



Escola d'Enginyeria de Telecomunicació i  
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# FINAL DEGREE PROJECT

**TITLE:** Scheduling in TSN Networks using machine learning

**DEGREE:** Bachelor's degree in Network Engineering

**AUTHOR:** Arnau Martínez Lopera

**DIRECTOR:** Anna Agustí

**Title:** Scheduling in TSN networks using machine learning

**Author:** Arnau Martínez Lopera

**Director:** Anna Agustí

## Overview

The massive adoption of Ethernet technology in multiple sectors, produces the need to provide deterministic solutions to ensure a Quality of Service (QoS) that meets the requirements of time-triggered flows. For this, the Time-Sensitive Networking (TSN) Task Group (TG) of the IEEE 802.1 developed a set of standards that define mechanisms for time-sensitive transmissions of data over Ethernet networks.

This project focuses on studying the feasibility of scheduling three classes of time-triggered flows with different time constraints over a simple network topology, which is made from two TSN (Time-Sensitive Networking) nodes connected through a link. Scheduling multiple time-triggered flows is a complex problem because the scheduling, if exists, must meet the time constraints of all these flows.

To address this challenge, we explore the potential of using supervised machine learning classification models to accurately predict the feasibility of scheduling a given set of time-triggered flows, meeting their time-constraints, in a Time-Sensitive Network (TSN).

Supervised models require a training dataset that contains a data matrix and a class label vector. To obtain the class label vector of each observation, we use an adaptation of the implementation developed in [27] of the Integer Linear Programming (ILP) model introduced in [33].

Two different models are considered: K-Nearest Neighbours (K-NN) and Support Vector Machine (SVM). These algorithms are tested and built from the application of the Leave One Out Cross-Validation (LOOCV) technique with the generated datasets, and the results obtained are compared and discussed.

Finally, a hybrid verification strategy is proposed to train and test machine learning models, drastically reducing the resources and computation time originally required to compute the class label of each observation of the dataset.

**Título:** Scheduling in TSN networks using machine learning

**Autor:** Arnau Martínez Lopera

**Directora:** Anna Agustí

## Resumen

La adopción masiva de la tecnología Ethernet en múltiples sectores, produce la necesidad de brindar soluciones deterministas para asegurar una Calidad de Servicio (QoS) que cumpla con los requerimientos de los flujos sensibles al time (en adelante TT por las siglas en inglés Time-Triggered). Para ello, el grupo de trabajo Time-Sensitive Networking (TSN) del IEEE 802.1 desarrolló un conjunto de estándares que definen mecanismos para transmitir flujos de datos con requisitos temporales estrictos a través de redes Ethernet.

Este proyecto se enfoca en estudiar la viabilidad de programar varios flujos de tres clases TT diferentes sobre una topología de red simple, compuesta de dos nodos TSN (Time-Sensitive Networking) conectados a través de un enlace. Programar la transmisión de múltiples flujos TT es un problema complejo ya que la solución, si existe, debe garantizar que se cumplen todos los requisitos temporales de todos los flujos a transmitir.

Para abordar este desafío, en este trabajo exploramos el potencial del uso de modelos de clasificación de aprendizaje supervisado para predecir con precisión, la viabilidad de programar un conjunto dado de flujos TT, cumpliendo con sus restricciones de tiempo, en una red (TSN).

Los modelos supervisados requieren un conjunto de datos de entrenamiento formados por una matriz de datos y un vector de etiquetas de clase. Para generar las etiquetas de clase, en este trabajo utilizamos una adaptación de la implementación desarrollada en [27] del modelo ILP definido en [33].

En este proyecto se consideran dos modelos diferentes: K-Nearest Neighbors (K-NN) y Support Vector Machine (SVM). Estos algoritmos se prueban y se construyen a partir de la aplicación de la técnica de validación cruzada llamada Leave One Out Cross-Validation (LOOCV) con los conjuntos de datos generados, para posteriormente comparar y discutir los resultados.

Finalmente, se propone una estrategia de verificación híbrida con el objetivo de entrenar y probar modelos de aprendizaje automático, reduciendo drásticamente los recursos y el tiempo de cómputo requerido originalmente para generar las etiquetas de clase en la base de datos.

# Table of Contents

Introduction .....	1
CHAPTER 1. Introducing Time Sensitive Networks (TSN).....	5
1.1. IEEE802.1Qav (Credit Based Shaper).....	6
1.2. IEEE802.1Qbv (Time Aware Shaper) .....	7
1.3. Complexity Problems .....	9
1.4. Applying TSN .....	9
CHAPTER 2. Dataset Generation .....	11
2.1. Integer Linear Programming (ILP).....	11
2.1.1 Preliminary definitions .....	12
2.1.2. ILP Model .....	13
2.2. Implementation.....	14
2.3. Enhancing the ILP implementation.....	16
2.4. Executing the ILP .....	16
2.4.1. Inputs .....	16
2.4.2. Outputs.....	18
2.5. Implemented Scenarios .....	22
2.5.1. Scenario 1 .....	22
2.5.2. Scenario 2 .....	24
2.5.3. Scenario 3 .....	25
CHAPTER 3. Feasibility Prediction Algorithms .....	27
3.1. Binary Classification Algorithms .....	27
3.2. K-Nearest Neighbours (K-NN).....	27
3.3. Support Vector Machine (SVM).....	28
CHAPTER 4. Model Testing.....	29
4.1. Testing K-NN.....	30
4.1.1. Scenario 1 .....	30
4.1.2. Scenario 2 .....	32
4.1.3. Scenario 3 .....	33
4.2. Testing SVM.....	34
4.2.1. Scenario 1 .....	34
4.2.2. Scenario 2 .....	36
4.2.3. Scenario 3 .....	36
4.3. Conclusion .....	37

CHAPTER 5. Hybrid Verification .....	39
5.1. Strategy Application .....	39
5.2. Results .....	40
CHAPTER 6. Conclusions and Next Steps .....	43
6.1. Conclusions.....	43
6.2. Next Steps.....	44
References.....	45
Annex.....	<b>Error! Bookmark not defined.</b>
Development Environment .....	<b>Error! Bookmark not defined.</b>

# List of Figures

<b>Fig. 1.1</b> VLAN tag.....	6
<b>Fig. 1.2</b> Two time slices example IEEE 802.1Qbv schedule. ....	7
<b>Fig. 1.3</b> Best effort frame interfering the next transmission.....	8
<b>Fig. 1.4</b> Guard bands usage. ....	8
<b>Fig. 1.5</b> Network topology .....	10
<b>Fig 2.1</b> Convex polytope .....	11
<b>Fig 2.2</b> ILP Core Implementation [27] .....	14
<b>Fig 2.3</b> StreamsConfig.json Configuration file example. Size (Bytes), Period (ms), Deadline (ms), Bandwidth (bps). ....	17
<b>Fig 2.4</b> ILP execution configuration “distributions”: distribution file path, “timeout”: (s) maximum iteration time .....	18
<b>Fig 2.5</b> Combinations to compute by the ILP algorithm.....	18
<b>Fig 2.6</b> Output computation example .....	19
<b>Fig 2.7</b> Gantt chart example.....	20
<b>Fig 2.8</b> Scatter plot example .....	22
<b>Fig 2.9</b> 3D scatter plot of scenario 1 .....	23
<b>Fig 2.10</b> 3D scatter plot of scenario 2 .....	24
<b>Fig 2.11</b> 3D scatter plot of scenario 3 .....	26
<b>Fig 4.1.</b> 3D scatter plot of scenario 1 with wrong classified instances (K-NN) .....	31
<b>Fig 4.2</b> 3D scatter plot of scenario 2 with wrong classified instances (K-NN) .....	32
<b>Fig 4.3</b> 3D scatter plot of scenario 3 with a wrong classified instance (K-NN) .....	33
<b>Fig 4.4</b> 3D scatter plot of scenario 1 with wrong classified instances (SVM) .....	35
<b>Fig 4.5</b> 3D scatter plot of scenario 2 with wrong classified instances (SVM) .....	36
<b>Fig 4.6</b> 3D scatter plot of scenario 3 with wrong classified instances (SVM) .....	37
<b>Fig 5.1</b> 3D scatter plot of scenario 3 with support vectors and a wrong classified combination.....	40
<b>Fig 5.2</b> 3D scatter plot of missing combinations. Includes wrong classified combinations and the additional combinations selected to retrain the model.....	41
<b>Fig 5.3</b> 3D scatter plot of missing and wrong classified combinations .....	42

## List of Tables

<b>Table 2.1</b> Traffic characteristics table of scenario 1 .....	23
<b>Table 2.2</b> Traffic characteristics table of scenario 2 .....	24
<b>Table 2.3</b> Traffic characteristics table of scenario 3 .....	25
<b>Table 4.1</b> Wrong classified instances of K-NN in Scenario 1 .....	31
<b>Table 4.2</b> Wrong classified instances of K-NN with LOOCV in Scenario 2 .....	33
<b>Table 4.3</b> Wrong classified instances of K-NN with LOOCV in Scenario 3 .....	34
<b>Table 4.4</b> Wrong classified instances of SVM in Scenario 1 .....	35
<b>Table 5.1</b> Additional combinations .....	41
<b>Table 5.2</b> Wrong predicted combinations after retraining.....	42

## Acronyms Table

Acronym	Description
TSN	Time Sensitive Networks
SDN	Software Defined Network
QoS	Quality of Service
TG	Task Group
TAS	Time Aware Shaper
ILP	Integer Linear Programming
SVM	Support Vector Machines
K-NN	K-Nearest Neighbours
ML	Machine Learning
AVB	Audio Video Bridging
LOOCV	Leave One Out Cross Validation
PCA	Principal Component Analysis
IEEE	Institute of Electrical and Electronic Engineers
OSI	Open Systems Interconnection
LAN	Local Area Network
VLAN	Virtual Local Area Network
PCP	Priority Code Point
GCL	Gate Control List
TDMA	Time-division Multiple Access
CBS	Credit Based Shaper
RAM	Random Access Memory
OOM Killer	Out Of Memory Killer
CPU	Central Processing Unit
LDA	Linear Discriminant Analysis
RBF	Radial Basis Function



## Introduction

Due to the inherent characteristics of Ethernet, this technology has been adopted in many sectors throughout these five decades of development. But even with the improvements over these years, networks based on this technology lacks on determinism. This means that lacks Quality of Service (QoS), being unable to guarantee maximum latencies, reduced delay variations and other characteristics that belong to deterministic networks.

To provide this QoS requirements, the Audio Video Bridging (AVB) Task Group, renamed later Time-Sensitive Networking (TSN) Task Group by the IEEE, have developed a set of standards that define mechanisms for time-sensitive transmissions of data over Ethernet networks. Although TSN provides a great variety of mechanisms to ensure QoS, it is a huge challenge to set up properly the parameters of these mechanisms to achieve the desired QoS requirements.

Given a set of time-critical flows (that we will refer as Time-Triggered flows or TT flows for short) in a TSN network using the IEEE802.1Qbv (Time Aware Shaper) standard, this project focus on the problem of computing a scheduling for this set of TT flows in which every flow must meet its QoS requirements.

There exist many algorithms to solve this problem, but they are usually complex and time consuming. Alternatively, we propose to speed up this evaluation process by implementing supervised machine learning algorithms with the aim of determining if a given set of time-critical flows, with a given set of characteristics, can be transmitted over a certain link, satisfying all the QoS requirements.

Since we propose supervised machine learning algorithms, we previously need to generate a labelled dataset to train these supervised models. That is why as a previous step, we need to lean on another algorithm to serve as a source of data. To do this, in this work, we adapt the implementation developed in [27] of the Integer Linear Programming (ILP) model proposed in [33].

It is important to note that the ILP algorithm is NP-complete [28], which means that the algorithm may not always find a solution within a reasonable timeframe. To address this issue, a timeout is set to the ILP implementation. Which means that, if the ILP does not find a solution before the timeout expires, we will not have the label class of the corresponding experiment. Hence, we must assume that the provided datasets may be incomplete.

In case of obtaining a result from the ILP implementation, a Boolean value, that we called feasibility, is used to determine if a specific combination of flows can be scheduled satisfying the time requirements of all flows.

In this work, the network topology considered is a single link and we try to schedule a combination of  $N$  Time-Triggered flows that belong to three different classes. All flows of

the same class share the same traffic characteristics in terms of bytes to be transmitted, period and deadline. In each observation we consider a different number of flows of each class to be transmitted over the link and we apply the ILP implementation to find a schedule. If the combination of flows can be scheduled, the combination is proven to be feasible. Otherwise, we consider the scheduling of the combination of flows as unfeasible. If the timeout expires, we do not have a class value for the experiment.

After building the datasets, we normalize the features to ensure equal relevance and train two different supervised Machine Learning classifications algorithms so the models can predict if a given combination of TT flows has a feasible scheduling or not. The two supervised models used in this work are K-Nearest Neighbours (K-NN) and Support Vectors Machine (SVM), and we use the Leave One Out Cross-Validation (LOOCV) technique, to evaluate and compare the performance of the two models.

Finally, a hybrid verification strategy is proposed to minimize the execution time of the ILP implementation based on the use of a machine learning algorithm. The resulting models obtained from this strategy should be capable of generating highly reliable results in fractions of a second avoiding multiple hours of scheduling analysis.

This project is structured as follows:

In the first chapter, we introduce TSN, highlighting its key properties and emphasizing its significance across various industries. Next, we briefly describe two widely used TSN scheduling mechanisms and define the main characteristics of time-triggered flows. Then, we present the network topology and the assumptions considered in this work.

In the second chapter, our focus lies in obtaining labelled datasets to train the machine learning algorithms. To accomplish this, we adapted the ILP algorithm implemented in [27]. This implementation allows us to generate the necessary input data along with their corresponding class labels, enabling us to train the supervised machine learning models. For that purpose, we define three different test scenarios. Each scenario is defined by the characteristics of the three different classes of TT streams considered and the total number of flows that must be scheduled over the link. Then, we execute the ILP for each scenario and every possible combination of flows to later present the results obtained.

The third chapter briefly introduce the two supervised Machine Learning models that we use in this work, that is K-Nearest Neighbours (K-NN) and Support Vectors Machine (SVM).

Subsequently, the fourth chapter implements the previously presented supervised algorithms adjusting their hyperparameters using the Leave One Out Cross Validation (LOOCV) method to build the models. From these models, we later expose their performance and make a comparison of their predictive capabilities.

The fifth chapter proposes an alternative strategy to train machine learning algorithms, minimizing the use of the ILP implementation, and taking advantage of the implemented

machine learning algorithms. This approach offers significant time savings, as the machine learning algorithms can produce reliable results within milliseconds, whereas a computational analysis or ILP algorithm could potentially take several hours to complete.

Finally, the conclusions section of this project serves as a comprehensive review of the conducted work, where we carefully assess the results and outcomes achieved through this project.

To continue with the developments made in this project and allow the reproducibility of the experiments carried out, the annex includes a link to access the developed code.



## CHAPTER 1. Introducing Time Sensitive Networks (TSN)

Due to its speed, low cost, and enormous versatility, Ethernet is the main communication solution for most industries, with over 50 years of history.

Even with the improvements it has received over the years, especially in terms of speed, the development of Ethernet is mostly governed by the Best Effort principle, which lacks determinism.

Deterministic characteristics such as, ensuring maximum latency with reduced delay variations, along with its reliability requirements, are necessary properties for some networks. I.e., for industrial, automotive, telecommunications or aerospace networks.

To provide this Quality of Service (QoS) requirements, the Time-Sensitive Networking (TSN) Task Group (TG) [1] of the IEEE 802.1 developed a set of standards that define mechanisms for time-sensitive transmissions of data over Ethernet networks.

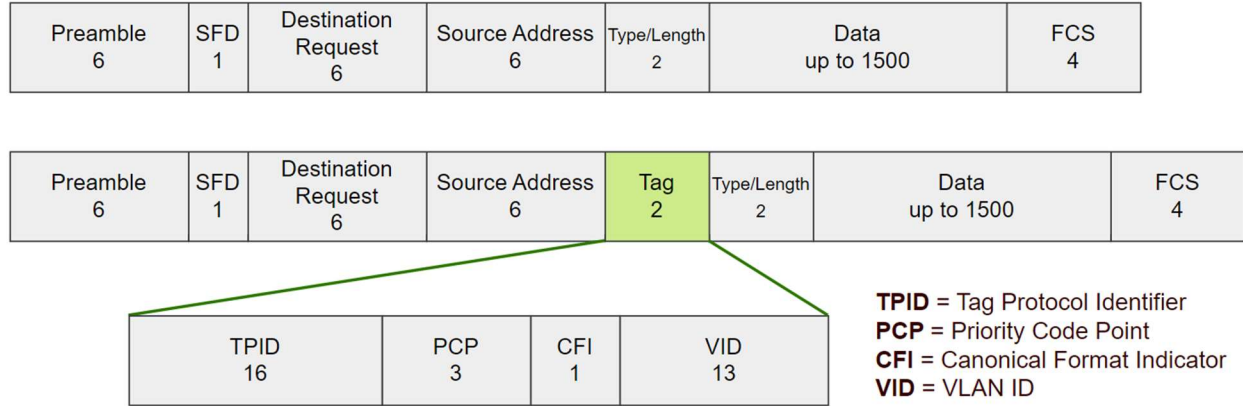
For that, the different standards specified by IEEE 802.1 TSN-TG present the following key properties:

1. **Time synchronization:** All participating devices work in a synchronized way and in real time, with microsecond precision. The standard IEEE802.1AS-Rev specifies the protocol and procedures used to ensure that the synchronization requirements are met.
2. **Scheduling and traffic shaping:** All the intervening devices work under the same rules regarding processing and packet forwarding. The goal is to allow coexistence in the same network, of different types of traffic with different priorities and latency requirements. The most relevant standards that collect this property are IEEE802.1Qbv and IEEE802.1Qav.
3. **Selection and Path reservations:** Every network device work under the same rules when reserving bandwidth and time slots and when choosing paths. For path reservation, it is possible of using more than one path for fault-tolerance. This is a property specially defined in IEEE802.1Qat and IEEE802.1Qca.

These standards make use of the virtual LAN (VLAN) concept, which is a logical independent network within the same physical network. VLANs works by including tags to data frames, in order to handle them as if they were in the same domain. This tags mechanism is defined in IEEE802.1Q from which the majority of TSN standards extend from.

The VLAN tagging also includes a prioritization scheme inside the tag, as shown in Fig. 1.1, named as Priority Code Point (PCP), which with 3 bits, it is possible to define the priority class of a particular frame in a range from 0 to 7. This traffic distinction will be later

used by a TSN mechanism to guarantee the coexistence of different traffic classes accomplishing its QoS requirements.



**Fig. 1.1** VLAN tag

TSN makes use of traffic shaping concept, which refers to bandwidth management technique that distributes frames in time, smoothing out the traffic and preventing buffer congestion, to ensure network performance for higher priority applications.

The main TSN scheduling, and traffic shaping mechanisms proposed for TSN Networks are the IEEE802.1Qav Credit Based Shaper and the IEEE802.1Qbv Time Aware Shaper which are explained in the following sections.

### 1.1. IEEE802.1Qav (Credit Based Shaper)

The IEEE802.1Qav standard plays a crucial role in defining traffic shaping mechanisms using priority classes. One of the commonly employed shaping mechanisms is the Credit Based Shaper, which follows a similar logic to the Leaky Bucket algorithm [28].

The Credit Based Shaper aims to shape the traffic by smoothing out its transmission rate, ensuring a more uniform distribution over time. By limiting the burst size, it helps reduce the occupancy of buffers, thereby minimizing congestion losses and mitigating interferences within the network.

However, it is important to note that implementing the Credit Based Shaper introduces additional network delay. This delay arises from the mechanism's design, as it regulates the transmission of data packets based on available credits. While this delay may be acceptable in many cases, it can be problematic when with very limited time constraints.

In such cases, even slight variations in network delay may disrupt the precise timing constraints of time-sensitive applications, rendering the Credit Based Shaper unsuitable.

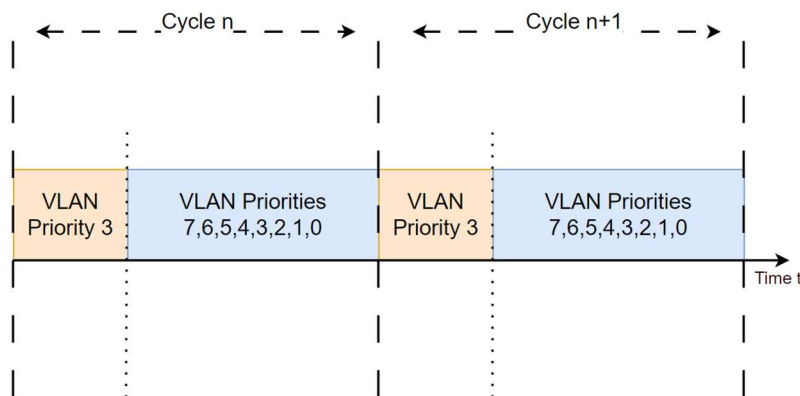
Therefore, when considering the implementation of traffic shaping mechanisms, including the Credit Based Shaper, it is crucial to carefully evaluate the specific requirements of the system.

## 1.2. IEEE802.1Qbv (Time Aware Shaper)

The IEEE 802.1Qbv standard describes the Time Aware Shaper (TAS), a scheduler that defines fixed length, repeating time cycles and assigns time slices of this cycle to different traffic classes. Thus, the transmission of time-critical flows can be granted. For each egress port of a switch, eight different queues (one for each Ethernet priority) are defined and the TAS determines which queues are allowed to transmit in each time slice. To this end, the TAS defines a Gate Control List (GCL) and the frames in a queue are eligible for transmission if the corresponding queue gate is open.

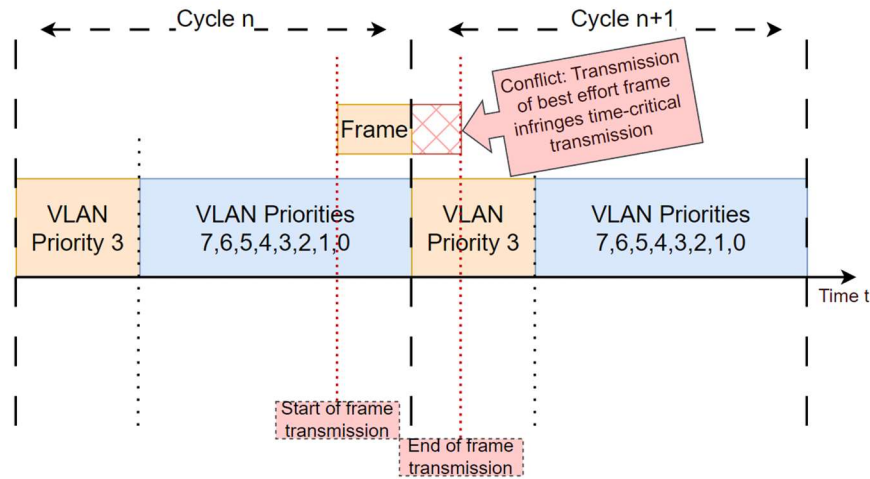
This standard is based on Time-division multiple Access (TDMA) mechanism [30], which establishes a shared medium to allow stream transmission, into many time slots that are cyclically repeated. Therefore, within each of these slots, it is possible to guarantee the exclusive use of the channel for several traffic classes. This way, it is possible to transmit time critical traffic deterministically, without interruptions, avoiding buffer accumulations.

In Fig. 1.2, two transmission channels have been established in the same cycle. Within the first temporary segment, only traffic tagged with VLAN priority 3 is transmitted. The second cycle section, groups the rest of the traffic transmissions ignoring VLAN priority 3 tag.



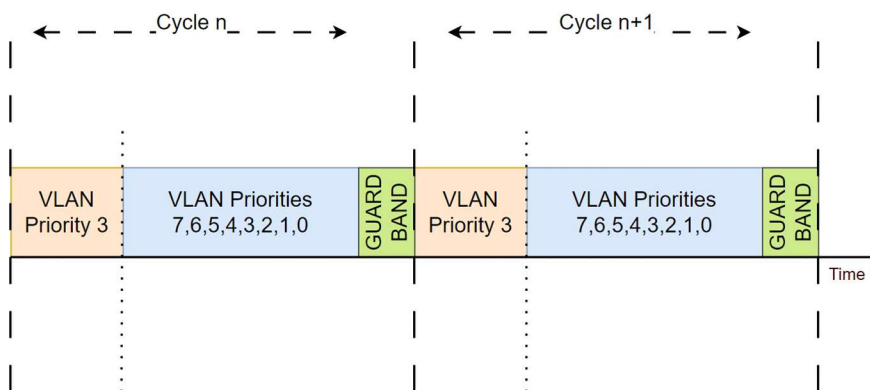
**Fig. 1.2** Two time slices example IEEE 802.1Qbv schedule.

As shown in Fig 1.3, it is possible that the transmission of a frame may not be completed when a new cycle starts.



**Fig. 1.3** Best effort frame interfering the next transmission.

To prevent this situation, a guard band can be defined before each time-critical traffic segment, as shown in Fig. 1.4. During this guard band time, the start of a new transmission is not permitted, so the termination of in progress transmissions is allowed. Therefore, the duration of this guard band should correspond to the transmission required time of the maximum Ethernet frame size, taking into account its headers and interframe spacing.



**Fig. 1.4** Guard bands usage.



As an alternative, frame pre-emption is presented in IEEE802.1Qbu, which consists in momentarily pause low priority frames transmission when higher priority frames need to be transmitted.

### 1.3. Complexity Problems

TSN protocols provide a great variety of possibilities to the network architecture, so these can be combined in a certain way, to achieve the necessary requirements. For example, in TSN network it is common to use TAS for higher priority traffic and, CBS for lower priority traffic.

The fact of having many configuration possibilities within each defined standard, offers flexibility and control to the network architect, although on the contrary results in a much more complex configuration process, which is exacerbated considering the possible interactions between the implemented protocols.

Furthermore, as mentioned in [28], configuring Time Aware Shaper to define the schedule of the Gate Control List of all devices, is proven to be a nondeterministic polynomial time problem, which means that there is no efficient algorithm found to solve it.

### 1.4. Applying TSN

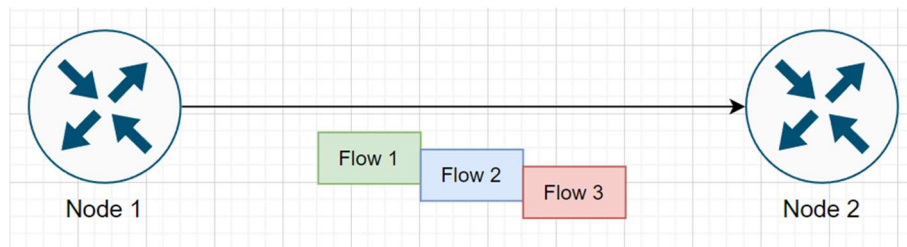
In this project we decided to only consider IEEE802.1Qbv standard to schedule time-triggered traffic, transmitting periodic data streams with hard real-time requirements. On the scheduling problem that we present, we focus on time-trigger traffic so the presence of best effort and AVB traffic is filtered.

In this project, we designed and implemented three distinct scenarios with the aim to explore the transmission of three classes of flows, each with different time requirements. For each analysed case, we transmit 10 time-critical streams and utilize a topology consisting of a single shared link connecting two nodes, as shown in Fig. 1.5. This choice allowed us to create controlled and simplified environments, significantly reducing the number of potential combinations that needed to be considered.

By simplifying the network topology, we could focus our efforts on understanding the specific challenges associated with the scheduling of time-critical flows in these setups.

This topology assumes that:

1. All generated traffic is unicast.
2. There is no packet loss due to transmission errors or buffer overflows.
3. Node processing time is negligible.
4. Propagation time is negligible.



**Fig. 1.5** Network topology

## CHAPTER 2. Dataset Generation

In this project, supervised machine learning models are proposed to predict the feasibility of transmitting a specific set of flows over a link. To train these models, a dataset is required. This dataset is a data matrix and a class label vector.

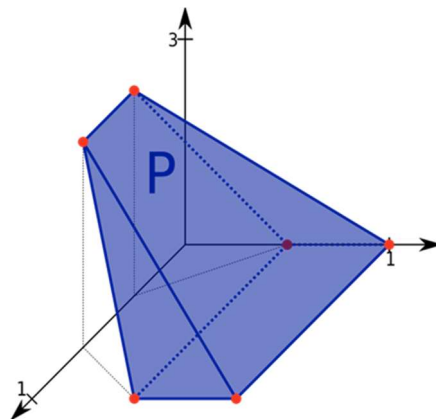
In each scenario three different Time-Triggered classes. Each class has a specific set of characteristics and time constraints. For each scenario, a dataset is built. Each row of this dataset represents a different experiment (or observation), i.e., defines a specific combination of flows of class 1, class 2 and class 3. The first three columns represent the number of flows of class 1, class 2 and class 3, respectively, considered in each experiment. The fourth column is the class label, and it is a boolean value that states if a feasible scheduling exists for each experiment.

To set the class label of each combination of flows, we use an adaptation of the implementation described in [27] of the Integer Linear Programming (ILP) approach proposed in [33].

### 2.1. Integer Linear Programming (ILP)

An Integer Linear Programming (ILP) is a method to maximize or minimize an objective function, subject to one or more restrictions or requirement, from linear relationships such as, equalities and inequalities.

The intersections generated from these inequalities, creates a convex polytope whose interior region is defined as a “feasible region” (see Fig. 2.1), since it fulfils all the constraints of the problem. Then, an objective function is applied to obtain an optimal solution.



**Fig 2.1** *Convex polytope*

### 2.1.1 Preliminary definitions

In this project, we want to schedule a total of  $N$  Time-Triggered flows or streams ( $s_j$ , where  $j = \{0, \dots, N - 1\}$ ) in a link. Each stream belongs to a stream class  $c_i$  that defines the characteristics of the flow. In this work we consider three different classes, and hence,  $i = \{1, 2, 3\}$ . Each stream of class  $i$  must transmit a total of  $B_i$  bytes during its period of length  $P_i$ . In the implementation used in this work, a maximum segment size ( $MSS$ ) in bytes is defined. Then, the number of frames that a stream of class  $i$  generates in  $P_i$  (hereafter,  $F_i$ ) is  $B_i/MSS$ . The subindex  $r$  identifies each frame of a given flow within its period, thus, for a stream of class  $i$  we have  $r = \{0, \dots, F_i - 1\}$ . The stream classes defined for the scenarios of this project consider values of  $B_i$  that are multiple of  $MSS$  and, therefore,  $F_i$  is an integer, and the transmission time ( $t_{tx}$ ) of any frame can be computed as  $t_{tx} = MSS/v_{tx}$ , where  $v_{tx}$  is the link speed. In addition, each class  $i$  has a **deadline**,  $d_i$ , that in this work is defined as the maximum time between the arrival of the last bit of the last frame generated in  $P_i$  (that is, frame  $F_i - 1$ ) at the destination and the start of the transmission of the first bit of the first frame generated in  $P_i$  at the source.

In summary, if a stream  $j$  belongs to class  $i$ ,  $j \in c_i$ , then  $j$  must transmit  $B_i$  bytes within a period  $P_i$ , which is a total of  $F_i$  frames of length  $MSS$  bytes, each with a transmission time  $t_{tx}$ . All  $F_i$  frames must be received at the destination before the deadline  $d_i$ , starting to count when the transmission of the first frame of the flow starts.

The **offset** of each frame is defined as the difference between the start transmission time of a frame with respect to the beginning of its period. That is, for a stream  $j$  of class  $i$ ,  $j \in c_i$ , the offset of frame  $r$  of flow  $j$ ,  $\phi_{j,r}$ , is the difference between the start transmission time of frame  $r$  of flow  $j$  and the beginning of  $P_i$ , where  $r = \{0, \dots, F_i - 1\}$ .

The **end-to-end latency** of a stream, in this implementation, is defined as the difference between the arrival time of the last bit of the last frame generated in a period and the beginning of the transmission of the first frame of the period. Considering a topology with only one link and assuming negligible the propagation delay, for a stream  $j$  of class  $i$ ,  $j \in c_i$ , the end-to-end latency is defined as shown in Eq. 2.1:

$$\lambda_j = \phi_{j,F_i-1} + t_{tx} - \phi_{j,0}, \quad j \in c_i \quad (2.1)$$

The **end-to-end latency lower bound** is defined by the time required to transmit all the frames that a flow generates in its period considering that no other stream would interfere. Considering a topology with only one link and assuming negligible the propagation delay, for a stream  $j$  of class  $i$ ,  $j \in c_i$ , the end-to-end latency lower bound is defined as show in Eq. 2.2:

$$\underline{\lambda}_j = F_i t_{tx}, \quad j \in c_i \quad (2.2)$$

The **hyperperiod**,  $H$ , also called base period, is the GCL cycle time and determines how often the schedule is repeated. It is calculated applying the Least Common Multiple (LCM) of all stream class periods. Hence, the  $F_i$  frames that a stream  $j$  of class  $i$  generates in  $P_i$

are repeated  $H/P_i$  times in an Hyperperiod. It is important to mention that the scheduling mechanism used in this work, considers that the same scheduling, that is the same offset values, are assigned to all  $H/P_i$  repetitions of the  $F_i$  frames of flow  $j$  of class  $i$  within the hyperperiod.

### 2.1.2. ILP Model

The objective of the scheduling problem considered here consists of minimizing the extra end-to-end latency introduced due to the interference of other streams, while trying to accomplish the time requirement of all flows.

The ILP model used in this work is as follows:

$$\min \sum_{j=0}^{N-1} (\lambda_j - \underline{\lambda}_j) \quad (2.3)$$

$$\text{s.t.} \quad \lambda_j \leq d_i \quad \forall j, j \in c_i \quad (2.4)$$

$$\underline{\phi}_{j,0} = 0 \quad \forall j \quad (2.5)$$

$$\phi_{j,F_i-1} \leq P_i - t_{tx} \quad \forall j, j \in c_i \quad (2.6)$$

$$\phi_{j,r-1} + t_{tx} \leq \phi_{j,r} \quad \forall j, j \in c_i, r = \{1, \dots, F_i - 1\} \quad (2.7)$$

$$\alpha P_i + \phi_{k,n} + t_{tx} \leq \beta P_q + \phi_{h,m} + M\sigma \quad (2.8)$$

$$\beta P_q + \phi_{h,m} + t_{tx} \leq \alpha P_i + \phi_{k,n} + M(1 - \sigma) \quad (2.9)$$

$$\begin{aligned} &\forall k, k \in c_i, n = \{1, \dots, F_i - 1\}, \alpha = \{0, \dots, (H/P_i) - 1\} \\ &\forall h \neq k, h \in c_q, n = \{1, \dots, F_q - 1\}, \beta = \{0, \dots, (H/P_q) - 1\} \end{aligned}$$

Eq. 2.3 is the objective function.

Without going into details of each constraint, here we just give an idea of their finality.

Eq. 2.4 ensures that the end-to-end latency of each flow is equal or less than the deadline of its class.

Eq. 2.5 states that the lower bound of the offset value assigned to the first frame of any flow is 0.

Eq. 2.6 defines that the maximum offset of the last frame of a flow in its period must be equal or less than the period of its class minus the transmission time of a frame.

Eq. 2.7 ensures that all frames of the same flow are transmitted in order and that their transmission do not overlap.

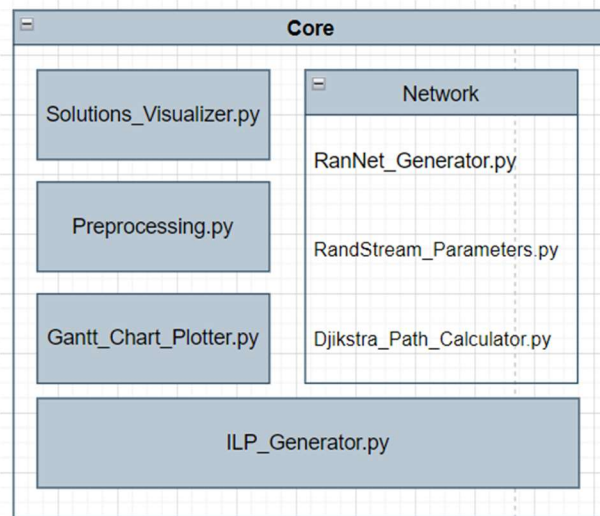
Eq. 2.8 and Eq. 2.9 are required to prevent overlapping between frames of different flows within an hyperperiod.  $M$  represents a theoretically infinitely large constant which causes either the inequality of Eq. 2.8 or Eq. 2.9 to be trivially satisfied if  $\sigma = 1$  or  $\sigma = 0$ , respectively.

## 2.2. Implementation

In this subsection, we provide a description of the implementation and explain its inputs and outputs. We also present the implemented studied cases from which we developed this project.

The ILP solution has been implemented using Python 3.9 and it is originally made up from a set of scripts on which the different responsibilities are divided during execution, see Fig. 2.2.

The core implementation used in this project is an adaptation of the implementation made in [27] of the ILP model of [33]. In this adaptation, the objective function and the restrictions presented in this second chapter are considered.



**Fig 2.2** ILP Core Implementation [27]

Throughout the following explanation, mentions are made of each of the implemented services both from the ILP Core and the additional implementation set.

The ILP model, has been implemented with the provided tools included in the Pyomo software collection package [5]. This is a BSD licensed project which can use multiple linear programming solvers such as CBC [6] or GUROBI. [7].

The main script of the ILP core implementation is the *Solutions\_Visualizer.py* through which all core classes are orchestrated and executed. Firstly, networks are randomly generated using the *RanNet\_Generator.py* script. This process involves generating the network topology and establishes connections between nodes.

Next, the path that the time-triggered flows should follow within the network is determined using the *Dijkstra\_Path\_Calculator.py* script. This step calculates the optimal paths for the flows based on Dijkstra algorithm.

Once the paths are determined, three classes of time-triggered traffic with specific temporal requirements are randomly generated using the *RandStream\_Parameters.py* and *Preprocessing.py* scripts. These classes define the characteristics and timing constraints of the flows.

Finally, the *ILP\_Generator.py* script is used to schedule the generated time-triggered flows based on the determined paths and temporal requirements.

Given that in this project we aim to use a fix topology, in this implemented adaptation we prevented the execution of *RanNet\_Generator.py* and *Dijkstra\_Path\_Calculator.py*. Instead, a fix network and fixed transmission paths are provided.

In addition, a set of scripts have been implemented which are essential for the proper functioning of this solution. We have developed three scripts (*init\_combinations.py*, *filter\_combinations.py*, *provide\_combinations.py*) to control the combinations that are executed in the ILP.

These scripts initialize, filter, persist and provide the flow combinations that will be executed by the ILP algorithm. The *init\_combinations.py* is capable of generating all possible flow combinations, to ensure that all cases are analysed. Since the generated combinations do not contain repetitions, each vector combination can only have one specific format. I.e., if a vector combination [1, 1, 3] has been already generated, [3, 1, 1] won't be included.

Originally, the core itself, was in charge of randomly generating flow combinations *RandStream\_Parameters.py*. Therefore, there could be cases in which certain flows combinations would not be analysed, or even cases in which many identical configurations would be fed to the ILP algorithm.

After, generating the considered number of combinations, the *filter\_combinations.py* script filters every set of flows whose bitrate sum exceeds the link bandwidth and store these results. Then the *provide\_combinations.py* provides the actual vector of flow combinations for every ILP Core iteration.

Next, the frame duration per flow type and the number of repetitions within a Hyperperiod are determined. This calculation is performed also in "Preprocessing.py" as it was originally implemented but preventing hard-coded parameters.

Afterward, the constraints inequalities explained in the previous section are applied to each of the flows, so that we can obtain the polyhedron to be optimized with the objective function. Then the ILP model tries to find the most optimal scheduling solution within the defined constraints using a GUROBI [7] solver.

Every flow combination whose scheduling result has a feasible solution or not, is written in a results file and a Gantt chart is stored by default to properly visualize the scheduling result.

## 2.3. Enhancing the ILP implementation

This ILP model presents hard-coded parameters due to its original purpose. The link bandwidth, the packet size, and the number of flow types to transmit among others, were strongly hard-coded parameters, hard to change without affecting the correct functioning of this implementation. Other parameters such as, the transmission periodicity and the end-to-end deadline constraint, were easy to modify.

Therefore, we decided to apply some changes to disengage the main key parameters, so that this ILP model could be controlled through a configuration file (JSON format). With these configuration files, now it is possible to easily change; end-to-end deadline, transmission periods, the link bandwidth, and the packet size of each flow class, as it is explained in section 2.4.1.

## 2.4. Executing the ILP

Now that the implementation of the entire ILP program has been explained, this subsection shows and clarifies the procedure to execute this solution explaining the required inputs and resulting outputs to verify the feasibility of scheduling a given set of streams. This feasibility label will later be used by the machine learning algorithms.

It is important to note that the execution of all scripts in this program were carried out using Conda [10] as the package manager and environment handler, this allows reproducibility of the same environment across different machines on which the ILP was executed. The packages used and instructions for recreating the environment can be found in the annex.

### 2.4.1. Inputs

The parameters for a specific scenario are defined in the “StreamsConfig.json” configuration file. Fig. 2.3 provides an example of the content within this file, which



includes the flow classes, denoted as an array "streams," and the total number of transmitted flows in a particular case, indicated by "n\_streams."

For each class, the configuration includes the size of the transmitted flow, specified as "size," the period of the flow denoted as "Period," and the maximum acceptable end-to-end delay defined as "Deadline." Additionally, the link speed, represented by "Bandwidth," is also specified within the configuration file.

```
StreamsConfig.json > ...
1  {
2      "n_streams" : 3,
3      "streams": [
4          {
5              "Size": 4500,
6              "Period": 600,
7              "Deadline": 400
8          },
9          {
10             "Size": 1500,
11             "Period": 300,
12             "Deadline": 100
13         },
14         {
15             "Size": 1500,
16             "Period": 600,
17             "Deadline": 100
18         }
19     ],
20     "Bandwidth": 120000
21 }
```

**Fig 2.3** StreamsConfig.json Configuration file example. Size (Bytes), Period (ms), Deadline (ms), Bandwidth (bps).

Once the characteristics of the streams have been set, the next step is to configure the "ILPConfig.json". Fig 2.4 shows an example of this file. The "timeout" parameter is the maximum execution time given to the ILP to say if the combination of flows can be scheduled or not. Hence, it is important to set up this feature according to the used machine resources. The "distributions" parameter should be set so it refers to the file path where "init\_combinations.py" has saved the combinations of flows to be computed. The "base\_path" parameter is necessary to set a default results folder where the output files named as "results\_filename" will be stored.

```

{...} ILPConfig.json > ...
1  {
2      "distributions" : "distr.txt",
3      "timeout": 600,
4      "results_filename": "test",
5      "base_path": ".\\result"
6  }

```

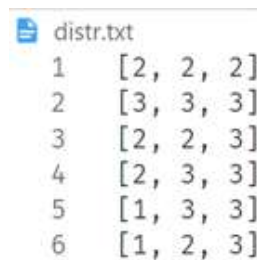
**Fig 2.4** ILP execution configuration “distributions”: distribution file path, “timeout”: (s) maximum iteration time

The next step is to run the “init\_combinations.py” script, which automatically generates and stores all possible flow combinations based on the total number of flows to be transmitted in a given scenario, without repetition.

Then, the “filter\_combinations.py” should be executed to filter those combinations whose transmission rate sum is greater than the capacity of the link. These filtered combinations are automatically saved in the resulting dataset so the machine learning can consider them later.

If we consider the example shown in Fig. 2.3 which are the combinations after applying “filter\_combinations.py”, the ILP must analyse are the ones shown in Fig 2.5. These combinations are then consumed by the ILP core implementation through the “provide\_combinations.py” service allowing the evaluation of their feasibility.

For that, the main script of the ILP solution “Solution\_Visualizer.py” will iterate over every combination and generates the corresponding outputs.



```

distr.txt
1  [2, 2, 2]
2  [3, 3, 3]
3  [2, 2, 3]
4  [2, 3, 3]
5  [1, 3, 3]
6  [1, 2, 3]

```

**Fig 2.5** Combinations to compute by the ILP algorithm

### 2.4.2. Outputs

After each iteration of computation, the solution captures both the environment in which the ILP was executed and its corresponding result. If applicable, a Gantt chart is generated to visually represent the proposed scheduling by the algorithm. The results of

each iteration are stored in separate files, with results saved in a text file format and in a CSV file format.

The following subsections explain the different outputs that can be obtained using this ILP solution. Every example included in the subsequent figures corresponds to the outputs generated from the computation of the combination [1,1,1], that is, one flow of class 1, one flow of class 2 and one flow of class 3. The characteristics of each flow class are the ones defined in the file shown in Fig. 2.3.

#### 2.4.2.1. Text file

The text file follows a fixed structure, as indicated in Fig. 2.6.

```

1  {
2      "Distribution": [1, 1, 1],
3      "Adjacency_Matrix": [[0, 1], [1, 0]],
4      "Stream_Source_Destination": [[0, 1], [0, 1], [0, 1]],
5      "Link_order_Descriptor": [[0], [0], [0]],
6      "Links_per_Stream": [[1], [1], [1]],
7      "Number_of_Streams": 3,
8      "Frames_per_Stream": [[1, 1, 1], [1], [1]],
9      "Deadline_Stream": "{0: 400, 1: 100, 2: 100}",
10     "Streams_Period": "{0: 600, 1: 300, 2: 600}",
11     "Streams_size": [300, 100, 100],
12     "Clean_offsets":
13     [
14         {"Task": "('S', 0, 'L', 0, 'F', 0)", "Start": 100.0},
15         {"Task": "('S', 0, 'L', 0, 'F', 1)", "Start": 200.0},
16         {"Task": "('S', 0, 'L', 0, 'F', 2)", "Start": 400.0},
17         {"Task": "('S', 1, 'L', 0, 'F', 0)", "Start": 0.0},
18         {"Task": "('S', 2, 'L', 0, 'F', 0)", "Start": 500.0}
19     ],
20     "Latencies": [400.0, 100.0, 100.0],
21     "Feasibility": true
22 }
```

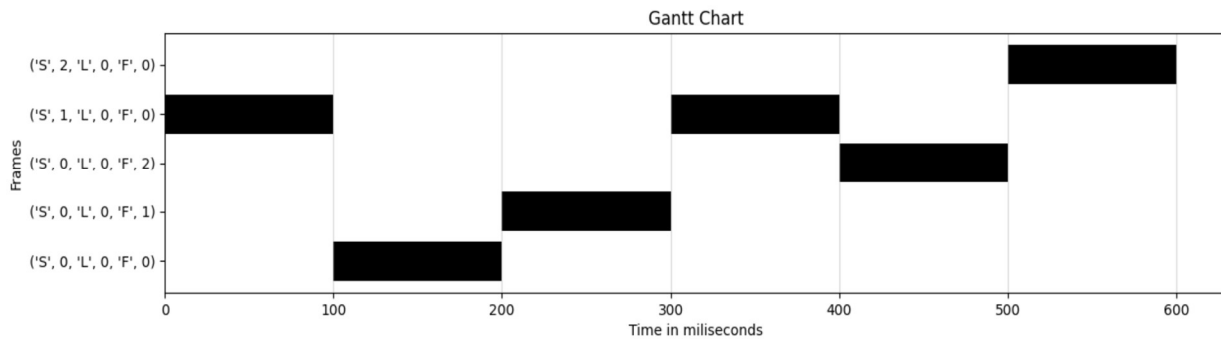
**Fig 2.6** Output computation example

- *Distribution*: Each vector position shows the number of transmitted flows per class (T1, T2, T3).
- *Adjacency\_Matrix*: Describes the network topology.
- *Stream\_Source\_Destination*: Indicates the sequence of hops that a flow must traverse to reach its destination.

- *Link\_order\_Descriptor*: Defines the path where each stream is transmitted.
- *Links\_per\_Stream*: List of the number of links travelled per stream.
- *Number\_of\_Streams*: Total number of transmitted streams.
- *Frames\_per\_Stream*: Matrix containing vector of one's for each transmitted frame per flow.
- *Deadline\_Stream*: Dictionary indicating the deadline in milliseconds per flow class.
- *Streams\_Period*: Dictionary with the periods per flow class.
- *Streams\_size*: Ordered vector showing the total transmission time for each flow class.
- *Clean\_Offsets*: A collection of frames specifying the timing at which each transmission should begin (Start). The "Task" string denotes the frame, where "S" represents the stream class, "L" denotes the link, and "F" identifies the frame number.
- *Latencies*: Ordered vector where the end-to-end latency is shown.
- *Feasibility*: Indicates whether the combination is schedulable or not.

#### 2.4.2.2. Gantt Chart

As previously mentioned, together with the text result, a Gantt Chart is generated to provide a visual depiction of the output. Fig. 2.7 shows an example of a Gantt chart generated from the results obtained when scheduling 3 flows, one of each class described in the file shown in Fig. 2.3.



**Fig 2.7** Gantt chart example

This example shows the influence of the deadlines in the scheduling problem.

In the given Gantt chart, 3 classes of streams with frames of 100ms are scheduled. It is observed that the three frames belonging to flows "S":0 and the single frame of "S":2, can

be evenly distributed within the 600ms, as their periods align with the Hyperperiod. In contrast, the flow "S":1 with 2 frames, has a period of 300ms, requiring a time difference of 300ms between frames of the same class.

In this example, the deadline used for "S":0 is 400ms and the end-to-end latency of this is also 400ms since, there are 3 frames that require 100ms and one of them has been delayed 100ms (see the definition of end-to-end latency).

However, if we were to configure the deadline for "S":0 to 300ms, it would be necessary to transmit all "S":0 frames as a burst, sending them consecutively without any time gap in between so that its end-to-end delay would be equal to 300ms. However, due to the influence of the "S":1 stream, there is no case where "S":0 can be transmitted as a burst without overlapping with "S":1.

#### 2.4.2.3. CSV File

This comma-separated file consists of 5 fields:

- T1, T2, T3: Are variables describing the number of flows assigned to each flow classes.
- Feasibility: Is the class label to be predicted. It is a boolean that indicates whether the flow combination is schedulable or not, according to the ILP algorithm.
- Exceeds Bandwidth: Is a boolean value that distinguishes combinations whose bitrate exceeds the bandwidth capacity. Since it is evident that these combinations cannot have a feasible scheduling, they are automatically included in the dataset without being computed by the ILP algorithm. This value will allow us to visualize combinations with evident scheduling unfeasibility, see 2.4.2.4.

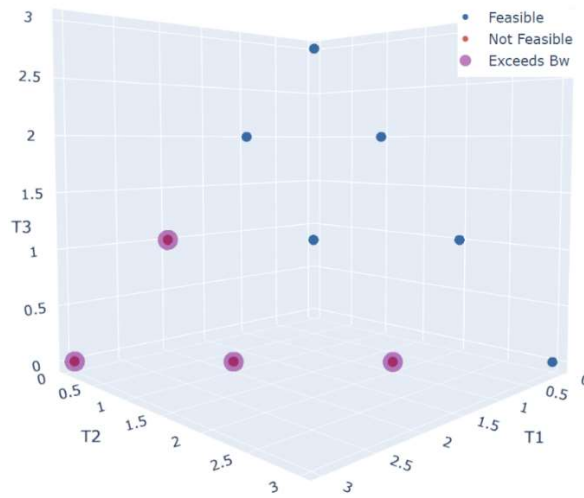
To train the ML models, the selected features are T1, T2 and T3. Although other features have been tested, none have managed to enrich the dataset and improve the performance of the machine learning models trained in chapter 4.

#### 2.4.2.4. Scatter Plot

The results of the different implemented scenarios are presented along with a three-dimensional graph where each set of flow distributions is shown as a dot, see Fig. 2.8.

Each axis references a flow class, and the dot colours show whether the distributions are feasible or not, according to traffic characteristics of each scenario. In addition, those combinations that are non-feasible due to exceeding the capacity of the shared link have been marked with purple.

The combination [1, 1, 1], where one stream of each class is transmitted, is located in the centre of the scatter plot in Fig. 2.8, and it refers to the example case analysed in 2.4.2.2.



**Fig 2.8** Scatter plot example

## 2.5. Implemented Scenarios

This section provides an overview of three scenarios and their corresponding results obtained from executing the ILP implementation. In each scenario, the same network topology is implemented, and changes are made to characteristics of the transmitted traffic.

The traffic characteristics table of every scenario describes the transmitted flows classes (T1, T2, T3), including its packet size, as well as its transmission period and its end-to-end transmission delay deadline that must be satisfied.

### 2.5.1. Scenario 1

- Total flows: 10
- Link bandwidth: 800Kbps
- Frame transmission time: 15ms

Table 2.1 describes the characteristics of each flow class in this scenario., Streams of class T1 transmit 3 frames of 1500 bytes each, with a period of 600ms. Streams of class T2 transmit 2 frames of 1500 bytes, with a period of 300ms. Lastly, streams of class T3 transmit 1 frame of 1500 bytes, with a period of 200ms.

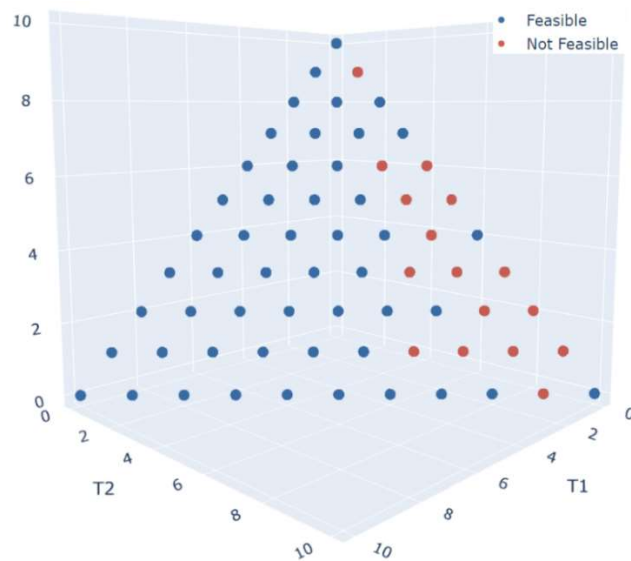
Given a link speed of 800Kbps, each 1500 bytes frame is transmitted in 15ms. In this scenario, a total of 10 flows are transmitted.

**Table 2.1** Traffic characteristics table of scenario 1

Flow Class	Packet Size (Bytes)	Period (ms)	Deadline (ms)
T1	4500	600	60
T2	3000	300	30
T3	1500	200	15

As show in Fig 2.9 the ILP has been able to obtain results on each of the combinations. None of these combinations exceed the capacity of the shared link, so they are all potentially schedulable. However, it can be observed how the group of not feasible appears dispersed on the right side of the graph where there is a greater presence of T2 flows.

The T3 type flows do not present an impediment for scheduling since they only need to allocate 15ms every 200ms. Although they must respect the end-to-end delay deadline, in the case of T1 flows they also facilitate scheduling since they are flows that must allocate 3 frames of 15ms in every 600ms.

**Fig 2.9** 3D scatter plot of scenario 1

### 2.5.2. Scenario 2

- Total flows: 10
- Link bandwidth: 1Mbps
- Frame transmission time: 12ms

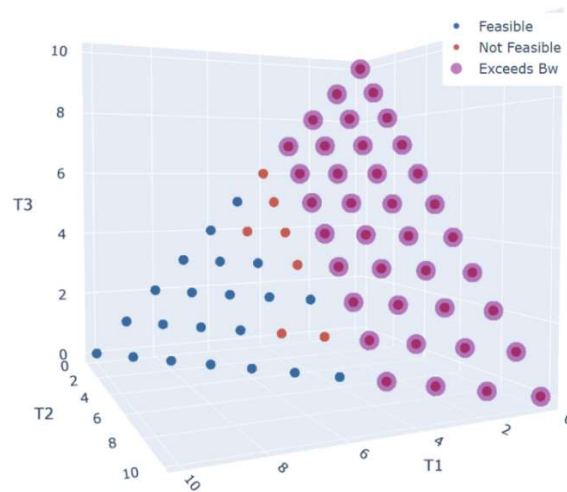
In this scenario a total of 10 flows are being transmitted over a 1Mbps link. Table 2.2 describes the characteristics of each flow class. Streams of class T1 are configured to transmit 3 frames of 1500 bytes within a time interval of 600ms between transmissions. The streams of class T2 also transmit 3 frames of 1500 bytes, but with a shorter time interval of 300ms. Lastly, the streams of class T3 transmit 2 frames of 1500 bytes every 200ms.

**Table 2.2** *Traffic characteristics table of scenario 2*

Flow Class	Packet Size (Bytes)	Period (ms)	Deadline (ms)
T1	4500	600	48
T2	4500	300	36
T3	3000	200	24

As Fig.2.10 shows, the ILP algorithm has been able to generate results on all combinations. However, in this case the number of iterations that have been carried out has been less, since many of the combinations have been previously filtered due to exceeding the capacity of the shared link.

The configurations that exceeded the capacity of the link (marked in purple) are mainly due to the presence of type 2 and 3 flows, since they are the flows with the highest bitrate. The rest of the non-feasible combinations are close to this large group.



**Fig 2.10** 3D scatter plot of scenario 2



### 2.5.3. Scenario 3

As the number of frames included in the scheduling process increases, the computation time required by the ILP implementation also escalates significantly. Consequently, there are cases where the resulting dataset may not include all the desired results due to a preconfigured timeout.

As an example of an incomplete scenario, the scenario 3 has been implemented reducing the “timeout” parameter. The resulting dataset then has missing results that we later discuss in chapter 5, where we try to predict those missing combinations.

Furthermore, this scenario combines the results obtained from computing the same scenario with different total transmitted flows: 10 and 12, resulting in the generation of distinct planes of data points. Despite the variations in the total number of flows, both cases share the same traffic characteristics.

This third scenario allows us to examine and compare the machine learning models behaviour over datasets with missing results and including different planes of data points.

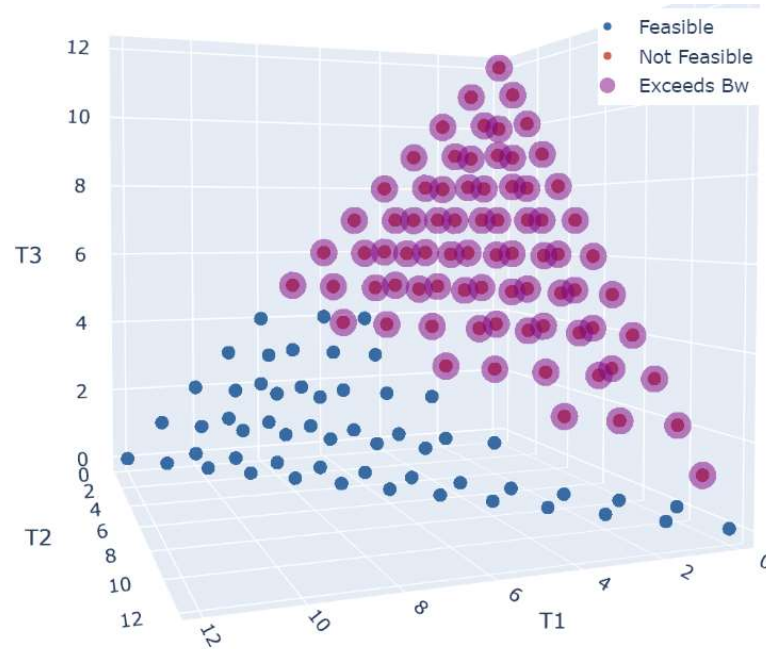
- Total flows: 10 and 12
- Link bandwidth: 1.5Mbps
- Frame transmission time: 8ms

In this scenario, streams are transmitted over a 1.5Mbps capacity link. Table 2.3 describes the characteristics of each flow class. Streams of class T1 transmit 3 frames of 1500 bytes every 600ms. Streams of class T2 transmit 3 frames of 1500 bytes every 300ms. Finally, streams of class T3 transmit 4 frames of 1500 bytes every 200ms. Given the link speed, the transmission time of each frame of 1500 bytes is 8ms.

**Table 2.3** Traffic characteristics table of scenario 3

Flow Class	Packet Size (Bytes)	Period (ms)	Deadline (ms)
T1	4500	600	24
T2	4500	300	24
T3	6000	200	32

As shown in Fig. 2.11, the graph presents a gap that separates feasible and non-feasible combinations. This gap refers to the portion of the problem space where the instances require the longest computation time when executing the ILP algorithm. By utilizing machine learning techniques, we can make predictions and gain insights into the areas of the space where the computation time is longer.



**Fig 2.11** 3D scatter plot of scenario 3

## CHAPTER 3. Feasibility Prediction Algorithms

Once we know and understand the generated data, we can consider using it as training for supervised machine learning models, with which to determine the feasibility of a certain flow configuration. In the following subchapters, we introduce a brief explanation of the predictors used in this project.

### 3.1. Binary Classification Algorithms

Classifications models can be classified as generative and discriminative [35].

Generative models are those that focus on the distribution of the dataset modelling the data to find the joint probability in which the two events occur simultaneously according to the results obtained. These are typically used to estimate probabilities and likelihoods by modelling data points and discriminating between classes or groups based on these probabilities. Because the model learns a probability distribution, it is possible to use that probability distribution to generate new data instances.

Instead, discriminative models aim to learn about the boundary between classes within a dataset in order to identify the decision boundary between classes to label the data and perform a classification.

For this project we have considered the use of two discriminative algorithms: Support K-Nearest Neighbor (K-NN) and Vector Machine (SVM).

### 3.2. K-Nearest Neighbours (K-NN)

The authors in [2], where a similar problem appears, propose to use a K-NN algorithm. Below is an introduction to the fundamentals of K-NN to understand the obtained results.

K-NN [36] is a non-parametric supervised algorithm that uses proximity to make classifications or predictions about the grouping of an individual data point. This is an instance-based lazy algorithm, meaning that instead of undergoing a training stage, it heavily relies on memory to store all its training data which is used to perform all the necessary computation to make a classification.

For a new instance, a classification is made by finding the K closest instances in the dataset. For this algorithm the Euclidean distance [26] has been used. To choose the K value, we must take into account that there are no predefined statistical methods to find the best K value.

About K value, it is important to consider that:

1. Using small values of  $K$  means that noise will have higher influence on the results.
2. Employing large  $K$  values will be more computationally expensive and will result in a less generic model.

In this project, we conducted multiple iterations to evaluate the appropriate  $K$  value to implement in a  $K$ -NN model.

### 3.3. Support Vector Machine (SVM)

A Support Vector Machine (SVM) [32] classifies observations by constructing a hyperplane also called decision boundary, that separates observations belonging to two groups (classes) of data. The goal of SVM is to find an optimal hyperplane that separates different classes of data points.

To generate the hyperplane, before a mathematical function known as kernel is performed. This function transforms the input data into a higher-dimensional feature space, where the classes can be separated by a hyperplane. This process is known as the kernel trick.

Later, the support vectors are selected. We call support vectors to those observations that lie on the margin surrounding the data-separating hyperplane, which are the only points that have a real impact on the model. We can even discard the rest without affecting the results.

There are two key concepts related to the hyperplane margin: hard margin and soft margin.

Hard margin aims to find a hyperplane that perfectly separates the classes without allowing misclassifications. This means that the hyperplane can even be a two-dimensional straight line or a flat plane. Therefore, hard margin models are suitable for situations where the data is perfectly separable. However, hard margin SVMs have some limitations. They are sensitive to outliers and noise since a single misclassified point can significantly affect the placement of the decision boundary. Soft margin SVMs address the limitations by allowing a certain degree of misclassification in the training data.

This trade-off is controlled by the  $C$  parameter, which adds a directly proportional penalty to the distance from the decision boundary, for each misclassified data point. For a small  $C$  the penalty is low while for a large  $C$  the reverse occurs.

When the data is not linearly separable, using a Radial Basis Function (RBF) as a kernel function is commonly used.

The Radial Basis Function (RBF) measures the similarity between data points based on their distance in the feature space, allowing SVMs to model nonlinear decision boundaries. To tune the RBF kernel the gamma parameter is used. The gamma value determines the reach of the kernel function, with higher values leading to a narrower decision boundary.

## CHAPTER 4. Model Testing

In this section, the supervised algorithms explained in the previous section are trained using the complete datasets from the scenarios of this project to compare which algorithm is able to perform a classification of feasible and unfeasible schedulings.

In the first two scenarios, where we have access to all the available data, we conducted a comparison of the resulting models to determine their effectiveness and identify the best-performing supervised model. However, it is important to note that in some cases, the scenario results may be incomplete due to a configured timeout (Scenario 3).

Before training any algorithm, a standard scaler is applied to the dataset to ensure that all features have the same impact in algorithm training and speed up training process. These algorithms use Euclidean distances to train its decision boundaries, so these are very sensitive and are more likely to yield poor results if standardization or normalization is not applied.

Although scaling from a given complete dataset with the same the scaling would be unnecessary, it is a good practice to implement it since exists cases in which the dataset would not be complete. Therefore, we decided to apply a standard scaler because we already know that all features are limited to the same range, and it is less sensitive to outliers than a “Minmax” normalization.

To train both algorithms, a method called “GridSearch” of the Sklearn Library [19] together with Leave One Out cross-validation (LOOCV) [38] is applied in order to build the most robust models with the actual available data.

This “GridSearch” method helps to automate the process of finding an optimal set of hyperparameters. This is done by providing a range parameter to train every combination and then evaluate them using an evaluation metric, which in our case is accuracy. This performance metric represents the proportion of correctly classified predictions out of the total number of instances.

In our case, these evaluation metrics result from the application of the LOOCV. This technique divides the provided dataset into subset, where each subset consists of all but one data point. The model is then trained on the remaining data points and tested on the single data point that was left out.

This process is repeated for each data point in the dataset and its performance is evaluated also using the accuracy metric. This evaluation allows us to identify the model with the best performance, which can be selected for further analysis.

Although cross-validation methods can be computationally expensive, we took advantage of having a small dataset to obtain the most representative models, less biased by the provided training data.

This chapter presents and reviews results obtained from training and applying the K-NN and SVM algorithms. Its subsections include a scatter plot resulting from applying the model, a table displaying instances that were incorrectly classified, and an analysis of the results.

## 4.1. Testing K-NN

The K-NN algorithm is an attractive option due to its simplicity and ease of implementation. It is important to note that, as a lazy learner, it does not require a training phase, as it only stores the training data.

However, it can be computationally expensive when working with large datasets or high-dimensional data. In this case, since the dataset is small, computational cost is not an issue, but it is important to keep it in mind that as the dataset grows, this may become a concern.

Here is an analysis of the results obtained from applying the machine learning algorithm to the dataset in the first two scenarios described in 2.5. As mentioned above, LOOCV has been applied to the K-NN algorithm in each scenario considering a Euclidean distance as a metric.

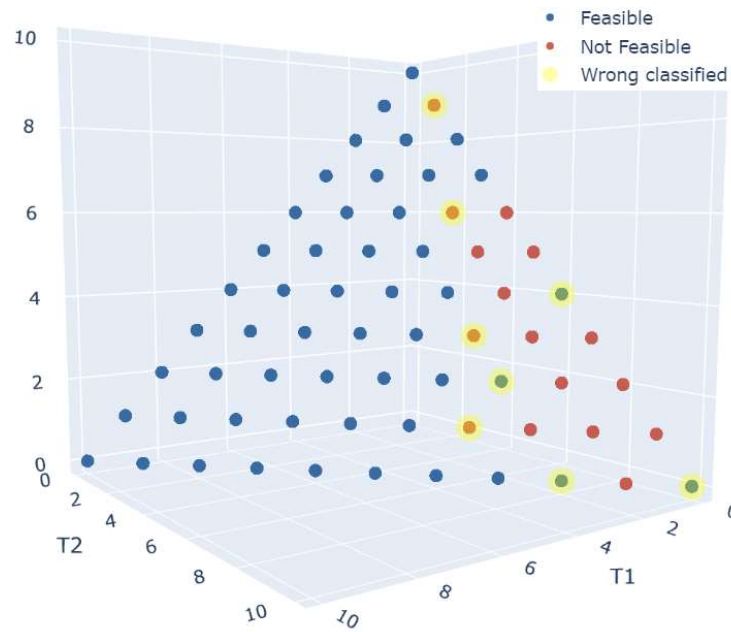
### 4.1.1. Scenario 1

An important factor to consider when applying K-NN is the maximum value of K. In this specific case, since the dataset contains 16 samples of unfeasible configurations, which is 24.24% of all the data, it would be wise to avoid models with a value of K greater than 16, as this could lead to the unfeasible instances being outvoted by feasible instances.

The best predictor obtained from the application of the LOOCV is when  $K=7$  resulting in an 87.88% of accuracy. Since the dataset is made up of 75.76% of instances in which the combination of flows is feasible, the application of this predictor gives us an extra 12.24%.

In the resulting scatter plot, which is Fig. 4.2, the predictor has not correctly predicted instances 1, 2 and 3 from Table 4.1, located on the right margin of the figure since they are outliers.

It is worth noting that using 7 neighbors in this scenario may be considered excessive, as it represents almost 50% of the unfeasible instances concentrated in the lower right margin. This supposes a greater difficulty for the classifier when classifying instances 4, 5, 6 and 7 from Table 4.1.



**Fig 4.1.** 3D scatter plot of scenario 1 with wrong classified instances (K-NN)

**Table 4.1** Wrong classified instances of K-NN in Scenario 1

Instance Id	T1	T2	T3	Feasibility
1	0	1	9	False
2	0	6	4	True
3	0	10	0	True
4	1	3	6	False
5	2	5	3	False
6	2	6	2	True
7	3	6	1	False
8	2	8	0	True

### 4.1.2. Scenario 2

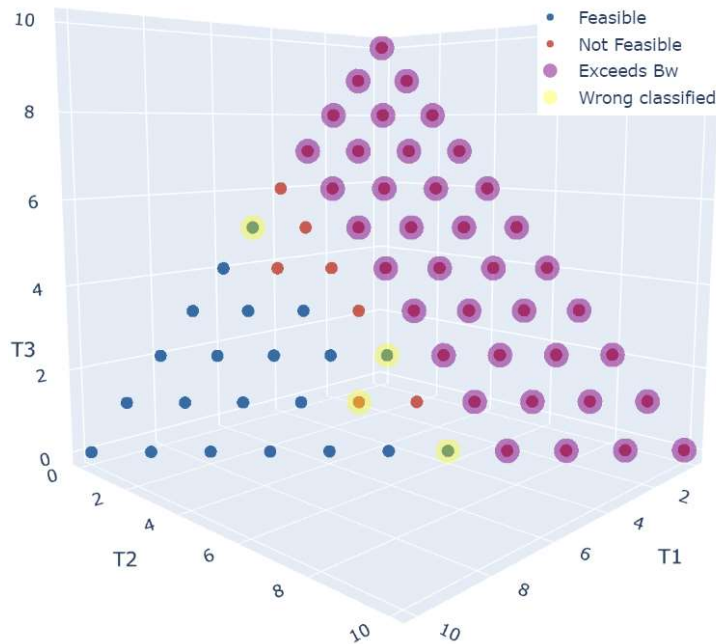
For this scenario, a maximum value of  $K$  equal to 21 has been used in the training of the K-NN algorithm models since the maximum of feasible instances is 21.

Within this dataset, there are 38 combinations that exceed the maximum capacity of the shared link. This represents a significant portion of the unfeasible combinations, considering that there are a total of 45 unfeasible instances.

Consequently, the challenge in this scenario lies in accurately predicting the infeasible instances among those combinations where the unfeasibility is not obvious. These cases require careful consideration.

The best predictor obtained from applying the LOOCV method is achieved when  $K=12$ , resulting in an accuracy of 94%. However, it is important to note that this accuracy is biased by the presence of unreliable instances within the dataset.

When considering the instances located on the diagonals where misclassifications occur in Fig. 4.2, 4 out of a total of 13 instances have been incorrectly classified, resulting in an error rate of approximately 30%, which is a poor performance. See incorrectly classified instances in Table 4.2.



**Fig 4.2** 3D scatter plot of scenario 2 with wrong classified instances (K-NN)



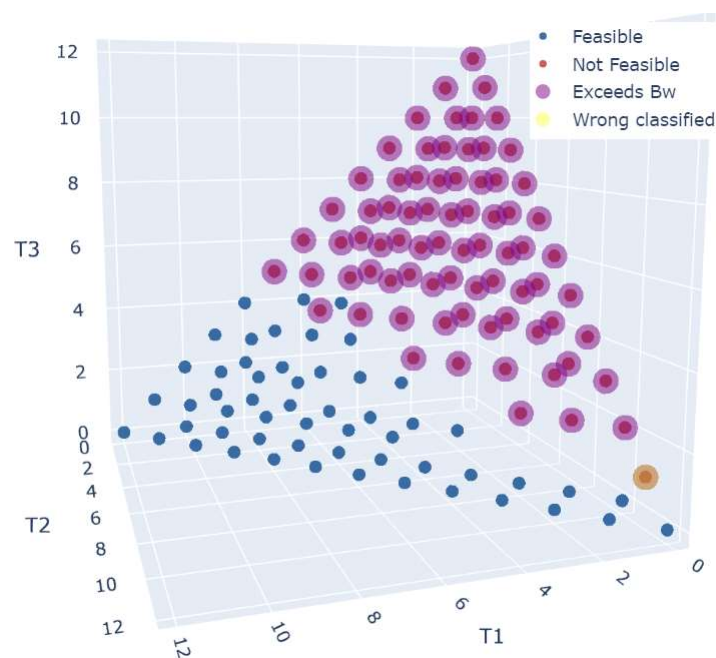
**Table 4.2** Wrong classified instances of K-NN with LOOCV in Scenario 2

Instance Id	T1	T2	T3	Feasibility
1	5	0	5	True
2	4	4	2	True
3	5	4	1	False
4	4	6	0	True

### 4.1.3. Scenario 3

In this scenario the dataset contains a larger number of instances compared to previous scenarios and there are no unfeasible combinations that do not exceed the bandwidth of the shared link. As a result, there is a clear separation between feasibility areas indicating that the classifier should be able to achieve 100% accuracy.

However, the K-NN model with K=5 achieves an accuracy of 99%, meaning that it incorrectly predicts a single combination (Table 4.3), as shown in Fig 4.3.

**Fig 4.3** 3D scatter plot of scenario 3 with a wrong classified instance (K-NN)

**Table 4.3** Wrong classified instances of K-NN with LOOCV in Scenario 3

Instance Id	T1	T2	T3	Feasibility
1	1	11	1	False

## 4.2. Testing SVM

The inner workings of the Support Vector Machine (SVM) algorithm can be intricate, but its main concept is relatively straightforward to understand. Particularly in scenarios where combinations can be represented as points in a three-dimensional graph, there is often a distinct separation between feasible and unfeasible instances.

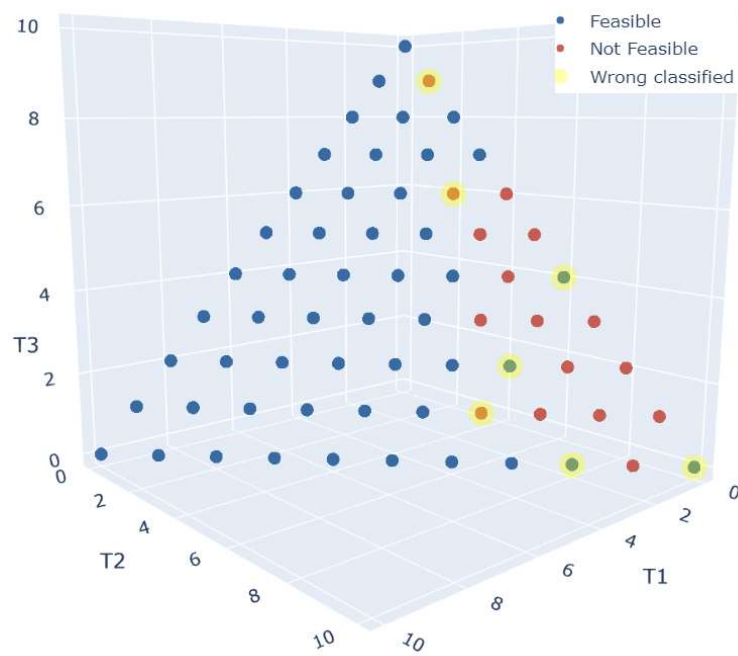
To build the SVM models, the focus was on adjusting the regularization “C” and “gamma” parameters so that the model creates an adequate hyperplane. Given that these parameters can differ depending on the provided training dataset, the resulting hyperparameters cannot be compared.

The considered “GridSearch” parameters are C values from 0.1 to 50 with a step size value of 1, and a range of values from 1.e-09 to 1.e+03 in 13 steps for the “gamma” value. The results obtained from the application of the SVM training with LOOCV using the RBF kernel, are presented in the next subsections.

### 4.2.1. Scenario 1

In this first scenario, the SVM model with hyperparameters  $C=5.1$  and  $\gamma=0.1$  serves as the optimal predictor, achieving an accuracy of 89.4%. This accuracy is slightly superior to that of the K-NN model. It is worth emphasizing that instances 1, 2, and 3 from Table 4.3, situated on the right margin of the Fig. 4.3, are considered outliers, and should not be expected to be correctly classified by a non-overfitted model.

These instances are introducing an error of approximately 4.45% into the model, so consequently, the maximum attainable accuracy for this dataset is 95.45%.



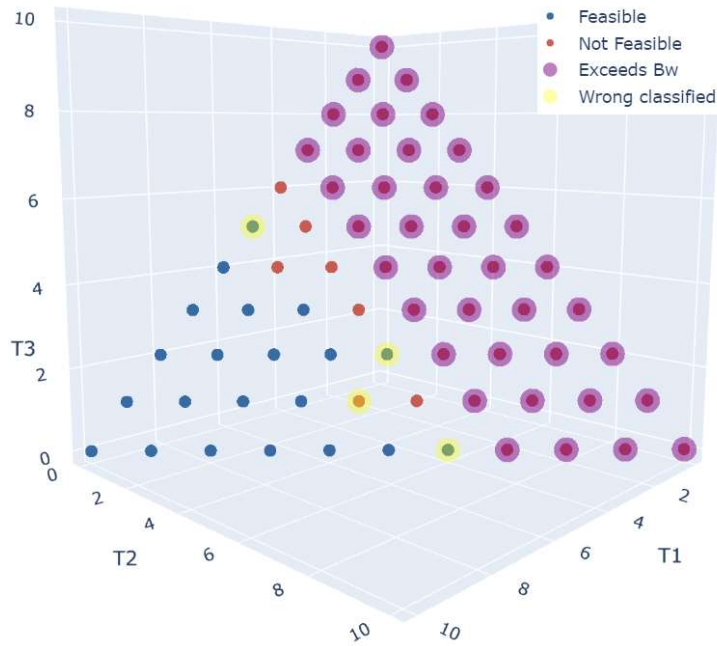
**Fig 4.4** 3D scatter plot of scenario 1 with wrong classified instances (SVM)

**Table 4.4** Wrong classified instances of SVM in Scenario 1

Instance Id	T1	T2	T3	Feasibility
1	0	1	9	False
2	0	6	4	True
3	0	10	0	True
4	1	3	6	False
5	2	6	2	True
6	3	6	1	False
7	2	8	0	True

### 4.2.2. Scenario 2

In this scenario, the application of the LOOCV identified SVM ( $C=4.1$ ,  $\gamma=0.01$ ) as the best estimator. The results achieved by this algorithm were the same as the K-NN model, see Fig. 4.4 where prediction results are shown.

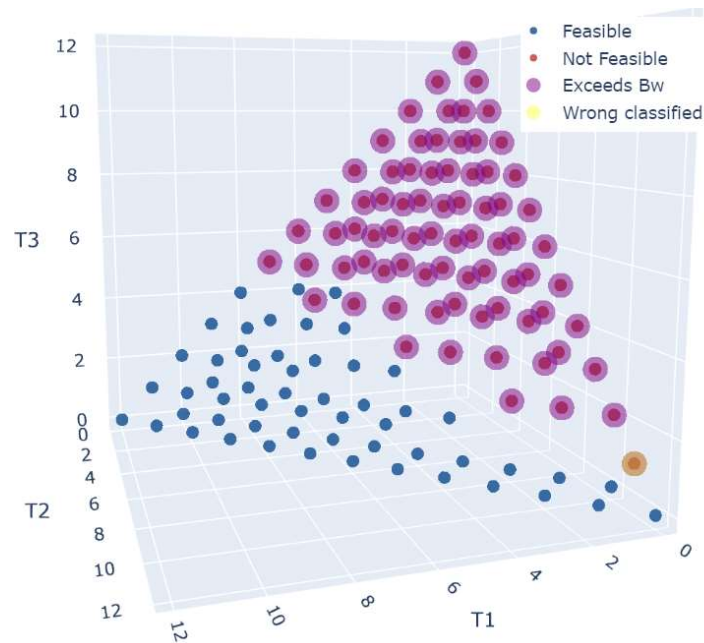


**Fig 4.5** 3D scatter plot of scenario 2 with wrong classified instances (SVM)

### 4.2.3. Scenario 3

In this scenario with data points belonging to different planes, the most optimal classifier is an SVM with a hyperparameter configuration of  $C=10.1$  and  $\gamma=0.1$ . This model with a 99% of accuracy, exhibits the same performance as the K-NN, what is expected since feasible and non-feasible zones are separated.

There is only one misclassified point which is located in the lower right margin of the dataset, see Fig 4.6. This misprediction suggests that the classifier may encounter issues when correctly classifying instances within that specific area.



**Fig 4.6** 3D scatter plot of scenario 3 with wrong classified instances (SVM)

### 4.3. Conclusion

In the first scenario, we found that SVM performed slightly better than K-NN when trained and applied to the implemented scenarios. However, it is important to highlight that these results are based on limited dataset therefore, we cannot obtain a conclusion determining the superiority of one model over the other.

The results of the models in scenario 3 showed a 99% accuracy, which is not a representative result for comparing the models due to the existence of a clear separation between feasible and non-feasible because of missing values in the dataset.

The current datasets used in this project have limitations in capturing variations present in real-world scenarios. Further exploration is needed to obtain reliable conclusions about a preferable model. This involves conducting experiments with a wider range of scenarios collecting more data to provide a comprehensive understanding of the models performance.



## CHAPTER 5. Hybrid Verification

Once a machine learning algorithm has been trained, it can produce immediate results for any given instance unlike other solutions such as the ILP implementation used in this project, which may not produce results in certain cases due to the configured timeout.

Missing results are precisely where a machine learning algorithm can be used so we can prevent the execution of time-consuming solutions. Given that we still need to lean on these time-consuming algorithms, a hybrid verification strategy approach is proposed, combining the use of machine learning algorithms and the ILP implementation.

In this section, we also intend to demonstrate the capability of accurately predicting unknown combinations of data in different planes, using a machine learning algorithm. For that purpose, we applied the mentioned strategy over the scenario 3 and then analysed the results. Given that the scenario 3 is a controlled scenario in which we tuned the “timeout” parameter, we can afford computing the complete scenario 3 to serve as a base truth, with which comparing the results obtained from the application of this hybrid verification method.

In the following sections, the strategy is explained, and the results obtained from its application on scenario 3 are analysed.

### 5.1. Strategy Application

This hybrid verification strategy consists of minimizing the use of the ILP implementation, by only executing it to obtain results from those combinations that may be more significant to improve the accuracy of a model predictor.

The challenge for the ML algorithm is to accurately predict missing data points or predicting those located in a zone where feasible and non-feasible combinations are not well distinguished. The data points within this zone are the most informative combinations in the whole dataset and may be the ones that optimizes the performance of the classifier.

Therefore, this strategy proposes to avoid the application of the ILP algorithm on data points where the feasibility is already evident to the predictor. Identifying the most informative data points enable us to optimize the scheduling verification process drastically reducing the total computation time required.

A first iteration of the strategy consists of:

1. Obtain results with the ILP implementation on distributions require less time to compute.
2. Train the models with existing data.
3. Analyse results.
4. Select possible crucial combinations to compute the ILP implementation.

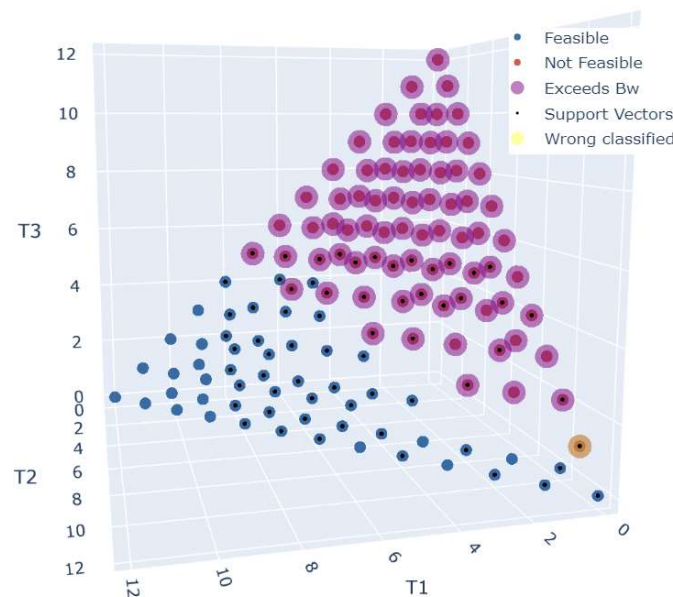
## 5.2. Results

In this section, the results obtained from the application of the aforementioned strategy over a SVM are presented.

The initial training data considered to apply this strategy appertains to the dataset of scenario 3. This dataset has a total of 131 instances, from which 55 are feasible combinations and the remaining are unfeasible.

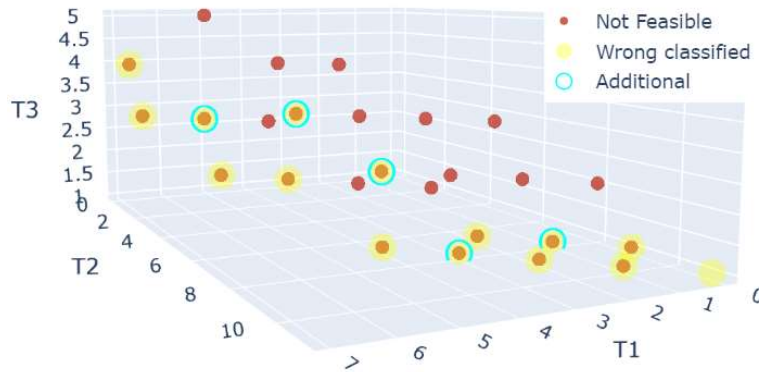
After training an SVM model with this dataset (4.2.3), we obtained a SVM model with a 99% accuracy, indicating that the predictor has no difficulty in classifying the provided data, see Fig. 5.1. As mentioned before, this high accuracy was expected due to the clear separation between the classes in the dataset.

If we compare these results with the complete dataset, we observe that the model achieves an error rate of approximately 9.55% (Fig. 5.2), which means that it incorrectly predicts 15 out of the 157 instances in the complete dataset.



**Fig 5.1** 3D scatter plot of scenario 3 with support vectors and a wrong classified combination





**Fig 5.2** 3D scatter plot of missing combinations. Includes wrong classified combinations and the additional combinations selected to retrain the model.

**Table 5.1** Additional combinations

Instance Id	T1	T2	T3	Feasibility
1	3	5	2	False
2	4	3	3	False
3	3	8	1	False
4	1	8	1	False
5	6	3	3	False

After this initial result, the next step is to retrain the model incorporating additional combinations to the initial training dataset, see additional combinations in Table 5.1.

In Fig 5.1 it is possible to observe that support vectors are located near the feasibility boundary. These support vectors were selected by “GridSearch” so that the margin between the class labels is maximized while minimizing the classification errors, which results in the most optimal decision boundary. Therefore, when incorporating new combinations to the dataset, it is crucial to prioritize those that are near to that decision boundary. Such combinations have a greater potential to improve the classifier performance.

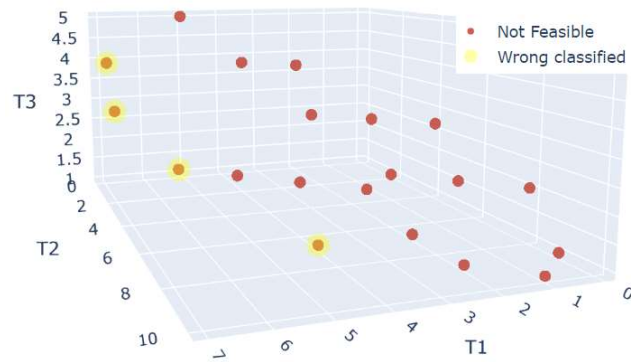
After retraining the classifier and comparing it with the complete dataset, we observed that it incorrectly predicted 4 combinations, see Table 5.2. This results in an error rate of 2.55%, what is significantly lower than the previous model.

Analyzing the incorrectly predicted combinations in Fig. 5.3, we observe that they are located in the left margin of the figure. Therefore, in a third iteration of this strategy, probably selecting combinations located within that zone suggests that the classifier would adjust its decision boundary improving its accuracy.

Therefore, through the addition of new combinations to the training dataset and observing the resulting increase in accuracy, we can affirm that the predictive capabilities of an SVM model can be greatly improved.

**Table 5.2** Wrong predicted combinations after retraining

Instance Id	T1	T2	T3	Feasibility
1	4	7	1	False
2	7	1	4	False
3	7	2	3	False
4	6	4	2	False



**Fig 5.3** 3D scatter plot of missing and *wrong classified combinations*

## CHAPTER 6. Conclusions and Next Steps

### 6.1. Conclusions

The present work has been a first contribution to provide trained Machine Learning models as an alternative to determine whether, a certain resource allocation within a Time Sensitive Network meets a set of timing constraints following [2].

Despite the complexity related to scheduling verification, it has been possible to generate multiple datasets from the Integer Linear Programming implementation of [27] which has been crucial to the proper development of this project. To obtain the dataset we applied the pertinent code modifications to adapt the results to the needs of this project.

For every dataset scenario, we have implemented the K-NN and SVM models and we have tested them, analysing, and graphically exposing their predictive performance when adjusting its hyperparameters.

To obtain a comparison that is not biased by the training data provided to both algorithms, we have applied the Leave One Out Cross Validation method so that, multiple models of each of the algorithms have been generated, ending up with the two best models for each machine learning algorithm.

With all this, it is shown that no algorithm stands out over the other, since the obtained results are based on limited dataset with clear separated instances in the last 2 scenarios. Therefore, we cannot obtain a conclusion determining which one is the most performant.

Taking advantage of the fast response of machine learning algorithms to quickly obtain results, we propose a hybrid verification strategy, in which the models predict every flow combination and the ILP implementation is only used to verify those whose prediction is more likely to fail.

The results demonstrate that the model positively responds to the inclusion of more relevant data during the training process, highlighting the importance of training a machine learning model with a representative dataset.

Therefore, we can conclude that, although the number of scenarios generated to test machine learning models are limited and the datasets are small, the preliminary findings suggest that machine learning models can be a promising tool to optimize the scheduling verification process.

## 6.2. Next Steps

Although, the applied models performed as planned, we would have preferred to carry out this project on with more realistic scenarios with a greater amount of data. For this, it is necessary, to improve the efficiency of the ILP implementation or to go deeper in other types of algorithms such as Satisfiability Modulo Theories (SMT) [20].

Regarding the application of prediction models, it would be interesting to also implement algorithms such as LDA or Naive Bayes, to compare the performance between generative and discriminative classification algorithms.

Finally, assuming that it is possible to obtain a greater amount of data, a fascinating next step would be to propose a new line of research around reinforcement learning using neuronal networks for scheduling verification.

## References

- [1] IEEE802, Time-Sensitive Networking (TSN) Task Group [Online] Available at: <https://1.ieee802.org/tsn/>
- [2] N. Navet, T. L. Mai, J. Migge, “Using Machine Learning to Speed Up the Design Space Exploration of Ethernet TSN Networks”, University of Luxembourg, 12-13 (2019)
- [3] Wikipedia, Time-division multiple access. [Online], Available at: [https://en.m.wikipedia.org/wiki/Time-division\\_multiple\\_access](https://en.m.wikipedia.org/wiki/Time-division_multiple_access)
- [4] Time-Sensitive Networking. [Online] Available at: [https://en.wikipedia.org/wiki/Time-Sensitive\\_Networking](https://en.wikipedia.org/wiki/Time-Sensitive_Networking)
- [5] Pyomo. [Online] Available at: <http://www.pyomo.org/>
- [6] CBC Solver. [Online] Available at: <https://coin-or.github.io/>
- [7] Gurobi Solver. [Online] Available at: <https://www.gurobi.com/>
- [8] Nicolas, N., Tieu. Long, M. and Jörn, M. “A Hybrid Machine Learning Schedulability Analysis Method for the Verification of TSN Networks”, *IEEE International Workshop on Factory Communication Systems*, 1-2 (2019)
- [9] José, R. in “Linux Out of Memory Killer” (2017). Available at: <https://neo4j.com/developer/kb/linux-out-of-memory-killer/>
- [10] Conda Package. [Online], Available at: <https://docs.conda.io/>
- [11] Psutil Library. [Online], Available at: <https://github.com/giampaolo/psutil>
- [12] Brownlee, J. “Feature Selection Methods” in *Applied Predictive Modeling.*, pp 499 (2013) Available at: <https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/>
- [13] Chao, S. R. Dougherty, E. The peaking phenomenon in the presence of feature-selection, *DBLP Computer Science*, 29 (11), 1672-1674 (2008) Available at: [http://gsp.tamu.edu/wp-content/uploads/sites/71/2017/01/pap\\_PRL\\_Peaking.pdf](http://gsp.tamu.edu/wp-content/uploads/sites/71/2017/01/pap_PRL_Peaking.pdf)

- [14] Kaiser Rule. [Online], Available at:  
[https://docs.displayr.com/wiki/Kaiser\\_Rule](https://docs.displayr.com/wiki/Kaiser_Rule)
- [15] Andrew Y. Ng. and Jordan, M. I. "On Discriminative vs. Generative classifiers: A comparison of logistic regression and naïve Bayes", *AI-Stanford*, 7-8 (2001). Available at:  
<http://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf>
- [16] Thirumuruganathan, A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm [Online] Available at :  
<https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/>
- [17] Machine Learning Sharing-KNN Algorithms and Numpy Implementation. [Online]. Available at :  
<https://developpaper.com/machine-learning-sharing-knn-algorithms-and-numpy-implementation/>
- [18] Scikit-Learn, Gaussian Process Kernels RBF. [Online], Available at:  
[https://scikit-learn.org/stable/modules/generated/sklearn.gaussian\\_process.kernels.RBF.html](https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html)
- [19] Scikit-Learn, GridSearchCV. [Online], Available at :  
[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.GridSearchCV.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)
- [20] Wikipedia, Satisfiability modulo theories. [Online], Available at:  
[https://en.wikipedia.org/wiki/Satisfiability\\_modulo\\_theories](https://en.wikipedia.org/wiki/Satisfiability_modulo_theories)
- [21] Kendarps, Why do you need to scale data in KNN [Online] Available at:  
<https://stats.stackexchange.com/questions/287425/why-do-you-need-to-scale-data-in-knn>
- [22] IEEE 802.1Q Frame Format. [Online] Available at:  
<https://support.huawei.com/enterprise/en/doc/EDOC1100088104>
- [23] Wikipedia, Time-division multiple access. [Online], Available at:  
[https://en.m.wikipedia.org/wiki/Time-division\\_multiple\\_access](https://en.m.wikipedia.org/wiki/Time-division_multiple_access)
- [24] StatisticalHelp, Gini Coefficient of Inequality. [Online] Available at :  
[https://www.statsdirect.com/help/default.htm#nonparametric\\_methods/gini.htm](https://www.statsdirect.com/help/default.htm#nonparametric_methods/gini.htm)

- [25] Sang Gyu Kwak, Jong Hae Kim. "Central Limit Theorem: The Cornerstone of modern statistics", *National Library of Medicine*, 2(70), 2-4 (2017) Available at: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5370305/>
- [26] Hla Stanford, Euclidean Distance in 'N'-Dimensional Space. [Online] Available at: [https://hlab.stanford.edu/brian/euclidean\\_distance\\_in.html](https://hlab.stanford.edu/brian/euclidean_distance_in.html)
- [27] Orozco Urrutia, G.D., "A Microservices-based Control Plane for Time Sensitive Networks", *Master's Thesis, UPC, Castelldefels, Spain*. [In progress]
- [28] G.Sierra, "Algoritmos de gestión de tráfico: Leaky Bucket, Token Bucket y Virtual", *Tecnura*, 15 (29), 78-80 (2011). Available at: [https://www.researchgate.net/publication/277261423\\_Algoritmos\\_de\\_gestion\\_de\\_trafico\\_Leaky\\_Bucket\\_Token\\_Bucket\\_y\\_Virtual](https://www.researchgate.net/publication/277261423_Algoritmos_de_gestion_de_trafico_Leaky_Bucket_Token_Bucket_y_Virtual)
- [29] Castaño Cid, J.O., "Proves amb equipament Time-Sensitive Networking (TSN)", *UPC Grade Thesis*, 3-4 (2018) Available at: <https://upcommons.upc.edu/handle/2117/121567>
- [30] Wikipedia, Time-division multiple access. [Online] Available at: [https://en.m.wikipedia.org/wiki/Time-division\\_multiple\\_access](https://en.m.wikipedia.org/wiki/Time-division_multiple_access)
- [31] Qing Li, Dong Li, Xi Jin, Qizhao Wang, Peng Zeng. "A simple and efficient Time-Sensitive Networking Traffic Scheduling Method for Industrial Scenarios", *MDPI Electronics*, 2-11 (2020)
- [32] Fletcher T. "Support Vector Machines Explained". [Online]. *UCL Computer Science*, 2-5 (2008)
- [33] Lander Raagaard M., Pop P., "Optimization algorithms for the scheduling of IEEE 802.1 Time-Sensitive Networking (TSN)", *DTU Technical Report*, 35-40 (2017)
- [34] LoBello, L. and Steiner, W. "A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems". *Proc. IEEE*, 107, 1094–1120 (2019)
- [35] [15] Andrew Y. Ng. and Jordan, M. I. "A comparison of logistic regression and naive Bayes". *StanFord University*, (2002)
- [36] Sanjukta, D. and Kashvi, T. "A Brief Review of Nearest Neighbor Algorithm for Learning and Classification". *ICCS* (2019)

- [37] *Scikit-Learn*, “Tuning the hyper-parameters of an estimator.” [Online]: [https://scikit-learn.org/stable/modules/grid\\_search.html](https://scikit-learn.org/stable/modules/grid_search.html)
- [38] *Scikit-Learn*, “Leave One Out Cross Validation.” [Online]: [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.LeaveOneOut.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeaveOneOut.html)
- [39] *Baeldung*, “Cross Validation: K-Fold vs Leave-One-Out.” [Online]: <https://www.baeldung.com/cs/cross-validation-k-fold-loo>



