Using Machine Learning to Speed Up the Design Space Exploration of Ethernet TSN networks

Nicolas NAVET, University of Luxembourg, Luxembourg
Tieu Long MAI, University of Luxembourg, Luxembourg
Jörn MIGGE, RealTime-at-Work, France

Version: 2.0, dated 20/05/2019 (first version dated 29/01/2019).

Abstract: In this work, we ask if Machine Learning (ML) can provide a viable alternative to conventional schedulability analysis to determine whether a real-time Ethernet network meets a set of timing constraints. Otherwise said, can an algorithm learn what makes it difficult for a system to be feasible and predict whether a configuration will be feasible without executing a schedulability analysis? In this study, we apply standard supervised and unsupervised ML techniques and compare them, in terms of their accuracy and running times, with precise and approximate schedulability analyses in Network-Calculus. We show that ML is efficient at predicting the feasibility of realistic TSN networks, above 93% of accuracy for the best ML algorithm tested whatever the exact TSN scheduling mechanism, and offer new trade-offs between accuracy and computation time that are especially interesting for design-space exploration algorithms.

Keywords: timing verification, machine learning, supervised learning, unsupervised learning, schedulability analysis, real-time systems, Time-Sensitive Networking (TSN), design-space exploration.

Contents

1 The design and configuration of TSN networks

1.1 TSN opens up a wealth of possibilities at the data link layer

Ethernet is becoming the prominent layer 2 protocol for high-speed communication in real-time systems. Over time it appears that it will be replacing most other wired high-speed local area network technologies, be it in the industrial and automotive domains, or even for telecom infrastructures. These applications domains often have strong Quality of Service (QoS) requirements that include performance (e.g., latency, synchronization and throughput requirements), reliability (e.g., spatial redundancy) and security (e.g., integrity), which cannot be met by the standard Ethernet technology.

The IEEE802.1 TSN TG (Time Sensitive Networking Technical Group), started in 2012, is a follow-up initiative to the IEEE AVB (Audio Video Bridging) TG meant to develop the technologies to address these QoS requirements. Companies, mainly from the industry and telecom domains, as well as hardware vendors, are driving the works in IEEE802.1 TSN TG with respect to their technical and business roadmaps. TSN TG has developed more than 10 individual standards, which, after their adoption as amendments to the current IEEE802.1Q specification, are integrated into the newest edition of IEEE802.1Q ([IEEE802.1Q-2018] at the time of writing). The reader interested in a survey of the TSN standards related to low-latency communication, and the ongoing works within TSN TG can refer to [Na18].

TSN protocols offer a wealth of possibilities to the network architect. For instance, temporal QoS can be implemented through *the use of priorities*, as offered by 802.1Q [IEEE802.1Q-2018], *traffic shaping* with the Credit-Based Shaper (CBS) of [IEEE802.1Qav] or *time-triggered transmissions* with the Time-Aware Shaper (TAS) of [IEEE802.1Qbv]. Other powerful mechanisms include *frame pre-emption* defined in [802.1Qbu], and flexible *per-stream shaper* as currently being worked on in [IEEE802.1Qcr]. Importantly, these protocols and mechanisms can be used in a combined manner, for instance TSN/TAS for the highest-priority traffic, then AVB CBS at the two immediate lower priority levels followed by a number of best-effort traffic classes each assigned to a distinct priority level (see [MiViNaBo18b] for a realistic automotive case-study showcasing the joint use of several TSN protocols). Besides temporal QoS, TSN provides support for other requirements like dependability, e.g. through frame replication [IEEE802.1CB] and per-stream filtering [IEEE802.1Qci], which further increases the complexity of the design problem.

1.2 The complexity of configuring TSN networks

All the technical possibilities offered by TSN provide a lot of flexibility and power to the designer to meet the requirements of an application and use the bandwidth in an efficient manner. It however makes the configuration step much harder for several reasons: the largely increased size of the space of possible configurations and the non-trivial interactions between the protocols. The sole problem of configuring time-triggered transmissions with TAS (*i.e.*, defining the Gate Control List schedules on all network devices) is known to be intractable (see [SOCrSt18]) in the sense that there is no polynomial-time algorithm to find an optimal solution. There are however efficient algorithms to solve this problem for the Qbv protocol [Ra17, SOCrSt18], as well as most other technology-specific configuration problems like setting the traffic shaper parameters [IEEE,NaMiViBo18] or choosing the priorities for a set of streams [HaScFr14]. In our view, the global problem of selecting the suitable TSN protocols for the application at hand, and configuring them in an efficient, if not optimal manner remains largely unaddressed. In the industrial practice, these design choices cannot be made on technical grounds only, they have to integrate other concerns like costs, time-to-market and the risk of adopting a new technology. This means that, in a specific industrial context, some technical solutions have to be ruled out because they would not allow meeting certain constraints like typically time-to-market.

1.3 Design Space Exploration for automating the design of TSN networks

Design-Space Exploration (DSE), that is design decisions based on the systematic exploration of the search space, encompasses a set of techniques used in various domains like electronic circuit design or 3D modelling that are increasingly leveraging the possibilities brought by ML, see for instance [EEN18].

The complexity of designing and configuring TSN networks calls for DSE algorithms to assist in the selection and configuration of TSN protocols. An algorithm that serves this purpose is *ZeroConfig-TSN* (ZCT), presented in [MiViNaBo18a,NaViMi18] and implemented in the RTaW-Pegase tool [Peg18]. Its overall goal is to create the most efficient solutions given communication requirements and set of TSN mechanisms chosen by the designer. As depicted in Figure 1, it works by iteratively creating candidate solutions using the allowed set of TSN mechanisms, configuring the mechanisms using dedicated algorithms, and then assess by schedulability analysis or simulation whether the requirements are met by the candidate solutions.

For networks, even with optimized implementations of simulator and schedulability analysis, this last step is compute intensive and drastically limits the size of the search space that can be explored. For instance, assessing the feasibility of the TSN network used in the experiments of this study with 500 flows scheduled with three priority levels requires an average 470ms computation time (Intel I7-8700 3.2Ghz) per configuration, resulting in about 131hours for 10^6 candidate solutions. To mitigate this problem, this work studies whether ML algorithms can be a faster alternative to conventional schedulability analysis to determine whether a real-time TSN network meets a set of timing constraints. A drastic speed-up in feasibility testing thanks to ML would facilitate the adoption of DSE algorithms in the design of E/E architectures.

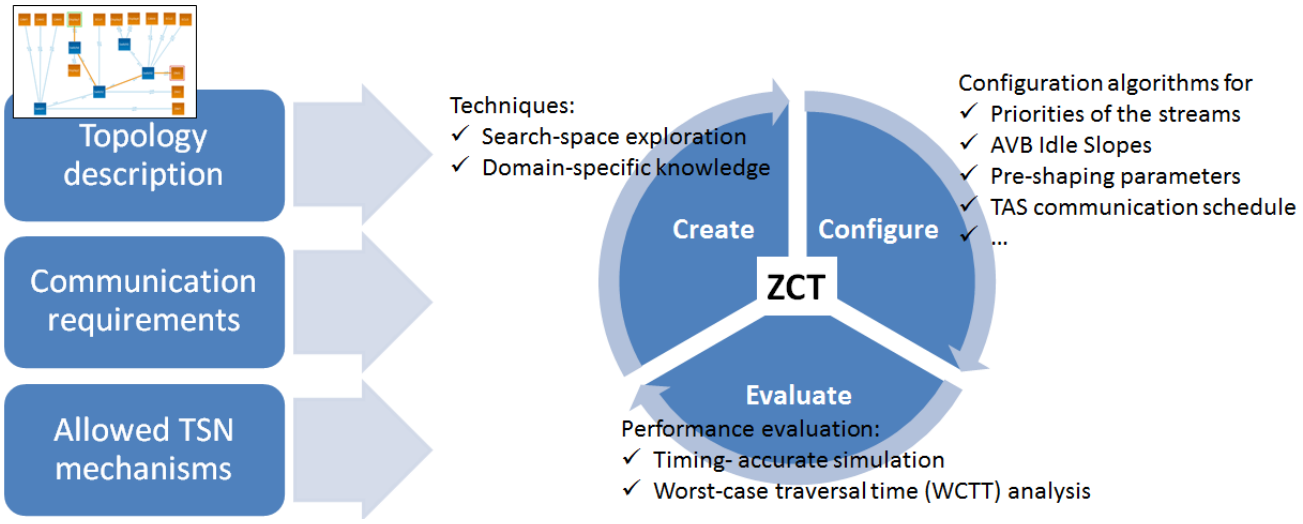

Figure 1: Illustration of design-space exploration for TSN networks with Zero-Config TSN (figure from [MiViNaBo18a]).

We believe high-level functions like ZCT will increasingly be used be it alone because they provide solutions to problems that cannot be solve manually as efficiently in the same amount of time. In our view, it is key that these algorithms incorporate as much domain-specific knowledge as possible, as, in our experiments with evolutionary algorithms, computing power alone does not lead to efficient solutions. With such techniques belonging to the realm of generative design, the designer has to accept to not exactly know how the solution is created. What is mandatory though is that the designer is provided with proofs that the retained solution meets the requirements.

1.4 Verification of timing constraints

The two main model-based approaches to assess whether a real-time system meets its temporal constraints are:

- *Schedulability analysis*, also called feasibility analysis, worst-case analysis, response time analysis or worst-case traversal time analysis in the case of networks, is a mathematical model of the system used to derive upper bounds on the performance metrics (communication latencies, buffer usage, etc). Typically, the model will focus on a subset of the possible trajectories of the systems that are proven to contain the most pessimistic scenarios.
- *Timing-accurate simulation*: a system is characterized by a certain state Sn and a set of rules to change from state n to n+1: Sn+1 = F(Sn+1) are defined in the simulation model. The evolution of the system depends on this set of evolution rules and the sequence of values provided by the random generator. The model can abstract everything that is not relevant from the timing point of view but must capture all activities having an impact on the performance metrics, like typically the waiting time of a packet in any network device.

Some temporal constraints of an application can be checked by schedulability analysis (e.g., deadline constraints for safety critical streams) while others should be verified by simulation (e.g., TCP throughput for data upload [HaKi16,NaMi18]). Indeed, simulation is the only solution when no schedulability analysis exists, e.g. for application-specific traffic patterns (diagnostic data) and complex high-level protocols as TCP. Simulation is also required for performance metrics like throughput that cannot be evaluated by conventional schedulability analysis. Simulation is more versatile than schedulability analysis in the sense that it suited for complex systems as well, and any quantities the designer is interested in can be collected during simulation runs.
The main drawback of simulation is that, even if the coverage of the verification can be adjusted to the requirements (see [Na14]), it only provides statistical guarantees and not firm guarantees. Simulation is thus better suited to evaluate performance metrics pertaining to the less critical streams. Interestingly, in today's complex automotive communication architectures, both schedulability analysis and simulation have usually to be employed for the same application, making up what can be referred to as "*mixed-criticality timing analysis*".

1.5 Contributions of the study

This work explores the extent to which supervised and unsupervised ML techniques can be used to determine whether a real-time Ethernet configuration is schedulable or not, and quantifies the prediction accuracies and computation times that can be expected. Intentionally, to ensure the practicality of our proposals, we study these questions using standard ML techniques that can run on desktop computers with relatively small amounts of training data. Although ML has been applied to diverse related areas including performance evaluation (see Section 7 for a review of the state-of-the-art), this is to the best of our knowledge the first study to apply ML to determine the feasibility of a real-time system.

1.6 Organisation of the document

The remainder of this document is organised as follows. In Section 2, we introduce the TSN network model and define the design problem. Section 3 presents the schedulability analyses that will serve as benchmarks for ML algorithms. In Section 4 and Section 5 we apply respectively supervised and unsupervised learning algorithms to predict feasibility. Section 6 provides a recap of the results achieved and a comparison with the performances that can be obtained with conventional schedulability analysis. In Section 7, we give an overview of the applications of ML techniques in related domains. Finally, Section 8 provides first insights gained about the use of ML

techniques for timing analysis and identifies several possible improvements and research directions.

2 Ethernet TSN model and design problem

In this work, we consider that the network topology (layout, link data rates, etc) has been set as the well as the TSN protocols that the network devices must support. The supported TSN protocols determine the space of scheduling solutions that are feasible (e.g., FIFO, priority levels plus traffic shaping, etc). This is realistic with respect to industrial contexts, like the automotive and aeronautical domains, where most design choices pertaining to the topology of the networks and the technologies are made early in the design cycle at a time when the communication needs are not entirely known. Indeed, many functions become available later in the development cycle or are added at later evolutions of the platform.

2.1 TSN model

We consider a standard switched Ethernet network supporting unicast and multicast communications between a set of software components distributed over a number of stations. In the following, the terms "traffic flow" or "traffic stream" refers to the data sent to the receiver of an unicast transmission or the data sent to a certain receiver of a multicast transmission (*i.e.*, a multicast connection with *n* receivers creates *n* distinct traffic flows). A number of assumptions are placed:

- All packets, also called frames, of the same traffic flow are delivered over the same path: the routing is static as it is today the norm in critical systems,
- It is assumed that there are no transmission errors and no buffer overflows leading to packet loss. If the amount of memory in switches is known, the latter property can be checked with schedulability analysis as it returns both upper bounds on stream latencies and maximum memory usage at switch ports,
- Streams are either periodic, sporadic (*i.e.*, two successive frames become ready for transmission at least *x* ms apart) or sporadic with bursts (*i.e.*, two successive burst of *n* frames become ready for transmission at least *x* ms apart). The latter type of traffic corresponds for instance to video streams from cameras that cannot fit into a single Ethernet frame and must be segmented into several frames.
- The maximum size of the successive frames belonging to a stream is known, as required by schedulability analysis.
- The packet switching delay is assumed to be 1.3us at most. This value, which of course varies from one switch model to another, is in line with the typical latencies of modern switches

2.2 Traffic characteristics

The traffic is made up of three classes whose characteristics are summarized in Table 1. The characteristic of the streams and their proportion is the same as in [NaMi18], which in turn is inspired from the case-study provided by an automotive OEM in [MiViNaBo18b, NaViMiBo17].

Audio Streams	– 128 or 256 byte frames – periods: 1.25ms – deadline constraints either 5 or 10ms – proportion: 7/46

Video Streams	– ADAS + Vision streams – 30*1500byte frame each 33ms (30FPS camera for vision) – 15*1000byte frame each 33ms (30FPS camera for ADAS) – 10ms (ADAS) or 30ms (Vision) deadlines – proportion: 7/46
Command & Control (C&C)	– from 53 to 300 byte frames – periods from 5 to 80ms – deadlines equal to periods – proportion: 32/46

Table 1: Characteristics of the three types of traffic. The performance requirement is to meet deadline constraints. The frame sizes indicated are data payload only.

In the experiments, the number of streams varies but the proportion of each stream (indicated in Table 1) is a fixed parameter of the stream generation procedure chosen as in [NaMi18]. Each stream is either unicast or multicast with a probability 0.5. The number of receivers for a multicast stream is chosen at random between two and five. The sender and receiver(s) of a stream are set at random.

2.3 Network topology

The topology considered in this this study, intentionally chosen simple, is the same as in [NaMi18] and similar in terms of structure to the prototype Ethernet network developed by an automotive OEM a few years ago [NaSeMi16]. As shown in Figure 1, the network comprises two switches and height nodes. The data transmission rate is 100Mbps on all links except 1Gbps on the inter-switch link to avoid the severe bottleneck that can occur with such "dumbbell" topology.

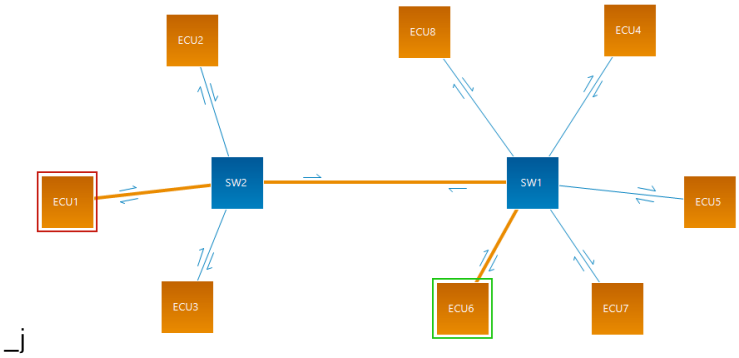

Figure 1: Topology of the prototype network used in the experiments. The unicast stream shown here goes from ECU1 to ECU6 (RTaW-Pegase screenshot).

Nb: experiments with the more complex network topology from [NaViMiBo17] are presented in [MaNaMi19b]. Although the exact results vary, the main takeaways from this report remain valid with the more complex topology.

2.4 Designing and configuring TSN networks

Besides eliciting the communication requirements and proposing a candidate topology, TSN design and configuration involves a number of additional sub-problems:

- Group traffic streams into traffic classes and set the relative priorities of the traffic classes: up to 8 priority levels are possible in TSN [IEEE802.1Q-2018],
- Select the QoS protocols for each traffic class: priorities alone, shaping with CBS, time-triggered transmission with exclusive bus access ("exclusive gating") with TAS, etc

- Configure each traffic class: parameters of CBS (*i.e.*, class measurement interval (CMI), values of the "idle slopes" per class per switch), Gate Control List (i.e. TT transmission schedule) for TAS in each switch, etc.
- If frame pre-emption is used [802.1Qbu], decide the subset of traffic classes that can be pre-empted by the rest of the traffic classes.

In the following, a *possible configuration*, or *candidate solution*, refers to a TSN network that has been fully configured as detailed above, while a *feasible configuration* is configuration that meets all the application's constraints.

The number of possible candidate solutions depends on the set of allowed mechanisms. In this study, we consider the following solutions corresponding to distinct trade-offs in terms of their complexity and their ability to meet diverse timing constraints:

1. FIFO scheduling (*FIFO*): all streams belong to the same traffic class and thus have the same level of priority. This is the simplest possible solution.
2. Priority with manual classification (*Manual*): the streams are grouped into the three classes shown in Table 1 and their priority is as follows: C&C class above audio class above video class. This can be seen as the baseline solution that a designer would try based on the criticality of the streams.
3. "Concise priorities" with height priority levels (*CP8*): "concise priorities" is the name of the priority assignment algorithm in RTaW-Pegase, which relies on the same principles as the Optimal Priority Assignment[1] algorithm for mono-processor system [Aud01] that has been shown to be optimal as well with an analysis developed with "the trajectory approach" for the transmission of periodic/sporadic streams in switched Ethernet networks [HaScFr14]. With concise priorities, unlike with manual classification, flows of the same type can be assigned to different traffic classes and more than three priority levels will be used if required by the timing constraints.
4. Manual classification with "pre-shaping" (*Preshaping*): we re-use the manual classification with three priority levels but apply a traffic shaping strategy called "pre-shaping" in transmission to all video-streams. This traffic shaping strategy, combines standard static priority scheduling with traffic shaping introduced by inserting idle times, pauses, between the times at which the successive frames of a segmented message (e.g. camera frame) are enqueued for transmission. All the other characteristics of the traffic remain unchanged. In [NaMiViBo18], this strategy has been shown to perform as well as CBS without the need for dedicated hardware. In our context, pre-shaping will benefit to video streams as it will reduce the interference brought by same priority video streams by intertwining the transmissions of the different video streams sharing the same links. The principles of the algorithm used to set the idle times, available under the name of "Presh" algorithm in RTaW-Pegase, are described in [NaMiViBo18].

This set of scheduling solutions has been selected to include two main QoS strategies, namely static priority scheduling and traffic shaping. Importantly, the lower bounds obtained in [BaScFr12,BoNaFu12] suggest that the existing schedulability analyses for these relatively simple scheduling mechanisms are accurate in terms of the distance between the computed upper bounds and the true worst-case latency.

[1] "Concise Priorities" and OPA differ by how unfeasible configurations are handled: concise priorities returns a priority assignment that tries to minimize the number of flows not meeting their constraints, while this is not part of standard OPA. This difference however does not play a role in this study. Whether OPA remains optimal in the TSN context with other analyses and other formalisms than the trajectorial approach such as Network-Calculus, as used in the paper, or Event-stream is to the best of our knowledge an open question. Experiments not shown here suggest that OPA is anyway efficient at finding feasible priority allocations with the schedulability analyses used in this report.

3 Verification with schedulability analysis

Worst-case traversal time (WCTT) analysis, also referred to as schedulability analysis, is the technique that is the best suited to provide the firm guarantees on latencies, jitters and buffer utilization needed by critical streams. There are different mathematical formalisms to develop WCTT analysis: Network-Calculus [LBTh01] or NC for short, Real-Time Calculus [ThChNa00], which is a specialization of NC for real-time systems, but also the trajectorial approach [LiGe16] and the event-stream approach as used for TSN in [ThErDi15]. Network-Calculus has probably been the dominant approach for the performance evaluation of safety critical embedded networks, one reason being that it has been used in aeronautics certification for the last 15 years, but recent results show that several of these formalisms can be unified into a larger theory [BoRo16].

3.1 A spectrum of schedulability analyses

Schedulability analyses, except in simple scheduling models like fixed-priority non-preemptive scheduling on CAN buses [Dav07], are pessimistic in the sense that they provide bounds on the quantity of interest instead of the exact values. The exact degree of pessimism (*i.e.*, tightness of the bounds) is hard to quantify, or even to foresee, although in some cases we can estimate it by comparison with lower bounds as done in [BaScFr12] and [BoNaFu12] (the latter study relies on the lower bounds from [BaScFr10]). It can be useful to develop several analyses for the same system, each offering a specific trade-off with respect to accuracy, computation time (e.g., linear or exponential time) and development time. Typically, as further discussed in [BoNaFu12], a network-calculus analysis can be characterized by three main attributes:

1. the internal representation of numbers (e.g., floating point or fractional representation),
2. the class of mathematical functions on which the computations are done (e.g. simple Increasing Convex or Concave functions (ICC) or more accurate Ultimately Pseudo Periodic functions (UPP), see [BoTh07]),
3. how input streams are modeled (e.g., stair-case or linear work arrival functions).

The more approximate the analysis, the faster it is for a given system as less computation is needed. Typically, not considering stair-case arrival functions but linear functions drastically reduces the number of computations needed at the expense of accuracy. For example in [BoNaFu12] the authors, using an earlier version of the RTaW-Pegase tool[2], report for AFDX networks a difference of computation time of approximately a factor 10 for the two analyses at the opposite ends of the spectrum of analyses in terms of complexity. Large differences in computation time between precise and approximate analyses exist for CAN networks as well as shown in [Fra19].

In the following, we will use in the experiments two analyses for static-priority scheduling in TSN offering distinct trade-offs between computation time and accuracy:

- The *approximate analysis*: ICC function class / fractional number representation / token bucket stream model. For this approximate analysis, WCTT computation execute in a time that is linear with the number of streams.
- The *precise analysis*: UPP function class / fractional number representation / stair-case stream model. For this accurate analysis, WCTT computations execute in a time that depends on the least common multiple (LCM) of the frame periods, which can lead to an exponential computation time if periods are coprime (i.e., periods do not share any other positive divider other than 1).

These two analyses, available in RTaW-Pegase, are based on results published in [BoTh07, BoMiNa11,Que12]. As the use of floating point numbers may give rise to numerical accuracy

[2] See https://www.realtimeatwork.com/software/rtaw-pegase/

problems, possibly leading to configurations deemed feasible while they are not, the two analyses implement rational number computation.

3.2 Feasibility: false positive and false negative

Whatever the schedulability analysis used, it should be safe in the sense that it actually provides worst-case results or at least upper-bounds on the metrics considered (*i.e.*, latencies, jitters and buffer usage). A schedulability analysis must thus rule out the possibility of *false positives*: configurations that are deemed feasible while they are not. On the other hand, the proportion of *false negatives* (*i.e.*, configurations deemed non-feasible while they actually are) directly depends on the tightness/accuracy of the analysis.
In statistical terms, false positives, called type 1 error, determine *the significance of the test*, while false negatives, called type 2 error, determines *the power of the test*. A sound schedulability analysis ensures no type 1 error but prediction of the feasibility using machine learning techniques, as investigated in this work, will lead to a certain amount of type 1 errors. This is why, in a design-space exploration workflow, we think that solutions deemed feasible by machine learning (or any other technique possibly leading to false positives) and retained to be part of the final set of solutions proposed to the designer, must undergo a conventional schedulability analysis to ensure their actual feasibility. However schedulability analysis will only be performed for the final set of solutions and not for each of the candidate solution which has been created during the exploration of the search space.

3.3 Schedulability analysis execution times

Among the four scheduling mechanisms presented in §2.4, FIFO and Manual do not entail any further configuration before a schedulability analysis can be executed. On the other hand, CP8 iteratively explores the search space to find a feasible priority allocation. Similarly, Preshaping tries in an iterative manner to find the shaping parameters leading to the optimal solution in terms of spreading out the load over time [NaMiViBo18]. In both cases, the time needed to assess the feasibility of a configuration includes computation that is not purely related to schedulability analysis, and both CP8 and Preshaping require the execution of several, usually many schedulability analyses each of the same complexity as the one for Manual scheduling (i.e., static priority scheduling). In order to not account for the time it takes to configure a policy, we provide hereafter the running times of the schedulability analysis for Manual scheduling.
Figure 1 shows the boxplots of the execution times for both analyses with RTaW-Pegase V3.3.6 on a single CPU core of a 3.2Ghz Intel I7-8700 for a varying number of streams. Four hundred random configurations with Manual scheduling are tested for feasibility for any number of streams shown in the plot. The first observation is that the execution times are small: 471ms on average with the precise analysis for configurations with 500 streams versus 13ms with the approximate analysis. This comes from both the efficiency of the implementations, which have been used and tuned for over 10 years, and, importantly, from the reasonnable value of the Least Common Multiple (LCM) of the frames' periods (160 seconds here with the 30 FPS cameras modeled as sending an image every 32ms instead of 33.33ms as typically done).. A second observation is that the execution times increase linearly with the size of the problem. In terms of the average execution times, the approximate analysis is much faster than the precise analysis with a speedup factor ranging from 34 to 46.

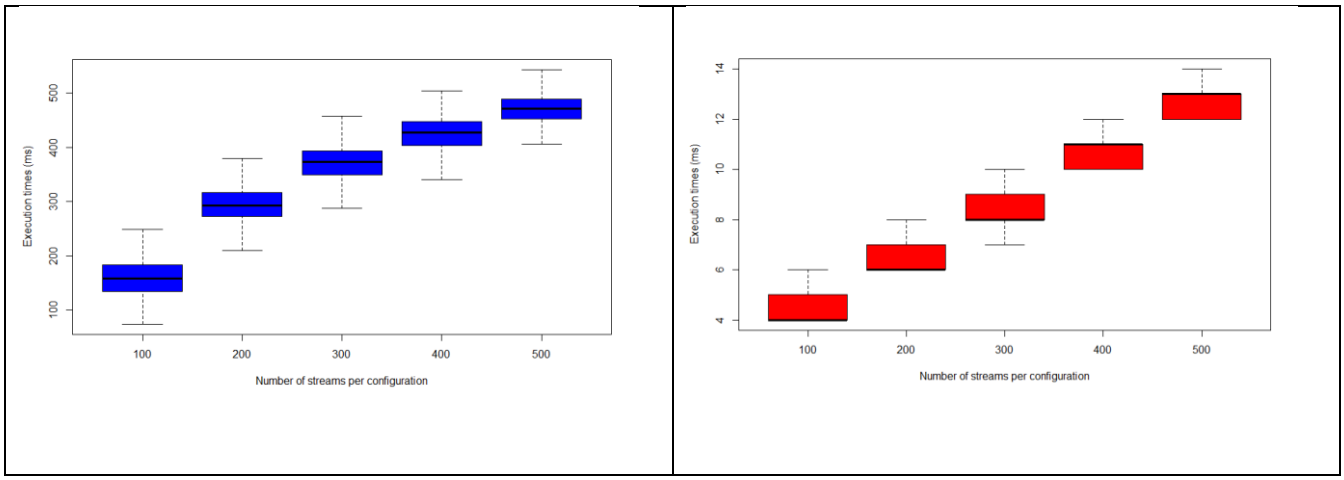

Figure 2: Boxplots ("whiskers" plots) of the computation times (ms) for the precise (blue plot) and approximate schedulability analyses (red plot) for a number of streams ranging from 100 to 500. The computation is performed with a single thread for all experiments. Each boxplot is created from samples of size 400. The bodies of the boxes are delimited by the 25% quantile (Q1) and the 75% quantile (Q3), while the upper, respectively lower "whisker", extends to 1.5 * (Q3-Q1) above, respectively below Q1 and Q3. Outliers, here the points beyond the whiskers, are not shown in the plots.

Table 2 shows the average execution times as well as the associated margins of errors (*i.e.*, half the width of the confidence interval expressed in percentage) that quantify the uncertainty due to the fact our measurements are not exhaustive, they cannot, but only consists of 400 runs for a given number of flows. As the sequences of execution times, whatever the analysis and the number of flows, do not follow a normal distribution (*i.e.*, the Jarque-Bera normality test is rejected at the 1% significance level), the standard formula to estimate the margins of errors cannot be applied here. As classically done in such situation, we used bootstrap simulations to estimate the margin of errors with the number of simulations set as suggested in [LB10]. The small values observed for the margins of error (less than 2.08%) suggest that in our context the number of measurements is sufficiently large.

	Precise analysis		Approximate analysis	
Number of flows per configuration	Average execution times	Margin of error	Average execution times	Margin of error
100	159.45ms	2.08%	4.61ms	1.60%
200	292.32ms	1.07%	6.35ms	0.75%
300	371.11ms	0.80%	8.45ms	0.60%
400	425.49ms	0.65%	10.56ms	0.48%
500	470.63ms	0.57%	12.79ms	0.41%

Table 2: Average execution times and corresponding margins of errors for the precise and the approximate analysis of the Manual scheduling solution. The first 10 and last 10 measurements have been discarded.

Let us consider a design space exploration algorithm that would evaluate 1000,000 configurations with 500 flows over the course of its execution, the approximate schedulability analysis would take less than 4 hours to execute on a single core versus about 131 hours for the precise analysis. The motivation of this work is to investigate whether machine learning techniques could allow drastically reduce this computation time, enabling thus a much more extensive exploration of the search space.

It should be noted that RTaW-Pegase offers multi-threaded versions of the two schedulability analyses used in this study with a speed-up that is close to linear with the number of CPU cores. To get a more precise comparison, we decided in this Section to not show measurements with multi-threading turned on, as the results depend on several additional factors, like the efficiency of the OS multi-threading services and the hardware to support them, the extent to which it is possible to parallelize a given analysis, etc. It should be pointed out that besides parallelising the schedulability analysis for a given configuration, it is also of course possible to execute in parallel schedulability analyses for several distinct configurations.

4 Predicting feasibility with supervised learning

In this section, we apply a supervised ML algorithm, namely k-Nearest Neighbours (k-NN), that makes use of a training set with labels to predict whether unlabelled configurations are feasible or not. k-Nearest Neighbours is a simple though powerful machine learning algorithm, described in standard ML textbooks like [HaTiFri09], that classifies unlabelled data points by placing them in the same category as their "nearest" neighbours, that is the closest data points in the feature space.

4.1 Supervised learning applied to network classification

Supervised learning is a class of ML algorithms that learn from of a training set in order to predict the value or the label of unseen data. The data in the training set are classified into categories: they are given "labels". On the other hand, algorithms belonging to unsupervised learning, as experimented in Section 5, tackle the classification problem without relying on a training set with labelled data.

In this study, as illustrated in Figure 3, the training set is a collection of TSN configurations that vary in terms of their number of flows and the parameters of each flow. A configuration in the training set will be labelled as *feasible* or *non-feasible*, depending on the results of the schedulability analysis: a configuration is feasible if and only if all latency constraints are met. As explained in §3.1, it should be noted that different schedulability analyses may return different conclusions depending on their accuracy. To label the training set, we use the accurate schedulability analysis, which minimizes the number of "false negatives". A configuration in the training set is characterized by a set of "features", that is a set of properties or domain-specific attributes that summarize this configuration. Those features will be the inputs of the ML algorithm, and choosing the right features plays a crucial role in the ability of an ML algorithm to perform well on unseen data.

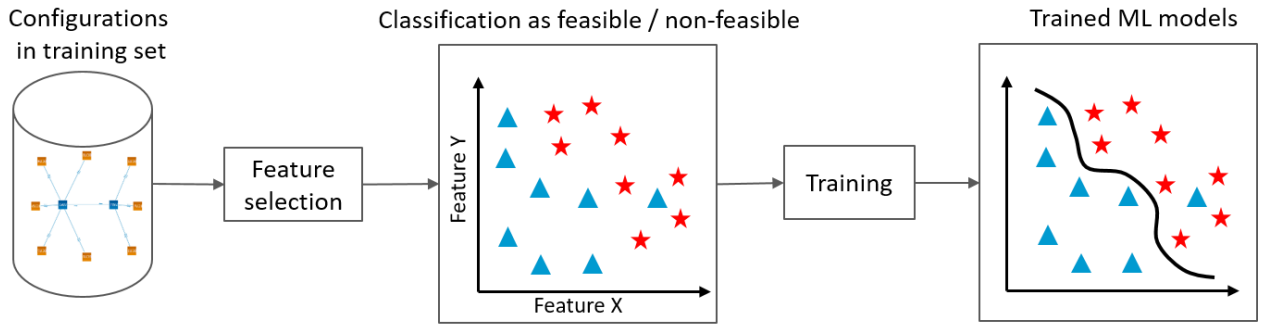

Figure 3 : Configurations in the training set are classified as feasible (blue triangle) or non-feasible (red star) based on the results of the precise schedulability analysis. The ML algorithm then tries to learn the characteristics of the configurations, that is the values of their features, that are predictive with respect to the feasibility of the configurations. Here the ML algorithm draws a

separation between feasible and non-feasible configurations that will be used to classify an unlabelled configuration.

4.2 Feature engineering and feature selection

Defining features to be used in a ML algorithm is called "feature engineering". This is a crucial step as the features, which are raw data or which are created from raw data, are meant to capture the domain-specific knowledge needed for the ML algorithm to be efficient on the problem at hand. Once a set of potential features has been identified, we have to select the ones that will be the most "predictive" and remove extraneous ones, as features with little or without predictive power will reduce the efficiency of an ML algorithm. Indeed, selecting all potential features together is not the best solution as it has now been well documented that more features does not equate to better classification after a certain point ("peaking" effect, see [DoHuSi09]).
As they are key issues in ML, feature engineering and feature selection have given rise to hundreds of studies over the past two decades (see [DoHuSi09, Li17] for good starting points). Feature engineering is typically done with domain knowledge, or it can be automated with feature learning techniques and techniques belonging to deep learning. Feature selection algorithms are typically, offered by any ML package (e.g., up to 40 different algorithms in *scikit-feature* [Li17]). Feature engineering and feature selection remain nonetheless difficult issues especially in domains where the number of raw data is huge (e.g., biological data such as genes, proteins, etc).
In our context, the raw data available to us are relatively limited in number: number of streams of each type, characteristics of the streams, topology of the network, etc. In addition, with the understanding of the TSN protocols and how schedulability analysis works, there are characteristics that we know will tend to make it difficult for a network to be feasible. For instance, if there is a bottleneck link in the network (*i.e*, the maximum load over all links is close to 1), or if the load is very unbalanced over the links, then it is more likely that the network will not be feasible than a network with perfectly balanced link loads. In a similar manner, with prior domain knowledge, it is possible to discard some features that will not be important factors for feasibility. In this study, we selected by iterative experiments the features among the raw data so as to maximize the prediction accuracy. We created one feature from the raw data, the Gini index of the load of the links, which evaluates the unbalancedness of link loads and thus the likelihood that there is one or several bottleneck links. This feature proves to increase noticeably the classification accuracy. From empirical experiments, we identified a set of five features with the most predictive power: the number of critical flows, the number of audio flows, the number of video flows, the maximum load of the network (over all links), the Gini index of the loads of the links.

4.3 The k-NN classification algorithm

Given a training set made up of data characterized by a vector of d features, k-NN works as follows:

- It caches all data of the training set.
- Given an unlabelled data p, it calculates the Euclidean distance from p to each cached data q as in Equation 1:

$$dist(p,q) = \sqrt{(p_1 - q_1) + (p_2 - q_2) + \cdots + (p_d - q_d)} \quad [1]$$

  with d the number of features.
- Based on the Euclidean distances, the algorithm identifies the k records that are the "nearest" to the unlabelled data p.
- K-NN classifies the data p in the category that is the most represented among its k nearest neighbours. Here we are solving a binary classification problem: if the majority of the k nearest neighbours is feasible, the unlabelled data is predicted to be feasible, otherwise, it is predicted to be non-feasible.

In the general case, the optimal value for k cannot be known beforehand, it has to be determined by iterative search, but the square root of the training set size typically provides a good starting point.

An advantage of k-NN is its reduced algorithmic complexity, which allows the use of large training sets. Actually, k-NN does not have a true training phase, it just stores the training data and postpone computation until classification. In terms of k-NN complexity, given n the number of data in the training set, the complexity of prediction for each unlabelled data is $O(n)$. For a comparison, Table 3 shows the complexity of two other very popular ML classification algorithms namely Decision Trees and Support Vector Machines [HaTiFri09]. Though the complexity of prediction in Decision Tree and SVM is smaller, the complexity of training is significant: it is polynomial in the size of the training set. Indeed, in the worst case, the complexity of training for Decision Trees is $O(n^2)$, and it is $O(n^3)$ for SVMs. It is worth noting that since the algorithms use the same number of features d, contribution of d is ignored for the sake of simplicity.

Algorithm	Complexity of Training	Complexity of Prediction
K-NN	--	$O(n)$
Decision Tree	Worst case: $O(n^2)$	$O(1)$
SVM	Worst case: $O(n^3)$	$O(1)$

Table 3 : Complexity of supervised learning algorithms as implemented in the popular *scikit* ML library [Pe11], with n being the size of the training set. K-NN does not require training step, and its prediction time is linear with the size of the training set, whereas the worst-case complexity of training Decision Trees and SVMs requires polynomial time but the prediction then runs in constant time.

4.4 Suitability of K-NN for TSN network classification

There is a diversity of ML algorithms to perform classification (see [HaTiFri09]), how to choose an algorithm that will perform well on the problem at hand is thus not a straightforward issue. As each classification algorithm has different ways of making use of the training data, an approach is therefore to try to gain insight into the training data and identify some of their structural characteristics, and try to determine if a given algorithm can be efficient on this data. A usually powerful manner of gaining an understanding of the data is to visualize it.

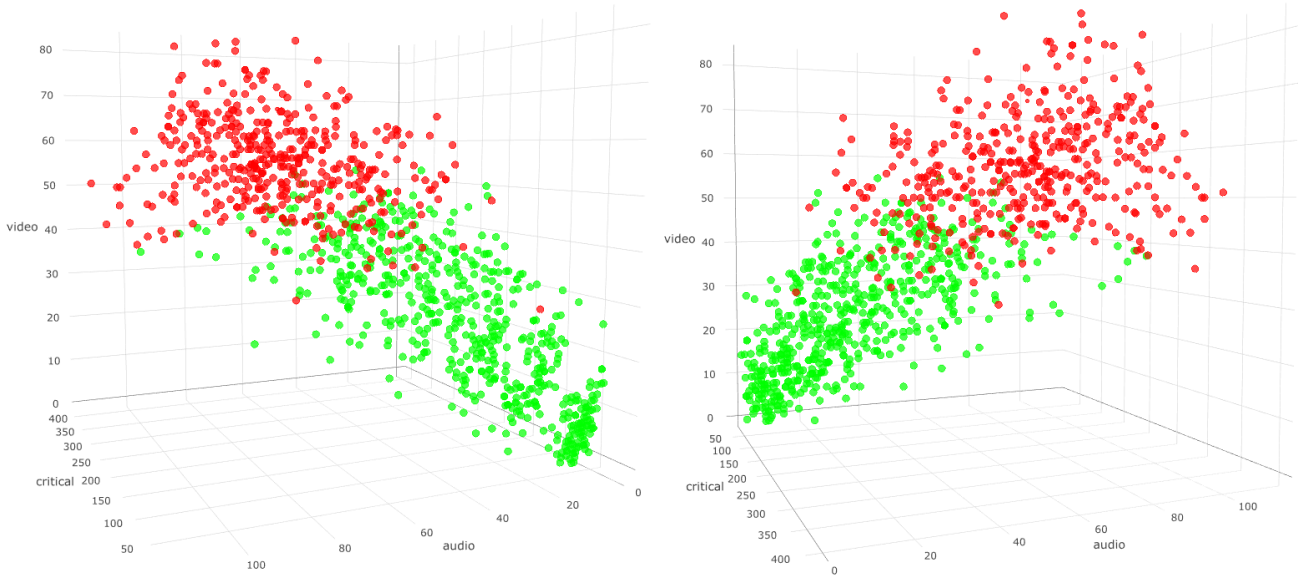

Figure 4: Configurations in the feature space (*i.e.*, number of audio, video and critical streams). The green dots are configurations feasible with Preshaping, while the red dots are configurations non-feasible with Preshaping. The configurations are split into two disjoint groups. In the feasible group, configurations have small number of flows, whereas in the non-feasible group, configurations have larger number of flows. Similar clustering in the feature space can be observed for all scheduling mechanisms considered in this study.

In our context, as shown in Figure 4, we observe that the two clusters, feasible and non-feasible configurations, are well disjoint in the 3D space. Precisely, we plot the training data according to three main features: the number of critical streams (x-axis), audio streams (y-axis) and video streams (z-axis). As can be seen on Figure 4, the configurations are clustered into two distinct groups: a group of feasible configurations (green points), which have small numbers of flows, and a group of non-feasible configurations (red points), which have large numbers of flows. The two groups tend to occupy different areas in the 3D space, and the data points , i.e., configurations, belonging to these groups are well clustered together. This suggests that if a new configuration is located in an area with a high density of feasible configurations, there is a large chance that this configuration is also feasible, and vice versa.
In other words, whether a new configuration is feasible or not can be predicted by a majority voting among the closest data points in the 3D space, or *d*-dimensional space in general (with *d* the number of features). As k-NN relies on the very same classification mechanism, this suggests that k-NN can be efficient at predicting the feasibility of TSN configurations. However, they are also small areas in the feature space where feasible and non-feasible configurations are mixed, where k-NN, or any methods trying to draws boundaries between groups like Support Vector Machines (SVM, see [HaTiFri09]), will not be able to make the right decision[3]. A more thorough analysis of the situations in which k-NN fails, and the relative predictive power of each of the features, can be found in [MaNaMi19a].

4.5 Generation of the training set

The training set is comprised of random TSN configurations based on the topology and traffic characteristics described in §2.2 with a total number of flows in the set [50, 100, 150, 200, 250, 300, 350, 400, 450, 500]. The latter interval for the training set is chosen to be sufficiently wide to cover the needs of many applications. The type of each flow and the parameters of each flow are chosen at random with the characteristics described in §2.2.
Like usually done in ML, the five features retained (see §4.2) are scaled into the range [0,1] based on the minimum and maximum possible value taken by a feature:

$$\text{normalized value} = \frac{\text{raw value} - \min}{\max - \min}$$

This normalization allows that all features possess a similar weight in the Euclidean distance calculation.
A crucial issue is then to choose the size of the training set; the larger the training set, the better the performance of ML but the higher the computation time. To estimate how many labelled configurations are needed for k-NN to be accurate, we increase the size of the training set until the prediction accuracy of k-NN does not show significant improvements.

[3] Experiments with SVMs not shown here have led to very similar performances as the ones obtained with k-NN.

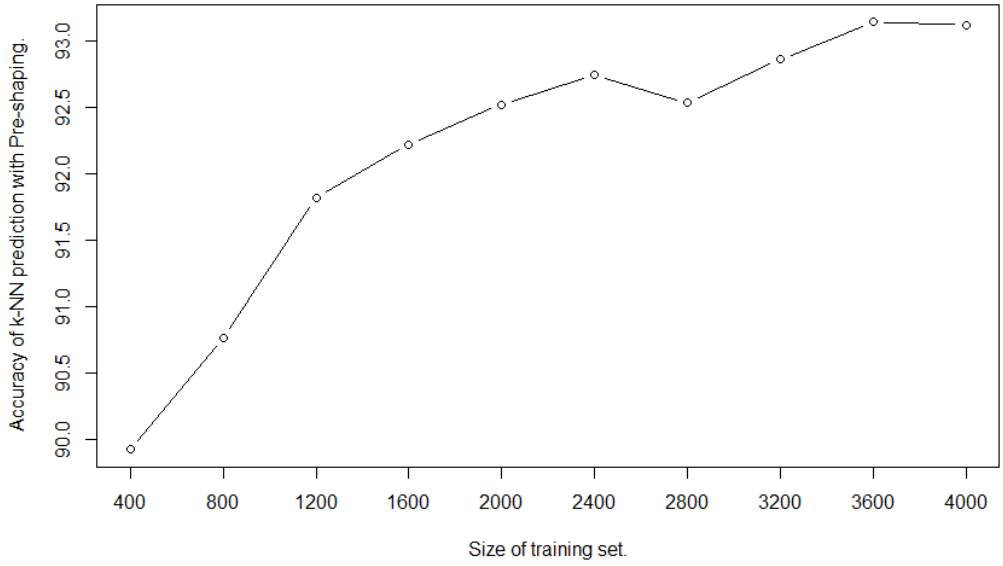

Figure 5 : Prediction accuracy of k-NN (y-axis) versus size of the training set (x-axis). The accuracy increases when the training set grows, however there is a plateau when the training set size reaches 3600. The fluctuation of the accuracy, e.g. between 2400 and 2800, can be explained by the non-deterministic characteristics of the TSN configurations evaluated. Results shown for Preshaping scheduling with 5-fold validation, with parameter k for k-NN equal to 20.

Figure 5 shows the relationship between prediction accuracy of k-NN, with k the number nearest neighbours set to 20, and size of the training set. Although the accuracy of k-NN increases when the training set is larger, it plateaus off past 3600. Therefore, in the rest of the study, we use training sets of size 4000. We conducted the same experiments for the three other scheduling solutions (FIFO, Manual, CP8) but only show here the prediction accuracy for Preshaping since, as the experiments will show, it is the scheduling solution whose feasibility is the most difficult to predict and requires the largest training set.

4.6 Performance criteria and evaluation technique

The following four metrics are retained to evaluate the performance of all the techniques considered in this work:

- The *overall Accuracy* (Acc) is the proportion of correct predictions over all predictions. The accuracy is the primary performance criterion in the following but it should be complemented with metrics making distinctions about the type of errors being made and considering class imbalance.
- The *True Positive Rate* (TPR), also called the *sensitivity* of the model, is the percentage of correct predictions among all configurations that are feasible.
- The *True Negative Rate* (TNR), also called the *specificity* of the model, is the percentage of correct predictions among all configurations that are not feasible.
- The *Kappa statistic* is an alternative measure of accuracy that takes into account the accuracy that would come from chance alone (e.g., suppose an event occurring with a rate of 1 in 1000, always predicting non-event will lead to an accuracy of 99.9%).

As classically done to combat model overfitting, that is a model being too specific to the training data and not generalizing well outside those training data, we use cross-validation with the k-fold[4] evaluation technique. In k-fold, the data set is divided into k equal sized subset (the k "folds"). A single subset is retained as testing set to determine the prediction accuracy while the remaining subsets are used all together as training set. The process is repeated k times until all subsets have served as testing set, then the accuracy of prediction is computed as the mean over all testing sets. k-fold evaluation guarantees that all configurations in the data set are used for feasibility prediction, i.e., there is no bias due to the prediction accuracy variability across the testing sets. In the following, we perform 5-fold evaluation, i.e., there are 4000 labelled configurations in the training set and 1000 unlabelled configurations in the testing set. With testing sets of size 1000, applying Theorem 2.4 in [LB10], the margin of error of the prediction accuracy is less than 3.1% at a 95% confidence level.

4.7 Experimental results

4.7.1 Parameter setting for k-NN

Since the optimal number of nearest neighbours k leading to the best k-NN performance is unknown, we test the values of k in the range [10, 100] by step 10 for each scheduling solution as shown in Table 4.

k	FIFO			Manual			CP8			Preshaping		
	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)
10	97.35	78.53	98.88	95.37	84.87	97.79	93.88	94.06	93.67	93.01	94.27	91.23
20	97.34	76.93	98.99	**95.64**	**85.78**	**97.91**	94.07	93.84	94.32	**93.01**	**94.10**	**91.49**
30	**97.42**	**76.93**	**99.08**	95.63	85.19	98.03	94.09	93.82	94.41	92.87	93.73	91.66
40	97.33	76.53	99.01	95.6	85.08	98.02	94.13	93.99	94.30	92.81	93.86	91.30
50	97.26	76.00	98.98	95.37	84.65	97.84	94.18	93.95	94.45	92.75	93.59	91.57
60	97.2	76.13	98.91	95.4	84.81	97.84	94.24	94.00	94.54	92.63	93.45	91.47
70	97.23	76.93	98.88	95.39	84.92	97.81	94.23	94.00	94.50	92.51	93.40	91.28
80	97.23	77.20	98.85	95.37	84.76	97.81	**94.29**	**93.93**	**94.69**	92.59	93.49	91.33
90	97.16	77.33	98.77	95.47	85.13	97.85	94.18	93.86	94.56	92.43	93.35	91.13
100	97.06	77.33	98.65	95.42	84.81	97.87	94.12	93.75	94.59	92.35	93.26	91.08

Table 4 : Prediction results of k-NN with k ranging from 10 to 100 for the four scheduling solutions. Bold font is used to identify the maximum prediction accuracy. Acc stands for accuracy, FP for false positive and FN for false negative. Values obtained by 5-fold evaluation with testing sets of size 1000 each.

We observe that the accuracy of k-NN is high for all values of k that are close to the optimal value. The difference between the best possible accuracy and the one obtained with a value of k either 10 above or below the optimal k value is always less than 0.9%. This suggests that in our problem k-NN is robust with respect to parameter k.
We note that the prediction accuracy of k-NN tends to decrease when the complexity of the scheduling mechanism increases. The more powerful the scheduling mechanism in terms of feasibility, we have FIFO < Manual < CP8 < Preshaping, the harder it is to predict its outcome on a given configuration.

4.7.2 Accuracy of the predictions

A prediction is wrong either due to a false positive or false negative. As Table 5 shows, there are differences between TPR and TNR across scheduling solutions. With FIFO, the TNR is much higher

[4] The parameter k in k-fold is neither related to the parameter k in k-NN, nor to the parameter K in the K-means algorithm. In the following, we keep the names of these standard techniques of the literature.

than the TPR. The reason may be due to the imbalance between feasible and non-feasible configurations in the training set, see Table 6. Indeed, with FIFO, non-feasible configurations largely outnumber feasible configurations with a proportion of 93.27%. Since there are much more "negative" training data, *i.e.*, non-feasible configurations, the machine learning algorithm may be more likely to conclude to non-feasibility even when the configuration is actually feasible.

Solution	Optimal k	Accuracy (%)	True Positive Rate (%)	True Negative Rate (%)	Kappa statistic (%)
FIFO	30	97.42	76.93	99.08	71.03
Manual	20	95.64	85.78	97.91	83.98
CP8	80	94.29	93.93	94.69	89.52
Preshaping	20	93.01	94.10	91.49	83.55

Table 5 : Performance of k-NN for the different scheduling solutions with the optimal k values. Accuracy, True Positive and True Negative Rates are obtained by 5-fold evaluation with testing sets of size 1000 each.

We might be concerned that with the imbalance in the testing set between feasible and non-feasible solutions (see Table 6), a high prediction accuracy can be obtained by chance alone. For instance, prediction accuracy for FIFO could be high just by consistently predicting non-feasible simple because the large majority of configurations are non feasible. In order to address this issue, we calculate the Kappa statistic, aka Kappa coefficient, which measures an "agreement" between correct prediction and true labels. If the Kappa coefficient is low, it is likely that true labels and predicted labels just match by chance. On the other hand, a Kappa coefficient higher than 60% was shown to be significant [MHMa12]. Table 5 shows the Kappa coefficients of k-NN prediction for the optimal value of k. Since the coefficients range from 71.03% to 89.52%, this suggests that the high accuracy of k-NN prediction is not obtained by chance alone.

Solution	Feasible configurations (%)	Non-feasible configurations (%)
FIFO	6.73	93.27
Manual	17.98	82.02
CP8	55.38	44.62
Preshaping	58.53	41.47

Table 6 : Percentage of feasible and non-feasible configurations for each scheduling solution in the training set with 4000 configurations. Logically, the percentage of feasible configurations increases with the possibilities offered by the scheduling mechanisms.

4.7.3 Robustness of the model

An important practical consideration is that a ML algorithm is able to perform well even if the unseen data does not meet perfectly the assumptions used in the training of the algorithm, that is during the learning phase. If the ML algorithm possesses some generalization ability to adapt to departure from the training assumptions, the model does not have to be retrained, and training sets re-regenerated in our context, each time the characteristics of the network change.
Here we study the extent to which changes in the traffic characteristics will influence the prediction accuracy of k-NN. We study this by keeping the training set unchanged, while changing some of the characteristics of the testing set. Precisely, the data payload size (denoted by s) of each critical flow in the testing set is set to a value randomly chosen in the range [s – x%, s + x%]. Table 7 reports the prediction accuracy of k-NN with a ± variation x up to 90% for the payload of the critical streams in the testing set. Figure 6 summarizes the numbers in Table 7 using boxplots, with the horizontal lines being the baseline prediction accuracies without changes in the training

set. Although, as can be seen in Figure 6, the performance of k-NN tends to decrease with these changes, the deterioration is limited: less than 2.17% whatever the scheduling solution. A possible first explanation is that some of the selected features, namely the maximum link load and the Gini index, are able to capture the variations of the data payloads and therefore remain predictive. Another explanation is that the average network load remains similar, as the variations are both positive and negative with the same intensity in both cases.

Var ± (%)	FIFO (k = 30)			Manual (k = 30)			CP8 (k = 80)			Preshaping (k = 20)		
	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)
0	97.42	76.93	98.97	95.64	85.78	97.91	94.29	93.93	94.69	93.01	94.10	91.49
10	96.88	66.14	99.19	95.19	83.33	97.74	**95.49**	96.33	94.47	92.78	94.91	89.93
20	96.4	66.88	98.97	95.1	81.09	98.44	**93.67**	94.00	93.26	92.11	93.71	89.80
30	**96.2**	65.12	98.90	94.61	79.49	98.34	94.25	93.20	95.59	91.66	94.09	88.38
40	96.91	72.06	98.72	95.59	81.78	98.72	94.17	94.06	94.30	92.23	93.37	90.74
50	96.6	67.61	98.82	95.33	81.81	98.69	94.22	93.28	95.34	**91.3**	92.99	89.00
60	97.0	70.13	99.24	**96.33**	88.47	98.17	93.92	93.99	93.84	93.24	94.64	91.33
70	**97.43**	70.83	99.49	95.34	84.95	97.68	94.75	93.60	96.14	**93.47**	95.23	91.28
80	96.59	69.08	98.85	**93.47**	76.45	97.65	93.96	94.24	93.63	92.39	94.06	90.16
90	96.85	65.47	99.00	94.31	81.55	97.13	95.1	95.44	94.71	92.14	94.17	89.49

Table 7 : Performance of k-NN with variations from -90% to 90% of the data payload for the critical flows in the testing set. Values in bold font are the minimum and maximum accuracy for each scheduling solution.

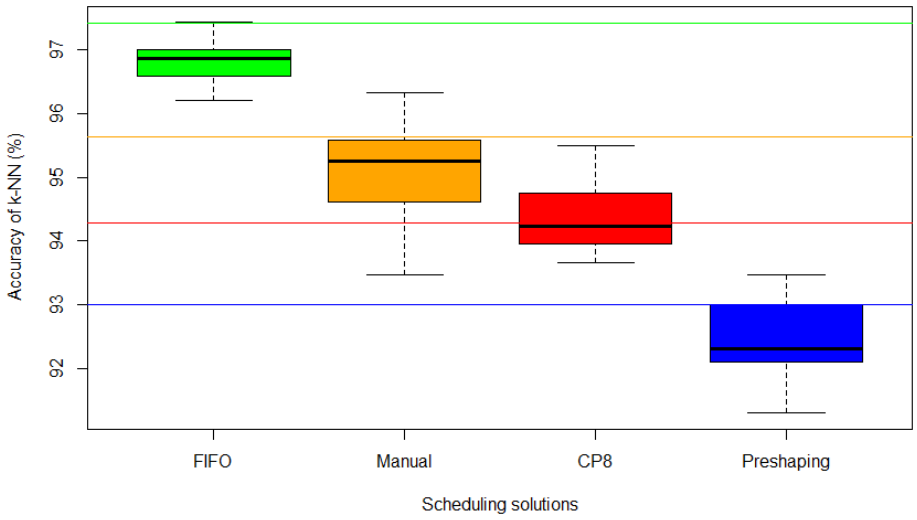

Figure 6 : Boxplot of the prediction accuracies obtained with k-NN with variations of the data payload of critical flows in the testing set between -90 and +90%. The horizontal lines are the baseline accuracies obtained when the traffic parameters used to generate the training and the testing set are identical.

5 Predicting feasibility with unsupervised learning

5.1 General principles

As illustrated in the previous section with the k-NN algorithm and further experimentations (not shown here) with SVMs leading to similar performances, supervised learning algorithms possess a good ability in predicting the feasibility of TSN configurations. However, supervised learning requires that the data in the training set are all labelled, i.e., here we have to execute a precise schedulability for each of the configurations in the training set. This is a compute intensive process, for instance for the experiments of the previous section, it took about three hours to determine the feasibility with the four scheduling solutions of the 4000 TSN training configurations on a 6 core 3.2Ghz Intel I7-8700.

In this section, we apply unsupervised learning, another class of ML techniques that learns to recognize patterns in a training set without relying on labels. Specifically, we focus on clustering algorithms, which group data into "clusters". Data in the same cluster are more "similar" to each other than those in other clusters. The process of clustering data creates knowledge that may be taken advantage of to predict feasibility. In our problem, the TSN configurations, are summarized by the five features listed in §4.2. There is a correlation between the feasibility of a configuration and the values of those features. Here the underlying assumption is that if we cluster together configurations with similar feature values, they may possess the same status with respect to feasibility.

In the following, we use the term *feasible cluster* when we assume that the majority of the configurations in this cluster are feasible. If this assumption holds, if an unlabelled configuration is assigned to a feasible cluster, then it is likely that this configuration is feasible as well. In other words, we apply a clustering algorithm to predict the feasibility of unlabelled configurations based on the clusters they are assigned to. Similarly, we denote by *non-feasible cluster* a group of configurations that are assumed to be in majority non-feasible.

Since, in unsupervised learning, configurations in the training set are unlabelled, we need a method to distinguish between feasible and non-feasible clusters once they have been created:

- Method 1: for each cluster that has been created, we select at random a number of configurations and give them labels based on the precise analysis. The cluster is identified as a feasible cluster if the majority of the configurations tested are feasible.
- Method 2: once the clusters have been created, we generate new configurations that undergo precise schedulability analysis in order to label them. Then we assign those configurations to the clusters whose center is the closest to them. If a majority of the labelled configurations assigned to a given cluster are feasible, then the cluster is identified as a feasible cluster. These newly created labelled configurations are making up what we refer to as the *voting set* in the following.

If the two methods are based on a voting scheme, the first requires to store all the configurations in the training set, which may be a problem in practice as clustering is typically done on very large number of configurations (100,000 in our experiments). In contrast, with the second method, only the vector of features of the configurations needs to be stored. For this reason, we choose to implement this second approach.

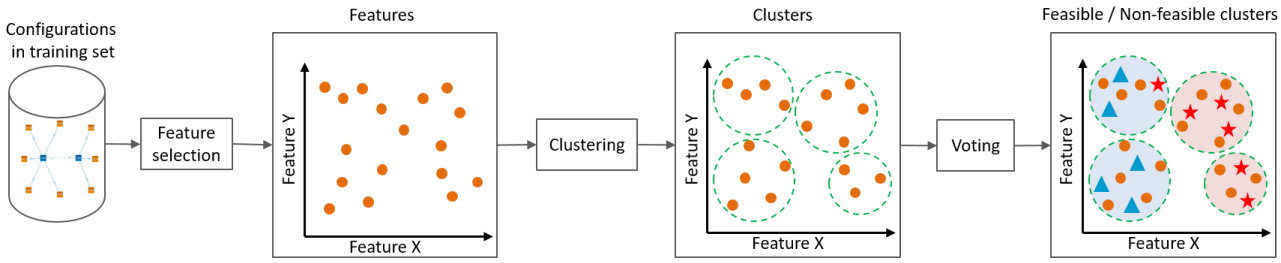

Figure 7 : After feature selection, configurations in the training set are grouped into clusters (dashed circles) by the K-means algorithm. The labelled configurations part of the voting set, either feasible (blue triangle) or non-feasible (red star), are used to identify feasible clusters (light blue circles) and non-feasible clusters (light red circles) based on the majority of labelled configurations within a cluster.

The approach proposed here could be considered a semi-supervised learning technique as we make use of both labelled and unlabelled data. However, semi-supervised clustering techniques [Bai13] typically make use of the labelled data, along with the unlabelled data, to build the clusters, while we are using labelled data only to determine the probable status of the clusters with respect to feasibility. For this reason, we consider here that this approach belongs to the class of unsupervised learning techniques.

5.2 The K-means clustering algorithm

In this study, we experiment with the K-means clustering algorithm as it is a well-understood low-complexity algorithm that is known to perform well in context where clusters are not of too widely varying densities and tend to be spherical.
Given a training set with unlabelled configurations and a predefined number of clusters K, K-means starts by selecting K points at random in the feature space to serve as the *cluster centers*. It then iterates between two steps:

- Allocate data point to clusters: compute the squared Euclidean distance of the features from a data point to all the cluster centers, then assign the data point to the cluster whose center is the nearest. Repeat for all data in the training set.
- Update centers: a new center is recomputed as the mean of the features of all data points belonging to this cluster.

This iterative algorithm finishes when the algorithm has converged, i.e., no more data points are moved from one cluster to another. This happens when the sum of the squared distances from the data points to their cluster center has been minimized, that is when the dissimilarity in terms of features within clusters is minimum.
In addition to its simplicity, K-means is among the fastest clustering algorithms. Although it has been shown in some specific cases to be superpolynomial [ArVa06], the complexity of K-means is linear with respect to the number of training data in most practical cases, compared to polynomial time for most other clustering algorithms [HaTiFri09].

5.3 Generation of the training set

A training set for K-means is generated with the same settings as in §4.5 for k-NN. [Do14] recommends the size of the training set to be no less than 2^d, preferably $5 * 2^d$ where d is the number of features. We decided to use 100,000 unlabelled configurations, which is much larger the guidelines in [Do14] to guarantee that the training set is large enough. In terms of computation time, generating 100,000 unlabelled TSN configurations takes only 9mn on 6 computational cores compared to 3 hours of computation for labelling 4000 TSN configurations using precise schedulability analysis, which represents a speedup of approximatively 20. If we include the cost

of creating the voting set (see §5.1) made up of 500 labelled configurations in our experiments, the speedup drops to approximatively a factor 6.

5.4 Experimental results

5.4.1 Number of clusters

On the contrary to other clustering techniques, K-means requires us to choose the number of clusters K. In many applications, the number of clusters is imposed by the problem, or can be determined with domain knowledge. In our problem, this is not the case and we have to choose a reasonable value for the number of clusters. We first note that as K-means minimizes the total sum of the squared distances within clusters, the more clusters there is, the smaller this quantity. However, having more clusters increases the running time and there is a point past which the gain of adding clusters becomes limited. This can be observed by plotting the sum of the within-cluster squared distances versus the number of clusters as shown in Figure 8, which is referred to as the "elbow" technique. As can be seen, K-means does not show significant improvements when the number of clusters exceeds 20. Therefore, in the following experiments we set 20 as the upper bound for the maximum number of clusters.

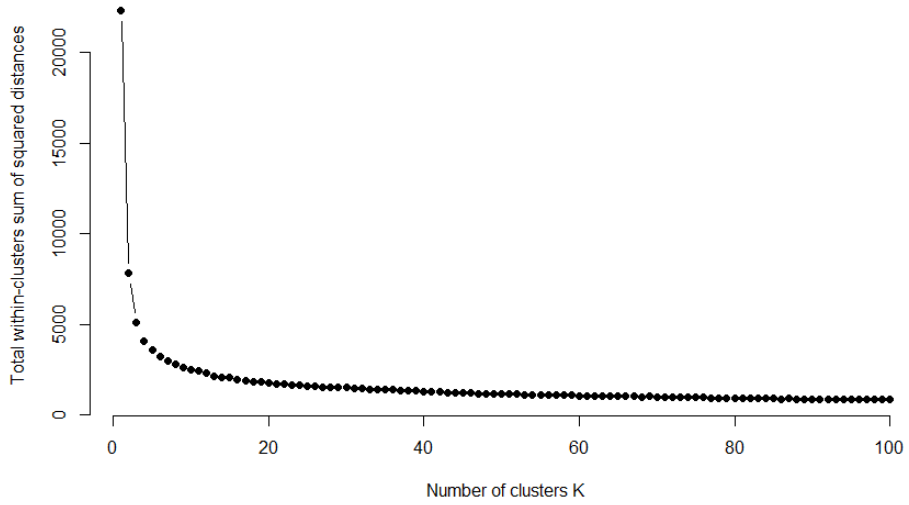

Figure 8 : Sum of the within-cluster squared distances, i.e. which is the measure of dissimilarity used in K-means, versus the number of clusters K. The "elbow" of the plot happens for K in range [2, 20].

5.4.2 Size of the voting set

As explained in §5.1 we use a number of labelled configurations, the voting set, to determine whether a cluster is a feasible cluster or not. A configuration in the voting set is allocated to the cluster whose center is the nearest in terms of features' distance. To have enough labelled configurations in each cluster, the size of the voting set must be large enough. However, there is a trade-off to be found as the larger the voting set, the more computation time needed.
Figure 9 shows the prediction accuracy of K-means for a voting set from 100 to 1000 configurations. We observe that below 400 the improvements in accuracy are significant but afterwards the accuracy plateaus off. In all the following experiments, the size of the voting set is set to 500.

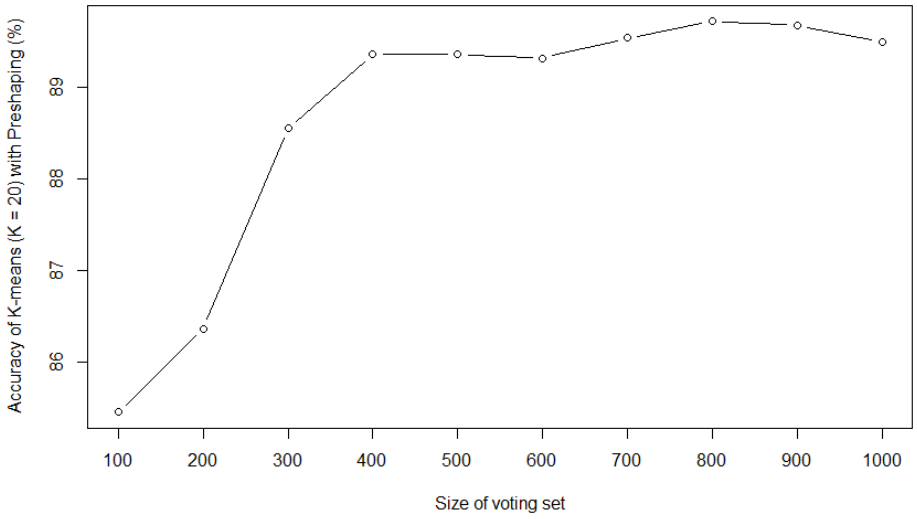

Figure 9 : Size of voting set versus prediction accuracy of K-means for Preshaping sheduling. The accuracy of K-means does not show significant improvement when the voting set exceeds 400. From this point onwards, clusters have a sufficient number of labelled configurations for majority voting. Each point is computed over 1000 test configurations with K = 20.

5.4.3 Accuracy of the predictions

Like for k-NN, the performance criteria for the evaluation are the accuracy and the rate of true positive (TPR) and true negative (TNR). Similarly, we use testing sets of 1000 unlabelled TSN configurations. Table 8 shows the performance of K-means in predicting the feasibility of TSN networks with the four scheduling solutions. We observe that globally the accuracy of K-means is better for large number of clusters, although there are differences depending on the scheduling solution. The accuracy with Manual is maximum for K equal 10 and remains high for larger values. The same pattern can be observed with CP8 with an optimal K value of 12. The accuracy with FIFO and Preshaping is the highest for K = 20. For these policies, the best values for K are around 50 leading to an improvement of less than 1% in accuracy over K=20. However, more clusters requires more training time, which reduces the interest of K-means over k-NN.

K	FIFO			Manual			CP8			Preshaping		
	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)
2	92.5	0.0	100.0	81.3	0.0	100.00	86.5	76.57	98.25	83.6	72.14	99.76
4	92.5	0.0	100.0	93.6	85.03	95.57	86.8	76.94	98.47	85.5	97.78	68.19
6	94.88	85.47	95.64	91.85	71.66	96.49	90.13	95.41	83.89	87.53	87.83	87.11
8	95.08	70.00	97.11	92.35	87.59	93.44	89.73	86.94	93.03	88.17	89.25	86.65
10	94.86	62.40	97.49	93.72	81.12	96.62	**92.56**	**93.08**	**91.94**	88.39	88.65	88.02
12	94.2	51.87	97.63	**94.56**	**77.27**	**98.54**	91.89	95.18	87.99	88.04	85.04	92.27
14	94.37	64.93	96.76	93.59	79.30	96.88	91.18	91.57	90.72	88.37	86.79	90.60
16	95.15	77.73	96.56	93.76	81.66	96.54	91.79	93.30	90.00	89.25	88.99	89.61
18	95.18	81.20	96.31	93.23	82.89	95.61	91.47	92.80	89.89	89.32	89.44	89.16
20	**95.52**	**80.80**	**96.71**	93.67	77.27	97.44	91.91	93.54	89.98	**89.49**	**89.37**	**89.66**

Table 8 : Prediction results for K-means with K ranging from 2 to 20 for the four scheduling solutions with testing sets of 1000 TSN configurations. Bold font identifies the maximum prediction accuracy. Acc stands for accuracy, TPR for True Positive Rate and TNR for True

Negative Rate. Predictions for FIFO and Preshaping are more accurate for the largest values of K unlike for Manual and CP8 scheduling.

Table 9 summarizes the performance of K-means with the optimal number of clusters in the interval [2,20]. The accuracy ranges from 89.5 to 95.5%. Like for k-NN, the more powerful (in terms of feasibility) the scheduling mechanism, the harder it is to predict the feasibility for a given network configuration. Compared to supervised learning with k-NN (see Table 5), the accuracy of K-means is lower by 1 to 3.5%. Table 9 also shows the Kappa coefficients, which suggests that K-means possesses a true predictive ability in our context.

Solution	Optimal K	Accuracy (%)	True Positive Rate (%)	True Negative Rate (%)	Kappa coefficient (%)
FIFO	20	95.52	80.80	96.71	95.16
Manual	12	94.56	77.27	98.54	93.82
CP8	10	92.56	93.08	91.94	91.44
Preshaping	20	89.49	89.37	89.66	88.01

Table 9 : Performance of K-means with the optimal number of cluster K for each scheduling solution and corresponding Kappa coefficients.

5.4.4 Robustness of the model

We study whether changes in the traffic characteristics would affect the accuracy of K-means by adding variations to the packet size of critical flows in the same manner as done in §4.7.3.
Table 10 reports the prediction accuracy of K-means, as obtained with the optimal K values, with a ± variation x up to 90% for the payload of the critical streams in the testing set. Figure 10 summarizes the numbers in Table 10 using boxplots with the horizontal lines being the baseline prediction accuracies without changes in the training set.
As can be seen in Figure 10, the performance of K-means tends to decrease with these changes for Manual and CP8, with a deterioration respectively of 1.85% and 1.84% in the worst-case. The loss in accuracy is less pronounced for FIFO (-0.81%) and Preshaping (-1.48%). As it is the case for k-NN, these results suggest that K-means is robust to changes in the data payload between the training and the testing set. If variations in the traffic characteristics lead to fluctuations in the accuracy, we do not observe a clear correlation between the accuracy and percentage of variation. For instance, in the case of CP8, the accuracy without variation is 92.56%, the highest value, while with a 90% ± variation it remains almost identical at 92.54%.

Var ± (%)	FIFO (K = 20)			Manual (K = 12)			CP8 (K = 10)			Preshaping (K = 20)		
	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)	Acc (%)	TPR (%)	TNR (%)
0	95.52	80.80	96.71	94.56	77.27	98.54	**92.56**	93.08	91.94	89.49	89.37	89.66
10	95.74	75.57	97.26	94.27	75.71	98.26	92.2	92.14	92.28	90.61	89.35	92.29
20	94.75	75.12	96.46	94.52	75.85	98.98	**90.72**	90.58	90.90	88.51	87.97	89.29
30	96.19	81.88	97.43	93.3	82.37	96.00	91.67	90.55	93.11	89.61	88.57	91.01
40	95.32	80.88	96.37	94.26	78.81	97.77	91.38	90.46	92.47	88.77	87.03	91.04
50	96.1	81.55	97.21	93.85	76.23	98.23	91.37	88.07	95.32	**88.01**	86.52	90.05
60	95.2	76.75	96.74	**95.19**	81.53	98.40	91.01	17.10	90.18	90.34	89.71	91.21
70	**96.77**	79.44	98.11	93.22	76.25	97.05	91.79	87.81	96.60	**90.82**	90.18	91.61
80	**94.71**	75.66	96.28	**92.71**	74.92	97.07	92.32	89.94	95.16	90.56	90.12	91.14
90	94.84	78.91	95.93	93.4	76.69	97.09	92.54	93.34	91.61	88.7	88.59	88.85

Table 10 : Performance evaluation of K-means for predicting the feasibility of TSN configurations with four scheduling solutions when the data payload size of the critical flows varies in the testing set. Experiments performed with the optimal number of clusters for each scheduling solution, as

shown in Table 8. Although the accuracy of K-means fluctuates, it does not show any trend with respect to the percentage of variation.

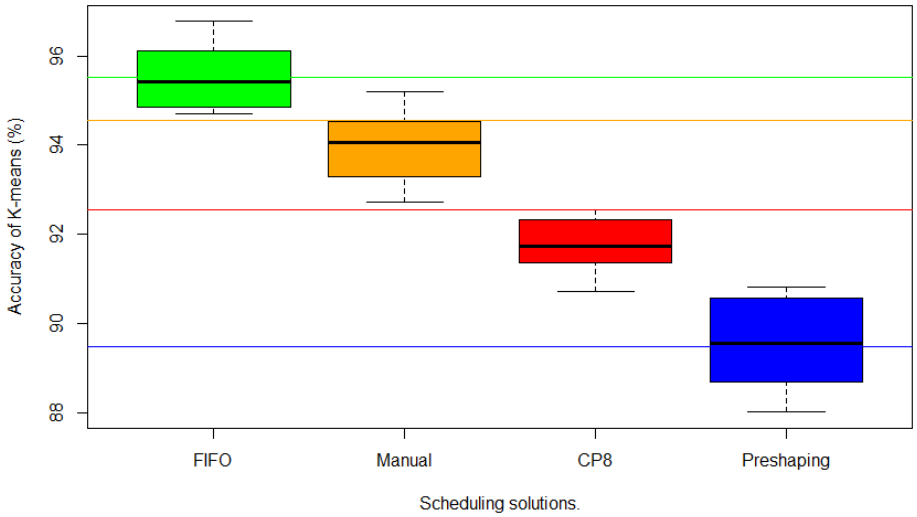

Figure 10 : Boxplot of the prediction accuracies obtained with K-means with variations of the data payload of critical flows in the testing set between -90 and +90%. The horizontal lines are the baseline accuracies obtained when the traffic parameters used to generate the training and the testing set are identical. The change of data payload size also leads to fluctuation in the prediction accuracy of K-means, which is as observed for K-NN.

6 On the use of ML to speed up the verification

In this work, we explore the application of ML to the problem of determining the feasibility of TSN configurations. We summarize in this section the experimental results and discuss the new trade-offs between accuracy and computation time offered by ML techniques.

6.1 Summary of the experiments

Table 11 provides a recap of the experimental results with the four methods used to determine feasibility: supervised learning with k-NN, unsupervised learning with K-Means, approximate schedulability analysis and precise schedulability analysis. The execution times shown in this paragraph were obtained on a 6-core 3.2Ghz Intel 8700 configured following the guidelines in [Pao10] to remove frequency scaling and turbo mode options, which are known sources of indeterminism. The total computation times reported in Table 11 are for all scheduling solutions, that is the feasibility of each TSN configuration is tested with the four scheduling mechanisms described in §2.4. In the case of the ML algorithms, the computation times include both the training time and the actual prediction time once the algorithms have been trained. This is the computation time that assessing the feasibility of TSN configurations in a design space exploration algorithm would require.

It should be kept in mind that, beyond computing WCTTs, the approximate and precise analyses for CP8 and Preshaping perform additional computation for setting parameters, respectively priority assignment and traffic shaping parameters. Most often, for a given TSN configuration, both CP8 and Preshaping examine several candidate solutions each requiring each a WCTT analysis, before concluding on the schedulability. This explains why, in Table 11, the approximate analysis is only 3 times faster than the precise analysis. Indeed, because the approximate analysis

leads to larger WCTT bounds, approximate analysis for CP8 requires trying more candidate solutions before a feasible one is found than with the precise analysis.

Method	**K-means**	**k-NN**	**Approximate NC analysis**	**Precise NC analysis**
Approach	Unsupervised learning	Supervised learning	Schedulability analysis	Schedulability analysis
Size of training set	100 000 unlabelled configurations + 500 labelled configurations for voting set	4 000 labelled configurations	--	--
Time to generate training set	≈ 31mn	≈ 3h	--	--
Training time	≈ 48s	--	--	--
Time to assess feasibility of 1000 configurations	**≈ 8ms**	**≈ 120ms**	**≈ 15mn** (include priority allocation for CP8)	**≈ 45mn** (include priority allocation for CP8 and computation of traffic shaping parameters for Preshaping)
Accuracy for FIFO	95.52 %	97.42 %	97.42 %	100%
TPR for FIFO	80.80%	76.93%	69.33%	100%
TNR for FIFO	96.71%	99.08%	100%	100%
Accuracy for Manual	94.56 %	95.56 %	94.2 %	100%
TPR for Manual	77.27%	85.78%	68.98%	100%
TNR for Manual	98.54%	97.91%	100%	100%
Accuracy for CP8	92.56 %	94.29%	85.4%	100%
TPR for CP8	93.08%	93.93%	71.22%	100%
TNR for CP8	91.94%	94.69%	100%	100%
Accuracy for Preshaping	89.49 %	93.01 %	Not available in toolset	100%
TPR for Preshaping	89.37%	94.10%	Not available in toolset	100%
TNR for Preshaping	89.66%	91.49%	Not available in toolset	100%

Table 11 : Summary of the experiments performed on a 6-core 3.2Ghz Intel I7-8700 with RTaW-Pegase V3.3.6 for the schedulability analyses. The time taken to assess the feasibility of 1000 configurations ranges from 8ms (K-means with a trained model) to 45mn (precise schedulability analysis). The worst accuracy (85.4% for CP8) is obtained with the approximate schedulability analysis that requires approx. 15mn of computation, which is more than the two ML techniques

k-NN and K-means. However, none of the two schedulability analysis lead to false positive. Execution time for approximate analysis of Preshaping scheduling estimated based on the ratio between the execution times of CP8 and Preshaping precise analysis.

It should be pointed out that the results of the precise schedulability analysis are considered in this study as 100% accurate, as this is the most accurate technique we have, and as we know for certain that it does not lead to false positive. In reality, the precise schedulability analysis is conservative, it returns upper bounds and not the exact worst-case values, which creates a certain amount of false negative configurations. Even if prior works [BaScFr12, BoNaFu12] suggest that the precise analysis for static priority is tight, its pessimism will create a certain amount of configurations deemed unfeasible while they actually are feasible. This phenomenon introduces a bias in the accuracy results presented here, which leads to the false positive rate (FPR) to be overestimated and the true negative rate (TNR) to be underestimated. As our main concern are false positives, the actual risk with supervised learning can be less than in the results shown hereafter, though it cannot be precisely quantified.
Table 12 shows estimates of the total running times on six CPU cores of the four verification techniques to determine the feasibility of 1000, 100 000 and 1000 000 TSN configurations. For ML algorithms, this running time includes the time to generate the training set, the training time, and the time to predict feasibility. Numbers in Table 12 are representative of what can be expected in terms of execution times on a standard desktop computer. These results suggest that ML is the most appropriate technique for design space exploration involving the creation of a large number of candidate solutions.

Number of TSN networks	K-means	k-NN	Approximate analysis	Precise analysis
1 000	31mn48s	3h	15mn	45mn
100 000	31mn49s	3h	1day1h	3days3h
1000 000	31mn56s	3h2mn	10days10h	31days6h

Table 12 : Estimation of the computation times to predict feasibility with all the four possible scheduling mechanisms for a varying number of TSN configurations. For k-NN and K-means algorithms, both training and prediction times are accounted for. Figures shown for TSN networks on a 6-core 3.2Ghz Intel I7-8700.

6.2 Efficiency area of the verification techniques

Figure 11 shows the trade-offs between accuracy and running times obtained with each of the techniques. We consider Manual scheduling with computation on a single CPU core, which leads to the most unbiased comparisons in terms of execution times.

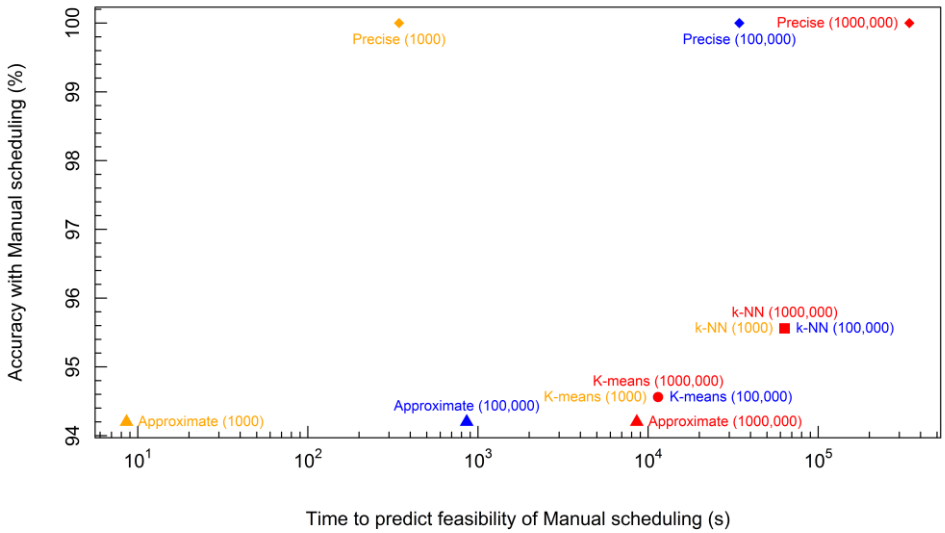

Figure 11 : Accuracy (%) versus running time on a single computational core (seconds in log scale) with Manual scheduling for 1000, 100000 and 1000000 TSN configurations. Squares, points, triangles and diamonds identify results with k-NN, K-means, Approximate and Precise analysis respectively. Running times of ML algorithms only increase marginally when the number of configurations grows, whereas the running time of the analyses increases linearly with size of the testing set. In terms of accuracy, ML algorithms are between approximate and precise analysis.

The experiments done in this study lead to the insights summarized below:

- For a small number of configurations (1000 here), the machine learning algorithms experimented are not competitive with precise schedulability analysis as they are both slower and less accurate.
- For a medium number of configurations (100,000 here), approximate analysis, K-means and precise analysis offer all different meaningful trade-offs between accuracy and running times. Supervised learning with k-NN is not competitive here. In many contexts however, like for Preshaping in this study, an efficient approximate analysis will not be available and then supervised learning becomes competitive for a relatively small number of configurations (*i.e.*, as soon as the number of configurations tested is larger than the size of the training set).
- For a large number of configurations (1000,000 here), all techniques are meaningful as none is dominated by another for both the accuracy and running times. However precise analysis may not be practical because of the execution times (16 hours with Manual and 31 days with all four scheduling analyses, see §3.3 and Table 12) compared to 3 hours for supervised learning and slightly more than 30mn for unsupervised learning with all four analyses on 6 cores. Once the training time has been amortized, machine learning techniques are very fast even for large number of configurations. Among ML algorithms, k-NN is slower but offers a better accuracy than K-means (+1 to 3.5%) and possesses a better ability to identify feasible configurations. For instance, for Manual scheduling, k-NN is able to predict correctly 86% of the feasible solutions versus 77% for K-means.
- Approximate analysis is less accurate than machine learning techniques in our experiments. Despite its speed, the benefits of using it are unsure to us, except if false positive must be ruled out.

It should be borne in mind that both the approximate and precise TSN schedulability analysis are very fast (resp. 13ms and 471ms on average for 500 flows, see Table 2). This comes from both the efficiency of the implementation and the small value of the Least Common Multiple (LCM) of the frames' periods (at most 160 seconds). The area of efficiency of the different techniques may thus be different for other schedulability analyses and message sets.

7 Related work

Over the last two decades, ML techniques have been successfully applied to very diverse areas such as bioinformatics, computer vision, natural language processing, autonomous driving and software engineering [Al18, As18]. In recent years, deep learning algorithms, that perform feature extraction in an automated manner unlike with traditional ML techniques, have been an especially active field of research (see [Po18] for a survey).

The two application domains of ML directly relevant to this work are networking and real-time systems. Between the two, ML in networking (see [Wa18] for an overview), especially networking for the Internet, has been by far the most active area. ML has been applied to solve problems such as intrusion detection, on-line decision making (parameters and routes adaptation), protocol design, traffic and performance prediction. For instance, in [AN14] an ML algorithm, based on the "expert framework" technique, is used to predict on-line the round-trip time (RTT) of TCP connections. This algorithm allows TCP to adapt more quickly to changing congestion conditions, decreasing thus on average the difference between the estimated RTT and the true RTT, which results in better overall TCP performance. In [Ma17] an algorithm belonging to Deep Belief Networks computes the packet routes dynamically instead of using conventional solutions based on OSPF (Open Shortest Path First). Another impressive application of ML is to be found in [WiBa13] where the authors implement a "synthesis-by-simulation" approach to generate better congestion control protocols for TCP comprising more than 150 control rules.

ML has also found applications in real-time systems, although the results appear to be more disparate and much less numerous. As early as 2006, [KaBa06] proposes the use of ML for the problem of automatically deriving loop bounds in WCET estimation. Later, researchers from the same group use a Bayesian network created from a training set made up of program executions to predict the presence of instructions in the cache [BaBaCu10]. More recently, a line of work has been devoted to ML algorithms for Dynamic Voltage and Frequency Scaling (DVFS) in battery-powered devices. For instance, [Ma18] presents a learning-based framework relying on reinforcement learning for DVFS of real-time tasks on multi-core systems. ML techniques are also applied to decide the order of execution of tasks on-line. This has been done in various contexts. For instance [MaDaPa18] implements a neural network trained by reinforcement learning with evolutionary strategies to schedule real-time tasks in fog computing infrastructures, while in [Ma16] multi-core task schedules are decided with Deep Neural networks trained by reinforcement learning using standard policy gradient control.

Very relevant to this work is [RiGuGH17] that presents a framework based on the MAST tool suite for real-time systems to generate massive amount of synthetic test problems, configure them and perform schedulability analyses. This framework dedicated to the study of task scheduling algorithms is similar in the spirit to the RTaW-Pegase framework used in this study, that can be operated through a JAVA API and that includes a synthetic problem generation component named "Netairbench". Such frameworks are key to facilitate and speed-up the development and performance assessment of ML algorithms, which requires extensive experiments, be they based on real or artificial data.

8 Discussion and perspectives

This study shows that standard supervised and unsupervised ML techniques can be efficient at predicting the feasibility of realistic SN network configurations in terms of computation time and accuracy. In particular our experiments show that ML techniques outperform a coarse grained schedulability analysis with respect to those two performance metrics. Importantly, the ML algorithms experimented in this work neither require huge amount of data nor important computing power, they can be part of the toolbox of the network designers and be run on standard desktop computers. This approach investigated in this work can be applied for verifying the feasibility in other areas of real-time computing, for instance it could be used for end-to-end timing chains across several resources whose schedulability analyses are typically very compute intensive. Further work is however clearly needed to assess the capabilities and limits of this approach in other contexts.

A key difference of ML-based feasibility verification with conventional schedulability analysis is the possibility of having a certain amount of "false positives" (up to 3.53% with k-NN): configurations deemed feasible while they are not. In our view, this may not be a problem in a design-space exploration process as long as the few retained solutions at the end are verified by an analysis that is not prone to false positives. If ML techniques are to be used to predict feasibility in contexts where no schedulability analysis is available, then the execution environment should provide runtime mechanisms (e.g., task and message dropping, backup mode) to mitigate the risk of not meeting timing constraints and ensure that the system is fail-safe.

A well-known pitfall of supervised ML is to rely on training data that are not representative of the unseen data on which the ML algorithm will be applied. This may cause ML to fail silently, that is without ways for the user to know it. For instance, k-NN is more likely to return a wrong prediction when, in the feature space, the neighbourhood of the unseen data is sparse. In this study, we tested the sensitivity of the ML algorithms to departure from the traffic characteristic assumptions and the algorithms proved to be robust. In this work the network topology of the configurations in the training set and the testing set are identical. Changes in the topology was outside the scope of the study, as, in the design of critical embedded networks, the topology is usually decided early in the design phases, before the traffic is entirely known. The problem will probably be more difficult for ML if the unseen configurations may have different topologies, much larger training sets with a diversity of topologies will be required and the overall prediction accuracy might be reduced. A metric to estimate the distance between the training data and the unseen data and, as suggested in [JuRo07], could help assess the uncertainty of prediction.

This study can be extended in several directions:

- In order to minimize the rate of false positives, a measure of the uncertainty of prediction could be used to decide to drop a prediction if the uncertainty is too high and rely instead on a conventional schedulability analysis. This will only be possible for systems for which there exists a precise schedulability analysis. This may lead to hybrid feasibility verification algorithms where the clear-cut decisions are taken by ML algorithms, while the more difficult ones are taken by conventional schedulability analyses. This approach is explored in [MaNaMi19a]
- We intentionally applied standard ML techniques using the kind of computing power for building the training sets that can be provided by standard desktop computers in a few hours. A better prediction accuracy may be achievable with 1) larger training sets, 2) additional features such as the priorities of the flows to capture additional domain-specific knowledge, and 3) more sophisticated ML algorithms like XGBoost for supervised learning or DBSCAN and Expectation–Maximization clustering using Gaussian mixture models for unsupervised learning. In particular, the latter algorithms are known to be more robust than K-means when the feature space is not spherical. Also promising are "ensemble

methods" which combine the results of several ML algorithms, for instance by majority voting.

- Semi-supervised learning techniques, making use for the training set of a small amount of labelled data together with a large amount of unlabelled data, possess a lot of potential on the problem of predicting feasibility. Indeed, as the CPU time needed to create synthetic TSN configurations is negligible compared to the CPU time needed to label the data by schedulability analysis, this would allow the ML algorithms to rely on training sets several orders of magnitude larger than with supervised learning.
- To perform well, the k-NN and K-means algorithms used in this work ideally requires to be provided with "hand-crafted" features capturing domain knowledge, such as the Gini index of the link loads in this study. Most likely not all relevant features have been identified. A future work is to apply on the same problem deep learning techniques (e.g., Convolutional Neural Network - CNN) that simplify feature engineering by automating feature extraction. However, to do so, deep learning algorithms typically require much larger training set, in the millions in many deep learning applications, and lead to less understandable results.

In this work, ML is applied to predict the feasibility with respect to the types of constraints that can be verified by a schedulability analysis (*i.e.*, worst-case latencies, jitters and buffer utilization). There are other timing constraints that can only be verified by simulation, such as throughput constraints. ML could be also very interesting for such constraints as a simulation, for its results to be sufficiently robust in terms of sample size, typically takes at least one order of magnitude longer to execute than a schedulability analysis.

Generally speaking, ML has the potential to offer solutions to other difficult and topical problems in the area of real-time networking. In our view, it could be especially helpful for admission control in real-time, which is an emerging need in real-time networks with the increasing dynamicity of the applications for both the industrial and the automotive domains. Another domain of application of ML, like exemplified in [Ma16], is resource allocation. In the context of TSN networks, ML is a good candidate to help build bandwidth-effective time-triggered communication schedules for IEEE802.1Qbv.

9 References

[Al18] M. Allamanis, E.T. Barr, P.T. Devanbu and C.A. Sutton, "A Survey of Machine Learning for Big Code and Naturalness", ACM Comput. Surv. 51(4), 2018.

[AN14] B.A. Arouche Nunes, K. Veenstra, W. Ballenthin, S. Lukin, K. Obraczka, "A machine learning framework for TCP round-trip time estimation", EURASIP Journal on Wireless Communications and Networking, 2014(1), March 2014. doi: https://doi.org/10.1186/1687-1499-2014-47

[ArVa06] D. Arthur and S. Vassilvitskii, "How slow is the k-means method?", in Proc. of the twenty-second annual symposium on Computational geometry (SCG '06). ACM, New York, NY, USA, pp144-153. 2006. doi: http://dx.doi.org/10.1145/1137856.1137880

[As18] A.H. Ashouri, W. Killian, J. Cavazos, G. Palermo, and C. Silvano, "A Survey on Compiler Autotuning using Machine Learning", ACM Comput. Surv. 51(5), 2018. DOI: https://doi.org/10.1145/3197978

[Aud01] N. Audsley, "On priority assignment in fixed priority scheduling", Information Processing Letters, vol. 79, pp. 39–44, 2001.

[BaBaCu10] M. Bartlett, I. Bate, and J. Cussens, "Instruction Cache Prediction Using Bayesian Networks", In Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence, IOS Press, Amsterdam, pp1099-1100, 2010.

[Bai13] E. Bair, "Semi-supervised clustering methods", Wiley Interdisciplinary Reviews: Computational Statistics, 5(5), pp 349-361, 2013.

[BaScFr10] H. Bauer, J.-L. Scharbarg, C. Fraboul, “Improving the Worst-Case Delay Analysis of an AFDX Network Using an Optimized Trajectory Approach“, IEEE Transactions on Industrial informatics, vol 6, No. 4, November 2010.
[BaScFr12] H. Bauer, J.-L. Scharbarg, C. Fraboul, “Applying Trajectory approach with static priority queuing for improving the use of available AFDX resources”, 48(1), pp101-103, 2012.

[BoNaFu12] M. Boyer, N. Navet and M. Fumey, "Experimental assessment of timing verification techniques for AFDX", Embedded Real-Time Software and Systems (ERTS2 2012), Toulouse, France, February 1-3, 2012.

[BoMiNa11] M. Boyer, J. Migge, and N. Navet, “A simple and efficient class of functions to model arrival curve of packetised flows”, in 1st International Workshop on Worst-case Traversal Time, in conj. with the 32nd IEEE Real-Time Systems Symposium (RTSS 2011), Vienna, November 2011.

[BoRo16] M. Boyer, P. Roux, “Embedding network calculus and event stream theory in a common model”, 21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, 2016, pp. 1-8. doi: 10.1109/ETFA.2016.7733565

[BoTh07] A. Bouillard and E. Thierry, “An algorithmic toolbox for network calculus”, Discrete Event Dynamic Systems, vol. 17, no. 4, October 2007.

[Dav07] R.I. Davis, A. Burns, R.J. Bril, J.J. Lukkien. “Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised”. Real-Time Systems, Volume 35, Number 3, pp. 239-272, April 2007.

[Do14] S. Dolnicar, B. Grün, F. Leisch, K. Schmidt, “Required sample sizes for data-driven market segmentation analyses in tourism”, Journal of Travel Research, vol. 53, no. 3, pp 296-306, 2014.

[DoHuSi09] E.R. Dougherty, J. Hua, C. Sima, “Performance of feature selection methods”. Current genomics, 10(6), pp. 365-374, 2009

[EEN18] Electronics Europe News, “Cadence, Nvidia to apply machine learning to EDA”, available at http://www.eenewseurope.com/news/cadence-nvidia-apply-machine-learning-eda-0, retrieved 17/01/2019.

[Fra19] P. Fradet, X. Guo, J.-F. Monin and S. Quinton, “CertiCAN: A Tool for the Coq Certification of CAN Analysis Results”, to appear at the 25th IEEE Real-Time and Embedded Technology and Applications Symposium, Montreal, 2019.

[JuRo07] I. Juutilainen and J. Röning, “A Method for Measuring Distance from a Training Data Set”, Communications in Statistics - Theory and Methods, 36(14), Taylor & Francis, pp2625-2639, 2007. https://doi.org/10.1080/03610920701271129

[HaKi16] S. Han and H. Kim, “On AUTOSAR TCP/IP Performance in In-Vehicle Network Environments”, in IEEE Communications Magazine, vol. 54, no. 12, pp. 168-173, Dec. 2016.

[HaScFr14] T. Hamza, J.-L. Scharbarg, C. Fraboul, “Priority assignment on an avionics switched Ethernet network (QoS AFDX)”. IEEE International Workshop on Factory Communication Systems - WFCS 2014, May 2014, Toulouse, France.

[HaTiFri09] T. Hastie, R.Tibshirani, J. Friedman, “The Elements of Statistical Learning: Data Mining, Inference, and Prediction”, Springer, 2009.

[IEEE802.1Qav] “IEEE Standard for Local and Metropolitan Area Networks – Virtual Bridged Local Area Networks Amendment 12 Forwarding and Queuing Enhancements for Time-Sensitive Streams,” IEEE Std 802.1Qav-2009, January 2009.

[802.1Qbu] “IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption,” IEEE Std 802.1Qbu-2016, Aug. 2016.

[802.1Qbv] “IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic,” IEEE Std 802.1Qbv-2015 March 2016.

[IEEE802.1Qcr] “IEEE Draft Standard for Local and Metropolitan Area Networks – Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Asynchronous Traffic Shaping”, draft V0.5, J. Specht editor, June 2018.

[IEEE802.1Qci] “IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing”, IEEE Std 802.1Qci-2017, Sept. 2017.

[IEEE802.1CB] “IEEE Standard for Local and metropolitan area networks – Frame Replication and Elimination for Reliability,” IEEE Std 802.1CB-2017, Oct. 2017.

[IEEE802.1Q-2018] IEEE, “IEEE Standard for Local and metropolitan area networks--Bridges and Bridged Networks - IEEE Std 802.1Q-2018”, 2018.

[KaBa06] D. Kazakov, I. Bate, “Towards New Methods for Developing Real-Time Systems: Automatically Deriving Loop Bounds Using Machine Learning”, IEEE Conference on Emerging Technologies and Factory Automation (ETFA), September 2006. doi: 10.1109/ETFA.2006.355425

[LB10] J.-Y. Le Boudec, “Performance Evaluation of Computer and Communication Systems”, EFPL Press, ISBN: 978-2-940222-40-7, 2010. Also available from http://perfeval.epfl.ch/.

[LBTh01] J.-Y. Le Boudec and P. Thiran, “Network Calculus”, ser. LNCS, vol. 2050, Springer Verlag, 2001.

[Li17] J. Li, K. Cheng, S. Wang, F. Morstatter, R.P. Trevino, J. Tang, H. Liu, “Feature selection: A data perspective”, ACM Computing Surveys, 50(6), 2017. https://doi.org/10.1145/3136625

[LiGe16] X. Li, L. George, “Deterministic delay analysis of AVB switched Ethernet networks using an extended Trajectory Approach”, Real-Time Systems, Volume 53, Issue 1, pp 121–186, 2016.

[Ma17] B. Mao, Z.M. Fadlullah, F. Tang, N. Kato. O. Akashi, T. Inoue, K. Miz, “Routing or Computing? The Paradigm Shift Towards Intelligent Computer Network Packet Transmission Based on Deep Learning”, 66(11), May 2017. doi: 10.1109/TC.2017.2709742

[Ma18] F.M. Mahbub ul Islama, M. Lin, L.T.Yang, K.-K.R. Choo, “Task aware hybrid DVFS for multi-core real-time systems using machine learning”, Information Sciences, vol. 433-434, pp315-332, April 2018. doi: https://doi.org/10.1016/j.ins.2017.08.042

[Ma16] H. Mao, M. Alizadeh, I. Menachey, S. Kandula, “Resource Management with Deep Reinforcement Learning”, Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets’16), pp50-56, November 2016.

[MaDaPa18] L. Mai, NN. Dao, M. Park, “Real-Time Task Assignment Approach Leveraging Reinforcement Learning with Evolution Strategies for Long-Term Latency Minimization in Fog Computing”, Sensors 18(9), August 2018. doi: 10.3390/s18092830.

[MaNaMi19a] T.L. Mai, N. Navet, J. Migge, "A Hybrid Machine Learning and Schedulability Method for the Verification of TSN Networks", 15th IEEE International Workshop on Factory Communication System (WFCS 2019), Sundsvall, Sweden, May 27-29, 2019. Preliminary version available as technical report.

[MaNaMi19b] T.L. Mai, N. Navet, J. Migge, “On the use of supervised machine learning for assessing schedulability: application to Ethernet TSN”, in submission, 2019.

[MHMa12] McHugh, L. Mary, "Interrater reliability: the kappa statistic" Biochemia medica: Biochemia medica 22, no. 3, pp 276-282, 2012.

[MiViNaBo18a] J. Migge, J. Villanueva, N. Navet and M. Boyer, “Performance assessment of configuration strategies for automotive Ethernet quality-of-service protocols”, Automotive Ethernet Congress, Munich, January 30-31, 2018.

[MiViNaBo18b] J. Migge, J. Villanueva, N. Navet, M. Boyer, “Insights on the performance and configuration of AVB and TSN in automotive networks”, Proc. Embedded Real-Time Software and Systems (ERTS 2018), Toulouse, France, January 31-February 2, 2018.

[Na14] N. Navet, S. Louvart, J. Villanueva, S. Campoy-Martinez, J. Migge, “Timing verification of automotive communication architectures using quantile estimation“, Embedded Real-Time Software and Systems (ERTS 2014), Toulouse, France, February 5-7, 2014.

[Na18] A. Nasrallah, A. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, H. ElBakoury, “Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research”, IEEE Communications Surveys & Tutorials, in print, 2019. Available on arXiv.org as arXiv:1803.07673v2, 2018.

[NaMi18] N. Navet, J. Migge, "Insights into the performance and configuration of TCP in Automotive Ethernet Networks", 2018 IEEE Standards Association (IEEE-SA) Ethernet & IP @ Automotive Technology Day, London, October 8-9, 2018.

[NaMiViBo18] N. Navet, J. Migge, J. Villanueva, M. Boyer, ONERA “Pre-shaping Bursty Transmissions under IEEE802.1Q as a Simple and Efficient QoS Mechanism”, World WCX™: SAE World Congress Experience, Detroit, USA, March 2018. Extended version to appear in SAE International Journal of Passenger Cars—Electronic and Electrical Systems.

[NaViMi18] N. Navet, J. Villanueva, J. Migge, “Automating QoS protocols selection and configuration for automotive Ethernet networks”, presentation at WCX18: SAE World Congress Experience (WCX018), session “Vehicle Networks and Communication (Part 2 of 2)”, Detroit, USA, April 11, 2018.

[NaViMiBo17] N. Navet, J. Villanueva, J. Migge, M. Boyer, "Experimental assessment of QoS protocols for in-car Ethernet networks", 2017 IEEE Standards Association (IEEE-SA) Ethernet & IP @ Automotive Technology Day, San-Jose, Ca, October 31-November 2, 2017.

[NaSeMi15] N. Navet, J. Seyler, J. Migge, "Timing verification of real-time automotive Ethernet networks: what can we expect from simulation?", presentation at the SAE World Congress 2015, "Safety-Critical Systems" Session, Detroit, USA, April 21-23, 2015.

[NaSeMi16] N. Navet, J. Seyler, J. Migge, "Timing verification of real-time automotive Ethernet networks: what can we expect from simulation?", Embedded Real-Time Software and Systems (ERTS 2016), Toulouse, France, January 27-29, 2016.

[Pao10] G. Paolini, “How to Benchmark Code Execution Times on Intel® IA-32 and IA-64 Instruction Set Architectures”, Intel Corporation, 2010.

[Pe11] F. Pedregosa et al, “ Scikit-learn: Machine Learning in Python”, Journal of Machine Learning Research, vol. 12, pp2825-2830, 2011.

[Pan18] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar, “Introduction to Data Mining”, 2nd edition, Pearson, isbn9780133128901, 2018.

[Peg18] “RTaW-Pegase: Modeling, Simulation and automated Configuration of communication networks“, available at url https://www.realtimeatwork.com/software/rtaw-pegase, retrieved on 28/12/2018.

[Po18] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. Presa Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, “A Survey on Deep Learning: Algorithms, Techniques, and Applications”, ACM Comput. Surv., 51(5), September 2018. doi: https://doi.org/10.1145/3234150

[Que12] R. Queck, “Analysis of Ethernet AVB for automotive networks using Network Calculus”, IEEE International Conference on Vehicular Electronics and Safety (ICVES), Istanbul, July, 2012.

[Ra17] M.L. Raagaard, P. Pop, M. Gutiérrez, W. Steiner, Runtime reconfiguration of time-sensitive networking (TSN) schedules for Fog Computing, IEEE Fog World Congress (FWC), 2017.

[RiGuGH17] J.M. Rivas, J.J. Gutiérrez, and M. González Harbour, "A supercomputing framework for the evaluation of real-time analysis and optimization techniques". Journal of Systems and Software, vol. 124 issue C, pp 120-136, February 2017). doi: https://doi.org/10.1016/j.jss.2016.11.010

[RuBo14] J.A. Ruiz De Azua, M. Boyer "Complete modelling of AVB in Network Calculus Framework" Int. Conf. on Real-Time Networks and Systems (RTNS 2014), Versailles, France, October 2014.

[SOCrSt18] R. Serna Oliver, S. Craciunas, W. Steiner, "IEEE 802.1Qbv Gate Control List Synthesis using Array Theory Encoding", IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Porto, April 2018. doi: 10.1109/RTAS.2018.00008

[ThChNa00] L. Thiele, S. Chakraborty and M. Naedele, "Real-time calculus for scheduling hard real-time systems," 2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings, Geneva, Switzerland, 2000, pp. 101-104 vol.4. doi: 10.1109/ISCAS.2000.858698

[ThErDi15] D. Thiele, R. Ernst, J. Diemer, "Formal worst-case timing analysis of Ethernet TSN's time-aware and peristaltic shapers", IEEE Vehicular Networking Conference (VNC), Kyoto, December 16-18, 2015.

[WiBa13] K. Winstein, H. Balakrishnan, "TCP ex Machina: Computer-Generated Congestion Control", Proc. ACM SIGCOMM Computer Commun. Rev., 43(4), pp.123–34, 2013.

[Wa18] M. Wang, Y. Cui, X. Wang, S. Xiao, J. Jiang, "Machine Learning for Networking: Workflow, Advances and Opportunities", IEEE Network, 32(2), March-April 2018.