# On the use of supervised machine learning for assessing schedulability: application to Ethernet TSN

Tieu Long Mai
University of Luxembourg
Luxembourg
long.mai@uni.lu

Nicolas Navet
University of Luxembourg
Luxembourg
nicolas.navet@uni.lu

Jörn Migge
RealTime-at-Work (RTaW)
France
jorn.migge@realtimeatwork.com

## 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? To get insights into this question, we apply a standard supervised ML technique, k-nearest neighbors (k-NN), and compare its accuracy and running times against precise and approximate schedulability analyses developed in Network-Calculus. The experiments consider different TSN scheduling solutions based on priority levels combined for one of them with traffic shaping. The results obtained on an automotive network topology suggest that k-NN is efficient at predicting the feasibility of realistic TSN networks, with an accuracy ranging from 91.8% to 95.9% depending on the exact TSN scheduling mechanism and a speedup of 190 over schedulability analysis for $10^6$ configurations. Unlike schedulability analysis, ML leads however to a certain rate "false positives" (*i.e.*, configurations deemed feasible while they are not). Nonetheless ML-based feasibility assessment techniques offer new trade-offs between accuracy and computation time that are especially interesting in contexts such as design-space exploration where false positives can be tolerated during the exploration process.

## CCS CONCEPTS

• **Computer systems organization** → **Real-time systems**; • **Networks** → *Link-layer protocols*; • **Computing methodologies** → **Supervised learning**;

## KEYWORDS

Timing verification, schedulability analysis, machine learning, Time Sensitive Networking (TSN).

**ACM Reference format:**
Tieu Long Mai, Nicolas Navet, and Jörn Migge. 2019. On the use of supervised machine learning for assessing schedulability: application to Ethernet TSN. In *Proceedings of 27th International Conference on Real-Time Networks and Systems, Toulouse, France, November 6–8, 2019 (RTNS 2019),* 11 pages.


*RTNS 2019, November 6–8, 2019, Toulouse, France*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

*Context: Ethernet TSN for high-speed real-time communication.* Ethernet is becoming the prominent layer 2 protocol for high-speed communication in real-time systems. However, many of such systems, be it in the industrial, automotive or telecom domains, have strong Quality of Service (QoS) requirements including performance (e.g., latency, synchronization and throughput requirements), reliability (e.g., spatial redundancy) and security (e.g., integrity) that cannot be met with the standard Ethernet technology. The IEEE802.1 TSN TG (Time Sensitive Networking Technical Group), started in 2012, develops technologies to address these QoS requirements. 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 ([17] 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 consult [29].

*Verification of timing constraints.* The two main model-based approaches to assess whether a real-time system meets its temporal constraints are schedulability analysis and simulation. Schedulability analysis, also called feasibility analysis, worst-case analysis, response time analysis or worst-case traversal time analysis (WCTT) 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). A simulation model on the other hand captures the behaviour of the real system through a set of rules, which, along with the sequence of values provided by the random generator, dictate the evolution of the model. To be timing-accurate the model must capture all activities having an impact on the performance metrics, like for example the waiting time of a packet in a network device. The main drawback of simulation is that, even if the coverage of the verification can be adjusted to the requirements (see [30]), it only provides statistical guarantees and not firm guarantees. In the remainder of this work, we focus on schedulability analysis as the verification technique.

*Design-space exploration for further automating the design of embedded architectures.* The complexity of designing and configuring complex embedded systems, like the Electrical and Electronic (E/E) architecture of a vehicle platform, calls for Design-Space Exploration (DSE) algorithms, that is design decisions based on the systematic exploration of the search space. Regarding network design, a first step in that direction is the ZeroConfig-TSN (ZCT) algorithm [28, 33], implemented in the RTaW-Pegase commercial tool [38], that assists designers in the selection and configuration of

TSN protocols. Beyond protocol selection and configuration several other important and difficult design problems are good candidates for DSE, such as the allocation of the functions to the computational resources or even the topological layout of the E/E architecture.

DSE algorithms typically consist of three distinct steps: creating candidate solutions, configuring these solutions and evaluating their performance. 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 350 flows scheduled with three priority levels requires an average 805ms computation time (Intel I7-8700 3.2Ghz) per configuration, resulting in about 224 hours 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.

*Contribution of the paper.* This work explores the extent to which supervised ML techniques can be used to determine whether a real-time Ethernet configuration is schedulable or not. The prediction accuracy and computation time that can be expected from ML are quantified by comparison with a precise and an approximate schedulability analysis. 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 5 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. To facilitate further research, the data and the R code used in this study are available as open-source (AGPL V3.0) at https://github.com/crtes-group-unilux/ML4TSN-Schedulability.

*Organisation.* The remainder of this document is organised as follows. In Section 2, we introduce the TSN network model and define the design problem. In Section 3 we apply a supervised learning algorithm to predict feasibility. Section 4 provides a recap of the results obtained and a comparison with the performances of conventional schedulability analysis. In Section 5, we give an overview of the applications of ML techniques in related domains. Finally, Section 6 provides first insights gained about the use of ML techniques for timing analysis and identifies a number of 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 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 Designing and configuring TSN networks

In the following, a *possible configuration*, or *candidate solution*, refers to a TSN network that has been configured, while a *feasible configuration* is a configured TSN network that meets all the application's constraints. The configuration of a TSN network involves a number of sub-problems:

- Group traffic streams into traffic classes and set the relative priorities of the traffic classes: up to 8 priority levels are permitted with TSN [17],
- Beyond the priorities of the traffic classes, optionally select for each traffic class an additional QoS protocol: shaping using the Credit-Based Shaper (CBS, defined in IEEE802.1Qav), time-triggered transmission with the Time-Aware Shaper (TAS, defined in IEEE802.1Qbv), etc.
- Configure each traffic class: parameters for CBS (i.e., class measurement interval (CMI), values of the "idle slopes" per traffic class and per egress port), Gate Control List (*i.e.*, the transmission schedule) for TAS, etc.
- If frame pre-emption (IEEE802.1Qbu) is used, decide the subset of traffic classes that can be pre-empted by the rest of the traffic classes.

In this study, we consider the following scheduling 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: command and control (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 relative criticality of the streams.
(3) "Concise priorities" with eight 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 algorithm for mono-processor system [4] that has been shown to be optimal with an analysis developed with "the trajectory approach" for the transmission of periodic/sporadic streams in switched Ethernet network [15][1]. 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

[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. Whether OPA remains optimal in the TSN context with other analyses and other formalisms such as Network-Calculus, as used in the paper, or Event-Streams [40] is to the best of our knowledge an open question.

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 segmented messages (e.g., camera frames) are enqueued for transmission. All the other characteristics of the traffic remain unchanged. The principles of the algorithm used to set the idle times, available under the name of "Presh" algorithm in RTaW-Pegase, are described in [32].

These scheduling solutions have been selected to include two main QoS strategies, namely static priority scheduling and traffic shaping, while they can be implemented with basic TSN networking devices offering only the eight priority levels. In addition, those four solutions are compatible with the standard static priority scheduling analysis. Two schedulability analyses in Network-Calculus (NC), based on results published in [8, 9, 37], are used in the experiments:

- The "approximate analysis" which computes WCTTs in linear time with respect to the number of streams.
- The "precise 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.

The reader can refer to [31] for typical computation times and information about the class of (min,+) functions used in each analysis. Importantly, the lower bounds proposed in [6, 7] and used in [10] suggest that the existing NC schedulability analyses for static priority scheduling are precise in terms of the distance between the computed upper bounds and the true worst-case latencies.

## 2.2 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 term "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 are counted as $n$ 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 losses. 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 [35].

## 2.3 Traffic characteristics and network topology

The traffic is made up of three classes whose characteristics are summarized in Table 1. The characteristic of the streams and their proportion is inspired from the case-study provided by an automotive OEM in [27, 34].

**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.**

| | |
|---|---|
| Audio Streams | • 128 or 256 byte frames<br>• periods: one frame each 1.25ms<br>• deadline constraints either 5 or 10ms<br>• proportion: 7/46 |
| Video Streams | • ADAS + Vision streams<br>• 30*1500byte frame each 33ms (30FPS camera for vision)<br>• 15*1000byte frame each 33ms (30FPS camera for ADAS)<br>• 10ms (ADAS) or 30ms (Vision) deadlines<br>• proportion: 7/46 |
| Command & Control | • from 53 to 300 byte frames<br>• periods from 5 to 80ms<br>• deadlines equal to periods<br>• proportion: 32/46 |

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. 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. In this study, we consider deadline constraints: the WCTT of each stream, computed by schedulability analysis, must be less than the stream's deadline.

The topology considered in this study is the one provided by an automotive OEM in [27]. As shown in Figure 1, the network comprises 5 switches and 16 nodes, with a data transmission rate equal to 100Mbps on all links, except for the 1Gbps link from ECU12 to Switch3.

# 3 PREDICTING FEASIBILITY WITH SUPERVISED LEARNING

In this section, we apply a supervised ML algorithm, namely k-Nearest Neighbours (k-NN), to predict whether TSN configurations are feasible or not. k-NN is a simple though powerful machine

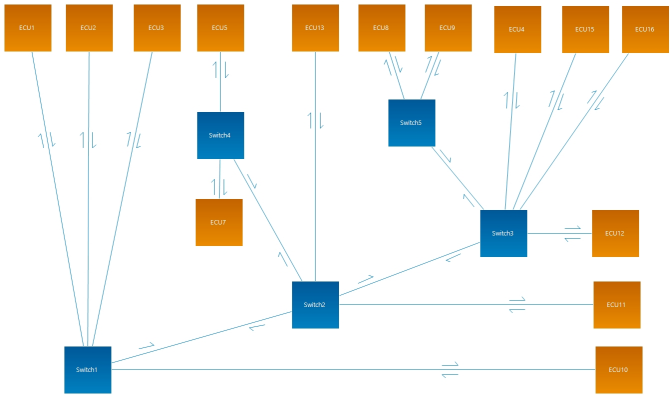


**Figure 1: Topology of the prototype network used in the experiments (topology from [27]).**

learning algorithm, described in standard ML textbooks like [16], that classifies unlabelled (i.e., unseen) data points by placing them in the same category as the majority of their "nearest" neighbours, which are the data points from the training set that are the closest in the feature space.

## 3.1 Supervised learning applied to network classification

Supervised learning is a class of ML algorithms that learn from a training set to predict the value or the label of unseen data. The data in the training set are classified into categories: they are given "labels". In this study, as illustrated in Figure 2, 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.

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" (*i.e.*, configurations deemed non feasible while they actually are). 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.

## 3.2 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, capture the domain-specific knowledge needed for the learning to take place. Once a set of candidate 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 (this is called the "peaking" effect, see [12]). Feature engineering and feature selection have given rise to hundreds of studies over the past two decades (see [12, 20] for good starting points). Feature engineering is typically done with expert knowledge, or it can be automated with feature learning techniques and techniques belonging to deep learning.

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. There are also 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 with the same total number of flows. 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 a new feature from the raw data, the Gini index [13] 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) and the Gini index of the loads of the links.

## 3.3 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:

(1) It stores all data belonging to the training set.
(2) Given an unlabelled data $p$, the distance from $p$ to each training set data $q$ is calculated by the Euclidean distance between the $d$ features of the two data in a $d$-dimensional space: $dist(p,q) = \sqrt{(p_1 - q_1) + (p_2 - q_2) + ... + (p_d - q_d)}$.
(3) Based on the Euclidean distances, the algorithm identifies the $k$ data in the training set that are the "nearest" to the unlabelled data $p$.
(4) 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.

The best value for k cannot be known beforehand, it has to be determined by iterative search (see §3.5). 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 complexity, given $n$ the number of data in the training set, the complexity of prediction for each unlabelled data with k-NN is $O(n)$. For a comparison, the prediction time of two other popular ML classification algorithms, Decision Trees and Support Vector Machines [16], is constant but the worst-case complexity of their training phase is polynomial in the size of the training set.

Visual exploratory data analysis shows that feasible and non-feasible configurations in the training set tend to occupy distinct areas in the feature space (see Figure 4 in [31]). In other words,

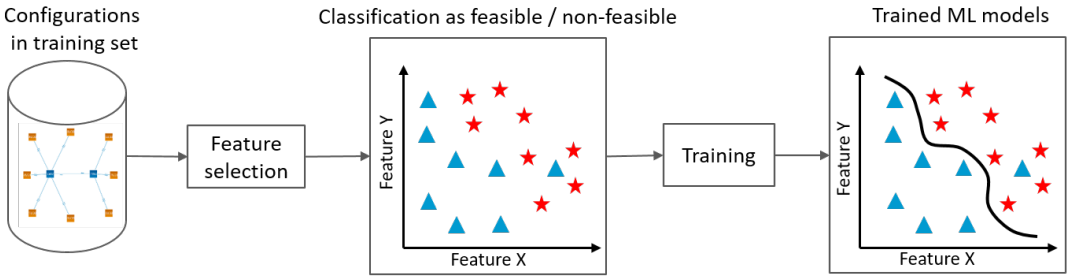


**Figure 2: Configurations in the training set are classified as feasible (blue triangle) or non-feasible (red star) based on the results of the precise schedulability analysis. The ML algorithm then tries to learn the values of the features that are predictive with respect to the feasibility of the configurations. Here the ML algorithm draws a separation between feasible and non-feasible configurations that will be used to classify an unlabelled configuration.**

whether a new configuration is feasible or not can usually be predicted by a majority voting among the closest data points in the feature space. As k-NN relies on the very same classification mechanism, this suggests that it is a technique well suited to our problem. 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 draw boundaries between groups like Support Vector Machines (SVM, see [16]), will not be able to make the right decision [2].

## 3.4 Performance criteria and evaluation technique

The following six metrics are retained to evaluate the performance of the techniques considered in this study:

- 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 that consider 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 False Positive Rate (*FPR*) is the percentage of configurations falsely predicted as feasible among all non-feasible configurations.
- The False Negative Rate (*FNR*) is the percentage of configurations falsely predicted as non feasible among all feasible configurations.
- 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%).

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

To combat model overfitting, that is a model being too specific to the training data and not generalizing well outside, we use cross-validation with the standard k-fold technique. With k-fold, the data set is divided into k equal-size subset. 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 average over all testing sets. k-fold evaluation guarantees that all configurations in the data set are used for prediction, *i.e.*, there is no bias due to the prediction accuracy variability across testing sets. In this study, we perform 5-fold evaluation, *i.e.*, there are 5200 labelled configurations in the training set (see §3.5) and 1300 unlabelled configurations in the testing set. With testing sets of size 1300, applying Theorem 2.4 in [19], the margin of error of the prediction accuracy is less than 2.72% at a 95% confidence level.

## 3.5 Experimental setup

The approach described in §3.1 requires to make certain experimental choices:

*3.5.1 Networks in the training set.* The training set is comprised of random TSN configurations based on the topology and traffic characteristics described in §2.3 with a total number of flows in the set [50, 75, 100, 125, 150,175, 200, 225, 250, 275, 300, 325, 350]. 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 §2.3. When the maximum link load exceeds 1 then the generated configuration is discarded as we know for certain that no feasible scheduling solution exists. With the aforementioned settings, the proportion of feasible and non-feasible configurations with respect to the maximum load over all links is as shown in Figure 3. On average, over the entire training set, the proportion of feasible configurations is 8.85% for FIFO, 22.04% for Manual, 52.77% for CP8 and 55.96% for Preshaping.

*3.5.2 Size of the training set.* A crucial issue is 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,

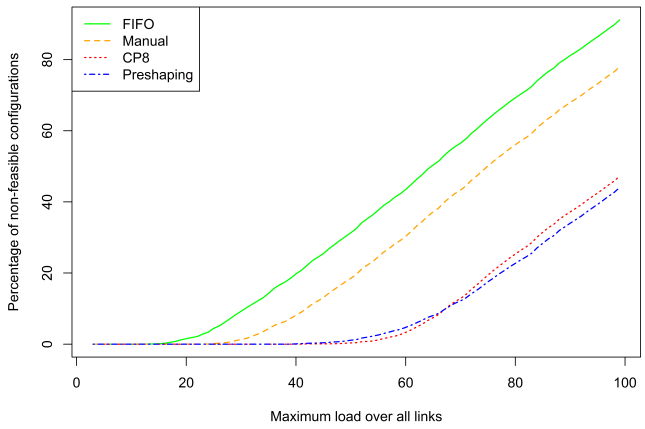


**Figure 3: The solid green, dashed orange, dotted red and dash-dotted blue curves resp. represent the percentage of non-feasible configurations in the training set for FIFO, Manual, CP8, Preshaping scheduling versus the maximum load over all links. Logically, the higher the maximum link load, the more likely a configuration will not be feasible whatever the scheduling solution.**

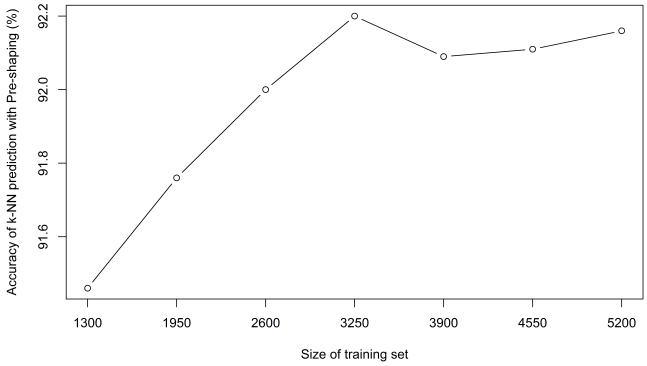


**Figure 4: Prediction accuracy of k-NN (y-axis) versus size of the training set (x-axis). The accuracy increases when the training set grows, however there is a plateau when the training set size reaches 3250. The small fluctuation of the accuracy between 3250 and 5200 (less than 0.11%), can be explained by the non-deterministic characteristics of the TSN configurations evaluated. Results shown for Preshaping scheduling with parameter k for k-NN equal to 20.**

we increase the size of the training set until the prediction accuracy of k-NN does not show significant improvements. As can be seen in Figure 4, the accuracy plateaus past 3250. In the rest of the study, we integrate a safety margin and use training sets of size 5200. This value is set based on 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.

*3.5.3 Min-max normalization.* Like usually done in ML, the five features retained (see §3.2) are rescaled into the range [0,1] based on the minimum and maximum possible value taken by a feature. This normalization allows that all features possess a similar weight in the Euclidean distance calculation.

*3.5.4 Parameter k of k-NN.* Since the number of nearest neighbours k leading to the best k-NN performance is unknown, we tested on the data making up the training set the values of k in the range [10,100] by step 10 and retained for each scheduling solution the best value of k (see column 1 in Table 2). The difference between the best possible accuracy and the one obtained with a value of k either 10 above or below the best k value is always less than 1.17%. This suggests that in our problem k-NN is robust with respect to its parameter k.

## 3.6 Experimental results

*3.6.1 Prediction accuracy.* As can be seen in Table 2 the prediction accuracy of k-NN with the best k values ranges from 91.84% to 95.94%. We note that it tends to decrease when the complexity of the scheduling mechanism increases. The more powerful the scheduling mechanism in terms of feasibility, we have here FIFO < Manual < CP8 < Preshaping (see last column of Table 2), the harder it is to predict its outcome on a given configuration, except for the accuracy of CP8, which is marginally better than Manual.

**Table 2: Performance of k-NN for the different scheduling solutions with the best k values (k*). Accuracy (Acc), True Positive, True Negative Rates and Kappa statistics (K) obtained by 5-fold evaluation with testing sets of size 1300 each. Experiments done with R package *class* v7.3-14. The last column (F) is the percentage of feasible configurations for each scheduling solution in the training set of 5200 configurations.**

| | k* | Acc(%) | TPR(%) | TNR(%) | K | F(%) |
|---|---|---|---|---|---|---|
| FIFO | 40 | 95.94 | 72.82 | 98.16 | 67.32 | 8.54 |
| Manual | 30 | 94.23 | 84.42 | 97.0 | 82.74 | 22.08 |
| CP8 | 30 | 94.65 | 94.88 | 94.39 | 89.84 | 51.38 |
| Preshaping | 60 | 91.84 | 92.23 | 91.36 | 83.38 | 54.62 |

A prediction is wrong either due to a false positive or false negative. As Table 2 shows, there are differences between TPR and TNR across scheduling solutions. With FIFO, the TNR is much higher than the TPR. The reason may be due to the imbalance between feasible and non-feasible configurations in the training set. Indeed, with FIFO, non-feasible configurations largely outnumber feasible configurations with a proportion of 91.46% (versus 45.38% non-feasible solutions with Preshaping). 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. We may also be concerned that with the imbalances in the testing set, 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 answer this question, we calculate the Kappa statistic, or 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% is usually considered significant [26]. Table 2 includes the Kappa coefficients of k-NN prediction for the best value of k. Since the coefficients range from 67.32% to 83.38%, this suggests that the high accuracy of k-NN prediction is not obtained by chance alone.

*3.6.2 Robustness of prediction.* 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. We studied the extent to which changes in the traffic characteristics would 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 $[(1-x)\cdot s, (1+x)\cdot s]$. The average network load remains similar whatever the value of $x$ though, as the variations are both positive and negative with the same intensity in both cases.

In nine successive experiments with testing set of size 1300, the value of $x$ is set from 0.1 to 0.9 by step of 0.1. Figure 5 summarizes the results obtained with the best k values, using boxplots, with the horizontal lines being the baseline prediction accuracies without changes in the training set. Although, the performance of k-NN tends to decrease with these changes, the deterioration is limited: less than 2.2% whatever the scheduling solution and the value of $x$. A 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.

## 4 COMPARISON WITH SCHEDULABILITY 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. Even if prior works [7, 10] suggest that the precise analysis for static priority is tight, its pessimism will create a certain amount of configurations deemed unfeasible while they are. This phenomenon introduces a bias in the accuracy results presented here, which leads to the FPR to be overestimated and the TNR to be underestimated. As our main concern are false positives, the actual risk with supervised learning may be less than in the results shown hereafter, though it cannot be precisely quantified.

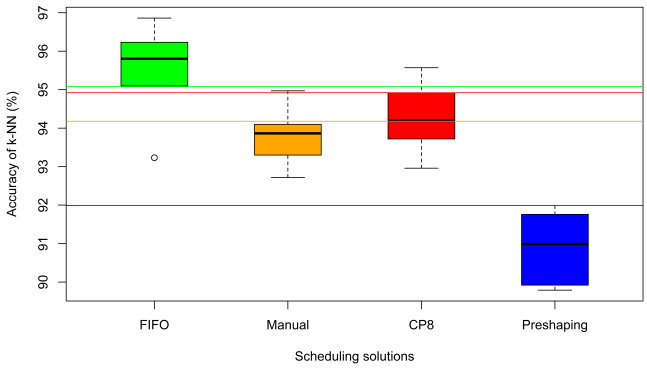


**Figure 5: Boxplot of the prediction accuracies obtained with k-NN with positive and negative variations of the data payload range for the critical flows in the testing set between 10% and 90% by step of 10%. Each box summarizes the results of the 9 experiments. The horizontal lines are the baseline accuracies obtained when the traffic parameters used to generate the training and the testing set are identical.**

Table 3 shows a summary of the accuracy obtained with the different techniques. k-NN is less accurate than the approximate analysis for the simple FIFO scheduling (−2.98%) and Manual (−2.31%), but outperforms it for CP8 scheduling (+8.34%). These results suggest that ML is an alternative to approximate analysis for design space exploration algorithms involving the creation of a large number of candidate solutions.

| | **k-NN** | **Approx. analysis** | **Prec. analysis** |
|---|---|---|---|
| Approach | Sup. learning | Sched. analysis | Sched. analysis |
| FIFO | Acc.: 95.94%<br>TPR:72.82% TNR:98.16%<br>FPR: 1.84% | Acc.: 98.92%<br>TPR:87.39% TNR:100%<br>FPR: 0.0% | Acc. :100%<br>TPR:100% TNR:100%<br>FPR: 0.0% |
| Manual | Acc.: 94.23%<br>TPR:84.42% TNR:97.0%<br>FPR: 3.0% | Acc.: 96.54%<br>TPR:84.32% TNR:100%<br>FPR: 0.0% | Acc. :100%<br>TPR:100% TNR:100%<br>FPR: 0.0% |
| CP8 | Acc.: 94.65%<br>TPR:94.88% TNR:94.39%<br>FPR: 5.61% | Acc.: 86.31%<br>TPR:75.35% TNR:100%<br>FPR: 0.0% | Acc. :100%<br>TPR:100% TNR:100%<br>FPR: 0.0% |
| Preshaping | Acc.: 91.84%<br>TPR:92.23% TNR:91.36%<br>FPR: 8.64% | Acc.: NA<br>TPR:NA TNR:NA<br>FPR: 0.0% | Acc. :100%<br>TPR:100% TNR:100%<br>FPR: 0.0% |

**Table 3: Summary of accuracy results. The worst accuracy (86.31% for CP8) is obtained with the approximate schedulability analysis, however, unlike with ML algorithms, none of the two schedulability analysis lead to false positive. Results are not available for the approximate analysis with Preshaping scheduling as the toolset does not support it.**

In certain contexts, especially in design-space exploration, the choice of the verification technique should also consider computation time besides accuracy. Figure 6 shows the trade-offs between

accuracy and running times obtained with each of the techniques for Manual scheduling with computation on a single CPU core. Considering Manual scheduling provides the fairer picture of the different trade-offs. Indeed, the FIFO analysis is much simpler than the others, while for CP8 and Preshaping the time needed to assess the feasibility of a configuration includes computation that is not purely related to schedulability analysis but parameters setting (resp. priorities and shaping parameters). They both 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).

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

- For a small number of configurations ($10^3$ here) the machine learning algorithm experimented is not competitive with precise and approximate schedulability analysis as it is both slower and less accurate. Approximate analysis and precise analysis offer Pareto optimal trade-offs between accuracy and running times. For a medium number of configurations ($10^5$ here), k-NN is still dominated by the approximate analysis. In many contexts however, like for Preshaping in this study, an efficient approximate analysis will not be available and then k-NN becomes competitive for a relatively small number of configurations. As k-NN execution time is negligible compared to schedulability analysis, this is basically the case as soon as the number of configurations tested is larger than the size of the training set (5200 in this study).
- For a large number of configurations ($10^6$ 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 large execution times. For instance, with Manual scheduling, the precise analysis would take about 132 hours compared to less than 3 hours for the approximate analysis and 42 minutes for supervised learning, which corresponds to a speedup of 190 in favor of ML over precise analysis. Once the training time has been amortized, machine learning techniques are very fast even for very large number of configurations.

It should be borne in mind that both the approximate and precise TSN schedulability analyses are very fast on the TSN configurations tested (resp. 14.77ms and 804.52ms on average for 350 flows). The areas of maximal efficiency of the different techniques will thus be different for other schedulability analyses and types of configurations. For instance, with the simpler network topology used in the technical report [31], k-NN is already Pareto optimal for $10^5$ configurations.

## 5 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 [1, 2]. 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 [36] 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 [41] 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 [3] 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 [24] 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 [42] 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 less numerous. As early as 2006, [18] 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 [5]. Then [14] proposes the use of Deep Learning Neural Networks to predict the rate of interference suffered by a task in a multicore systems. Recently, a line of work has been devoted to ML algorithms for Dynamic Voltage and Frequency Scaling (DVFS) in battery-powered devices. For instance, [21] 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 [22] implements a neural network trained by reinforcement learning with evolutionary strategies to schedule real-time tasks in fog computing infrastructures, while in [25] 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 [39] 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 open-source framework dedicated to the study of task scheduling algorithms is similar in the spirit to the RTaW-Pegase tool [38] used in this study, that can be operated through a JAVA API and that includes a synthetic problem generation module 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.

## 6 DISCUSSION AND PERSPECTIVES

This study shows that standard supervised ML techniques can be efficient at predicting the feasibility of realistic TSN network configurations in terms of computation time and accuracy. In particular our experiments show that ML techniques outperform in some contexts 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

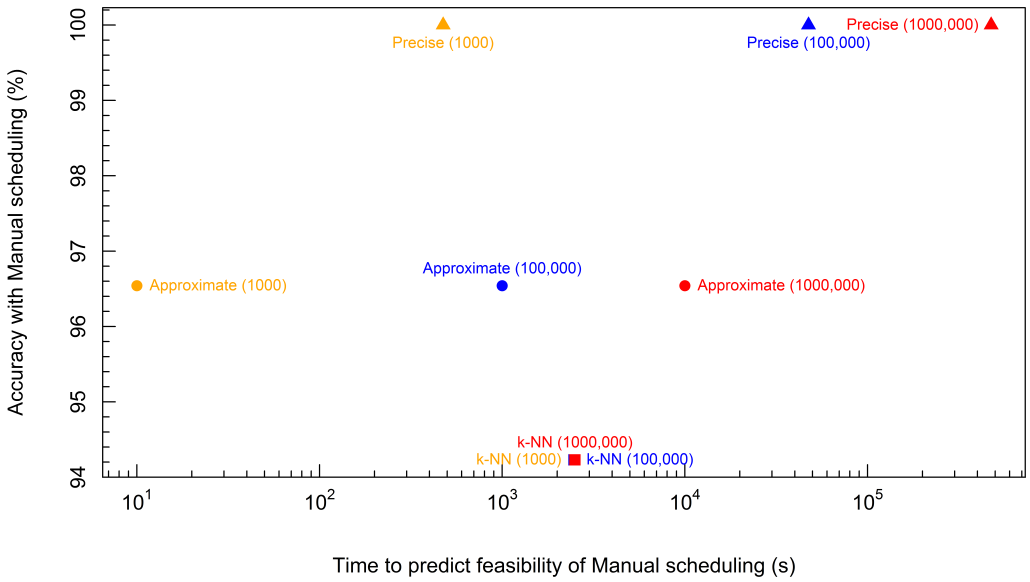


**Figure 6: Accuracy (%) versus total running time (seconds in log scale) on a single computational core for $10^3$ (yellow), $10^5$ (blue) and $10^6$ (red) TSN configurations with Manual scheduling. Running times are averages over sets of configurations with the number of flows ranging from 50 to 350. Squares, points and triangles identify results with k-NN, 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. Experiments conducted on one core of an Intel I7-8700 processor with k-NN algorithm from the *class* v7.3-14 R package and schedulability analysis from RTaW-Pegase v3.4.3 Java library [38].**

nor important computing power, they can be part of the toolbox of the network designers and run on standard desktop computers. 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", from 1.84% to 8.64% in our experiments. This may not be a problem in a design-space exploration process as long as the few retained solutions at the end, the ones proposed to the designer, 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) 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 (see experiments in [23]). In our experiments, k-NN proved to be robust to departure from the data payload assumptions. 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.

This study can be extended in several other directions:

- In order to minimize the rate of false positives, a measure of the uncertainty of prediction can be used to decide to drop a prediction if the uncertainty is too high and rely instead on a conventional schedulability analysis. This leads to envision hybrid feasibility verification algorithms where the clear-cut decisions are taken by ML algorithms, while the more difficult ones are taken by conventional schedulability analysis. This approach is explored in [23].
- 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 [11]. Also promising are "ensemble methods" which combine the results of several ML algorithms, for instance by majority voting.

- Semi-supervised learning, making use for the training set of a small amount of labelled data together with a large amount of unlabelled data, may prove to be more efficient that supervised learning for 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, semi-supervised learning 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 algorithm used in this work 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 that automate feature extraction. However, to do so, deep learning algorithms may require much larger training set.

In this work, ML is applied to predict the feasibility with respect to deadline constraints verified by schedulability analysis. There are other timing constraints that can only be verified by simulation, such as throughput constraints with complex protocols like TCP. ML could also be efficient in such contexts 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.

The 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. Besides the feasibility problem, ML has the potential to offer solutions to other difficult problems in the area of real-time networking. In our view, it could be especially helpful for admission control in real-time, 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 [25], is resource allocation. In the context of TSN, ML is a good candidate to help build bandwidth-effective time-triggered communication schedules for IEEE802.1Qbv.

## REFERENCES


[1] M. Allamanis, E.T. Barr, P. Devanbu, and C. Sutton. 2018. A Survey of Machine Learning for Big Code and Naturalness. *Comput. Surveys* 51, 4, Article 81 (July 2018), 37 pages. http://doi.acm.org/10.1145/3212695

[2] H. A. Amir, K. William, J. Cavazos, G. Palermo, and C. Silvano. 2018. A Survey on Compiler Autotuning using Machine Learning. *CoRR* abs/1801.04405 (2018). arXiv:1801.04405 http://arxiv.org/abs/1801.04405

[3] N. Arouche, A. Bruno, K. Veenstra, W. Ballenthin, S. Lukin, and K. Obraczka. 2014. A machine learning framework for TCP round-trip time estimation. *EURASIP Journal on Wireless Communications and Networking* 2014, 1 (26 Mar 2014), 47. https://doi.org/10.1186/1687-1499-2014-47

[4] N. C. Audsley. 2001. On Priority Asignment in Fixed Priority Scheduling. *Inf. Process. Lett.* 79, 1 (May 2001), 39–44. http://dx.doi.org/10.1016/S0020-0190(00)00165-4

[5] M. Bartlett, I. Bate, and J. Cussens. 2010. Instruction Cache Prediction Using Bayesian Networks. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1099–1100. http://dl.acm.org/citation.cfm?id=1860967.1861224

[6] H. Bauer, J. Scharbarg, and C. Fraboul. 2010. Improving the Worst-Case Delay Analysis of an AFDX Network Using an Optimized Trajectory Approach. *IEEE Transactions on Industrial Informatics* 6, 4 (Nov 2010), 521–533. https://doi.org/10.1109/TII.2010.2055877

[7] H. Bauer, J.-L. Scharbarg, and C. Fraboul. 2012. Applying Trajectory approach with static priority queuing for improving the use of available AFDX resources. *Real-Time Systems* 48, 1 (01 Jan 2012), 101–133. https://doi.org/10.1007/s11241-011-9142-9

[8] A. Bouillard and E. Thierry. 2008. An Algorithmic Toolbox for Network Calculus. *Discrete Event Dynamic Systems* 18, 1 (01 Mar 2008), 3–49. https://doi.org/10.1007/s10626-007-0028-x

[9] M. Boyer, J. Migge, and N. Navet. 2011. 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)*.

[10] M. Boyer, N. Navet, and M. Fumey. 2012. Experimental assessment of timing verification techniques for AFDX. In *6th European Congress on Embedded Real Time Software and Systems*. Toulouse, France. https://hal.archives-ouvertes.fr/hal-01345472

[11] T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[12] E.R. Dougherty, J. Hua, and C. Sima. 2009. Performance of Feature Selection Methods. *Current Genomics* 10, 6 (2009), 365–374. https://doi.org/10.2174/138920209789177629

[13] F. Farris. 2010. The Gini index and measures of inequality. *The American Mathematical Monthly* 117, 10 (2010), 851–864.

[14] David Griffin, Benjamin Lesage, Iain Bate, Frank Soboczenski, and Robert I. Davis. 2017. Forecast-Based Interference: Modelling Multicore Interference from Observable Factors. In *25th International Conference on Real-Time Networks and Systems (RTNS 2017)*.

[15] T. Hamza, J. Scharbarg, and C. Fraboul. 2014. Priority assignment on an avionics switched Ethernet Network (QoS AFDX). In *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*. 1–8. https://doi.org/10.1109/WFCS.2014.6837580

[16] T. Hastie, R. Tibshirani, and J.H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

[17] IEEE. 2018. *IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks* (IEEE Std 802.1Q-2018 ed.).

[18] D. Kazakov and I. Bate. 2006. Towards New Methods for Developing Real-Time Systems: Automatically Deriving Loop Bounds Using Machine Learning. In *2006 IEEE Conference on Emerging Technologies and Factory Automation*. 421–428. https://doi.org/10.1109/ETFA.2006.355425

[19] J.-Y. Le Boudec. 2010. *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland.

[20] J. Li, K. Cheng, S. Wang, F. Morstatter, R.P. Trevino, J. Tang, and H. Liu. 2017. Feature Selection: A Data Perspective. *ACM Comput. Surv.* 50, 6, Article 94 (Dec. 2017), 45 pages. https://doi.org/10.1145/3136625

[21] F.M. Mahbub ul Islam, M. Lin, L.T. Yang, and R. C. Kim-Kwang. 2018. Task aware hybrid DVFS for multi-core real-time systems using machine learning. *Information Sciences* 433-434 (2018), 315 – 332. https://doi.org/10.1016/j.ins.2017.08.042

[22] L. Mai, N.-N. Dao, and M. Park. 2018. Real-Time Task Assignment Approach Leveraging Reinforcement Learning with Evolution Strategies for Long-Term Latency Minimization in Fog Computing. *Sensors* 18, 9 (2018). https://doi.org/10.3390/s18092830

[23] T. L. Mai, N. Navet, and J. Migge. 2019. A Hybrid Machine Learning and Schedulability Analysis Method for the Verification of TSN Networks. In *15th IEEE International Workshop on Factory Communication Systems (WFCS)*. https://doi.org/10.1109/WFCS.2019.8757948

[24] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani. 2017. Routing or Computing? The Paradigm Shift Towards Intelligent Computer Network Packet Transmission Based on Deep Learning. *IEEE Trans. Comput.* 66, 11 (Nov 2017), 1946–1960. https://doi.org/10.1109/TC.2017.2709742

[25] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. ACM, New York, NY, USA, 50–56. https://doi.org/10.1145/3005745.3005750

[26] M.L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica: Biochemia medica* 22, 3 (2012), 276–282.

[27] J. Migge, J. Villanueva, N. Navet, and M. Boyer. 2018. Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks. In *Embedded Real-Time Software and Systems (ERTS 2018)*. Toulouse, France. https://hal.archives-ouvertes.fr/hal-01746132

[28] J. Migge, J. Villanueva, N Navet, and M. Boyer. 2018. Performance assessment of configuration strategies for automotive Ethernet quality-of-service protocols. In

*Automotive Ethernet Congress*. Munich.

[29] A. Nasrallah, A. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. El Bakoury. 2019. Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research. *IEEE Communications Surveys Tutorials* 21, 1 (2019). https://doi.org/10.1109/COMST.2018.2869350

[30] N. Navet, S. Louvart, J. Villanueva, S. Campoy-Martinez, and J. Migge. 2014. Timing verification of automotive communication architectures using quantile estimation. In *Embedded Real-Time Software and Systems (ERTS 2014)*. Toulouse, France. http://nicolas.navet.eu/publi/ERTSS2014_quantiles.pdf

[31] N Navet, T.L. Mai, and J. Migge. 2019. *Using Machine Learning to Speed Up the Design Space Exploration of Ethernet TSN networks*. Technical Report. University of Luxembourg. http://hdl.handle.net/10993/38604

[32] N. Navet, J. Migge, J. Villanueva, and M. Boyer. 2018. Pre-shaping Bursty Transmissions under IEEE802.1Q as a Simple and Efficient QoS Mechanism. *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 11 (04 2018). https://doi.org/10.4271/2018-01-0756 Authors preprint available at http://www.realtimeatwork.com/wp-content/uploads/preprint-JPCE-Preshaping-2018-web.pdf.

[33] N. Navet, J. Villanueva, and J. Migge. 2018. Automating QoS protocols selection and configuration for automotive Ethernet networks. In *SAE World Congress Experience (WCX018), session "Vehicle Networks and Communication (Part 2 of 2)"*. Detroit, USA.

[34] N. Navet, J. Villanueva, J. Migge, and M. Boyer. 2017. Experimental assessment of QoS protocols for in-car Ethernet networks. In *2017 IEEE Standards Association (IEEE-SA) Ethernet & IP @ Automotive Technology Day*. San-Jose, Ca.

[35] Plexxi. 2016. Latency in Ethernet Switches. Retrieved 2019/04/25, http://www.plexxi.com/wp-content/uploads/2016/01/Latency-in-Ethernet-Switches.pdf.

[36] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M.P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar. 2018. A Survey on Deep Learning: Algorithms, Techniques, and Applications. *ACM Comput. Surv.* 51, 5, Article 92 (Sept. 2018), 36 pages. http://doi.acm.org/10.1145/3234150

[37] R. Queck. 2012. Analysis of Ethernet AVB for automotive networks using Network Nalculus. In *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*. 61–67. https://doi.org/10.1109/ICVES.2012.6294261

[38] RealTime-at-Work. 2009. RTaW-Pegase: Modeling, Simulation and automated Configuration of communication networks. Retrieved 2019/01/24, https://www.realtimeatwork.com/software/rtaw-pegase.

[39] J.M. Rivas, J.J. Gutiérrez, and M. González Harbour. 2017. A supercomputing framework for the evaluation of real-time analysis and optimization techniques. *Journal of Systems and Software* 124 (2017), 120 – 136. https://doi.org/10.1016/j.jss.2016.11.010

[40] D. Thiele, P. Axer, and R. Ernst. 2015. Improving Formal Timing Analysis of Switched Ethernet by Exploiting FIFO Scheduling. In *Proceedings of the 52Nd Annual Design Automation Conference (DAC '15)*. ACM, New York, NY, USA, 41:1–41:6. https://doi.org/10.1145/2744769.2744854

[41] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang. 2018. Machine Learning for Networking: Workflow, Advances and Opportunities. *IEEE Network* 32, 2 (March 2018), 92–99. https://doi.org/10.1109/MNET.2017.1700200

[42] K. Winstein and H. Balakrishnan. 2013. TCP Ex Machina: Computer-generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. ACM, New York, NY, USA, 123–134. https://doi.org/10.1145/2486001.2486020