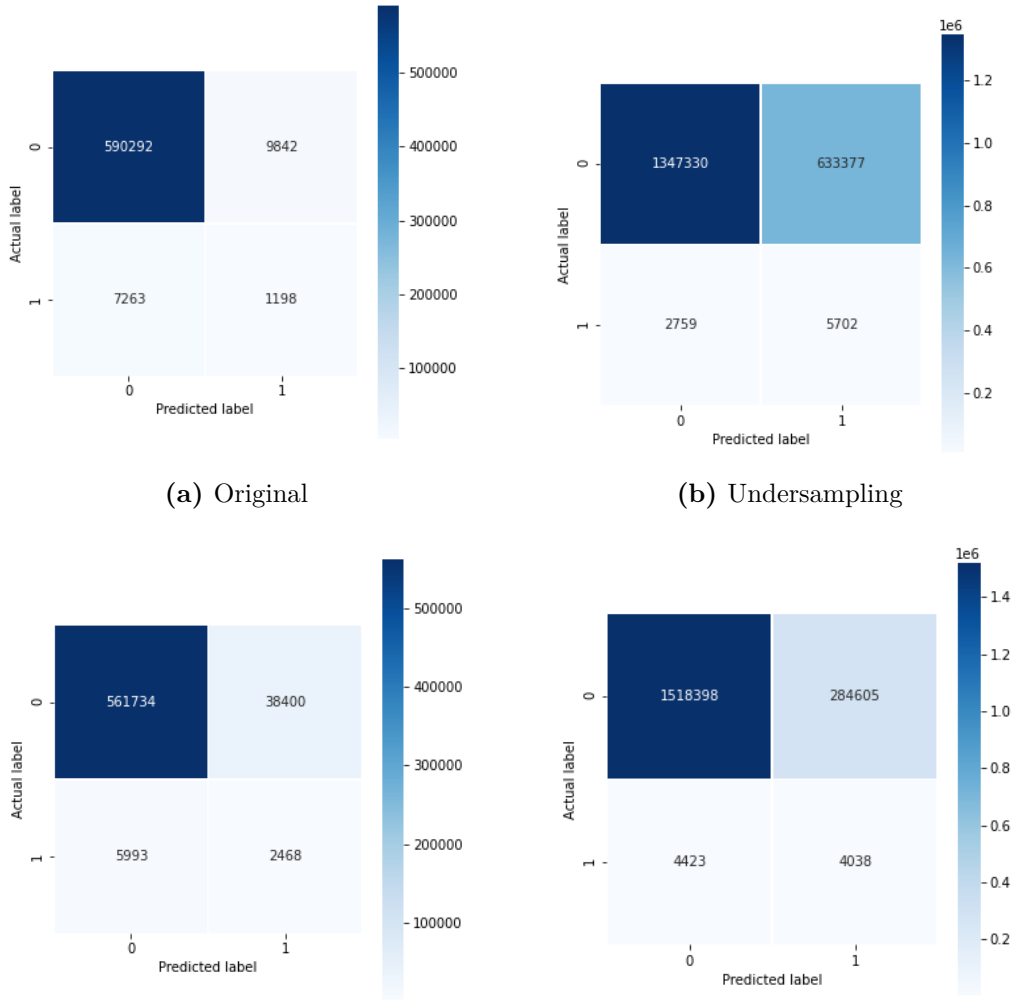
Training dataset		Original	Undersampling	Oversampling	Combination
Accuracy		0.97	0.68	0.93	0.84
Precision	Macro average	0.55	0.50	0.52	0.51
	Class 0	0.99	0.99	0.99	0.99
	Class 1	0.11	0.01	0.06	0.01
Recall	Macro average	0.56	0.68	0.61	0.66
	Class 0	0.98	0.68	0.94	0.84
	Class 1	0.14	0.67	0.29	0.48
F1 score	Macro average	0.55	0.41	0.53	0.47
	Class 0	0.99	0.81	0.96	0.91
	Class 1	0.12	0.02	0.10	0.03

Table 5.1: Summary of metrics for the results of decision trees on different training datasets

According to the experimental results:

- All four models output decent accuracy, except for the undersampled training dataset. The same distinct behaviour also applies to the recall and F1 score for "no-loss" time bins, class 0.

- All models predict class 0 very well because the precisions are always very high with the same value of 0.99, but they predict class 1 very poorly because of the awfully low precisions that range from 0.01 to only 0.11, resulting in a stationary macro average precision of around 0.5. They all present a poor ability from an overall perspective for class 1 due to very low F1 scores ranging from 0.02 to 0.12 and, thus, result in a weak overall prediction performance with a mediocre macro average F1 score ranging from the minimum of 0.41 to the maximum of merely 0.55.



(a) Original **(b)** Undersampling

(c) Oversampling **(d)** Combination

Figure 5.3: Confusion matrices for the results of decision trees on different training datasets

- Even though the model trained on original training dataset without sampling generates the highest F1 score to show the best general performance mainly due to the best prediction output for class 0, it still cannot be considered for further analyses since it can only identify 1198 actual "loss" time bins out of 8461 resulting in the lowest recall of only 0.14 for class 1, which proves the negative influence of imbalanced dataset and totally fails our objective.

- The undersampling technique is very good at predicting time bins with loss in the future due to the highest number of correct classifications as well as the

recall of class 1. On the contrary, we have a huge number of incorrect classifications with a quantity of 633377 for class 0 on account of the information loss from the tremendous decrement of the "no-loss" training dataset. It would be a lot of waste of network resources if we frequently reacted to each of these incorrect predictions.

- A model based on oversampling strategy solves the problem of undersampling by only generating 38400 incorrect predictions for class 0 since the information is adequate. However, because of the errors and uncertainties brought by unreal and generated data, it merely identifies 2468 "loss" time bins, which is a little bit better than the original training dataset but still not preferable.
- Obviously, the combination of undersampling and oversampling produces a balanced performance between the other two sampling strategies, with higher recall for class 1 but lower recall for class 0 compared to oversampling and higher recall for class 0 but lower recall for class 1 compared to undersampling. It can be considered as a compromise between the introduction of errors due to oversampling and information loss due to undersampling.

In a word, all decision tree models have their own merits and disadvantages, but they all perform poorly by either generating too many incorrect classifications for "no-loss" time bins or completely failing the prediction with too few correct classifications for "loss" time bins. Even with an equilibrium in the middle from the combination strategy, the performance is barely acceptable. However, the aforementioned poor results actually meet our expectation because: i.) We did not tune any models with different parameters for better performance since we regard decision tree as a base model to provide an overview of how machine learning method works on the problem and an intuitive guidance of how to develop other methods; ii.) The prediction problem is a hard one with a lot of intrinsic issues and the sampling techniques can alleviate the pains caused by imbalanced dataset to some degree but cannot solve the problem completely; iii.) Decision tree is a relatively simple and weak machine learning architecture that tends to be surpassed by other sophisticated and powerful methods for a complicated problem.

5.1.3 Discussion about metrics

Imbalanced data brings challenges not only to the prediction problem but also to the metrics that evaluate the model's performance because most of the common metrics that are widely used for classification work under the assumption of a balanced class distribution. According to the experimental results derived above,

we cannot rely on all the metrics described in section 4.2 and we will explain the reasons and finally decide those that will be used for the rest of this thesis in the following:

- Both **accuracy** and **precision** are not reliable due to the domination of class 0. The accuracy and precision of class 0 will always be very high since, in most cases, we have an excess of correct classifications of class 0, which plays a crucial and decisive role in the metrics' calculation. The accuracy will be determined by the prediction performance for class 0 and barely affected by class 1. In other words, the accuracy is almost equal to the recall of class 0. At the same time, the precision of class 1 will always be very low because even if we are able to achieve a good prediction for "loss" time bins, the quantity of correct classifications is still not comparable with the quantity of misclassifications for class 0. As a result, the macro average precision will always be around 0.5. The accuracy is useful only when the model performs very poorly for class 0. However, such a bad result can be indicated by other meaningful metrics as well.

- **F1 score** can be used to indicate the overall performance, but because it is partially affected by precision, the F1 score is still not a substantially good choice. For example, if we refer to the result with the highest F1 score of 0.55, we will end up with a model that cannot identify the "loss" time bin and totally fails the objective. Therefore, we can only consider the F1 score as a measurement when the improvement is large because it can reflect the model enhancement bypassing the constraints due to precision from an overall point of view.

- **Confusion matrix** will always be included since it's simply an explicit illustration of the model outputs rather than a measurement of model performance.

- The most appropriate and reliable metric is related to **recall**. As the individual accuracy for "loss" or "no-loss" time bins, the class recall can directly indicate the prediction performance for either class. The macro average recall can reflect the global model's capacity to correctly make classifications, taking into account both classes.

Hence, in the following machine learning approach development, we will still present the outcomes related to all the metrics for reference but focus on the recall to evaluate the models. Note that, in some cases, we still refer to all the metrics for parallel comparisons.

On top of that, based on the metrics chosen, we can determine the criteria for model evaluation to select the best model:

1. **Firstly, selecting models with recall for class 0 to be at least 0.80, which means we can only afford at most 20% misclassifications for "no-loss" time bins.**

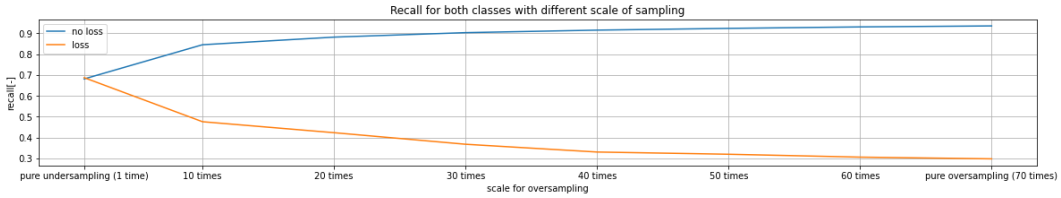
2. **Secondly, referring to the one with the highest recall for class 1 among the ones selected above, which means the model with the largest number of correct classifications for "loss" time bins.**

These constraints are defined aiming at balancing the performance between classes so that we are able to generate fewer wrong classifications for the "no-loss" time bin and recognize the "loss" time bin in the future as much as possible at the same time. Nevertheless, it cannot be ignored that even with a recall of 0.80 for class 0, we still end up with a great deal of misclassifications because we had superabundant "no-loss" time bins originally. This is an issue that cannot be avoided due to the imbalanced data, and since the objective of this thesis is to predict packet loss, we have to refer to such thresholds for now to find feasible solutions.

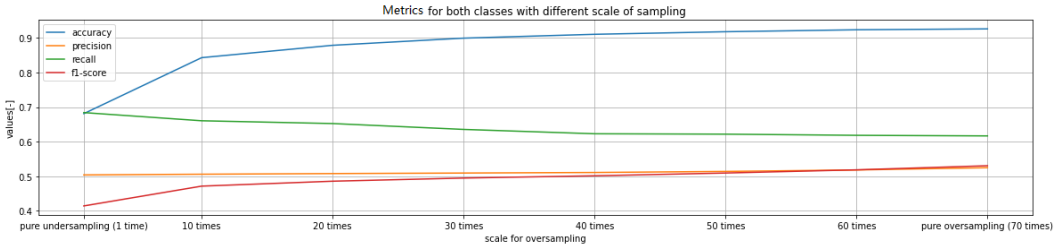
5.1.4 Effect of sampling

In the previous analysis of the combination sampling strategy, we increased the quantity of "loss" training set to 10 times, but apparently, different degrees of oversampling also affect the model output. In this section, we present the model evaluation results in terms of various inflation scales ranging from 1 time to 70 times with a step of 10. Fig.5.4 shows the results of the test phase for 8 models with different scales of oversampling.

On one hand, as we can see in Fig.5.4a, with the increment of oversampling from the minimum of no oversampling at all to the maximum of the same quantity of the original "no-loss" training set, the recall increases for class 0 but decreases for class 1, which means there's no way to improve the performance for both classes at the same time. And since the performance drop for class 1 is more severe with respect to the improvement for class 0, we experience an overall decline for macro average recall as in Fig.5.4b. However, the slight decrement of average recall cannot be used for model selection because the model tends to miss the goal of predicting packet loss if we keep increasing the amount of "loss" training set. On top of that, when the oversampling scale equals to 10 times, we reach a recall for class 0 greater



(a) Class recall



(b) Accuracy, macro average precision, recall and F1 score

Figure 5.4: Class recall and other metrics for the model evaluation with combination strategy in terms of different oversampling scale

than 0.8 for the first time, which meets our threshold defined in section 5.1.3 and corresponds to a relatively higher recall for class 1, so that in the following machine learning method development, we will refer to the 10-times oversampling when a combination sampling technique is needed.

On the other hand, according to Fig.5.4b, all metrics except recall possess a rising trend. But the model we select is the one with a sampling quantity of 10 times where the values for metrics including accuracy, precision, and F1 score are the second worst, which in turn proves the uselessness of those metrics for model evaluation in our specific case.

5.2 Neural network

The second machine learning method we have developed is the neural network (NN). In particular, we will refer to deep neural network (DNN) and long short-term memory (LSTM) neural network, both with a combination sampling strategy of random undersampling and oversampling by SMOTE.

5.2.1 Deep neural network

DNN can be used for classification problems with inputs of feature values and outputs of the approximation for class label. Starting from now on, we will also take into account the effect of different lengths of time windows before the target time bin.

Effect of time window:

We have defined a time window of study with 5 seconds and 10 samples in it, which results in 110 features in total for 11 statistics. However, the statistics in the time bin far away from the target time bin in the future may not be informative with respect to those near the future time bin, so in this section, we distinguish the length of time window to include different numbers of features[2] and implement multiple NNs with a similar simple architecture except for the input shape[3] to briefly understand the effect of the length of time window and select the one with the best performance for subsequent actual NN implementation.

Fig.5.5[4] indicates the accuracy of the training set and other metrics of the test set for different lengths of time window. On one hand, we do experience a slight performance drop as we reduce the number of features for a corresponding decreasing window size, especially when it comes to the length of 500 ms and 1000 ms, which means choosing only the very past statistics is not enough. On the other hand, the results differ very little between the window sizes from 5000ms to 1500ms, which indicates that statistics included far away from the target time bin influence the model the same as those in the middle time bins, and selecting

[2]We examine the length of time window starting from 0.5s (500 ms) to 5s (5000 ms), implying that the number of features will range from 11 (11 statistics × 1 sample in 0.5s) to 110 (11 statistics × 10 sample in 5s).

[3]Notes: i.) the number of neurons in the input layer corresponds to the number of input features. For example, if we are working with a 2 s-long time window, the number of features is 44 (11 statistics × 4 samples in 2 s), and the number of neurons in the input layer is also 44; ii.) there are two hidden layers, with the first layer having neurons of two thirds of the input layer and the second layer having one third of the input layer, and the activation function is *relu*; iii.) the output layer has two neurons with an activation function of *softmax*; iv.) we refer to *Adam* optimizer with a learning rate of 0.01, the loss is computed by *sparse categorical cross-entropy* and the optimizing target metric is accuracy.

[4]The minus sign for each length represents that it's the time window before the current time instance (target time bin for prediction).

sizes from 1500ms to 5000ms only matters very little for the model performance. However, at a window size of 3500 ms, we have an extraordinary value of accuracy for the test set due to the good prediction for class 0, while the other metrics remain similar. Thus, for DNN, we will refer to 3500 ms as the length of time window for further model implementation, and as a result, we have 77 features (11 statistics $\times$ 7 samples in 3.5 s) left.

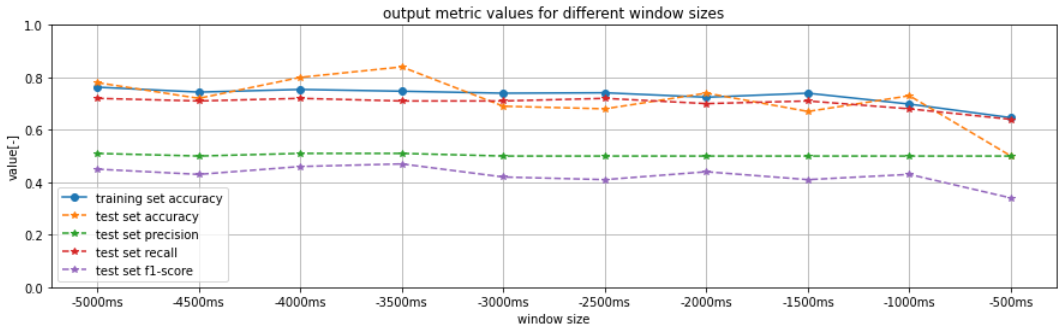


Figure 5.5: Resulting metrics of DNN for different time window size

Model tuning:

In order to optimize the NN for better performance, the most important step is to tune the model by modifying a group of parameters to find the one with the best output. We refer to a bunch of typical parameters, including number of layers, learning rate, and so on, to perform one hundred trials of random search among all the possible combinations of parameters to finally derive the best model configurations. A complete summary of all parameters as well as the consequential best combination and other predetermined configurations are shown in Table.5.2. Finally, we derive a DNN architecture as shown in Fig.5.6.

Final experimental result:

The previous DNN is trained to stop after 10 epochs for simplicity, so we need to keep training the neural network until it starts to overfit on the training dataset by including and observing a validation dataset[5] for early stopping. As a result, after

[5]In this case, we select 60% as the training set, 10% as the validation set, and the rest 30% as the test set, and stop the model training after the accuracy of the validation dataset keeps

Parameters		Value range	Best value
Tuned parameters	Number of hidden layers	[1,2,3,4]	3
	Number of neurons	[20,40,60,80]	1^{st}:60, 2^{nd}:60, 3^{rd}:20
	Activation function	[*relu*,*tanh*]	*tanh*
	Dropout	[True, False]	False
	Learning rate	$[10^{-4} : 10^{-2}]$	0.00074
Fixed parameters	Number of input neurons	77	
	Number of output neurons	2 (*softmax*)	
	Optimizer	*Adam*	
	Cost function	*sparse categorical cross-entropy*	
	Evaluation metric	Accuracy*	

* The accuracy is applicable here because the training dataset has the same amount for both classes thanks to the combination sampling strategy.

Table 5.2: Tuned and fixed parameters for DNN

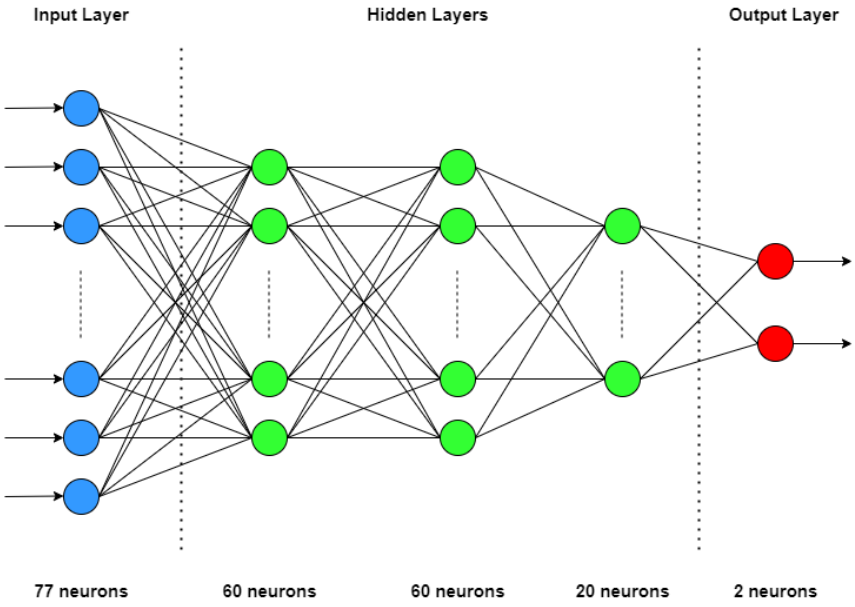


Figure 5.6: Final architecture of deep neural network

25 epochs, we have a final well-tuned and carefully-trained DNN with an accuracy on training dataset of 0.8512.

The summary of consequential metrics with highlights of recall is in Table.5.3

decreasing for 3 epochs.

and the confusion matrix is in Fig.5.7. Firstly, the recall for class 0 reaches merely above our threshold, which means the ability to not misclassify "no-loss" time bins is acceptable in our case. At the same time, the DNN model is able to identify 5866 out of 8461 "loss" time bins in the future with a recall of 0.69, which is the highest recall (i.e. the largest amount of correct classification of class 1) we can get so far. Compared to the decision tree model with the highest recall of 0.67 for class 1, DNN may not outperform very much in terms of packet loss, but has a significant improvement for class 0 because, previously, we could only get a recall of 0.68 for the "no-loss" time bin, while DNN is able to achieve 0.82 now. Moreover, the DNN model overfits a little bit with an accuracy of 0.8512 on the training set and a lower one of 0.82 on the test set, but the overfitting is much severer for class 1 with a recall of 0.69. In a word, just as expected, DNN indeed improves the performance with an acceptable result, but it is without doubt that the overall performance is still not adequate, which is proved not only by the recall but also by the macro average F1 score of 0.47.

Metrics		Value
Accuracy		0.82
Precision	Macro average	0.51
	Class 0	0.99
	Class 1	0.02
Recall	Macro average	0.76
	Class 0	0.82
	Class 1	0.69
F1 score	Macro average	0.47
	Class 0	0.90
	Class 1	0.04

Table 5.3: Metrics of final DNN model evaluation

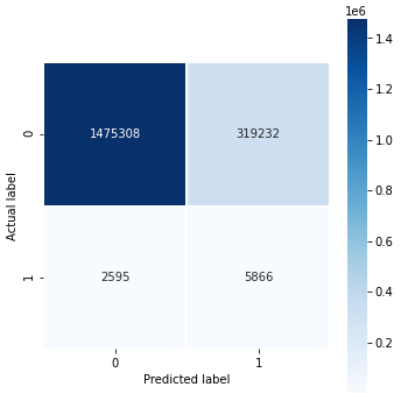


Figure 5.7: Confusion matrix of final DNN model evaluation

5.2.2 LSTM neural network

LSTM NN is able to process and memorize the entire sequence of input so that it can better learn the trend of statistics in the time window of study. In this section, we also refer to the combination strategy to deal with imbalanced data. Besides the differences between a neuron and an LSTM cell, two major differences between conventional NN and LSTM NN are the input layer and network connections: i.) the input layer of LSTM is formed by a sequence of values with a predefined order

and each value represents a timestamp that will enter into an LSTM cell. Only this value will enter the corresponding cell so that the LSTM cells also follow the sequence by processing the timestamps one by one and memorizing the sequence with a monotonically increasing amount; ii.) the links between two LSTM layers or between a LSTM layer and a dense layer are not fully connected. It depends on the output hidden states of the previous layer and the architectures of *one-to-many*, *many-to-many* and so on.

Effect of time window:

First of all, same as DNN, we also try to detect the impact of different lengths of time windows, and in this case, we exclude the condition of 500 ms since there's only one sample for each statistic, which is not enough to form a sequence. With a simple and predetermined LSTM NN[6], we derive the results and present it in Fig.5.8.

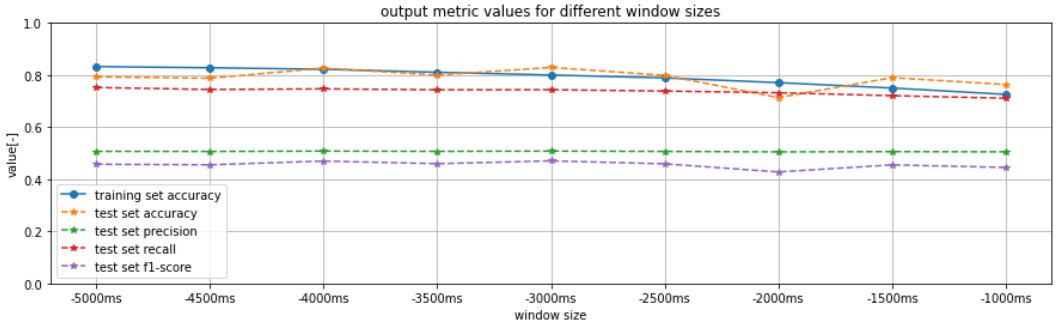


Figure 5.8: Resulting metrics of LSTM NN for different time window size

Similar to DNN, the LSTM NN model also experiences a performance drop as we reduce the window size, especially for time windows from 1000 ms to 2500 ms. This means that both neural network models follow the basic principle of

[6]Notes: i.) in this case, the features are no longer independent but must form a queue (a sequence of timestamps) in the order from earlier to later in the time window for each statistic, and as a result, each timestamp has 11 different statistics as features and enters into the corresponding LSTM cell; ii.) there's only one LSTM layer with the number of LSTM cells being equal to the number of timestamps in a sequence, and the activation function is *relu* and the output shape is $number\ of\ timestamps \times 9$; iii.) the output layer has one *sigmoid* neuron, the optimizer is *Adam* with a learning rate of 0.001, the target metric is accuracy and the loss is computed by *binary cross entropy*.

data-driven approaches; the more data included, the better the model tends to be. However, unlike DNN, with nearly constant values across the time window from 2500ms to 5000ms, the performance degeneration is more serious with a smaller number of features in smaller time windows. In other words, a longer time window will include more information to form a longer and more representative sequence for LSTM NN to better learn the trend among features for a statistic. Therefore, we will refer to the complete time window of study with a length of 5000 ms for further model optimization in the following.

Model tuning:

The LSTM NN model optimization is done in a similar way to DNN with 50 times of random search among the parameters presented in Table.5.4, and the the final architecture of the LSTM NN model is in Fig.5.9.

Parameters		Value range	Best value
Tuned parameters	Number of LSTM layer (exclude input layer)	[0,1,2]	2
	LSTM output shape	[16,32,48,64]	1^{st} LSTM: 48 2^{nd} LSTM: 16 3^{rd} LSTM: 16
	Number of possible dense layer	[0,1,2]	0
	Number of neurons	8 to 64 (step of 8)	-
	Activation function	[*relu*,*tanh*]	1^{st} LSTM: *tanh* 2^{nd} LSTM: *relu* 3^{rd} LSTM: *relu*
	Dropout	[True,False]	False
	Learning rate	$[10^{-4}:10^{-2}]$	0.00274
Fixed parameters	Input shape	10 $timestamps \times 11$ $features$	
	Number of output neurons	1 (*sigmoid*)	
	Optimizer	*Adam*	
	Evaluation metric	Accuracy	
	Cost function	*binary cross entropy*	

Table 5.4: Tuned and fixed parameters for LSTM NN

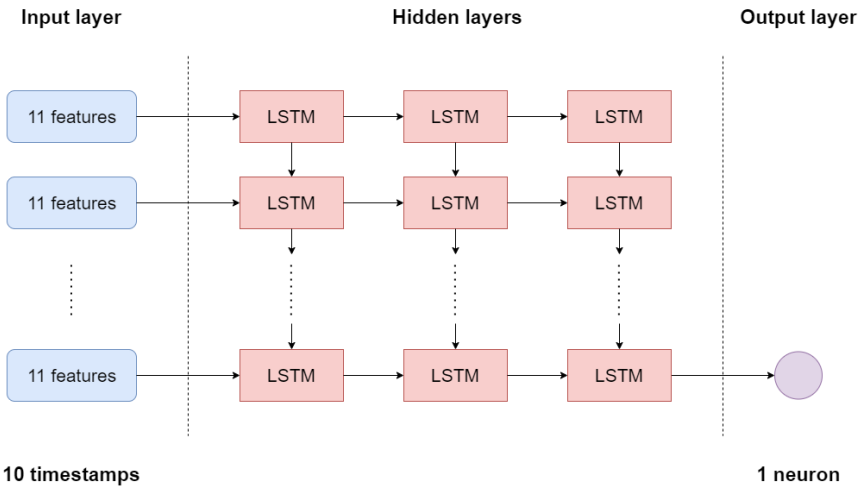


Figure 5.9: The architecture of the final LSTM neural network

Final experimental result:

Based on the best LSTM model derived above, we keep training it with a validation dataset for early stopping of the same configurations in DNN, and after 15 epochs in total, we finally derive a model with an accuracy of 0.8357 in the training set.

According to the results of the test dataset in Table.5.5 and Fig.5.10, the LSTM NN model also satisfies the threshold and outperforms DNN a little bit in terms of predicting "no-loss" time bins with a 1% improvement for the recall of class 0, while maintaining the performance for class 1 with the same recall. Although the model enhancement leads to 21292 (319232 misclassifications for class 0 in DNN minus 297940 misclassifications for class 0 in LSTM) fewer misclassifications, obviously, both neural network models are still not substantially adequate for our objective.

5.3 Random forest

The third machine learning method for classification is random forest. Unlike previous approaches, we also distinguish different versions of implementation to deal with imbalanced datasets.

Metrics		Value
Accuracy		0.83
Precision	Macro average	0.51
	Class 0	0.99
	Class 1	0.02
Recall	Macro average	0.76
	Class 0	0.83
	Class 1	0.69
F1 score	Macro average	0.47
	Class 0	0.91
	Class 1	0.04

Table 5.5: Metrics of final LSTM model evaluation

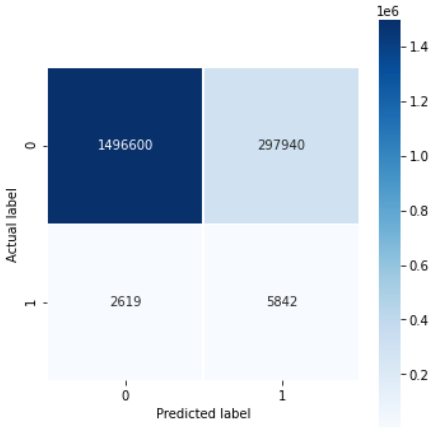


Figure 5.10: Confusion matrix of final LSTM model evaluation

5.3.1 Different implementations

The most common way to implement random forest is through the *"RandomForestClassifier"*[7] from *scikit-learn* which is one of the most famous and reliable *Python* modules for machine learning, but it also suffers from problems caused by the imbalanced dataset. Fortunately, there exists another *Python* module that is specified and developed relying on *scikit-learn* for the imbalanced classification problem, named *imbalanced-learn*, which introduces *"BalancedRandomForestClassifier"*[8] that is able to handle imbalanced datasets.

The *"BalancedRandomForestClassifier"* provides all the functionalities of the *"RandomForestClassifier"* and additionally, each tree in the forest will be provided with a balanced bootstrap sample[29] by first drawing a bootstrap sample from the minority class and then randomly drawing the same number of samples from the majority class. It also modifies the way to optimally split the tree by searching through a set of m_{try} randomly selected variables instead of searching through all variables. Furthermore, another way for the random forest to cope with an imbalanced dataset is by assigning a weight to each class to set up a heavier penalty for misclassifying the minority class with a larger weight. The class weights are incorporated into the random forest algorithm in two places: i.) in the tree induction procedure, class weights are used to weight the *Gini* criterion for the

[7] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. We will also call it the original random forest classifier.

[8] https://imbalanced-learn.org/stable/references/generated/imblearn.ensemble.BalancedRandomForestClassifier.html, we will also call it the balanced random forest classifier.

split; ii.) in the terminal nodes of each tree, class weights are again taken into consideration to determine the class prediction by a weighted majority vote[29].

In our case, we implement and compare random forest algorithms in three different ways:

1. The **original random forest classifier without class weight**[9] trained on a balanced training dataset with the combination sampling strategy (random undersampling and 10-times oversampling by SMOTE).
2. The **balanced random forest classifier with class weight** trained on the original imbalanced training dataset without any sampling techniques. The class weight is simply the multiple between the majority class and the minority class. In our case, the quantity of "no-loss" time bins is 70 times that of "loss" time bins, so that the class weight is 70 for class 1 and 1 for class 0.
3. The **balanced random forest classifier without class weight** trained on the original imbalanced training dataset without any sampling techniques.

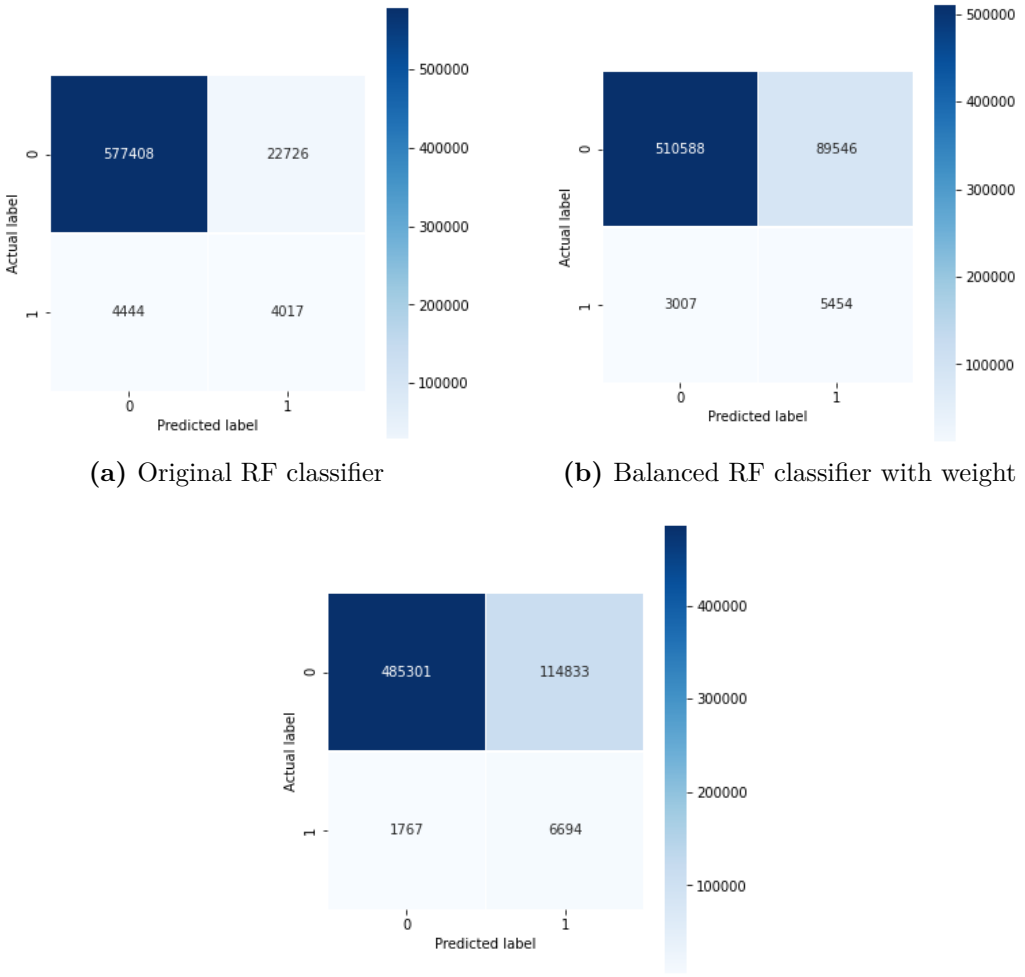
Then, for simplicity, we construct the random forest with 400 trees but without other specific configurations, and the results in terms of metrics and the confusion matrix derived from the three different random forest classifiers on the same test dataset are presented in Table.5.6 and Fig.5.11.

RF classifier		Original RF classifier	Balanced RF classifier with weight	Balanced RF classifier without weight
Accuracy		0.96	0.85	0.81
Precision	Macro average	0.57	0.53	0.53
	Class 0	0.99	0.99	0.99
	Class 1	0.15	0.06	0.06
Recall	Macro average	0.72	0.75	0.80
	Class 0	0.96	0.85	0.81
	Class 1	0.47	0.64	0.79
F1 score	Macro average	0.60	0.51	0.50
	Class 0	0.98	0.92	0.89
	Class 1	0.23	0.11	0.10

Table 5.6: Summary of metrics for the results of different random forest classifiers

In the first place, all three models satisfy the predefined constraints, and among them, according to our criteria for best model selection, **the best one with the**

[9]For original random forest classifier, we only include in this thesis the one without class weight because the one with class weight performs very poorly and cannot solve the problem caused by an imbalanced dataset, so it's not worth discussing.



(a) Original RF classifier

(b) Balanced RF classifier with weight

(c) Balanced RF classifier without weight

Figure 5.11: Confusion matrices for the results of different random forest classifiers

highest macro average recall of 0.80 and the highest recall of 0.79 for class 1 is the balanced random forest classifier without class weight trained on an imbalanced dataset. Out of 8461 "loss" time bins in the test set, it can correctly predict 6694, which is the largest amount up to now and is 828 more than the second-largest from DNN. Given that an improvement of a quantity of 828 is 9.79% of class 1 in the test dataset, this model advances significantly in terms of predicting packet loss.

Moreover, the original random forest classifier results in the highest macro average F1 score of 0.60, with a larger improvement in terms of the overall performance

with respect to the other two balanced random forest classifiers, and this is mainly because the misclassifications for class 0 are only 22726, which is comparable to the correct classifications of 4017 for class 1, leading to an acceptable precision for the first time, and eventually affecting the F1 score. This classifier can be applied if we prefer to not waste network resources reacting to packet loss events that are supposed to be normal cases without loss, and at the same time, the classifier can still identify around half the actual losses. However, we still refer to the balanced random forest classifier without weight as the best model for our primary objective.

Additionally, based on the results following the order from left to right in Tab.5.6, the performance regarding recall experiences a decreasing trend for class 0 but an increasing trend for class 1, which means: i.) the original RF classifier without specific internal countermeasures for imbalance is still affected by the imbalanced dataset and outperforms towards the majority class; ii.) the balanced RF classifier indeed relieves the issues caused by the imbalanced dataset in a way; iii.) the setting of class weight will force the model to work better for the majority class instead of the other way around, which is counterintuitive and the balanced RF classifier with weight can be regarded as a compromise with an intermediate performance between the other two models.

5.3.2 Effect of time window

In order to improve upon the best random forest model derived above, we also investigate the effects of different lengths of time windows as in section 5.2.1 and the results are in Fig.5.12. From the plot, the overall trend follows a steady state with a slight decline as we reduce the window size. For random forest models, just with the statistics in the last time bin, the prediction can still be done with a mild performance drop. However, the model will yield a vast number of misclassifications for "no-loss" time bins even with a tiny defect, so we will still refer to the full five-seconds-long time window of study with relatively best performance to include statistics as much as possible.

5.3.3 Model tuning and final experimental result

Finally, we are able to tune the balanced random forest classifier without class weight and with a time window of 5 s by modifying certain parameters to further optimize the performance. By performing one hundred trials of random search among all the possible combinations of parameters, we arrive at a group of parameters for

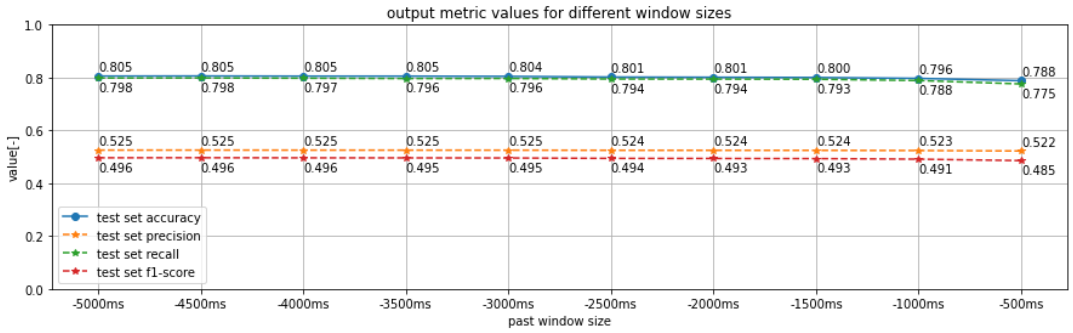


Figure 5.12: Resulting metrics of balanced random forest classifier without class weight for different time window size

the best performance. A summary of tuned and best parameters for random forest is shown in Table.5.7.

Parameter	Value range	Best value
Number of trees	100 to 1000 (step of 100)	600
Maximum depth of the tree	10 to 100 (step of 10) & None*	90
Maximum number of features to consider when looking for the best split	["auto", "log2", None]**	"auto"
Minimum number of samples required to be at a leaf node	[1,2,4,6]	2
Minimum number of samples required to split an internal node	[2,4,6,8,10]	4
Bootstrap	[True,False]	False

* None means the default setting where nodes are expanded until all leaves are pure or until all leaves contain less than the minimum number of samples required to split an internal node.

** "auto": $maximum\ number\ of\ features = \sqrt{number\ of\ features}$;
"log2": $maximum\ number\ of\ features = \log_2(number\ of\ features)$;
None (default): $maximum\ number\ of\ features = number\ of\ features$.

Table 5.7: Tuned parameters for balanced random forest classifier

Consequently, the results regarding metrics and confusion matrix based on the balanced random forest classifier with optimized parameters derived above are presented in Table.5.8 and Fig.5.13. Eventually, we do have an improvement with a 1% increment for recall of class 1 with respect to the original balanced random forest classifier, but the performance enhancement of 67 more correct classifications for class 1 and 1037 fewer misclassifications for class 0 is so trivial that it can be ignored, which indicates that the model tuning process for random forest in our specific case is not very useful. Nevertheless, the balanced random forest classifier

without class weight is still the best model according to the criteria.

Metrics		Value
Accuracy		0.81
Precision	Macro average	0.53
	Class 0	0.99
	Class 1	0.06
Recall	Macro average	0.80
	Class 0	0.81
	Class 1	0.80
F1 score	Macro average	0.50
	Class 0	0.89
	Class 1	0.10

Table 5.8: Metrics of final balanced RF model evaluation

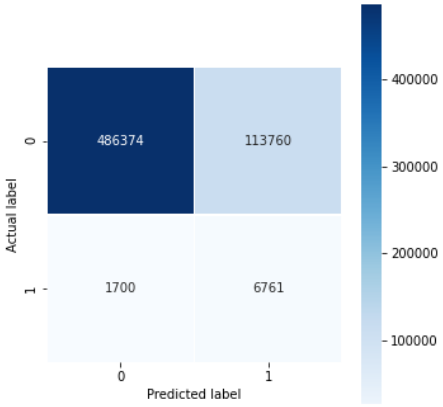


Figure 5.13: Confusion matrix of final balanced RF model evaluation

5.4 Gradient boosting tree

The fourth machine learning approach is GBDT, implemented by *XGBoost*. Similarly, we still follow the procedures for the investigation of time window and model tuning. Additionally, as suggested by the official documentation[10], we can set up a parameter named *"scale_pos_weight"*[11], which is the ratio between the quantity of the majority class and the minority class, to handle the imbalanced data so that we can simplify the model development procedure by using the original dataset without the need for sampling strategies.

5.4.1 Effect of time window

To begin with, we also inspect the impact of various lengths of time window with different quantities of features in the past of the target time bin by only varying

[10]https://xgboost.readthedocs.io/en/stable/tutorials/param_tuning.html

[11]It is the ratio between the amount of negative samples and positive samples and is used as a multiplier applied to every positive label weight, which is similar to the class weight in random forest. In our case, the value is 70.92.

the inputs and specifying the *"scale_pos_weight"* for the implementation of a basic *XGBoost* GBDT classifier, and the results are in Fig.5.14. Apparently, the model follows the same trend as before; the longer the time window, the better the performance, and it experiences a noteworthy performance decline, especially when it comes to the time windows with a duration starting from 500 ms to 2500 ms. Therefore, we will still stick to the 5000 ms-long time window with 10 samples and 110 features for the following model development.

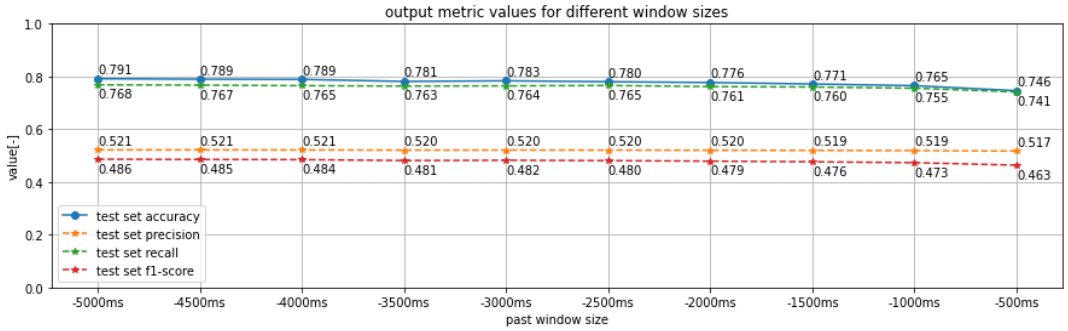


Figure 5.14: Resulting metrics of *XGBoost* GBDT classifier for different time window size

5.4.2 Model tuning and final experimental result

In order to improve the performance and finalize the result, we tune the model by modifying a group of parameters. Then, we apply a random search among all the combinations of parameters for 100 trails to output those with the best performance. Moreover, unlike random forest models with parallel trees, GBDT trains decision trees step-by-step, which means that, theoretically, GBDT can have any number of training iterations to build numerous successive trees, leading to possible overfitting on the test dataset. Thus, we introduce a validation dataset to avoid overfitting by early stopping the model training with limited iterations (number of trees) based on the observation of AUC (Area Under Curve) under the PR (Precision-Recall) curve as suggested by the documentation[12]. Additionally, for binary classification, the learning objective is to optimize the logistic regression probability. A summary of all parameters and the corresponding best combination is in Table.5.9.

[12]https://xgboost.readthedocs.io/en/stable/parameter.html

Parameter	Value range	Best value
Learning rate, which shrinks the feature weights	[0.01, 0.3]	0.270
Minimum loss reduction required to make a further partition on a leaf node	[0, 1]	0.232
Maximum depth of a tree	[2, 8]	7
Ratio of subsampling	[0.6, 1]	0.885
Subsample ratio of features	[0.7, 1]	0.991
L2 regularization term on weights	[1, 50]	15
L1 regularization term on weights	[0, 50]	41
"scale_pos_weight"	[60, 80]	64.832

Table 5.9: All tuning parameters and the best combination for *XGBoost* GBDT classifier

Previously, we trained the model with only 100 iterations for simplicity, and based on the best parameters, we keep training the *XGBoost* GBDT classifier until it starts to overfit on the validation dataset. Consequently, after 427 iterations, the model stops and the results are in Table.5.10 and Fig.5.15. On one hand, the model

Metrics		Value
Accuracy		0.96
Precision	Macro average	0.59
	Class 0	0.99
	Class 1	0.18
Recall	Macro average	0.73
	Class 0	0.97
	Class 1	0.49
F1 score	Macro average	0.62
	Class 0	0.98
	Class 1	0.27

Table 5.10: Metrics of *XGBoost* GBDT classifier evaluation

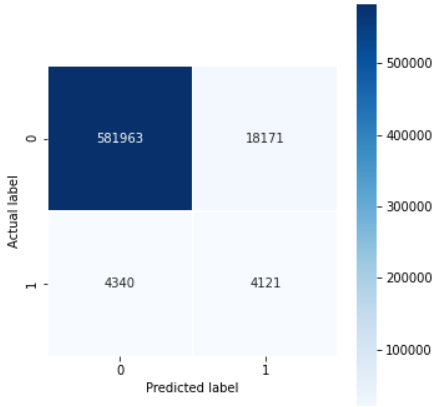


Figure 5.15: Confusion matrix of *XGBoost* GBDT classifier evaluation

performs very well for class 0, with a recall of 0.97 and only 18171 misclassifications. On the other hand, it can only identify half of the "loss" time bins, leading to a poor performance on class 1, which means that in our specific case, *XGBoost* GBDT classifier outperforms towards the majority class. On top of that, it is worth noting that the macro average F1 score of 0.62 is the highest one so far, indicating the

best overall performance.

However, the current training strategy tends to optimize the model to achieve outstanding performance from a holistic perspective, which is reflected in the final highest F1 score because the model only stops based on the evaluation of the AUC under the PR curve, which is biased towards the majority class due to the domination of "no-loss" time bins. Hence, another solution that balances the performance between classes is to regard the number of training iterations (trees) as a parameter and include it in another random search to find the stopping point where we have a relatively balanced performance by targeting the macro average recall instead of the default option.

As a result, we allocate another group of parameters[13] with a fixed training iteration of 203 and the final results are in Table.5.11 and Fig.5.16. Fortunately,

Metrics		Value
Accuracy		0.85
Precision	Macro average	0.53
	Class 0	0.99
	Class 1	0.07
Recall	Macro average	0.80
	Class 0	0.85
	Class 1	0.74
F1 score	Macro average	0.52
	Class 0	0.92
	Class 1	0.12

Table 5.11: Metrics of final *XGBoost* GBDT classifier evaluation

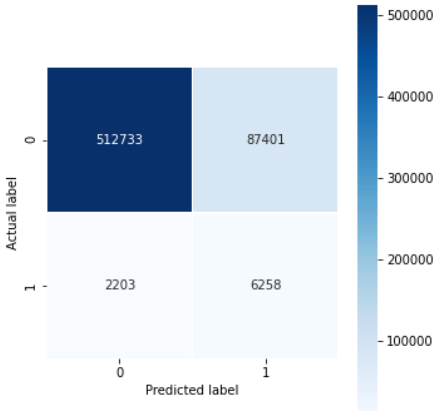


Figure 5.16: Confusion matrix of final *XGBoost* GBDT classifier evaluation

we significantly improve the performance for class 1 with a 25% increase in recall, and at the same time, the classifier can also reach a recall of 0.85 for class 0 to meet the threshold. It is without doubt that the upgraded model generates 69230 (87401-18171) more misclassifications for "no-loss" time bins when we try to reach a balance intentionally and artificially. In general, *XGBoost* GBDT classifier can be considered as an effective model with incredible prediction ability for "no-loss" time bins and decent performance for "loss" time bins, but even with *"scale_pos_weight"*,

[13]The other parameters are: 0.138 (learning rate), 0.115 (minimum loss reduction), 5 (max depth), 0.730 (subsampling ratio), 0.840 (subsample ratio of features), 10 (L2 regularization), 38 (L1 regularization), 77.912 (*"scale_pos_weight"*).

it still suffers from imbalance to pay certain trade-offs, which requires further optimizations and solutions.

5.5 Isolation forest

The fifth machine learning approach is isolation forest for anomaly detection, which considers "loss" time bins as outliers and tries to separate them from normal cases of "no-loss" time bins.

5.5.1 Training strategy

An isolation forest is an unsupervised model that doesn't construct and induce trees based on class labels but determines whether an input is an anomaly or not according to the final depth of the corresponding node. In order to enable isolation forest to work in a supervised manner, we need to first specify the portion of anomalies across the training dataset before the training phase so that the isolation forest model can define what the maximum depth is to consider an input as an anomaly. In other words, the trained isolation forest model can set a threshold for anomaly scores to distinguish outliers and normal cases based on the portion defined above.

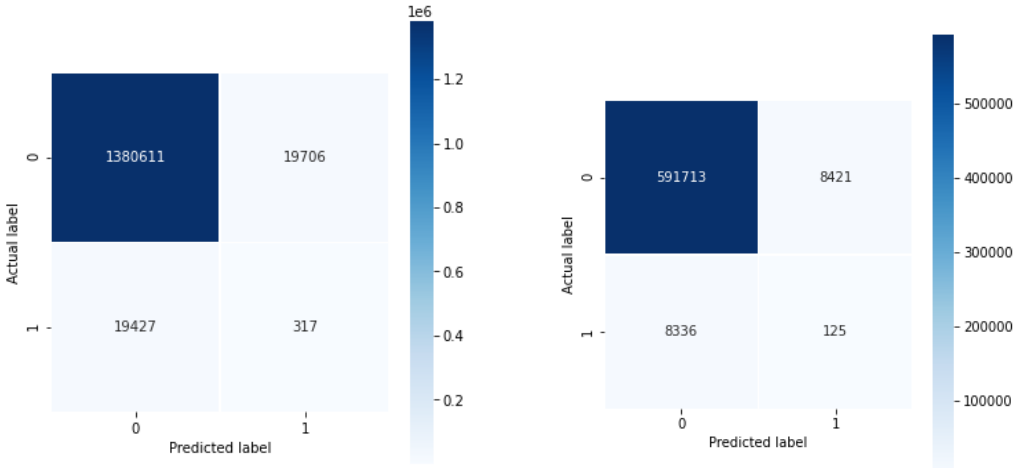
In our case, we still apply the seventy-thirty segmentation to define the training set and test set. In the training dataset, the percentage of "loss" time bins is 1.39% which will then be used by the *contamination* parameter for the isolation forest model. Finally, the model consists of 200 trees and is trained without any specific configurations for simplicity. Additionally, we refer to the original dataset without any sampling techniques, and the issues caused by an imbalanced dataset can be neglected because machine learning approaches for anomaly detection are specifically designed to cope with imbalanced data since anomalies are inherently rare events, and thus the data for anomaly detection problems are supposed to be imbalanced. Note that the length of time window for an isolated forest is still 5 seconds without any modifications.

5.5.2 Experimental result

After the training phase, we can apply the model to evaluate the anomaly conditions for both the training and test datasets. The resulting metrics are in Table.5.12 and the confusion matrices are in Fig.5.17.

Metrics		Training set	Test set
Accuracy		0.97	0.97
Precision	Macro average	0.50	0.50
	Class 0	0.99	0.99
	Class 1	0.02	0.01
Recall	Macro average	0.50	0.50
	Class 0	0.99	0.99
	Class 1	0.02	0.01
F1 socre	Macro average	0.50	0.50
	Class 0	0.99	0.99
	Class 1	0.02	0.01

Table 5.12: Metrics of isolation forest model evaluation for training set and test set



(a) Training dataset **(b)** Test dataset

Figure 5.17: Confusion matrices: the results of isolation forest model for training and test sets

According to the results, although the model is able to identify most of the

"no-loss" time bins with, so far, the highest recall of 0.99 for class 0 in both datasets to avoid massive misclassifications appearing in all of the classification models implemented before, the performance is still very poor because it cannot predict class 1 and results in an extremely low recall of 0.02 for the training set and 0.01 for the test set, which means the isolation forest model cannot recognize outliers of "loss" time bins in our case and completely fails the thesis objective.

5.6 Autoencoder

The last machine learning approach is the autoencoder for anomaly detection as well. An autoencoder is a neural network with a special architecture, and it will result in a relatively larger reconstruction error if the input is an anomaly since it only captures the latent representation of normal cases.

5.6.1 Training strategy

The Autoencoder is also an unsupervised machine learning tool, and the general workflow is similar to the isolation forest implemented in the previous section. However, the major difference between them is that the autoencoder only takes in samples for normal cases without any infiltration from outliers. This is because we want the NN to be only able to learn and reconstruct normal cases, and for any anomalous inputs, the autoencoder model will endeavor to reproduce them in the way it reproduces normal cases, leading to a larger difference between the input for the anomaly and the corresponding output. Therefore, we only include 70% from the "no-loss" dataset for training and use the rest 30% together with all of the "loss" dataset for testing.

In particular, we build a simple autoencoder[14] with architecture in Fig.5.18, in which we have 5 hidden layers and 8 neurons for the bottleneck, and regard the mean absolute error (MAE) between input and output as the reconstruction error, i.e. the cost function. By also considering the test set as a validation set for early stopping, we have trained and stopped the autoencoder model after 15 epochs.

[14]Other parameters:
i.) input shape and output shape: 77 neurons (7 samples in 3.5s × 11 statistics);
ii.) activation function: *relu* for hidden layers and *sigmoid* for input and output layer;
iii.) optimizer: *Adam* with a learning rate of 0.001.

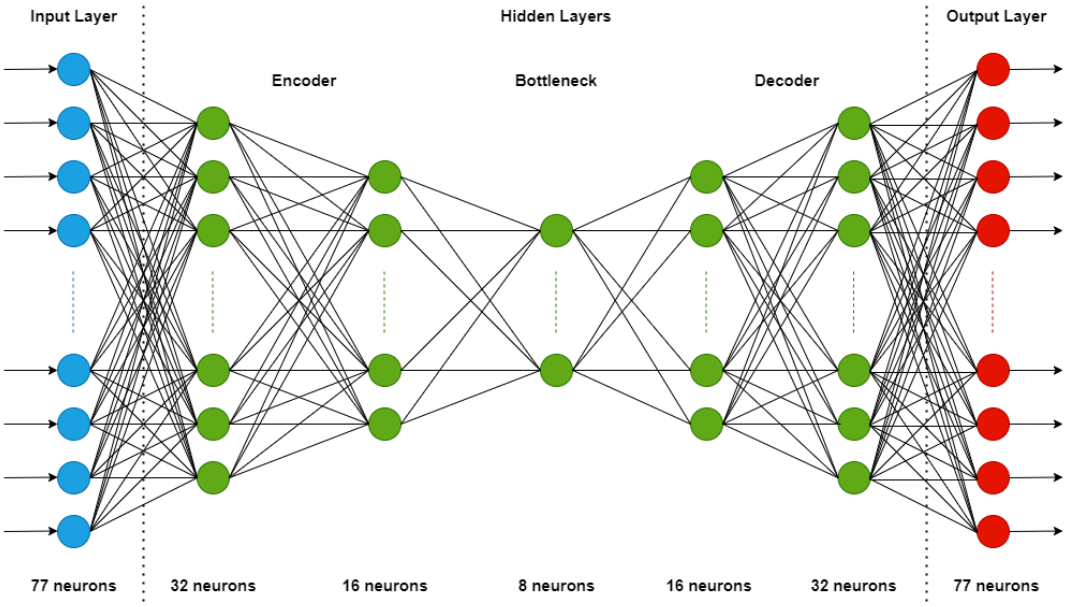


Figure 5.18: The architecture of autoencoder

5.6.2 Experimental result

The output of the autoencoder model trained before is the reconstruction of any input feature values within time bins. For the test dataset, we need to compute the MAE between each pair of input and output, and the way to tell anomalies apart from normal cases is based on the scale of the MAE. The "loss" time bins have higher values, while the "no-loss" time bins have lower values. Hence, for all the resulting MAE, similar to the contamination in an isolated forest, we can find a threshold above which the corresponding inputs are "loss" time bins. The threshold is simply allocated in all MAEs using the percentile based on the portion of anomalies in the test set. In our case, "loss" time bins occupy 4.489% of the test dataset and thus the percentile is $1 - 4.489\% = 95.511\%$. Fig.5.19 helps visualize MAEs and the location of the threshold, and as a result, the threshold is equal to 0.1305. As we can see in the figure, most MAEs are located on the left side of the threshold and concentrated around 0.05.

By applying the threshold on all MAEs, we are able to finally derive the loss condition for each corresponding input time bin, and the results are in Table.5.13 and Fig.5.20. The autoencoder model slightly outperforms the isolation forest with a relatively higher recall of 0.05 for class 1, but the general performance is still very poor and unacceptable, which indicates that it cannot be used for packet loss prediction either.

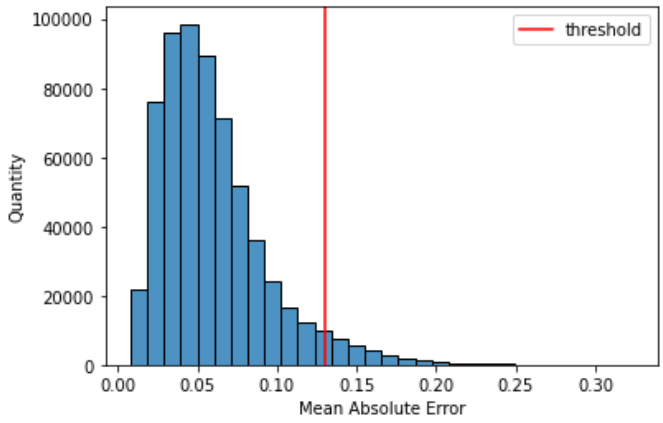


Figure 5.19: Distribution of mean absolute errors and the threshold

Metrics		Value
Accuracy		0.91
Precision	Macro average	0.50
	Class 0	0.96
	Class 1	0.05
Recall	Macro average	0.50
	Class 0	0.96
	Class 1	0.05
F1 score	Macro average	0.50
	Class 0	0.96
	Class 1	0.05

Table 5.13: Metrics of autoencoder model evaluation

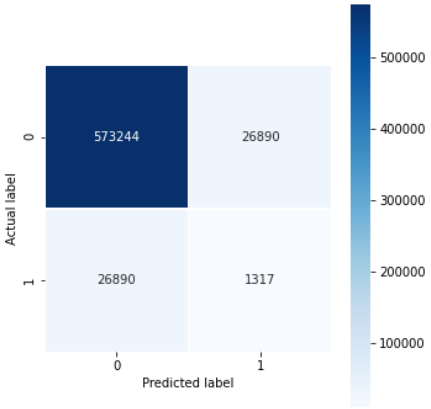


Figure 5.20: Confusion matrix of autoencoder model evaluation

5.7 Summary

At this stage, we have completed the basic machine learning approach development in terms of decision trees, neural networks, random forest, gradient boosting trees from classification and isolation forest, autoencoder from anomaly detection.

Generally speaking, classification methods play a dominant role in problem solving, while anomaly detection methods fail the objective completely. A summary of performances regarding class recall and macro average F1 score of all models developed above is in Table.5.14. According to the criteria, only the decision tree model with an undersampling strategy cannot generate a recall for class 0 above

0.80, and the best model is the balanced random forest model without class weight trained on the original dataset with the highest recall of 0.80 for class 1.

Table 5.14: *Summary of all machine learning approaches

Category	Model	Sampling strategy	Recall (class 0)	Recall (class 1)	Macro average F1 score
Classification	Decision tree	Original	0.98	0.14	0.55
	Decision tree	Undersampling	0.68	0.67	0.41
	Decision tree	Oversampling	0.94	0.29	0.53
	Decision tree	Combination	0.84	0.48	0.47
	DNN		0.82	0.69	0.47
	LSTM		0.83	0.69	0.47
	Random forest		0.96	0.47	0.60
	Balanced random forest with class weight	Original	0.85	0.64	0.51
	Balanced random forest without class weight		0.81	0.80	0.50
	XGBoost GBDT targeting overall performance		0.97	0.49	0.62
	XGBoost GBDT targeting class recall		0.85	0.74	0.52
Anomaly detection	Isolation forest		0.99	0.01	0.50
	Autoencoder		0.96	0.05	0.50

* The colored cells are notable values discussed in the literature. In particular, red indicates the lowest value and green is the highest.

On one hand, several classification methods are able to identify around 70% to 80% "loss" time bins and ensure the incorrect classifications for "no-loss" time bins are less than 20% at the same time. On the other hand, anomaly detection seems to be able to recognize most "no-loss" time bins, but it cannot predict "loss" time bins at all with too few correct classifications for class 1 to reach the thesis goal. The possible reason is that anomaly detection requires a larger difference between normal cases and outliers, but in our case, due to the complexities of the problem and the dynamic properties of the network, the distinctions between "loss" and "no-loss" time bins are not obvious enough, not to mention the variations among RTP flows. On top of that, the extremely good prediction for class 0 is due to the threshold based on the portion of "loss" time bins instead of the strengths of the anomaly detection algorithms themselves. For example, if we modify the class determination criteria by artificially moving the threshold in Fig.5.19 towards the left, the autoencoder model will induce more misclassifications for class 0 but still cannot generate a lot of correct predictions for class 1, which further emphasizes the failure of anomaly detection methods. Moreover, it's not worth continuing with any model tuning as well as optimization for anomaly detection simply because there's no potential to foresee an earthshaking improvement. For the exact same reason, some more analyses and models with similar poor performance regarding length of time window, LSTM autoencoder, etc., are not included in this thesis.

In particular, for classification methods, all models follow the same behavior of working well on "no-loss" time bins with higher recall for class 0 when working poorly on "loss" time bins with lower recall for class 1 and vice versa, which illustrates that it is not possible to achieve good performance for both classes at the same time. On top of that, the best model of balanced random forest presents the highest recall for class 1 but the lowest for class 0 leading to the largest amount of misclassifications for "no-loss" time bins. Note that due to the imbalanced dataset in which we have more than two million "no-loss" time bins, a model will induce a huge number of misclassifications for "no-loss" time bins even if the decrement in recall of class 0 is very small. For instance, according to Fig.5.11, the difference in recall between the original random forest and balanced random forest without weight is 0.15 (0.96-0.81) but it yields a huge difference of 92107 (114833-22726) for misclassifications in class 0. Although it cannot be ignored that incorrect predictions for "no-loss" time bins will generate an excess of wrong outcomes, which may lead to a waste of network resources for loss reaction, we still refer to the balanced random forest without weight as the best model, not only because of the criteria defined before but also the primary objective of this thesis. Since there's no way to improve the model for both classes at the same time, we want to find a balanced behavior in between to try our best to predict packet loss as much as possible without sacrificing and penalizing too many normal cases at the same time, and in our case, statistically speaking, a recall of 80% is an acceptable value.

Moreover, from an overall point of view, the first *XGBoost* GBDT classifier with the original dataset performs the best, with the highest macro average F1 score of 0.62, while most other models hover around 0.50, because it has the advantage of effectively identifying "no-loss" time bins with the second-highest recall of 0.97, and at the same time, it can still predict 49% of "loss" time bins, which is a mediocre outcome, while the decision tree model with the highest recall of 0.98 for class 0, can only successfully predict 14% "loss" time bins. Additionally, although the second *XGBoost* GBDT classifier is not the best model (second best) according to the criteria, it can still be considered an excellent one because it only induces 503 (6761-6258) less correct predictions for "loss" time bins but avoids 26359 (113760-87401) misclassifications for "no-loss" time bins. Meanwhile, with respect to models with comparable recall for class 0, like the balanced random forest with class weight, it has the highest recall for class 1 with an improvement of at least 5%. In a word, the *XGBoost* GBDT classifier has great potential to further optimize and the flexibility to tune the performance between classes, so we will also consider it for in-depth analysis.

To sum up, both balanced random forest classifier without weight and *XGBoost* GBDT classifier can produce balanced results with decent performance. Hence, in

the following, we will take into account both of them because of the outstanding performance and similar training set configuration without sampling techniques. As a result, we are able to see how far we can go from the two chosen classifiers by performing deeper analyses to further optimize the models and effectively solve the problem in the next chapter.

Chapter 6

In-depth analysis

In this chapter, we will focus on the elaboration of the balanced random forest classifier without class weight and the *XGBoost* gradient boosting tree classifier to perform more in-depth analyses in terms of constraints due to real cases, three ways for model optimization, and extra data from other sources. Note that in some of the following analyses, we only check the performance of the balanced RF classifier because either model optimizations or degeneration apply to *XGBoost* GBDT classifier in the same way.

6.1 Constraints due to real case scenarios

For all of the previous analyses and model developments, we always follow procedures from a data-driven perspective, taking into account only a few realities of real-time RTP communication for the sake of simplicity, comparability, and efficiency. In this section, we bring back all the constraints in real-case scenarios and investigate the impact on the model performance.

6.1.1 Predicting the farther future

Previously, we defined the problem as the prediction of packet loss in the exact next time bin of 500 ms (i.e. $Y_{t \to t+0.5}$) as in Equation.4.1. However, in reality, this kind of prediction is feasible but not useful because the system needs time to collect packets, calculate statistics over time aggregations, perform prediction using the

predefined model, and finally react to predicted packet loss, which means that the issues of packet loss can be tackled only after all of these procedures finish with a certain time consumption. That is to say, if we target the exact next time bin of 500 ms ($t \rightarrow t + 0.5$) for any time instance t, the packet loss would have already occurred after all of the calculations and reactions, rendering everything we have done obsolete and useless. Therefore, we need to predict a farther future[1]. For example, a time bin may be in the next 2000 ms instead of the subsequent one.

Predicting strategy:

To achieve this, we need to be aware of the time consumption of all procedures. In our case, the predicted time of a target time bin is around 47.39 ms for a balanced random forest classifier without weight and is even less with only 2.81 ms for *XGBoost* GBDT classifier, and the total time consumption including packet collection, statistic calculation, prediction, and reaction is less than 500 ms, which indicates that the potential packet loss that can be effectively responded to has to occur at least 500 ms from the current time instance. To put it in another way, we must aim for and predict the time bin at least in the next 1000 ms, i.e. $Y_{t+0.5 \rightarrow t+1}$. In such a scenario, since we focus on the time window of study, we still predict the subsequent time bin, $Y_{t \rightarrow t+0.5}$, but discard the exact past time bin of 500 ms ($X_{i,t-0.5 \rightarrow t}$) and refer to statistics in time bins from $t-5$ to $t-0.5$ ($X_{i,t-5 \rightarrow t-4.5}$ to $X_{i,t-1 \rightarrow t-0.5}$) to resemble the real-case scenario assuming that we are observing the communication at $t-0.5$. A more representative illustration is in Fig.6.1.

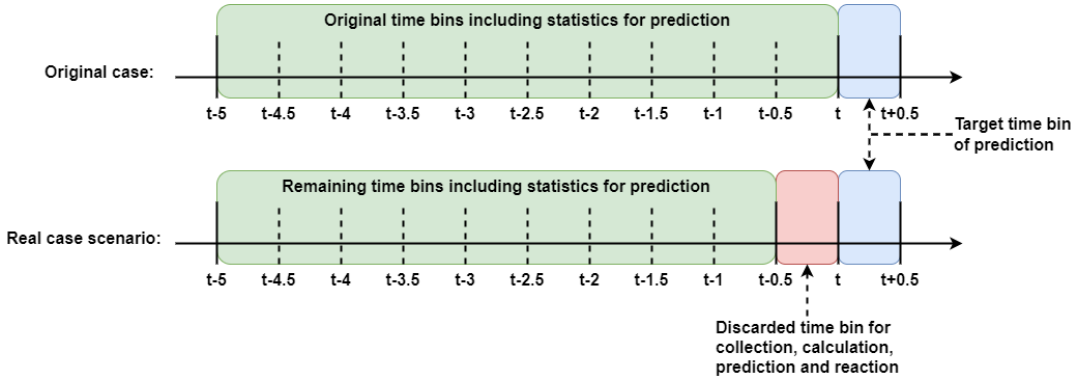
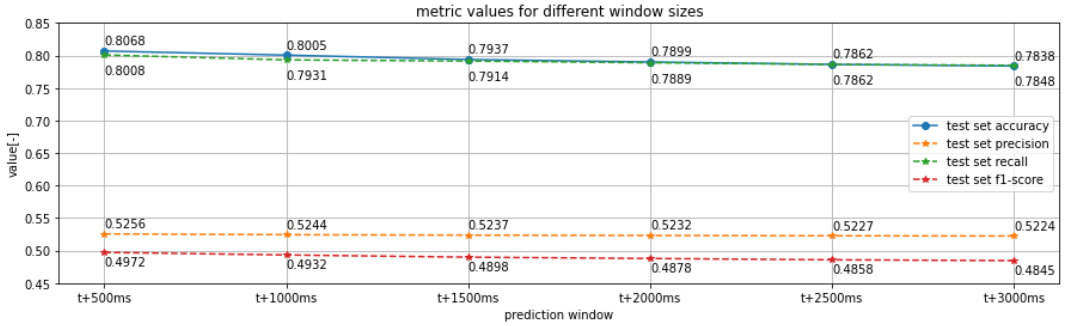


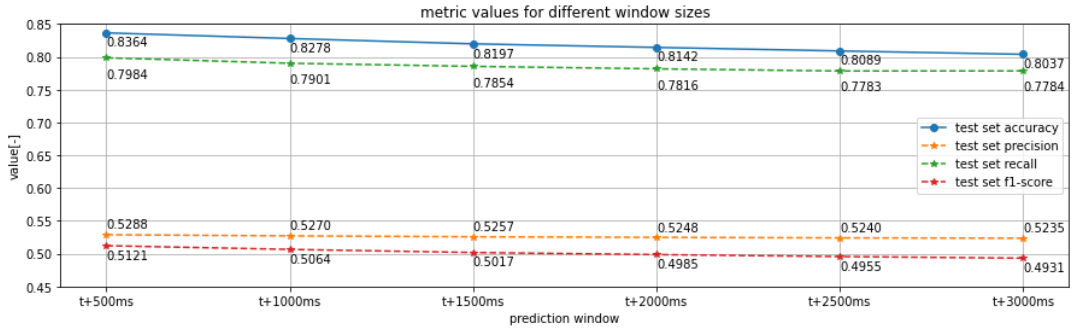
Figure 6.1: Description of new prediction strategy

[1]We did not consider this aspect during previous model development because, at that time, we didn't know the time consumption to determine the time bin for prediction.

In particular, it is not harmful to investigate time bins not only in the next 1000 ms but also in the next 1500 ms to 3000 ms[2]. Then, we retrain the two models with a different quantity of features in a shorter time window and compare their performance with that of the full time window for the original prediction. The results in terms of metrics are in Fig.6.2. As expected, with less information and more uncertainties included due to skipped time bins, the performances of both models indeed experience a degeneration as they predict farther and farther into the future, but the declines are not very serious with slight reductions, which means that it is also feasible to predict a more distant future. Hence, the implementation of a more complex and superior mechanism with a relatively larger time consumption but a more outstanding reaction to packet loss is absolutely possible if needed.



(a) Balanced RF classifier



(b) *XGBoost* GBDT classifier

Figure 6.2: Variation of metrics of the two chosen models with different predicting target time bin in the future

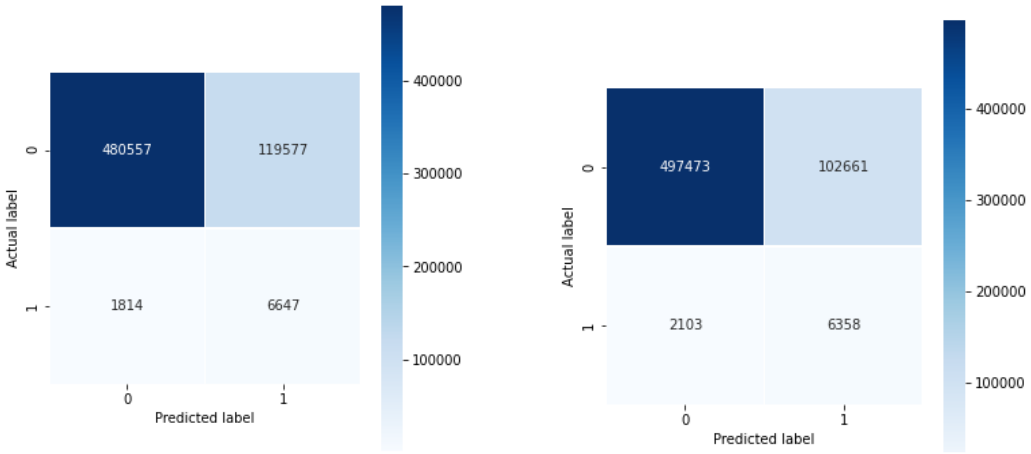
[2]In these cases, we simply discard more corresponding time bins in the past and use fewer available statistics.

Experimental result:

At last, the detailed results about the new target time bin in the next 1000 ms are in Table.6.1 and Fig.6.3. With respect to the original results in section 5.3.3 and

Metrics		Balanced RF classifier	*XGBoost* GBDT classifier
Accuracy		0.80	0.83
Precision	Macro average	0.52	0.53
	Class 0	0.99	0.99
	Class 1	0.05	0.06
Recall	Macro average	0.79	0.79
	Class 0	0.80	0.83
	Class 1	0.79	0.74
F1 socre	Macro average	0.49	0.51
	Class 0	0.89	0.90
	Class 1	0.10	0.11

Table 6.1: Metrics of the two chosen models in real case scenario targeting a farther future



(a) Balanced RF classifier **(b)** *XGBoost* GBDT classifier

Figure 6.3: Confusion matrix of the two chosen models in real case scenario targeting a farther future

section 5.4.2, the performance has a 1% drop in recall of class 0 for both models, and the recall for class 1 also has a 1% drop for the balanced RF classifier but remains the same for *XGBoost* GBDT classifier, which is totally acceptable and, consequently, the possibility and feasibility of model implementation considering a farther future for the real-case scenario is proved and supported.

6.1.2 Shuffling strategy for dataset

Normally, for the development of a machine learning approach, the datasets for training and testing should be randomly selected in order to represent the original dataset as much as possible, and in previous works, we always completely shuffle "loss" and "no-loss" datasets before methodology development, which means time bins from different flows are mixed together and data of flows are distributed almost homogeneously across training and test datasets. However, this is barely possible in real-case scenarios because, in reality, we ought to rely on a pre-trained model without the possibility of model upgrade to make predictions, which means that the incoming RTP traffic has never been included in model development and is very different from those used for model training, and thus, the statistics about new time bins vary from what we have in possession. Although both RTP flows, either used for model training or retrieved from new RTP sessions for evaluation, have similar properties, it is still an issue for a data-driven problem and a data-based model if the information of new data for evaluation is unknown and the patterns have never been analyzed and included. For this reason, we foresee a performance drop when it comes to real-case scenarios, and in this section, we will inspect the possible impacts using the balanced RF classifier without class weight and *XGBoost* GBDT classifier.

In order to simulate reality, we need to avoid the mixture of time bins from all flows and infiltration between training and test datasets. Therefore, instead of shuffling the entire dataset, we only shuffle[3] the dataset according to *"flow_id"* (section 3.2.4) that stands for a certain RTP flow to make sure that time bins in the same flow contained in the training set will not appear in the test set and vice versa[4].

However, shuffling flows for datasets induces two more problems: the evaluation of performance and finding the most proper combination of flows in the training set to generate the best performance. On one hand, different datasets for training due to various random shuffles lead to different performances, and it's reasonable

[3]We do not perform shuffles during the previous methodology development simply because we need to randomly shuffle the dataset multiple times to derive the average behaviour and the best performance for each model, which is totally a waste of time since the performance drop will appear regardless of the models.

[4]In this case, we still refer to the 70-30 partition for training and testing so that there's no way to guarantee a clean shuffle at the edge of the partition. But this phenomenon occurs just for one flow, and only several time bins from the same flow will appear in both sets, which is not a huge problem with respect to millions of time bins.

to refer to the average behavior instead of the best, worst, or intermediate one to evaluate the performance from a general point of view. Such an average behavior and related information can indicate the overall model performance and stationarity. On the other hand, the best performance can be considered the upper limit of the model in certain best-case scenarios. In other words, it is necessary to find a shuffling result to allocate flows that can represent the entire dataset and cover most of the patterns in the ideal case. As a result, we have done 50 trials of random shuffle to retrain 50 models based on different combinations of flows for both classifiers.

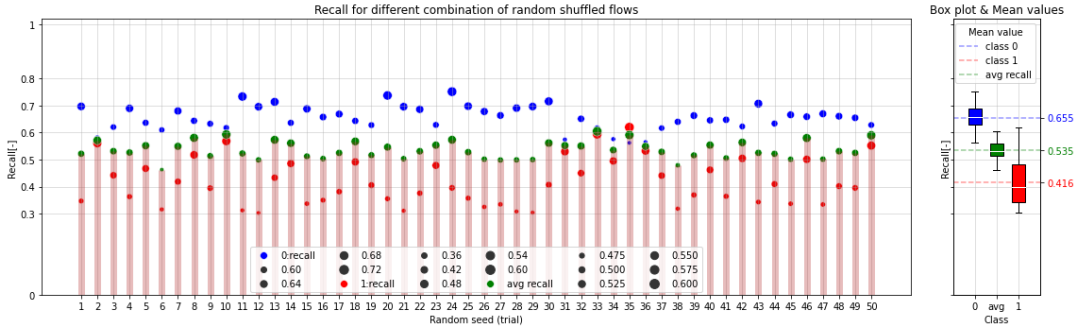
Experimental result:

The results in terms of recall are in Fig.6.4, in which the blue dots represent recalls for class 0, red dots represent recalls for class 1, and green dots are macro average recalls, and the box plot as well as mean values for all shuffles are on the right side. Additionally, the best performance among all shuffles indicates the upper limit of models in the best case scenario, while the mean behavior represents the general performance. According to the results[5]:

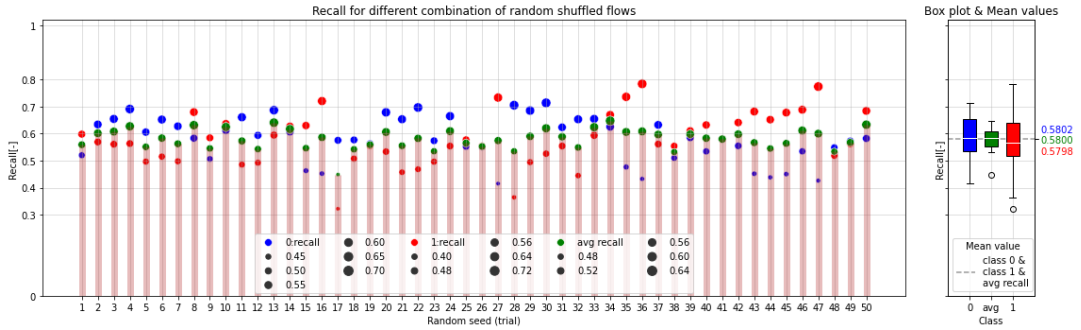
- For the balanced RF classifier, recalls for class 0 range between 0.6 and 0.7 with a mean value of 0.655 and recalls for class 1 range from 0.3 to 0.6 with a mean value of 0.416, which presents a huge performance drop of around 15% in recall for class 0 and 40% in recall for class 1, compared with the original result. The mean value for the macro average recall is only 0.535, leading to an extremely poor model like a coin toss with a half-half performance. Even in the best case scenario, the model can reach the best performance at trial 33 with recall of class 0 being only 0.616, recall of class 1 being only 0.591, and macro average recall being only 0.604, which is still 20% less than the original result, not to mention the incredible amount of 230565 misclassifications for class 0.

- For *XGBoost* GBDT classifier, the significant performance drop also applies. The mean values for all recalls are around 0.58, which is about 20% less than the original result, and the best performance is at trial 34, with recalls of

[5]In this case, we refer to models with a relatively simple configuration instead of the best parameters to save time, which means that the model may experience an improvement with parameter tuning, but it won't make a huge difference given the significant performance decline.



(a) Balanced RF classifier



(b) *XGBoost* GBDT classifier

Figure 6.4: Recalls (class 0 & 1) and macro average recall for the chosen models with different random shuffles of flows

0.625 (class 0), 0.670 (class 1), and 0.647 (macro average), which are relatively better than the balanced RF classifier but still very poor.

- Comparing both models, on one hand, the recalls for class 0 are relatively stable while the recalls for class 1 have larger fluctuations, which indicates that the real-case scenario affects "loss" time bins more than "no-loss" time bins since we have less data for class 1 and the patterns for "loss" time bins are also diverse. On the other hand, unlike in original performance when a balanced RF classifier generates a more balanced result, *XGBoost* GBDT classifier outputs a relatively balanced result in this case, and on top of that, it is less affected by shuffling flows given the better performance in best case scenario and mean value of macro average recall. Meanwhile, in a lot of cases, *XGBoost* GBDT classifier shows strange behavior where the performance of the minority class, class 1, is better, which only occurs once for the balanced RF classifier.

In general, the shuffling flows significantly affect both models, causing them to

experience a tremendous performance drop that cannot be neglected. Moreover, we didn't include another layer of constraint of predicting a farther future in this section, so the eventual result will be even worse considering all real-case scenarios, which requires more efforts and analyses to further optimize the models.

6.2 Model optimization

In order to tackle the issues caused by real-case scenarios where the models barely work, we need more technical solutions to improve the performance and eventually achieve the thesis goal. In this section, we come up with three approaches in terms of feature selection based on Recursive Feature Elimination; feature augmentation based on the preceding existence of packet loss; and "loss" time bin definition based on the correlation with the quantity of packet loss in a time bin.

6.2.1 Recursive feature elimination

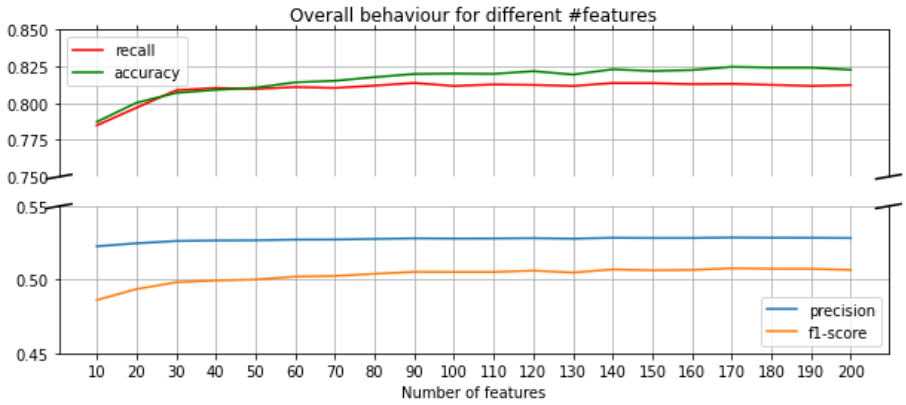
The first optimization for the models is related to feature selection. Previously, we selected features of statistics based on the fine characterization of packet loss in section 3.2.3 and referred to those with strong correlations in the time window of study following a time sequence. However, on one hand, statistics selected based on visualizations may not be sufficient and representative, and on the other hand, a discrete feature of a certain statistic in an individual time bin may also contain critical information for prediction even if adjacent statistics in its neighborhood are not included in features.

Therefore, a more technical and comprehensive feature selection process is performed through Recursive Feature Elimination (RFE)[30] which is a feature selection algorithm that recursively considers smaller and smaller subsets of all features by dropping out those that are least important and finally retaining the most crucial ones.

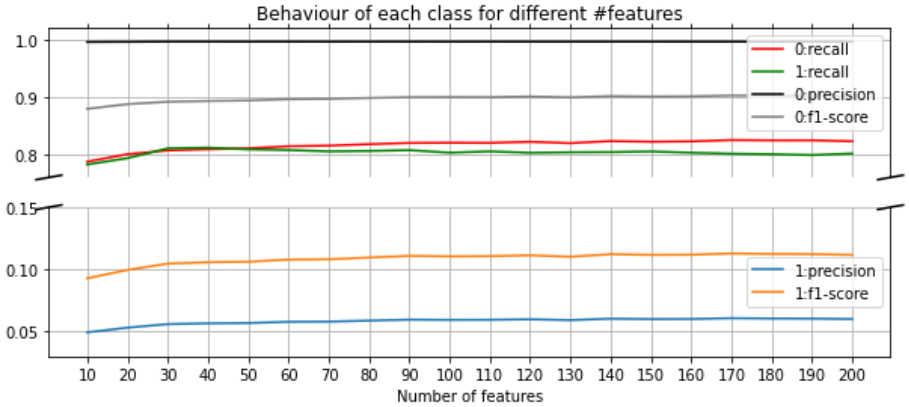
Outcome of RFE:

In our case, we have 96 types of statistics in total initially, and by removing those that absolutely have nothing to do with packet loss based on visualizations and domain knowledge, we have in hand 32 statistics left for RFE, including the 11 ones

selected and used before. We carry out RFE[6] to choose the number of features from 10 to 200 with a step of 10, and the results regarding metrics[7] for each quantity of features are posted in Fig.6.5. According to both figures, the model of the balanced RF classifier without class weight indeed experiences a performance improvement with a knee of around 30 features as we include more features, but the behavior



(a) Overall metrics



(b) Class metrics

Figure 6.5: Evaluation metrics for the results of Recursive Feature Elimination

[6]Notes: i.) In order to enable RFE to work, we build a larger dataset with 320 features (32 statistics × 10 samples in 5 seconds) in total; ii.) in this case, we only use the balanced RF classifier as reference, and there's no need to perform another round of RFE for *XGBoost* GBDT classifier.

[7]Besides the overall metrics, we also include class metrics to avoid potentially exceptional behaviors for a specific class.

starts to become stable with very small fluctuations when the number of features is greater than 80. Hence, in order to reduce the quantity of features for simplicity as well as efficiency and make comparison with respect to original features, we refer to 110 as the final number of features derived based on RFE.

Between new and original features, we only have 37 in common. Seventy-three selected automatically from new features are different from the original ones selected based on characterizations and visualizations, and they are mainly about 6 categories[8]:

- ***"interarrival_mean"***: mean value of interarrival time;
- ***"interarrival_moment3"***: the third moment of interarrival time;
- ***"len_udp_mean"***: mean value of UDP length;
- ***"len_udp_skew"***: skewness of UDP length;
- ***"rtp_inter_timestamp_std"***: standard deviation of differences between RTP timestamps;
- ***"inter_time_sequence_max_value_count_percent"***: percentage of the maximum value of differences between timestamp and sequence number.

For example, Fig.6.6 shows the fine characterization done in section 3.2.3 for the mean value of interarrival time (*"interarrival_mean"*) for the three cases. This statistic is not selected due to the relatively weak correlation with a gentle increment as approaching packet loss. But RFE relies on it and the reasons might be: i.) the patterns are not intense but still visible and distinguishable compared to the constant behavior in case 3 when there's no loss; ii.) both cases 1 and 2 share almost the same behavior, which indicates that the mean value of interarrival time applies for most "loss" time bins regardless of concentrated or sparse loss; iii.) besides the difference in trend, the difference between case 1&2 and case 3 in terms of numerical value is very large. The mean value of interarrival time is always a little bit larger than 0.27 when no packet loss plays its role, while it

[8]The remaining features are discrete statistics allocated in individual time bins without following chronological orders. For example, the Kurtosis of interarrival time only has two features: *"interarrival_kurtosis_minus_500ms"* in the time bin from 500 ms ago and *"interarrival_kurtosis_minus_3000ms"* from 3000 ms ago. Most of the new features still follow the time sequence in the time window of study with 10 samples, which means RFE also identifies the trends and utilizes the characteristics of packet loss for prediction.

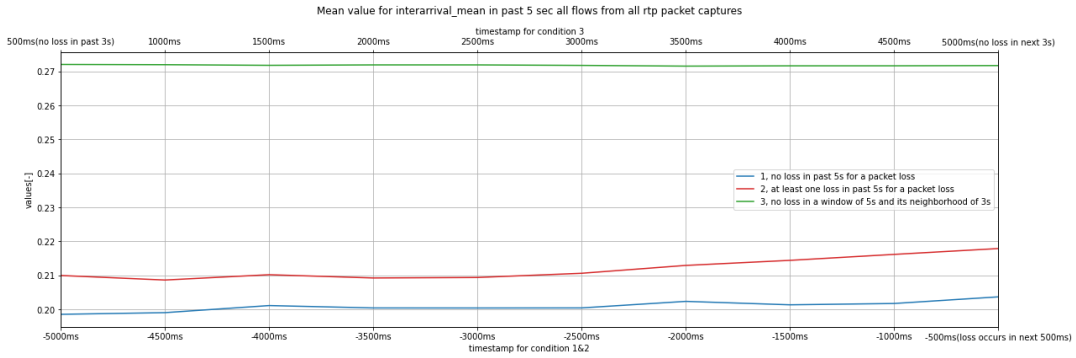


Figure 6.6: Fine characterization of mean value of interarrival time

spans over a smaller range starting from 0.20 to 0.22 when packet loss affects the communication.

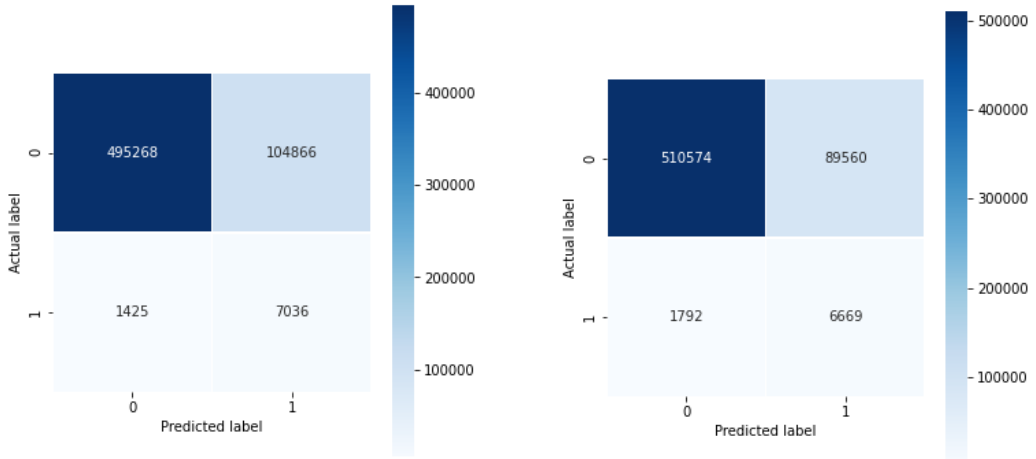
Experimental result:

Finally, we retrain the models based on the new features from RFE, and the results for the test set are in Table.6.2 and Fig.6.7. Compared with the original results for both models in section 5.3.3 and section 5.4.2, RFE is truly able to provide more informative and advantageous features for the problem, and we confirm the model improvement with 8894 (113760-104866) fewer misclassifications of "no-loss" time bins and 275 (7036-6761) more correct classifications of "loss" time bins for the balanced RF classifier, and nearly the same amount of correct classifications of "no-loss" time bins and 411 (6669-6258) more correct classifications of "loss" time bins for *XGBoost* GBDT classifier. Nevertheless, statistically speaking, improvements of 2% (0.83-0.81) and 3% (0.83-0.80) in recalls of both classes for the balanced RF classifier are not significant, while for *XGBoost* GBDT classifier, the recall of class 1 has a slightly larger rise of 5% (0.79-0.74) but the recall (both are 0.85) of class 0 remains the same, which indicates that, although the original features are not the best choice since they are defeated by RFE, they are still sufficient and meaningful, and features selected according to trends based on visualizations and characterizations of packet loss can be considered as successful[9].

[9]Notes: i.) in some of the following in-depth analyses, we will still refer to the original features for the sake of comparisons; ii.) we also try to retrain DNN based on new features, and the improvement is proved by the recall of 0.87 for class 0 and 0.70 for class 1, which are 5% and 1% greater than the original ones of DNN.

Metrics		Balanced RF classifier	*XGBoost* GBDT classifier
Accuracy		0.83	0.85
Precision	Macro average	0.53	0.53
	Class 0	0.99	0.99
	Class 1	0.06	0.07
Recall	Macro average	0.83	0.82
	Class 0	0.83	0.85
	Class 1	0.83	0.79
F1 socre	Macro average	0.51	0.52
	Class 0	0.90	0.92
	Class 1	0.12	0.13

Table 6.2: Metrics of the two chosen models with new features based on RFE



(a) Balanced RF classifier **(b)** *XGBoost* GBDT classifier

Figure 6.7: Confusion matrix of the two chosen models with new features based on RFE

6.2.2 Considering packet loss as features

The second model optimization is based on the augmentation of features. Besides available statistics about packets aggregated in time bins in the past, it is also possible to consider the presence of packet loss themselves (i.e., class label) in the past of a target time bin as features for prediction. Therefore, we update the original dataset with ten new columns of packet loss condition (binary class label) for each of the 10 time bins in the time window of study to have 120 features eventually. Note that in this case, we need to discard the first 10 samples for each individual dataset with continuous time series since the time bins in the past are not adequate for them.

Preliminary analysis:

In order to assess the impact of the feature augmentation and determine whether to proceed, we use and retrain the model of the balanced RF classifier without weight as a reference, and the results are in Table.6.3 and Fig.6.8.

Metrics		Value
Accuracy		0.86
Precision	Macro average	0.53
	Class 0	0.99
	Class 1	0.07
Recall	Macro average	0.82
	Class 0	0.86
	Class 1	0.78
F1 score	Macro average	0.53
	Class 0	0.93
	Class 1	0.13

Table 6.3: Metrics of balanced RF classifier with loss conditions as features

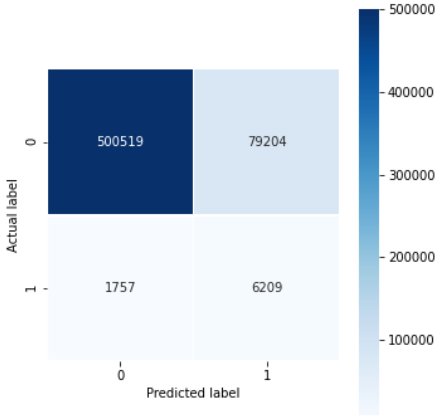


Figure 6.8: Confusion matrix of balanced RF classifier with loss conditions as features

According to the results, adding features about packet loss will improve the performance for class 0 with a 5% increment (0.86-0.81) in recall but impede the prediction for class 1 with a 2% (0.80-0.78) drop compared to the original result, which means that the newly added features about loss conditions optimize the model towards "no-loss" time bins, and this makes sense because for most "no-loss" time bins, the past 10 time bins do not have losses either, and all feature values in the past time window of study are equal to 0, which is more uniform to present a more remarkable trend with respect to "loss" time bins having an irregular distribution of packet loss in the past. Moreover, a decrement of 2% results in only 485 (6694-6209) more misclassifications for class 1 but an improvement of 5% for class 0 leads to 35629 (114833-79204) more correct classifications. Meanwhile, compared to those models[10] with relatively higher and comparable recalls for class 0 smaller than 0.90, augmented features about loss conditions can significantly improve the prediction for "loss" time bins with increments for recall ranging from

[10]i.e., DT with combination strategy has 0.84 (class 0) and 0.48 (class 1), DNN has 0.82 and 0.69, LSTM NN has 0.83 and 0.69, and balanced RF with class weight has 0.85 and 0.64, *XGBoost* GBDT classifier towards class recall has 0.85 and 0.74.

4% (0.78-0.74) to 30% (0.78-0.48), not to mention the better recall for class 0 as well.

Although the best model remains the same without packet loss conditions as features according to the criteria, we can still consider this strategy as an effective one with the potential to improve the model by fulfilling the thesis objective and avoiding superabundant incorrect predictions at the same time. On top of that, here we only investigate the performance of the original model without considering other aspects, and potentially, it could play a more important role in real-case scenarios to compensate for the diversity among flows, so in the following, we will continue with more analyses and optimizations by considering the new features as well as related aspects and making comparisons.

Considering flow shuffling:

The improvement has been proved by the preliminary analysis, and in this section, we check the influence over performance by performing 10 trials[11] of shuffling flows for both models with the augmented features of packet loss condition in the past.

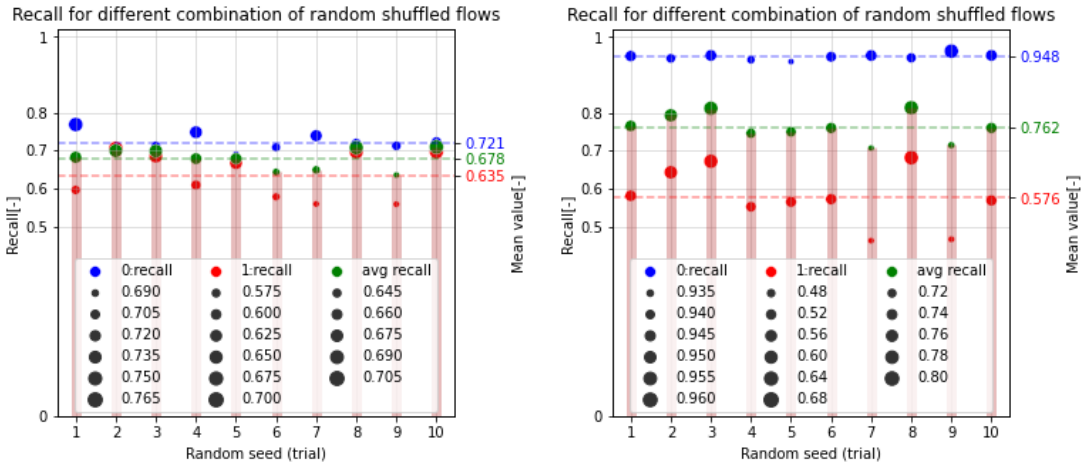
According to the results[12] related to recalls of each trial in Fig.6.9:

- The balanced RF classifier experiences an obvious improvement with respect to the original result in Fig.6.4a especially when it comes to class 1, with an increment of 21.9% (0.635-0.416) in the mean value of recall. Together with the improvement of 6.6% (0.721-0.655) in recall of class 0 and 14.3% (0.678-0.535) in macro average recall, the new features of packet loss condition can significantly alleviate the problem caused by flow shuffling to reach an acceptable performance.

- The *XGBoost* GBDT classifier presents a huge improvement for the recall of class 0 with 36.8% (0.948-0.580) and the macro average recall with 18.2% (0.762–0.580), while it remains almost the same for class 1 compared with the original result in Fig.6.4b, which illustrates that the augmented features can

[11]In this case, we don't need to do it for 50 trials as before because we want to roughly investigate the performance and perform further analysis, taking into account every aspect later to save time.

[12]Note that here we only consider the mean value of recalls to investigate the general performance and do not take into account the best performance since we have only performed 10 trials.



(a) Balanced RF classifier **(b)** *XGBoost* GBDT classifier

Figure 6.9: Recalls for the chosen models with different random shuffles of flows considering packet loss as features

help identify most "no-loss" time bins regardless of flows and maintain the same prediction ability for "loss" time bins at the same time.

- When comparing both models, they share the same behavior as in the basic model development, where the balanced RF classifier produces a more balanced performance. At the same time, both models have relatively steady behavior with small variations for recalls, especially for "no-loss" time bins, which means that the new features can optimize the models to stabilize the performance in any case.

In a word, the augmented features of loss condition in the past 5 seconds can substantially help the models distinguish different time bins even with the disturbance caused by various flows because: i.) unlike other features with normalized and diverse values, packet loss condition is a binary feature; ii.) for most "no-loss" time bins, their past 10 samples are also "no-loss" time bins, which helps the models easily identify class 0. However, although both models experience significant improvements, the performance of *XGBoost* GBDT classifier is still better because, on one hand, the advantage of the balanced RF classifier for class 1 with an enhancement of 5.9% (0.635-0.576) is not comparable with the drawback for class 0 with a huge drop of 22.7% (0.948-0.721)[13], and on the other hand, a recall of 0.721 for "no-loss"

[13]A difference of 5.9% for "loss" time bins only indicates 500 samples, but 22.7% for "no-loss"

time bins doesn't meet the criteria and will yield massive misclassifications that are totally not tolerable. Consequently, we only refer to *XGBoost* GBDT classifier to derive the final result in the following.

Considering all related aspects:

In this section, we use *XGBoost* GBDT classifier as our final machine learning model to obtain the final result, considering every related aspect (4 in total) in terms of model optimization and constraints caused by the real-case scenario:

- **Constraint 1:** Referring to the real-case scenario of predicting $t+1$ discussed in section 6.1.1;
- **Constraint 2:** Shuffling flows to simulate the real-case scenario described in section 6.1.2, and finding the average behavior for general performance and a specific combination of flows for the best performance;
- **Optimization 1:** Referring to the best features derived from Recursive Feature Elimination in section 6.2.1 rather than the original ones selected based on packet loss characterization;
- **Optimization 2:** Using packet loss conditions in the time window of study before the target time bin as features.

Fig.6.10 shows the results[14] of *XGBoost* GBDT classifier with a similar plot for 50 trials of shuffling flows as in section 6.1.2. First of all, with all possible optimizations and constraints, the final model presents a thorough improvement with acceptable performance based on the mean values of recalls. Secondly, the model performs very well for class 0 by identifying 94.6% of "no-loss" time bins but poorly for class 1 by only successfully predicting 54.0% of "loss" time bins. Moreover, the behavior for "no-loss" time bins is very stable while the prediction for "loss" time bins has a lot of fluctuations, which indicates that the variations among flows have a large impact, so that the model can sometimes predict packet

time bins leads to 136230 samples!

[14]Notes: i.) in this case, we use a relatively simple but constant parameter configuration with a validation dataset, which means that we may still have a slight improvement with parameter tuning, and the reason we do not perform model tuning is that it is needed for each trial; ii.) we also go through the exact same procedure for the balanced RF classifier, but the results are not good as expected and thus we do not include them in this thesis.

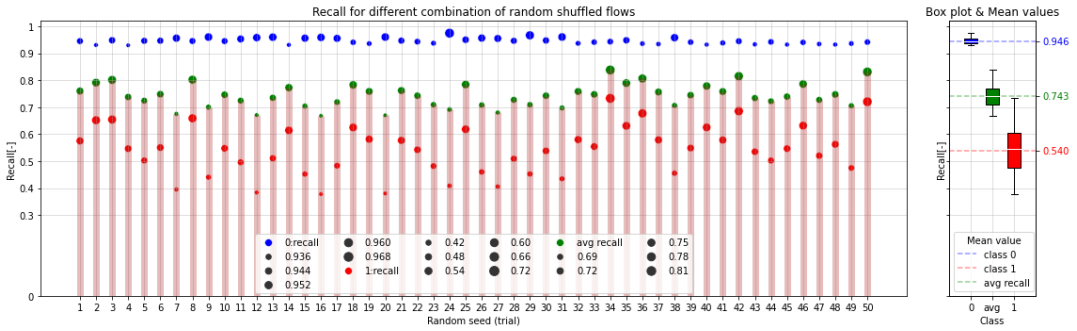


Figure 6.10: Recalls for *XGBoost* GBDT classifier with different random shuffles of flows considering all related aspects

loss efficiently but sometimes barely work. Finally, the model generates a fabulous result with a recall of 0.944 (class 0), 0.744 (class 1), and 0.838 (macro average) at trial 34, and thus, in the most ideal case, *XGBoost* GBDT classifier is very reliable given the difficulties of the problem.

To sum up, *XGBoost* GBDT classifier can be considered a useful but not perfect model, which can identify most of the normal time bins without losses and predict more than half of the time bins with packet losses in the meantime. Due to the imbalance between classes, the outstanding performance of class 0 helps avoid a great deal of misclassifications and, consequently, saves plenty of network resources for unnecessary reactions. But it is without doubt that the model can only solve half of the problems induced by packet loss, which partially achieves the thesis goal.

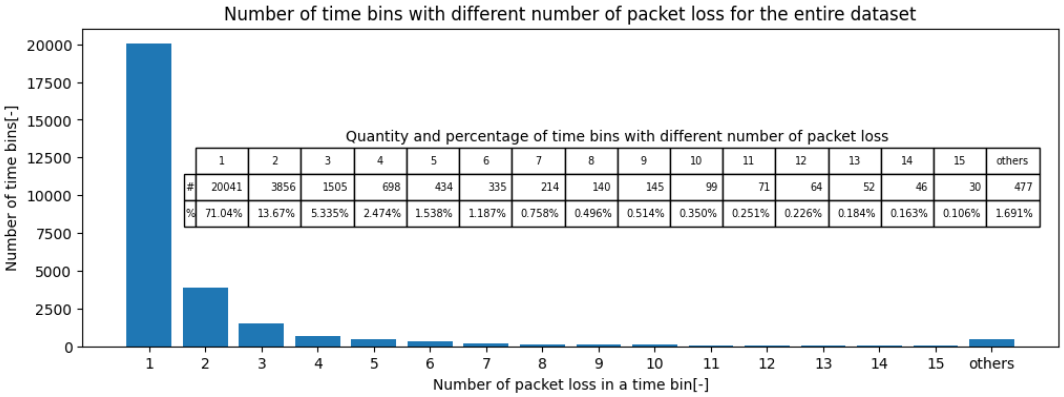
6.2.3 Correlation with quantity of packet loss

The last optimization is related to the definition of "loss" time bin by considering the amount of packet loss in a bin. In our dataset, "loss" time bins have different amounts of packet loss, and previously, we didn't consider the various quantities during analyses and model development. Technically speaking, less packet loss in a time bin is due to a mild congestion or other trivial network problems, which will slightly affect the RTP sessions and result in fewer influences over statistics of affected time bins with indistinct and ambiguous trends compared to time bins with a larger quantity of packet loss caused by relatively more serious network issues. Hence, "loss" time bins with more losses are supposed to be more easily predicted because of a more distinctive trend and characteristic. To sum up, there exist correlations between prediction performance and the number of packet losses,

so in this section, we try to elaborate on the two chosen models by concentrating on the quantity of packet loss in each "loss" time bin.

Analysis for quantity of packet loss in time bins for correct or incorrect predictions:

Firstly, in order to reach the goal, we update the dataset by adding one more column recording the number of packet losses in each bin, named *"num_packet_loss"*. The bar chart in Fig.6.11 together with a detailed table illustrates the distribution of time bins with different quantities of packet loss in the entire dataset, and most (71.04%) of "loss" time bins only have one loss.

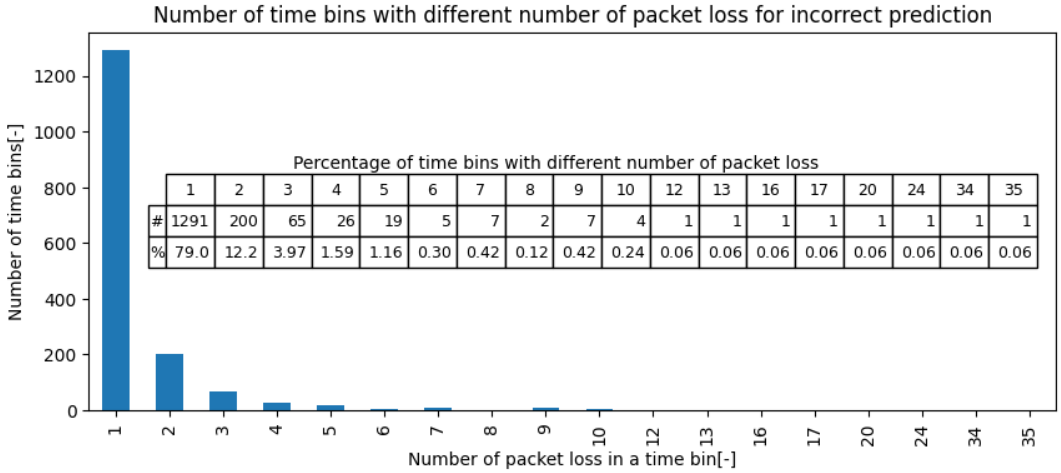


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	others
#	20041	3856	1505	698	434	335	214	140	145	99	71	64	52	46	30	477
%	71.04%	13.67%	5.335%	2.474%	1.538%	1.187%	0.758%	0.496%	0.514%	0.350%	0.251%	0.226%	0.184%	0.163%	0.106%	1.691%

Figure 6.11: Number of time bins with different quantities of packet loss for the entire dataset

Secondly, we check the distribution of "loss" time bins with various quantities of loss for correct as well as incorrect classifications of the result from the model of balanced RF classifier without weight as in Fig.5.13, and compare it with the original distribution in the entire dataset:

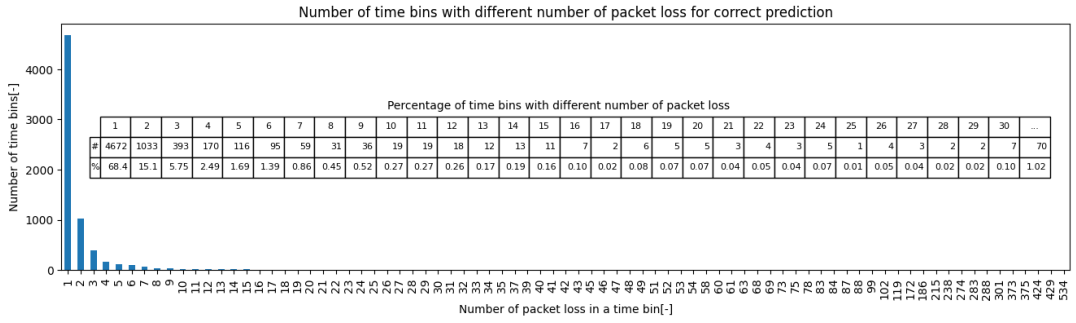
- Fig.6.12 indicates the distribution of time bins with different amounts of packet loss in outputs of misclassification. Among all of the incorrect classifications of "loss" time bins, those with only one loss occupy 79.0% which is larger than the entire dataset with a percentage of 71.04% and in the meantime, time bins with more losses have smaller portions. Apparently, as expected, target bins with only one loss are harder to predict with more misclassifications.



	1	2	3	4	5	6	7	8	9	10	12	13	16	17	20	24	34	35
#	1291	200	65	26	19	5	7	2	7	4	1	1	1	1	1	1	1	1
%	79.0	12.2	3.97	1.59	1.16	0.30	0.42	0.12	0.42	0.24	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06

Figure 6.12: Number of time bins with different quantities of packet loss for the incorrect classifications

- Fig.6.13 presents all the possible numbers of packet losses in any time bin among correct classifications and their corresponding number of bins. Unlike before, 68.4% of correct classifications just have one loss, which is similar to the entire dataset with a small variation, and time bins with more losses occupy more percentages. On top of that, the correct predictions include a specific portion of time bins with even more packet losses up to a maximum of 534, while in the misclassifications, the maximum number of packet losses is only 35, which means that the model is able to successfully make predictions when it comes to a time bin with a larger quantity of packet losses due to severe network problems that cause mighty strong impacts on the RTP flows and, consequently, lead to significant trends in statistics for prediction.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	...
#	4672	1033	393	170	116	95	59	31	36	19	19	18	12	13	11	7	2	6	5	5	3	4	3	5	1	4	3	2	2	7	70
%	68.4	15.1	5.75	2.49	1.69	1.39	0.86	0.45	0.52	0.27	0.27	0.26	0.17	0.19	0.16	0.10	0.02	0.08	0.07	0.07	0.04	0.05	0.04	0.07	0.01	0.05	0.04	0.02	0.02	0.10	1.02

Figure 6.13: Number of time bins with different quantities of packet loss for the correct classifications

Based on the results, we prove and confirm the correlation between prediction performance and the quantity of packet loss in the predicting time bin. The more losses we have in a time bin, the better model performance we end up with. Technically, it seems that the relevance is not very strong since in both classifications, time bins with one loss are the majority. This is a consequence of the domination of the one-loss time bin in the entire dataset, which is an intrinsic property of the problem.

Different definition of "loss" time bin:

Although the thesis objective is to predict the future packet loss regardless of the quantity, solo loss in a time bin can be assumed to be a non-detrimental issue in our network since we have hundreds and even thousands of received packets aggregated in a bin. Hence, it is more valuable to perform in-depth analyses in terms of time bins with losses greater than 1, not only because of the assumption but also because of the foreseeable better model performance.

In fact, among all of the "loss" time bins in the test set, those with more than one loss have 343 for incorrect prediction and 2155 for correct prediction, with a prediction accuracy (recall) of 86.27%. Furthermore, those with more than two losses have 143 for incorrect prediction and 1122 for correct prediction, resulting in a further higher prediction accuracy (recall) of 88.70%. Both of them are greater than the original recall (80%) of class 1 with a significant improvement. Therefore, another possible solution for model improvement is to change the definition of "loss" time bin by only considering "loss" time bins with more losses and discarding those with fewer losses. That is to say, we remove[15] a portion of the original dataset to feed and retrain the models. In particular, we maintain the quantity of "no-loss" time bins and specify two cases:

1. Regarding time bins with losses greater than 1 as "loss" time bins and removing time bins with losses equal to 1. In this case, we have 8166 "loss" time bins left;

[15]Notes: i.) the removal of data leads to a more imbalanced dataset, but this is not an issue since both models are able to handle imbalances internally; ii.) in this specific case, time bins with fewer losses will be totally neglected and we do not care about the prediction for them during model development. In reality, this is not a problem for discarded "loss" time bins because, on one hand, it is a preferable outcome if the prediction is class 1 since the bins do have losses, and on the other hand, it is an acceptable result if the prediction is class 0, since we assume those with fewer losses are not to be "loss" time bins and they are not harmful.

2. Regarding time bins with losses greater than 2 as "loss" time bins and removing time bins with losses equal to 1 or 2. In this case, we have 4310 "loss" time bins left.

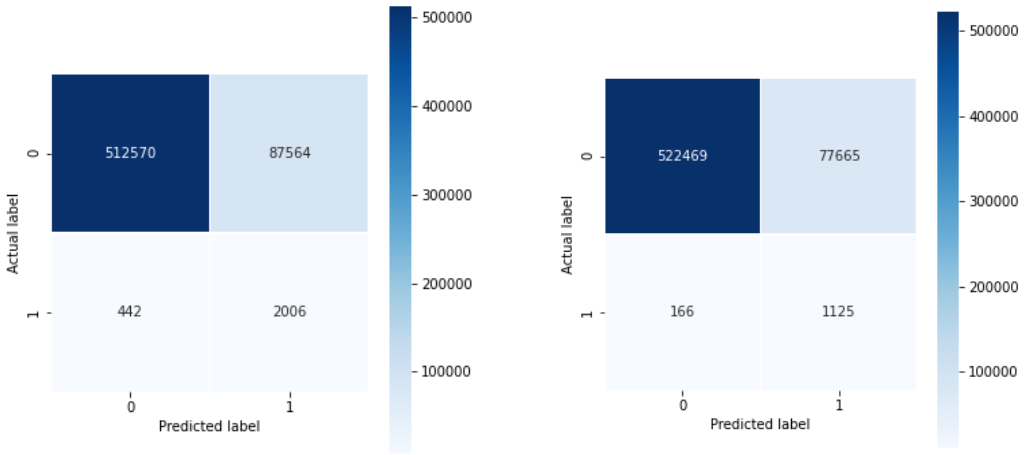
Afterwards, we follow the same procedure of training and testing for the model of balanced RF classifier[16], and the results for both cases are in Table.6.4 and Fig.6.14. First of all, both cases present an improved performance for both classes

Metrics		**Case 1**	**Case 2**
Accuracy		0.85	0.87
Precision	Macro average	0.51	0.51
	Class 0	0.99	0.99
	Class 1	0.02	0.01
Recall	Macro average	0.84	0.87
	Class 0	0.85	0.87
	Class 1	0.82	0.87
F1 socre	Macro average	0.48	0.48
	Class 0	0.92	0.93
	Class 1	0.04	0.03

Table 6.4: Metrics of balanced RF classifier for case 1&2 with different definitions for "loss" time bin

with larger augmentations in recalls. Secondly, case 2 of "loss" time bins with losses greater than 2 generates a substantially significant improvement by reaching 0.87 for recalls of both classes, which is, as expected, better than not only the original dataset but also case 1, including time bins with losses equal to 2. More importantly, case 2 results in the best performance up to now according to the criteria, and at the same time, it also produces a decent recall of 0.87 for class 0, leading to only 77665 misclassifications. This is so far the first time that we are able to balance both classes without having to pay a trade-off and improve performance for them at the same time. However, although we derive an outstanding outcome by ignoring time bins with fewer losses, it is without doubt that we cut down the majority of "loss" time bins to have only a few thousand pieces of data left, which is totally not comparable with the millions of "no-loss" time bins. In a word, we artificially simplify the problem to chase a better result, but it may not be worth the effort to identify just thousands of outliers out of millions of normal cases.

[16]Here we only use one model to check the effect of the quantity of packet loss for simplicity and use both models in the later stage when we consider all related aspects.



(a) Case 1 **(b)** Case 2

Figure 6.14: Confusion matrices: the results of balanced RF classifier for case 1&2 with different definitions for "loss" time bin

Nevertheless, determining whether an approach is meaningful or not is, of course, another problem that is related to the requirements of specific applications or QoE and is out of the scope of the thesis.

Considering all related aspects:

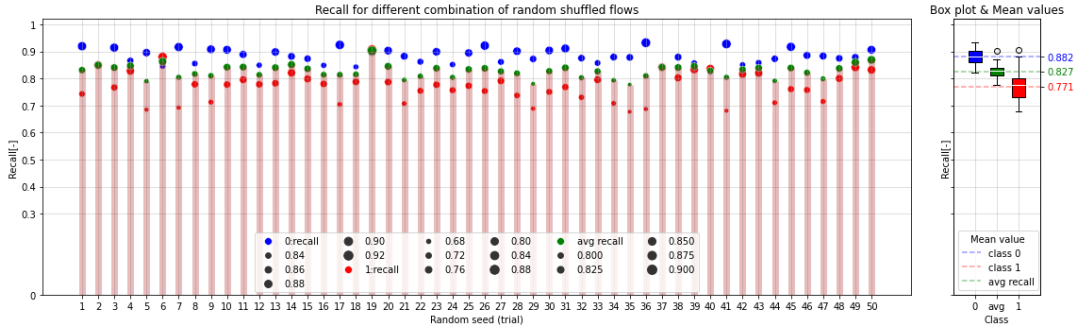
Based on previous analyses, we perceive a great potential to improve the models, and in this section, we will present the final result by taking into account all of the related aspects (5 in total) as in section 6.2.2 and concentrating on the correlation with quantity of packet loss to include one more condition:

- **Optimization:** Only considering time bins with losses more than 2 as "loss" time bins and ignoring "loss" time bins with losses less than or equal to 2 by removing them from the entire dataset.

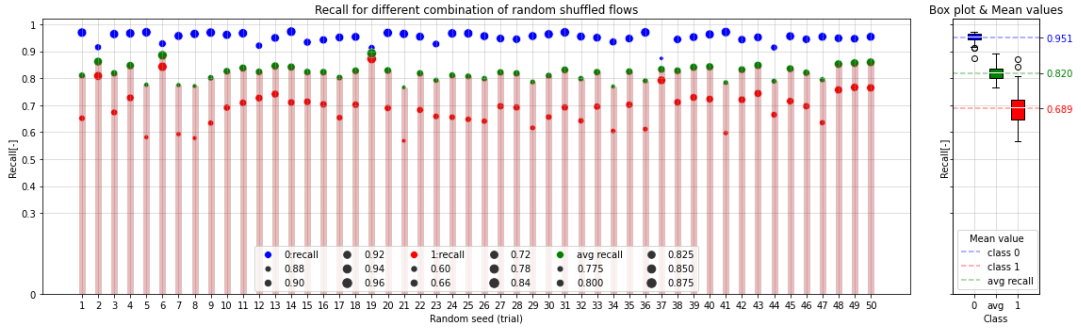
Consequently, the results of both models in terms of recall for 50 trials of different flow shuffles as in section 6.1.2 are presented in Fig.6.15.

According to the results:

- Unlike the result in section 6.2.2, where the balanced RF classifier doesn't work



(a) Balanced RF classifier



(b) *XGBoost* GBDT classifier

Figure 6.15: Recalls for the chosen models with different random shuffles of flows considering the correlation with quantity of packet loss and all the other aspects

at all, the inclusion of correlation with quantity of packet loss significantly improves the performance to eventually present a feasible model with a decent mean value of recalls of 0.882 (class 0), 0.771 (class 1) and 0.827 (macro average). Additionally, in the best case at trial 19, the accuracy for each class can reach 90% (0.898 for recall of class 0, 0.907 for recall of class 1, and 0.903 for macro average recall) and more importantly, this is the first time that we are able to identify 90% of "loss" time bins considering the constraints caused by real case scenarios.

- For the *XGBoost* GBDT classifier, the new approach indeed helps the model improve with respect to the original result in Fig.6.10 with a boost of 14.9% (0.689-0.540) for class 1 and the ability of maintaining the same outstanding performance (0.951 & 0.946) for class 0. Although the mean value of recall for class 1, 0.689, is not a splendid result, it is still more than acceptable given the difficulty of the problem, not to mention the excellent performance for "no-loss" time bins and the sensational results at trials 6 and 19 with recalls of 0.928 (class 0), 0.843 (class 1), 0.886 (macro average) and 0.913 (class 0),

0.872 (class 1), 0.892 (macro average) in the most ideal cases.

- By comparison, both models show effective prediction capabilities for "loss" time bins with losses more than 2 from a general perspective, while guaranteeing a stable identification for "no-loss" time bins regardless of the flows. However, there are also two main differences: i.) They exhibit the same behavior in the basic model development, where the balanced RF classifier generates a more balanced result and the *XGBoost* GBDT classifier outperforms towards the majority class, "no-loss" time bins; ii.) The performance of class 1 is more steady for the balanced RF classifier, which is able to predict 70% of losses in most of the trials, while the *XGBoost* GBDT classifier presents relatively dispersed results in which some cases are even lower than 60%, which means that the variations among flows in real-case scenarios affect *XGBoost* GBDT classifier more.

Generally speaking, both models have similar performance according to the almost same mean value of macro average recall (0.827 & 0.820), but based on different requirements of applications or users, we can still differentiate between the models. For example, if we need more stable behavior or require better identification for packet loss, the balanced RF classifier could be a more appropriate choice, and if we do not want to waste resources reacting to wrong classifications, *XGBoost* GBDT classifier is the preferable one since most cases are "no-loss" time bins.

To sum up, by considering the correlation between prediction performance and the quantity of packet losses in a time bin, we make magnificent progress in avoiding an excessive amount of misclassifications for "no-loss" time bins and successfully predict "loss" time bins with a larger number of losses. Although it cannot be ignored that the majority of "loss" time bins are neglected and discarded, we can still consider the models to be effective and feasible, because the decent and boosted performance for both classes when it comes to time bins with more losses which have a severer impact over RTP communications and are more likely to affect QoE, still demonstrates the capabilities of solving the prediction problem to some extent and partially achieving the goal of this paper given the difficulties of the problem. At this stage, we have completed the in-depth analyses for the models themselves by considering two constraints in real-case scenarios and three possible optimizations, and eventually, we propose viable solutions either for the entire dataset or the reduced dataset to predict packet loss in different ways. In the next chapter, we will deal with more data from other sources with the models to investigate the performance, feasibility, and latent issues across multiple RTP applications.

6.3 Analysis of extra data

In addition to the two main applications, Webex and Jitsi, there are also multiple extra *pcap* files for the collection of RTP packets from other applications[17]. In this section, we will follow the same procedures of running *Retina* and data preprocessing to derive the final dataset to perform further analysis and model development.

6.3.1 Data description

The new data is generated by applications that also rely on RTP to deliver real-time content but introduce certain modifications for their own interests and unique functions. Hence, *Retina* cannot parse some *pcap* files from applications like *GoToMeeting* and *Telegram*. Additionally, some applications, such as *Zoom*, generate a "Mark" to specify the start of a segment in a flow, but the corresponding sequence number is not monotonically increased by 1 with respect to the previous packet[18], which means *Retina* will generate errors when it comes to the packet loss calculation between adjacent packets using sequence number, and thus, it is necessary to discard *pcap* files from those applications. Finally, we derive 210 *csv* files from *Retina* outputs.

By performing the exact same data preprocessing for all the outputs as in section 3.2, we gain the final dataset with 384165 time bins in total. Moreover, the bar chart in Fig.6.16 shows the quantity of "loss" and "no loss" time bins, and the comparison with the original dataset. In general, the overall number of time bins is much less than the original dataset. But we have 31431 "loss" time bins occupying 8.18% of the new dataset, which is larger than both the absolute quantity and percentage of "loss" time bins in the old data. Although we have a greater number of "loss" time bins, leading to a smaller bias between classes, the new dataset is still imbalanced.

The reasons for which we did not include these data into previous analysis and

[17]They are: *Facebook, Facetime, Google Meet, GoToMeeting, Houseparty, Instgram, Skype, Telegram, Whatsapp, Zoom, Microsoft teams, Webex, Jitsi.* Although *Webex* and *Jitsi* are the original applications, the new data from them is collected in a different way and not included in the original model development, so that we still consider them as "new" applications.

[18]For example, the sequence number of the previous packet is 100 and the current one with "Mark" is 105, but they are actually consecutive packets that are supposed to have sequential sequence numbers.

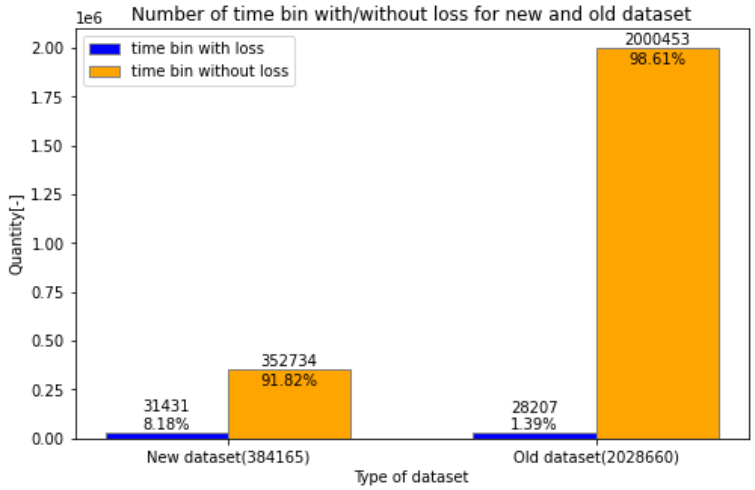


Figure 6.16: Quantity of "loss" and "no-loss" time bins in new and old dataset

model development are: i.) the ground truth about the number of packet losses from log files is not available for these applications in a convenient way; ii.) the quantity of data is not adequate; iii.) variations among multiple applications will introduce more constraints and noise; iv.) the new data is collected through a different network that may generate even more uncertainties. However, since we have already revealed the difficulties of packet loss prediction and proposed several solutions, it is good practice to take into account more data to further investigate the feasibility of usage of the models for other applications.

6.3.2 Experimental result

In order to comprehensively study the behavior of new data and make comparisons, we have proposed and performed five model evaluations based on different cases[19] of training and testing:

- **Case 1**: Using the model of the balanced RF classifier without class weight trained on the original dataset as in section 5.3.3 to test the new dataset;
- **Case 2**: Splitting new data into training and test sets, then retraining and

[19]Notes: i.) In the first three cases, we only use the balanced RF classifier as a reference to gain a preliminary idea of the performance since the final results are in cases 4 & 5; ii.) In cases 2 & 3, we do not shuffle flows because we want to make comparisons with the original model.

testing the balanced RF classifier;

- **Case 3**: Combining the original data and new data all together, then performing training and testing;
- **Case 4**: Similar to case 3, but considering all related aspects as in section 6.2.2 for the entire combined dataset and only using *XGBoost* GBDT classifier;
- **Case 5**: Similar to case 3, but considering all related aspects as in section 6.2.3 by removing time bins with losses less than 3, and using both models.

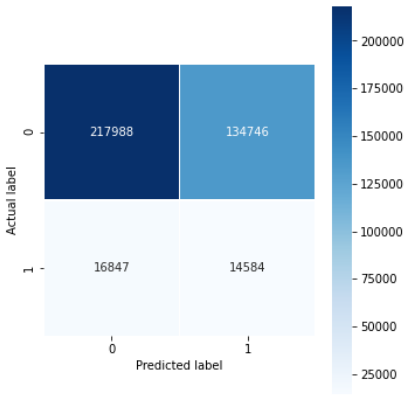
According to the experimental results regarding metrics and confusion matrix for the five cases in Table.6.5 and Fig.6.17:

Case*		1	2	3	4	5.1**	5.2***
Accuracy		0.61	0.84	0.84	0.95	0.94	0.97
Precision	Macro average	0.51	0.65	0.55	0.65	0.55	0.59
	Class 0	0.93	0.99	0.99	0.99	0.99	0.99
	Class 1	0.10	0.32	0.11	0.30	0.10	0.19
Recall	Macro average	0.54	0.85	0.83	0.88	0.94	0.93
	Class 0	0.62	0.83	0.84	0.95	0.94	0.97
	Class 1	0.46	0.87	0.82	0.81	0.93	0.90
F1 score	Macro average	0.45	0.68	0.55	0.70	0.58	0.65
	Class 0	0.74	0.90	0.91	0.97	0.97	0.98
	Class 1	0.16	0.46	0.20	0.43	0.19	0.31

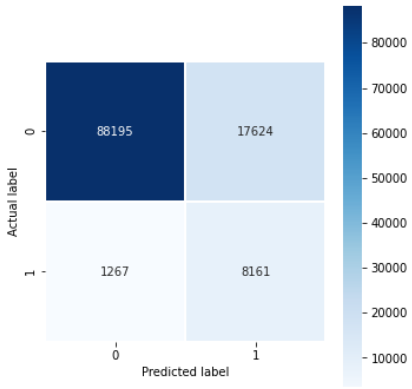
* In case 4 & 5, we present the best result out of 50 trials of random shuffles of flows: trial 17 for case 4 and trial 39 for case 5.
** Balanced RF classifier without class weight.
*** *XGBoost* GBDT classifier.

Table 6.5: Summary of metrics for results of the models on new dataset in different cases

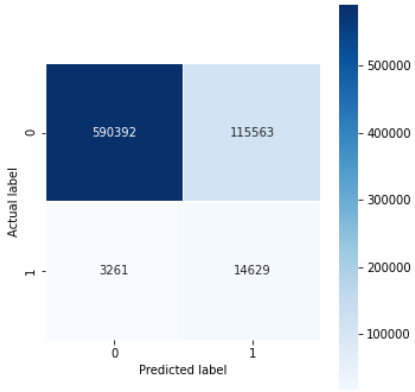
1. In case 1, the balanced RF classifier trained on the original dataset presents a significant performance drop, resulting in recalls of 0.62 (class 0), 0.46 (class 1), and 0.54 (macro average). Besides the variations among applications and other minor differences, the major reason is the unknown and the uncertainty embedded in the new data since they have never been included in the model training, which resembles the real-case scenario of shuffling flows in section



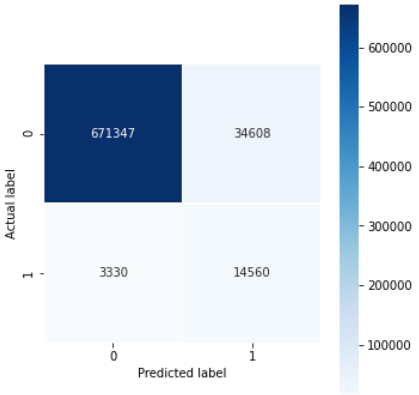
(a) Case 1



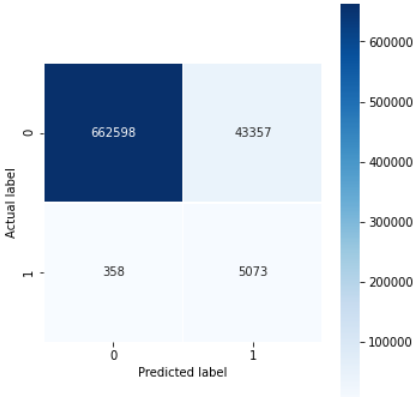
(b) Case 2



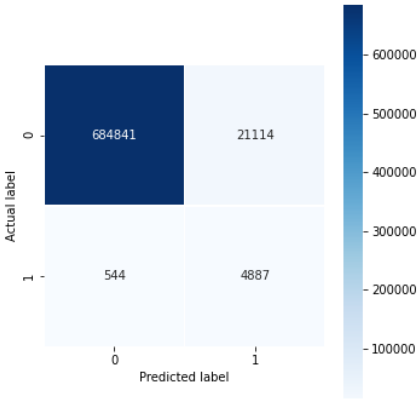
(c) Case 3



(d) Case 4



(e) Case 5.1, balanced RF classifier



(f) Case 5.2, *XGBoost* GBDT classifier

Figure 6.17: Confusion matrices for results of the models on new dataset in different cases

6.1.2. The similar results of the recall with a huge decline prove once again the impact of the real-case scenario.

2. The outstanding performance in case 2, which is 2% greater than the recall of class 0 and 7% greater than the recall of class 1 for the original results in Table.5.8 due to the larger amount of "loss" time bins and the less imbalanced dataset, indicates the feasibility of the model for multiple applications.

3. Similarly, case 3 with a combined dataset also produces a better performance with a higher recall for class 0 but a lower one for class 1 compared to case 2 because the imbalance is relatively severe, forcing the model to outperform towards the majority class.

4. Fig.6.18 shows the results of *XGBoost* GBDT classifier in case 4, with recalls for class 0 hovering around 0.945, which is almost the same as the original result in Fig.6.10, and recalls for class 1 having a substantial improvement of 15.7% (0.697-0.540) in the mean value with respect to the original result, which indicates that more data regarding "loss" time bins could introduce more informative knowledge and further optimize the model. Additionally, the recalls for class 1 are more concentrated with more stable behavior compared to the original result, with relatively dispersed values ranging from 0.38 to 0.75, which means that, with more available data, the model tends to be less affected by stochasticity due to different shuffled flows and converge to its actual performance in real case scenarios. Moreover, in the best case scenario at trial 17, *XGBoost* GBDT classifier is able to predict 81% "loss" time bins, which is a magnificent outcome and even better than all of the basic models in chapter 5 where everything is ideal without constraints and obstacles.

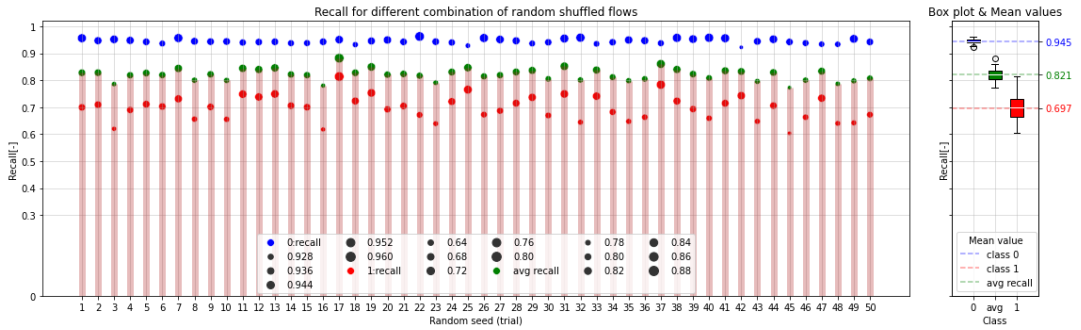
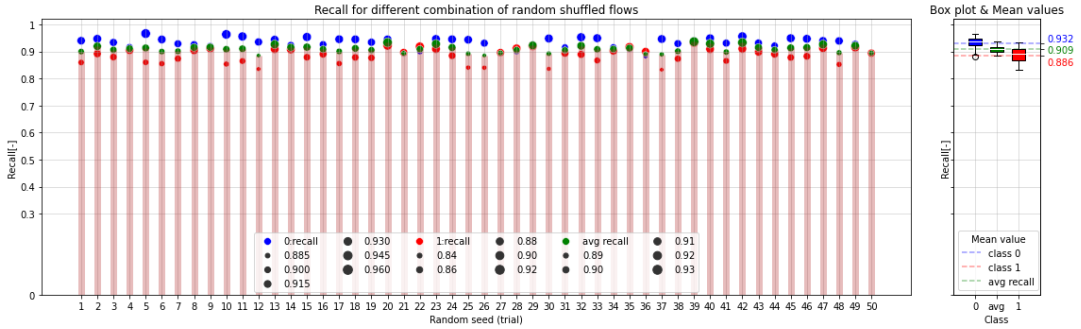


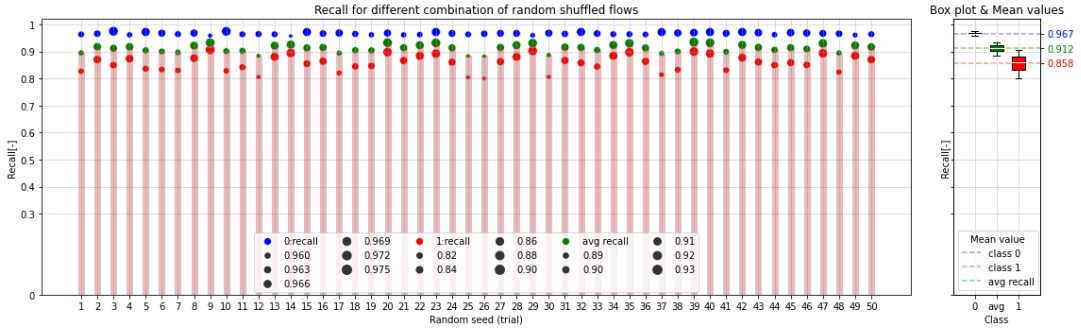
Figure 6.18: Recalls of *XGBoost* GBDT classifier in case 4 considering all related aspects for the entire combined dataset

5. Case 5 presents results for the two selected models considering the correlation with quantity of packet loss and the similar plots as in Fig.6.15 are posted

in Fig.6.19. We can conclude that: i.) Both models experience performance enhancement with increments in the mean values of recalls ranging from 1.6% (0.967-0.951) to 16.9% (0.858-0.689), indicating that the benefits of more data also apply when we only take into account the time bins with more losses; ii.) Both models still follow the same behavior of the balanced RF classifier, producing more balanced results, but the difference is getting smaller due to the improvements for both classes. Thus, the balanced RF classifier might be the proper choice since it outperforms towards class 1 to successfully predict more "loss" time bins, which better complies with the requirements and fulfills the objective of this thesis; iii.) The outcomes are more steady with slight fluctuations, especially for the behaviour of class 1 for *XGBoost* GBDT classifier, with values ranging from 0.56 to 0.85 previously but from 0.80 to 0.90 now; iv.) The best performance is at trial 39 for both models, with unexceptionable recalls of 0.94 & 0.97 for class 0, 0.93 & 0.90 for class 1 and 0.94 & 0.93 for the macro average values, which indicates the extremely powerful prediction capabilities for the models in the most ideal case.



(a) Balanced RF classifier



(b) *XGBoost* GBDT classifier

Figure 6.19: Recalls for the chosen models in case 5 considering all related aspects for the reduced combined dataset without packet losses less than three in a bin

On top of that, all cases 3, 4, and 5, where we combine the new and old datasets from multiple applications to have more data, illustrate that, not only can the models follow a data-driven approach to generate superior performance for both classes with a larger quantity of available data, they can also find commonalities among different samples, which presents the interoperability, versatility, and feasibility of the balanced RF classifier without class weight and *XGBoost* GBDT classifier across all applications.

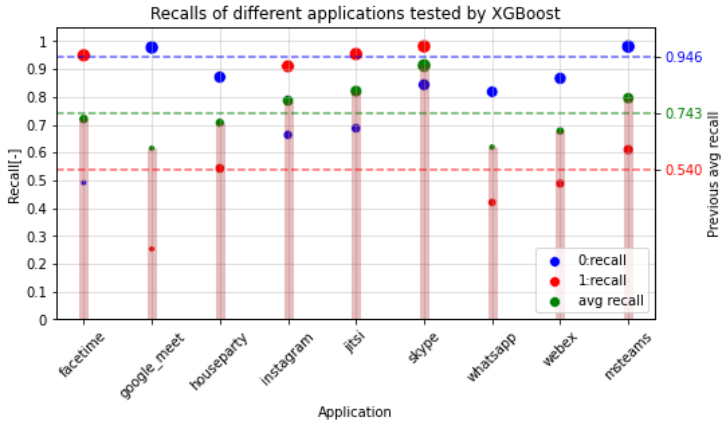
6.3.3 Performances of different applications

Based on the previous analyses, it is feasible to develop machine learning models across applications using mixed data. Unfortunately, available data with labels is a valuable resource and not easily accessible, which is a common problem in the machine learning domain. Therefore, it is good practice to investigate the operational capability of the chosen models developed and trained based on the initial target applications with adequate data on each new application so that we are able to find the possibilities of using the predefined model to predict packet loss for other applications that have never been analyzed and eventually save time for model retraining and data collection as well as labeling. Specifically, we will compare the general performances in terms of the mean value of recall of original models with model evaluation on each application in three cases[20]:

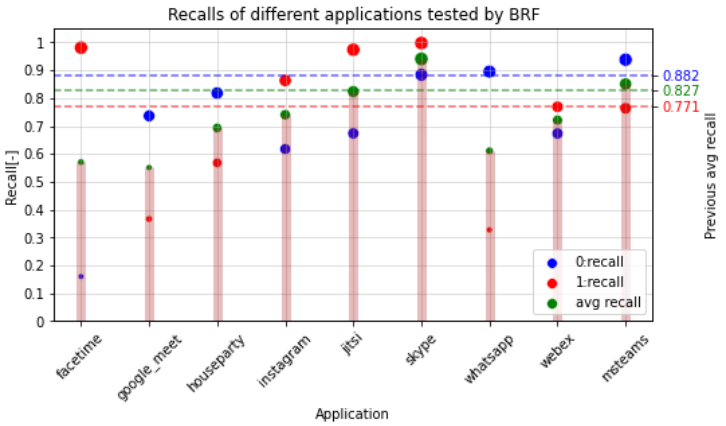
- **Case 1:** Considering all related aspects and using *XGBoost* GBDT classifier trained on the original dataset with the best performance at trial 34 as in section 6.2.2 to test the entire dataset for the new data from each application.
- **Case 2:** Using the balanced RF classifier trained on the reduced original dataset with the best performance at trial 19 as in section 6.2.3 to test all of the new time bins, excluding those with losses equal to 1 or 2 from each application.
- **Case 3:** Similar to case 2, but using *XGBoost* GBDT classifier.

According to the results of each application for the three cases in Fig.6.20:

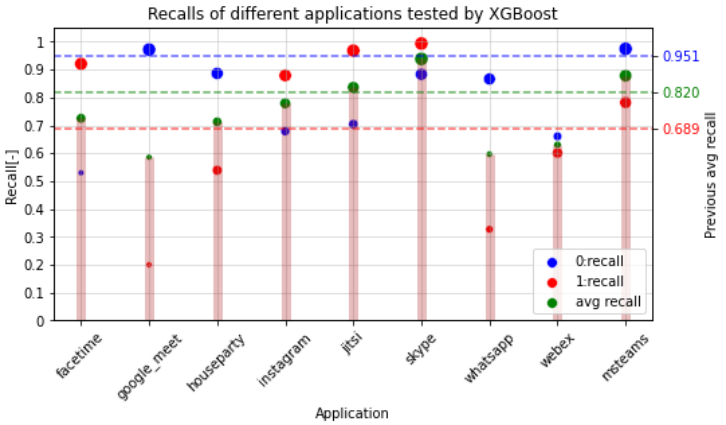
[20]The reason why we use the model trained on parts of the original dataset instead of the entire one is that we want to simulate the real-case scenario and avoid overfitting on the new dataset.



(a) Case 1



(b) Case 2



(c) Case 3

Figure 6.20: Recalls of the result of each application in the three cases with different classifiers and datasets

- For all cases, the models do not work for *Facetime*, *Google Meet* and *Whatsapp*, except that *XGBoost* GBDT classifier can successfully identify "no-loss" time bins in cases 1 & 3 but barely predict "loss" time bins for *Google Meet*.
- For *Houseparty* and *Webex*, *XGBoost* GBDT classifier in case 1 has similar performance with respect to original results, but the models in cases 2 & 3 experience significant performance decline with reduced dataset, and this is different from our previous solution in which the consideration of quantity of packet loss helps optimize the models. On top of that, the models perform poorly for *Webex* in all cases, even if the application is the initial one included in the original dataset, which indicates that data from the same application also has huge differences internally.
- The models perform acceptably for *Instgram* and *Jitsi* with a higher macro average recall in case 1 and a similar value in case 3. However, the performance for each class is the opposite, with higher recall for class 1 compared to original results, and this phenomenon also applies to *Facetime* and *Skype*, which means that the models can easily predict "loss" time bins for these applications but have trouble identifying "no-loss" time bins. Additionally, similar to *Webex*, *Jitsi* is also one of the original applications but generates relatively poor performance, which again illustrates the variations among data patterns in the same application and proves the difficulties of the prediction problem.
- The models work very well for *Skype* and *Microsoft Teams* with higher macro average recalls in all three cases, but it cannot be ignored that the performance for *Skype* is the opposite, and thus the models will produce a large amount of misclassifications for "no-loss" time bins, which might be a problem for network management even if the prediction for packet loss is fabulous. More importantly, when it comes to *Microsoft Teams*, the performances are perfect with even higher recalls for both classes in all cases, which indicates that both *XGBoost* GBDT classifier and the balanced RF classifier trained on the original applications present outstanding feasibility as well as adaptation and can be applied to *Microsoft Teams*.

In a word, following the data-driven principle, the models developed based on certain data are not suitable for others given the diversity of RTP flows in our specific problem. Although there's one applicable solution of *Microsoft Teams*, machine learning model development still requires information and knowledge from as many sources as possible in most cases. However, it is also important to note that the data collection for each application is not standardized and the network conditions are not constant, so if we want to dig deeper into various RTC applications in terms of packet loss phenomena, we need a more comprehensive

analysis and a well-designed procedure in order to obtain a more precise and solid solution.

Finally, we officially finished all the analyses and model development and are ready to draw conclusions in the next chapter.

Chapter 7

Conclusion

7.1 Summary

To begin with, the thesis objective is to forecast packet loss using past statistics of received packets in RTP traces in order to reveal the possibilities of implementing a predictive mechanism to improve QoE. In general, the thesis contains the manipulation and analyses of the dataset related to RTP packets, the machine learning approach development regarding classification and anomaly detection, the discussion about constraints in real case scenarios, the proposed solutions for model optimization, and the analyses for extra data from multiple other applications. Eventually, the thesis comprehensively presents the properties of the problem and proves the feasibility of utilizing machine learning models for predicting packet loss.

In particular, first we elaborate on the open source command-line tool *Retian* to calculate the exact quantity of packet loss in a time bin and derive the well-structured dataset. Meanwhile, besides the various analyses of time bins with or without loss, we focus on the characterization of packet loss to identify the patterns in the time window of study, to distinguish "loss" and "no-loss" time bins, and finally, determine 11 specific features (statistics) for machine learning models.

Secondly, by following the basic procedure of model building, tuning, and evaluation, we perform the model development for six algorithms: Decision Tree, Neural Network, Random Forest, Gradient Boosting Tree, Isolation Forest, and Autoencoder. On top of that, in order to deal with the issues caused by the imbalanced dataset, we thoroughly investigate the effects of sampling techniques,

evaluation metrics, various versions of the algorithm and specific parameters. As a result, classification methods outperform anomaly detection from a general point of view, and among all the methods, balanced RF classifier without class weight and *XGBoost* GBDT classifier are the best ones due to their balanced and outstanding performance in terms of recall, with values of 0.81 & 0.85 for class 0, 0.80 & 0.74 for class 1, and 0.80 for the macro average.

Furthermore, we inspect the impact of real-case scenarios of predicting a farther future as well as shuffling RTP flows and end up with significant performance drops. Afterwards, in order to solve the problem and optimize the best models taking into account constraints in reality, we come up with three solutions: i.) using Recursive Feature Elimination to select more informative features instead of those based on packet loss characterization; ii.) regarding packet loss condition before any target time bins as features; iii.) only considering time bins with losses more than 2 as "loss" time bins since more losses have a larger impact on RTP communications. Consequently, we successfully improve the performance and optimize the models to have an acceptable result for the entire dataset and an outstanding one for the reduced dataset.

Additionally, we analyze the new data from multiple other applications with more losses to investigate the performance of the models over the combined dataset. Fortunately, we achieve a more stable and magnificent performance in all cases. Moreover, we also check the feasibility of the models trained on the original dataset over each application and end up with only one applicable application, *Microsoft Teams*. Finally, a summary of recalls for all of the final results is presented in Table.7.1.

To sum up, based on all of the analyses and results, we can conclude that:

1. For RTP-based RTC applications, packet loss is a rare event that occurs irregularly. At the same time, RTP traces have numerous complicated properties, including dynamic evolution, variations among RTP flows, various possible causal phenomena for losses, different network conditions, etc. As a result, packet loss prediction is a hard task that requires elaborate analysis, and in our opinion, the works in this thesis could be a good starting point.

2. Besides the difficulties of the problem, the major issue during model development is the imbalanced dataset, which forces the model to outperform towards the majority class. Consequently, it is impossible to simultaneously improve the performances for both classes without paying a trade-off, and the only solution is to sacrifice one class to achieve either an excellent identification for "no-loss" time bins with a mediocre performance for "loss" time bins or a

Case*	Model	Mean value			Best value		
		class 0	**class 1**	**macro average**	**class 0**	**class 1**	**macro average**
0	Balanced RF	0.655	0.416	0.535	0.616	0.591	0.604
0	*XGBoost* GBDT	0.580	0.580	0.580	0.625	0.670	0.647
1	*XGBoost* GBDT	0.946	0.540	0.743	0.944	0.744	0.838
2	Balanced RF	0.882	0.771	0.827	0.898	0.907	0.903
2	*XGBoost* GBDT	0.951	0.689	0.820	0.913	0.872	0.892
3	*XGBoost* GBDT	0.945	0.697	0.821	0.951	0.814	0.882
4	Balanced RF	0.933	0.886	0.909	0.939	0.934	0.936
4	*XGBoost* GBDT	0.967	0.858	0.912	0.970	0.900	0.935

* Case description:

- 0: Entire original dataset with shuffling flows as in section 6.1.2;
- 1: Entire original dataset considering all related aspects, including packet loss as features as in section 6.2.2;
- 2: Reduced original dataset considering all related aspects for "loss" time bins with losses more than 2 as in section 6.2.3;
- 3: Entire combined dataset with new data considering all related aspects, including packet loss as features as in case 4 of section 6.3.2;
- 4: Reduced combined dataset with new data considering all related aspects for "loss" time bins with losses more than 2, as in case 5 of section 6.3.2.

Table 7.1: Summary of recalls for all the final results

relatively balanced performance with a better capacity for predicting outliers but plenty of misclassifications for normal cases.

3. The real-case scenarios in which we need to predict a farther future to effectively respond to potential losses and deal with the unknowns from new RTP flows that have never been included in the training phase significantly affect the model performance in a negative way, which emphasizes again the intrinsic difficulties of the real-time prediction problem for RTP traces.

4. The three optimization methods are able to conquer the obstacles and effectively improve the model performance from two perspectives: i.) for the prediction over the entire dataset, i.e., all the time bins, we have one machine learning approach that can successfully identify time bins without losses but performs not very effectively with acceptable performance for "loss" time bins; ii.) for parts of the dataset, i.e., time bins with losses equalling to 1 or 2 are discarded, we have two methods with better performances but different optimized directions: one with balanced and intermediate outcomes and the other one with outstanding performance for "no-loss" time bins but relatively poor results for "loss" time bins, which allow us to make decisions based on the requirements of applications or QoE.

5. Given the fact that the majority class of "no-loss" time bins occupies more than 95% of the entire dataset in any case, even a tiny misclassification ratio could lead to a huge amount of wrong predictions, which may consume a

lot of network resources to perform unnecessary reactions. Hence, a good solution could be the one with a powerful prediction capacity for normal cases. Although the corresponding relatively poor performance for predicting packet loss somehow slightly deviates from the thesis objective, a prediction accuracy ranging around 60% can still be considered a reliable and acceptable one given the difficulties of the problem, let alone the even better performance with more available data and outstanding results in some ideal cases.

6. The additional data from extra applications indeed help improve the performance since we have more "loss" time bins that contribute more information and knowledge regarding packet loss to compensate for the defects due to class imbalance. Moreover, the feasibility of model development across multiple applications is acknowledged, but the possibility of using one pre-trained model to test data from other sources is partially rejected. All of the aforementioned aspects prove that we are following a data-driven approach.

Finally, by considering possible related aspects and edge conditions, we finish the data analysis, model development, and optimization to derive decent outcomes in various ways, which successfully prove the feasibility of predicting packet loss for RTC applications and substantially illustrate the potential of implementing the algorithm into practical scenarios.

7.2 Limitation and future work

In our case, the data is collected through the connection of WiFi or Ethernet, in which the major cause of losses is network congestion. However, communications through cellular networks, like 4G and 5G, also play a crucial role in daily life but are not considered in this thesis. And cellular networks also include various factors affecting packet loss, which deserve more research. Therefore, a possible future work could be related to this, focusing on mobile applications.

Moreover, the thesis only studies the packet loss phenomenon in 500 ms-long time bins with discrete timestamps. Although we achieve decent results eventually, it is still good practice to jump out of the box to eliminate the restriction of time bins. Hence, another future work could be modifying the duration of target time bins or even concentrating on patterns of received packets themselves with continuous time series instead of relying on time aggregations.

In a word, as mentioned before, this thesis could be a good start for business applications by presenting the feasibility of packet loss prediction for improving

QoE. On top of that, the actual implementation, taking into account all commercial and technical factors, absolutely needs more analysis and development, which requires a joint effort by all.

References

[1] Telegeography. *Global Internet Map 2021*. 2020. URL: https://global-internet-map-2021.telegeography.com (cit. on p. 1).

[2] J. Clement. *Global Mobile Data Traffic from 2017 to 2022*. 2020. URL: https://www.statista.com/statistics/271405/global-mobile-data-traffic-forecast/#professional (cit. on p. 1).

[3] A. Feldmann et al. «The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic». In: 20 (2020), pp. 1–18 (cit. on p. 2).

[4] «Video Conferencing Market Size, Share & COVID-19 Impact Analysis, By Component, By Conference type, By Deployment, By Enterprises Size, By Application, and Regional Forecast, 2022-2029». In: (Apr. 2022) (cit. on p. 2).

[5] Theron Mohamed. *Zoom expects to boost revenue by 200% and profits by 300% this year. These 5 charts show the video-conferencing upstart's explosive growth*. 2020. URL: https://markets.businessinsider.com/news/stocks/zoom-expects-200-revenue-300-profit-growth-5-charts-2020-6-1029277092 (cit. on p. 2).

[6] *Lifesize 2019 Impact of Video Conferencing Report: More Collaborative Workplace Cultures Have Led to Spike in Video Communication*. 2019. URL: https://www.lifesize.com/en/press/impact-video-conferencing-report/ (cit. on p. 2).

[7] About Diane. *5 Ways Video Conferencing Protects the Environment*. URL: http://biggreenpurse.com/video-conferencing-protects-environment/#sthash.LIEMrVp5.kteiWgjy.dpuf (cit. on p. 2).

[8] «ITU-T Recommendation P.10: Vocabulary for performance and quality of service, Amendment 5». In: (2016), pp. 25–26 (cit. on p. 2).

[9] Ulrich Reiter, Kjell Brunnström, Katrien De Moorand Mohamed-Chaker Larabi, Manuela Pereira, Antonio Pinheiro, Junyong You, and Andrej Zgank. «Factors Influencing Quality of Experience». In: (2014), pp. 55–72 (cit. on p. 2).

[10] Gianluca Perna, Dena Markudova, Martino Trevisan, Paolo Garza, Michela Meo, and Maurizio M. Munafò. «Retina: An open-source tool for flexible analysis of RTC traffic». In: (Jan. 2022) (cit. on pp. 3, 12).

[11] Lopamudra Roychoudhuri and Ehab S. Al-Shaer. «Real-Time Analysis of Delay Variation for Packet Loss Prediction». In: (2004) (cit. on pp. 4, 11).

[12] Fernando Silveira Filho and Edmundo de Souza e Silva. «A method for predicting packet losses with applications to continuous media streaming». In: (2006) (cit. on p. 4).

[13] Hooman Homayounfard. «Packet-Loss Prediction Model Based on Historical Symbolic Time-Series Forecasting». In: (Oct. 2013) (cit. on p. 4).

[14] Manish P Ganvir and Dr. S.S.Salankar. «Time Series Forecasting of Packet Loss Rate Using Artificial Neural Network Based on Particle Swarm Optimization». In: (Mar. 2015) (cit. on p. 4).

[15] João Victor Costa CarmonaID, Edemir Marcus Carvalho de Matos, Bruno Souza Lyra Castro, Fabrı´cio Jose´ Brito Barros, Mie´rcio Cardoso de Alca^ntara Neto, and Evaldo Goncalves Pelaes. «Video loss prediction model in wireless networks». In: (2019) (cit. on p. 5).

[16] Dr. Kalpana Saha and Tune Ghosh. «Study of Packet Loss Prediction using Machine Learning». In: (2020) (cit. on p. 5).

[17] S.B. Moon. «Measurement And Analysis Of End-To-End Delay And Loss In The Internet». In: (2000) (cit. on p. 5).

[18] V. Paxson. «Measurements and Analysis of End-to-End Internet Dynamics». In: (1997) (cit. on p. 5).

[19] O. Hermanns and M. Schuba. «Performance investigations of the IP multicast architecture». In: (1996) (cit. on p. 5).

[20] Audio-Video Transport Working Group, H. Schulzrinne, GMD Fokus, S. Casner, Inc. Precept Software, Xerox Palo Alto Research Center, R. Frederick, V. Jacobson, and Lawrence Berkeley National Laboratory. «RTP: A Transport Protocol for Real-Time Applications». In: (1996) (cit. on pp. 7, 12, 14).

[21] Gianluca Pernay, Dena Markudovay, Martino Trevisany, Paolo Garzay, Michela Meoy, Maurizio M. Munafo, and Giovanna Carofiglio. «Real-Time Classification of Real-Time Communications». In: (2021) (cit. on pp. 8, 9).

[22] C. Holmberg, S. Hakansson, and G. Eriksson. «Web Real-Time Communication Use Cases and Requirements». In: (2015) (cit. on p. 8).

[23] Dhwani R. Bhadra, Charmi A. Joshi, Priya R. Soni, Nikita P. Vyas, and Rutvij H. Jhaveri. «Packet Loss Probability in Wireless Networks: A Survey». In: (Apr. 2015) (cit. on p. 10).

[24] Dziugas Baltrunas, Ahmed Elmokashfi, Amund Kvalbein, and Ozgu Alay. «Investigating Packet Loss in Mobile Broadband Networks under Mobility». In: (June 2016) (cit. on p. 10).

[25] Pablo Perez, Jesús Macías, Jaime J. Ruiz, and Narciso García. «Effect of Packet Loss in Video Quality of Experience». In: (June 2011) (cit. on p. 10).

[26] M. Zennaro, E. Canessa, K. R. Sreenivasan, A. A. Rehmatullah, and R. L. Cottrell. «Scientific Measure of Africa's Connectivity». In: (2006) (cit. on p. 10).

[27] Jerome H. Friedman. «Greedy Function Approximation: A Gradient Boosting Machine». In: (2001) (cit. on p. 18).

[28] N. V. Chawla, K. W. Bowyer, L. O.Hall, and W. P. Kegelmeyer. «SMOTE: synthetic minority over-sampling technique». In: (2002) (cit. on p. 65).

[29] Chao Chen, Andy Liaw, and Leo Breiman. «Using Random Forest to Learn Imbalanced Data». In: (2004) (cit. on pp. 79, 80).

[30] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. «Gene selection for cancer classification using support vector machines». In: (2002) (cit. on p. 103).