A Configuration Framework for Multi-level Preemption Schemes in Time Sensitive Networking

Mubarak Ojewale
Patrick Meumeu Yomsi
Luís Almeida

CISTER-TR-220606

2022

A Configuration Framework for Multi-level Preemption Schemes in Time Sensitive Networking

Mubarak Ojewale, Patrick Meumeu Yomsi, Luís Almeida

CISTER Research Centre

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: mkaoe@isep.ipp.pt, pmy@isep.ipp.pt, lda@fe.up.pt

https://www.cister-labs.pt

Abstract

To reduce the latency of time-sensitive flows in Ethernet networks, the IEEE TSN Task Group introduced the IEEE 802.1Qbu Standard, which specifies a 1-level preemption scheme for IEEE 802.1 networks. Recently, serious limitations of this scheme w.r.t. flows responsiveness were exposed and the so-called multi-level preemption approach was proposed to address these drawbacks. As is the case with most, if not all, real-time and/or time-sensitive preemptive systems, an appropriate priority-to-flow assignment policy plays a central role in the resulting performance of both 1-level and multi-level preemption schemes to avoid the over-provisioning and/or the sub-optimal use of hardware resources. Yet on another front, the multi-level preemption scheme raises new configuration challenges. Specifically, the right number of preemption level(s) to enable for swift transmission of flows; and the flow-to-preemption-class assignment synthesis remain open problems. To the best of our knowledge, there is no prior work in the literature addressing these important challenges. In this work, we address these three challenges. We demonstrate the applicability of our proposed solution by using both synthetic and real-life use-cases. Our experimental results show that multi-level preemption schemes improve the schedulability of flows by over 12% as compared to a 1-level preemption scheme, and at a higher abstraction level, the proposed configuration framework improves the schedulability of flows by up to 6% as compared to the dominant Deadline Monotonic Priority Ordering.

A Configuration Framework for Multi-level Preemption Schemes in Time Sensitive Networking

Mubarak Adetunji Ojewale
mkaoe@isep.ipp.pt
CISTER Research Center, ISEP, Polytechnic Institute of Porto
Portugal

Patrick Meumeu Yomsi
pmy@isep.ipp.pt
CISTER Research Centre, ISEP, Polytechnic Institute of Porto, Portugal

Luis Almeida
lda@fe.up.pt
CISTER Research Centre and DEEC/FEUP, University of Porto, Portugal

ABSTRACT

To reduce the latency of time-sensitive flows in Ethernet networks, the IEEE TSN Task Group introduced the IEEE 802.1Qbu Standard, which specifies a 1-level preemption scheme for IEEE 802.1 networks. Recently, serious limitations of this scheme w.r.t. flows responsiveness were exposed and the so-called *multi-level preemption approach* was proposed to address these drawbacks. As is the case with most, if not all, real-time and/or time-sensitive preemptive systems, an appropriate *priority-to-flow assignment* policy plays a central role in the resulting performance of both 1-level and multi-level preemption schemes to avoid the over-provisioning and/or the sub-optimal use of hardware resources. Yet on another front, the multi-level preemption scheme raises new configuration challenges. Specifically, the *right number of preemption level(s)* to enable for swift transmission of flows; and the *flow-to-preemption-class assignment* synthesis remain open problems. To the best of our knowledge, there is no prior work in the literature addressing these important challenges. In this work, we address these three challenges. We demonstrate the applicability of our proposed solution by using both synthetic and real life use-cases. Our experimental results show that multi-level preemption schemes improve schedulability of flows by over 12% as compared to a 1-level preemption scheme, and at a higher abstraction level, the proposed configuration framework improves the schedulability of flows by up to 6% as compared to the dominant Deadline Monotonic Priority Ordering.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Redundancy*; • **Networks** → Network reliability.

KEYWORDS

Priority Assignment, Frame Preemption, Configuration, TSN

ACM Reference Format:
Mubarak Adetunji Ojewale, Patrick Meumeu Yomsi, and Luis Almeida. 2022. A Configuration Framework for Multi-level Preemption Schemes in Time Sensitive Networking. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems (RTNS '22), June 7–8, 2022, Paris, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534879.3534891

1 INTRODUCTION

The IEEE Time Sensitive Networking (TSN) Task Group has recently published a set of standards to patch Ethernet with features that allow to satisfy the stringent timing, bandwidth and Quality of Service (QoS) requirements of emerging real-time application like the Advanced Driver-Assistance Systems (ADAS) and Adaptive cruise control (ACC) in the automotive domain. This set of standards are referred to as "TSN standards" [19] or simply "TSN" hereafter. TSN ensures that time-sensitive Ethernet flows satisfy their timing requirements by using two different approaches: the *time-triggered approach* and the *event-triggered approach.* The time-triggered approach requires that network nodes are synchronized and transmission decisions are based on static (pre-computed) time schedules. In this category, the Time Aware Shaper (TAS) (defined in IEEE 802.1Qbv standard [20]) is the most prominent. Another notable mention is the Cyclic Queue Forwarding (CQF) (defined by the IEEE 802.1Qch standard [23]). While the time-triggered approach ensures predictable, low-latency, and low-jitter transmission of Ethernet frames, it exhibits a number of challenges. Among these is the complex configuration synthesis of the dispatch schedule at each node and the significant link utilization trade-off [34]. In addition, the time-synchronization requirement is not needed (nor applicable) to some real-time systems, especially in use-cases that require dynamic behavior and/or unsynchronized end devices. In these types of systems, the event-triggered approach is often adopted. Here, the transmission of a flow is expedited as soon as the network conditions warrant such. A prominent example in this category is *frame preemption* (defined in the IEEE 802.1Qbu [22] and in the IEEE 802.3br [21] standards). Other notable examples include the *Credit Based Shaper* (CBS) (defined by the IEEE 802.1Qav standard [4]) and the *Asynchronous Traffic Shaper* (ATS) (defined by the IEEE 802.1Qcr standard [24]).

In the IEEE 802.1Qbu and the IEEE 802.3br standards, the TSN frame preemption mechanism is specified through the so-called 1-*level preemption scheme* as follows. The frames are organized in two classes: (*i*) the *express frames*, which are considered urgent and therefore eligible for expedited transmission and (*ii*) the *preemptable frames*, which are considered less urgent. Concretely, express frames can preempt preemptable frames. Still, two frames in the same class cannot preempt one another, thus some blocking remains. Several studies have shown that such a scheme significantly improves flows' schedulability and that its performance is comparable to that of TAS, which is more complex and expensive to implement [16, 18, 45]. Recently, Ojewale et al. [32, 33]; Gogolev and Bauer[16]; Ashjaei et al. [2]; and Lo Bello et al. [6] pointed out several limitations of the 1-level preemption scheme as specified in the standards. Most importantly, these authors showed that this scheme is vulnerable to performance degradation when the number of express frames is high. In addition, preemptable frames with firm timing requirements may suffer from long blocking periods due to priority inversion since frames in the same preemption class cannot preempt one another. Figure 1 illustrates such a scenario.

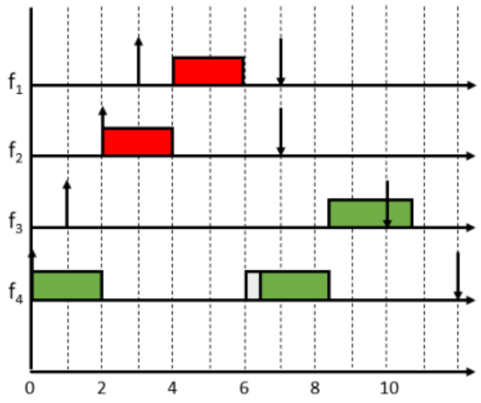


Figure 1: Priority inversion issue with the 1-level.

In Figure 1, four flows (f_1, f_2, f_3, f_4) are considered and the flow priorities are set such that "the smaller the index of a flow the higher its priority", i.e., f_1 is assigned the highest priority and f_4 the lowest priority. An upward arrow represents the arrival time of a flow and a downward arrow represents the deadline. We assume that the flows are transmitted by following a 1-level preemption scheme and are organized in two preemption classes: (1) the *highest preemption class (express)*, wherein frames are represented by using "red boxes"; and (2) the *lowest preemption class (preemptable)*, wherein frames are represented by using "green boxes". The light gray boxes represent the cost associated to the occurrence of a preemption. Finally, we assume that the deadline of flow f_3 is firm. In this scenario, flow f_4 arrives first and starts its transmission. At time $t = 1$, flow f_3 (with a higher priority than f_4) arrives but it cannot preempt f_4 because they belong to the same preemption class. At time $t = 2$, f_2 (which is express) arrives and preempts the transmission of f_4. At time $t = 3$, f_1 (with a higher priority than f_2) arrives but it cannot preempt f_2 because they belong to the same preemption class. Finally, upon the completion of the express flows, f_4 resumes its transmission despite f_3 is ready and pending. This is due to the actual specification of the 1-level preemption scheme and illustrates a priority inversion. Finally, due to this long blocking, f_3 misses its deadline.

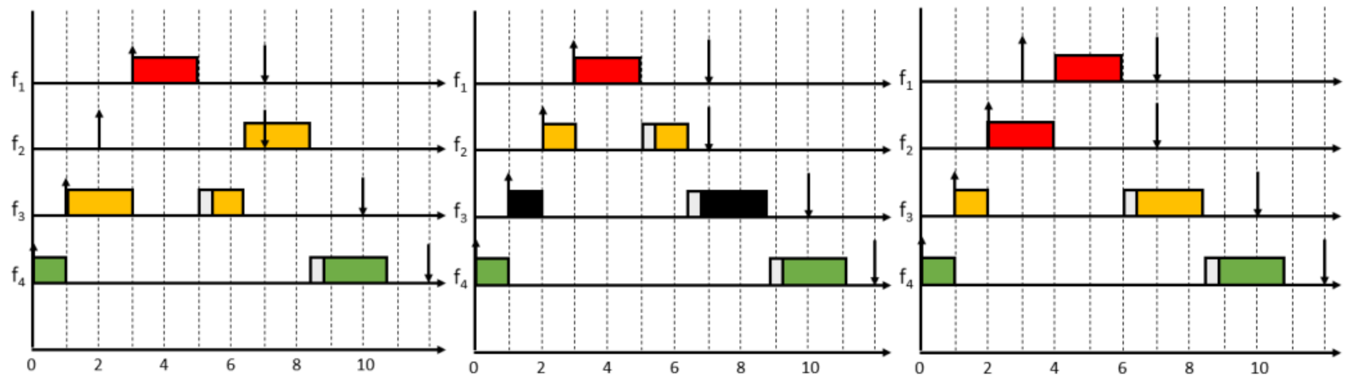
The above-described limitations imply that the 1-level preemption scheme does not provide the means to efficiently support the coexistence of flows with diverse timing requirements in the same network. To get around these obstacles, Ojewale et al. [32, 33] proposed the so-called multi-level preemption scheme, which brought about non-negligible performance improvement over the 1-level scheme [34]. Specifically, this new scheme allows for time-sensitive preemptable frames to preempt other lower-priority ones upon an educated frame-to-preemption class mapping strategy.

As is the case with most, if not all, real-time and/or time-sensitive preemptive systems, an appropriate *priority-to-flow assignment* policy plays a central role in the resulting performance of both 1-level and multi-level preemption schemes to avoid the over-provisioning and/or the sub-optimal use of hardware resources. In this scope, the so-called *Audsley's Optimal Priority Assignment algorithm* (AOPA) has become the reference in many real-time systems[1] provided that there are no priority inversions [13, 38]. This is not the case in preemptive TSN [32, 34]. Indeed, Davis et al. [14] noted that AOPA is not applicable in fixed priority schemes with differed preemptions and/or preemption thresholds, and this is the case with preemptive TSN. Consequently, the so-called *Deadline Monotonic Priority Ordering* (DMPO) [14] is often employed in practice and is known to dominate most of the other priority assignment heuristics in terms of schedulability [25]. Nonetheless, DMPO is not suited for priority assignment in preemptive TSN neither, since it provides a fully ordered priority list for the flowset. Ethernet supports up to eight priority levels only. With this limitation, a fully ordered priority list will lead to another bin packing problem, which is known to be strongly NP-Complete [27]. An efficient priority assignment scheme should not only provide the best possible priority order for flows but should also assign multiple flows into the same priority levels in the best possible manner.

Note that *multi-level preemption scheme* brings about a whole new dimension to the configuration synthesis in addition to the priority assignment challenge. In fact, under a 1-level preemption scheme, configuration decisions are somewhat simple and straightforward. They are limited to deciding whether each flow belongs to the express class or to the preemptable class. The picture darkens considerably when the multi-level preemption scheme is adopted. Here, the system designer must provide answers to two key questions: (1) how to define the number of preemption levels to enable? and (2) how to proceed with the flow-to-preemption-class mapping? In response to these concerns, it is worth noting that while Ethernet can support at most a seven-level preemption scheme[2], each additional preemption level comes with significant hardware overheads that can increase the cost of switch manufacturing [33, 39]. To ensure an optimal use of hardware resources, the system designer must ensure that just the needed number of preemption levels are supported for the transmission of flows over the network. Figure 2 highlights the importance of the preemption level synthesis problem.

In Figure 2, we consider the same four flows (f_1, f_2, f_3, f_4); priority assignment; and release scenario as in Figure 1. However, we now assume four possible preemption classes: (1) the *highest preemption class*, wherein frames are represented by using "red boxes"; (2) the *medium-higher preemption class*, wherein frames are represented by using "yellow boxes"; (3) the *medium-lower preemption class*, wherein frames are represented by using "black

[1]Here "optimality" refers to the capability of this algorithm to provide a priority-to-flow assignment that allows all flows to meet their timing requirements if such a scheme exists.
[2]Ethernet features eight priorities level, and thus at most eight preemption classes. By assuming that a flow in a preemption class can preempt any other flow in another preemption class with a lower priority, it follows that Ethernet can support at most a seven-level preemption scheme.



(a) Flow f_2 misses its deadline with a 2-level preemption scheme.

(b) All flows meet their deadlines with a 3-level preemption scheme.

(c) All flows meet their deadlines with a 2-level preemption scheme.

Figure 2: Motivation example for the multi-level preemption configuration.

boxes"; and finally (4) the *lowest preemption class*, wherein frames are represented by using "green boxes". For this setting, there is no configuration under a non-preemptive nor a 1-level preemption scheme that allows all flows to meet their timing constraints.

In Figure 2a, flows are partitioned in three preemption classes and f_2 misses its deadline as it is unable to preempt f_3 – the two frames belong to the same preemption class. In Figure 2b, the situation improves for f_2 and it meets its deadline, nonetheless this is obtained at the cost of an additional preemption class. In Figure 2c, an appropriate configuration (again with three preemption classes) is used and all frames meet their deadlines. From these observations, it follows that the performance of the multi-level preemption scheme strongly depends on the adopted configuration.

Contribution. In this work, we advance the state-of-the-art by addressing the aforementioned configuration questions. Given a set of flows and a TSN network, we first provide an offline priority assignment scheme for the flow set. Then, we provide an offline framework for determining the appropriate number of preemption levels on the one hand; and the flow-to-preemption-class assignment on the other hand. Put all together, the end goal of the proposed framework is twofold: (1) to ensure that all flows meet their deadlines; and (2) to ensure that hardware resources are utilized in an efficient manner. To the best of our knowledge, no prior work has addressed the configuration synthesis of TSN with multi-level preemption. Note that frame preemption can be implemented with or without TSN shapers such as TAS and/or CBS (see [22], page 48). In this work, we opt for an implementation without shapers so that we can focus solely on the evaluation of multi-level preemption without the added complexity of other protocol mechanisms.

Paper Organization. The rest of this paper is organized as follows. Section 2 presents the system model as well as key parameters used throughout this manuscript. Section 3 details our proposed framework. Experimental evaluations and performance assessment are presented in Section 4. Section 5 discusses some related work. Finally, Section 6 concludes the paper and provides some interesting future research directions.

2 SYSTEM MODEL

▹ **Network specification.** We assume an Ethernet backbone network for a real-time distributed system. We represent the network as a directed graph $G \overset{\text{def}}{=} (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of all nodes and $\mathcal{L}$ is the set of all physical links in the network. We assume that every link is bi-directional; full-duplex; and operates at a single reference speed, say $s > 0$. The tuple G is given with the interpretation that: (1) $\mathcal{N} = \text{EP} \cup \text{SW}$, where $\text{EP} \overset{\text{def}}{=} \{\text{EP}_1, \text{EP}_2, \ldots\}$ represents the set of all end-points and $\text{SW} \overset{\text{def}}{=} \{\text{SW}_1, \text{SW}_2, \ldots\}$ is the set of all switches. Each EP_q (with $q \geq 1$) has a single output port and can receive and send network traffic while SW consists only of forwarding nodes, each with a finite number of output ports, through which the traffic is routed. Each SW_ℓ (with $\ell \geq 1$) is enabled with multi-level preemption capability and decides, based on its internal routing table, to which output port a received frame will be forwarded. We also assume that each switch Input/Output port has 8 priority queues and that each queue is assigned to flows of the corresponding priority level.

▹ **Traffic specification.** We consider $\mathcal{F} \overset{\text{def}}{=} \{f_1, f_2, ..., f_n\}$ a network traffic with $n \geq 1$ flows. Each flow $f_i \overset{\text{def}}{=} \langle src_i, dst_i, T_i, D_i, S_i, P_i, \text{PC}_i \rangle$ consists of a potentially infinite number of instances (a.k.a. frames) and is characterized by: (1) src_i, the source endpoint; (2) dst_i, the destination endpoint; (3) T_i, the minimum inter-arrival time between two consecutive frames of f_i, i.e., by assuming that the first frame of f_i is released at src_i at time $a_{i,1} \geq 0$, then $a_{i,\lambda+1} - a_{i,\lambda} \geq T_i$ for all $\lambda \geq 1$, where $a_{i,\lambda}$ is the release time of the λ^{th} frame; (4) D_i, the relative deadline, i.e., $d_{i,\lambda} \overset{\text{def}}{=} a_{i,\lambda} + D_i$ is the latest time instant by which the λ^{th} frame of f_i must reach dst_i; (5) S_i, the size of f_i (in bytes); (6) P_i (with $0 \leq P_i \leq 7$), the priority; and finally (7) PC_i the preemption class. For the sake of convenience, we assume that 0 is the highest priority and 7 is the lowest[3]. For preemption classes, we assume in the same vein that the smaller the value, the higher

[3]The specification in the standards suggests the opposite. However, we opted to keep both the frame priority and preemption class in the same format (ascending order starting from 0) in this work to improve readability.

the preemption class. Flows with the same priority always belong to the same preemption class but the converse is not true. In other words, flows with the same preemption class may have different priorities (see Figure 3). Finally, the priority and the preemption class parameters lie at the core of the configuration problem addressed in this work and are determined in Section 3.

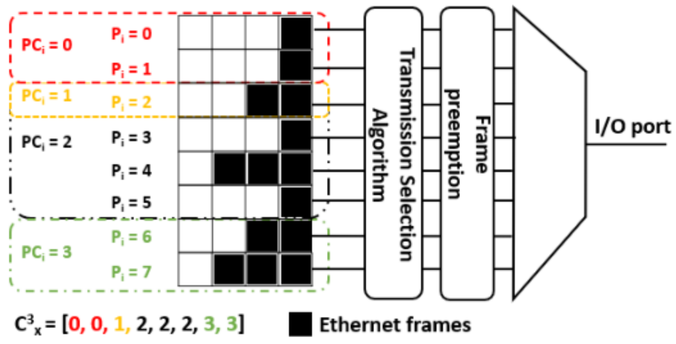


Figure 3: Preemption classes; priority queues and Configuration.

▹ **Flow configuration.** For a flow set $\mathcal{F}$, we denote by $C^m \overset{\text{def}}{=} \{C_1^m, C_2^m, C_3^m, \dots\}$ the set of all possible flow configurations, assuming an m-level preemption scheme[4] (with $0 \leq m \leq 7$). Each configuration C_x^m (with $x \geq 1$) is an *integer list* that defines the preemption class of each priority level. Concretely, $C_x^m \overset{\text{def}}{=} [c_{x,0}^m, c_{x,1}^m, \dots, c_{x,p-1}^m]$, where p is the number of different priorities in $\mathcal{F}$ and $c_{x,\sigma}^m$ (with $0 \leq \sigma \leq p-1$) is the preemption class of all flows $f_i \in \mathcal{F}$ with priority $P_i = \sigma$. In other words, a preemption configuration is a non-decreasing and surjective function. Figure 3 illustrates an example of flow configuration C_x^3 for a 3-level preemption scheme (i.e., with 4 preemption classes). Here, we assume that $x = 1$ in order to refer to the "first configuration" out of the set of all possible configurations C^3. Flows with priorities 0 and 1 are assigned to the preemption class 0, i.e, $c_{1,0}^3 = 0$ and $c_{1,1}^3 = 0$; flows with priority 2 are assigned to the preemption class 1; flows with priorities 3, 4 and 5 are assigned to the preemption class 2; and finally, flows with priorities 6 and 7 are assigned to the preemption class 3. Therefore, the resulting configuration is $C_1^3 = [0, 0, 1, 2, 2, 2, 3, 3]$. Note that we assume a network-wide configuration, i.e., C_1^3 applies to all switches. Furthermore, for any m-level preemption scheme the following rules are enforced.

R_1– Every flow f_i can be assigned to one and only one preemption class;

R_2– To reduce priority inversion, any flow, say f_j, with a lower priority than another flow, say f_i, cannot be assigned to a higher preemption class than that of f_i;

R_3– To conserve hardware resources, every preemption class must have at least one flow assigned to it.

In addition to these three rules, we introduce the following definitions for any flow set $\mathcal{F}$ and network $\mathcal{G}$.

[4]An m-level preemption scheme implies $m + 1$ preemption classes.

Definition 2.1 (Valid configuration). Any configuration will be stamped as "valid" if upon the flow-to-preemption-class assignment, it conforms to Rules R_1, R_2 and R_3.

Definition 2.2 (Solution). Any valid configuration will be considered a "solution" if all flows $f_i \in \mathcal{F}$ meet their deadlines.

Rule R_2 implies that C_x^m (with $x \geq 1$) is always sorted in an ascending order, i.e., $c_{x,\sigma}^m \leq c_{x,\ell}^m$ for all $0 \leq \sigma \leq \ell \leq p-1$. Rule R_3 implies that each integer from 0 to m must appear *at least once* in each valid configuration. Consequently, the task of generating all valid configurations C^m can be mapped to the problem of generating all ordered multisets [7] of cardinality p with elements taken from the set $\{0, \dots, m\}$, where p is the number of unique flow priorities in $\mathcal{F}$. Rule R_3 also implies that all elements $\{0, \dots, m\}$ must appear at least once in each multiset. In the following section, we first provide a priority assignment scheme in Section 3.1, and then a preemption class assignment scheme in Section 3.2

3 PROPOSED FRAMEWORK

3.1 Priority assignment

In this section, we present the priority assignment scheme for TSN flows based on the traditional *k-means clustering algorithm* [17]. This is a popular unsupervised Machine Learning (ML) algorithm that partitions items in an unlabeled list into k different clusters (with $k \geq 1$). Specifically, it computes k focal points called "centroids" within the data space in an iterative manner. Clusters are formed around the centroids by assigning each item to the closest centroid to it. Finally, the location of each centroid is updated to the center of all data points assigned to it after each iteration. In the context of this work, the "items" to be assigned are TSN flows and the k clusters are the flow priorities. We recall that Ethernet supports eight (8) priority levels, thus $k \leq 8$. The k-means algorithm performs clustering based on some selected characteristics of the items, referred to as "features". These features capture the domain-specific knowledge of the real-life objects/concepts that the items represent. In this scope, the features need to be defined and sometimes transformed into a format that the k-means algorithm can process. The process of defining and preparing features for ML algorithms is called "feature engineering". In the following section, we summarize the feature engineering for our TSN priority assignment problem.

3.1.1 Feature engineering. For the priority assignment problem, five of the seven flow parameters defined in Section 2 are considered for the clustering process, since the other parameters are yet to be determined (i.e., the priority and the preemption class). In other words, the tuple $\langle src_i, dst_i, T_i, D_i, S_i \rangle$ is considered for each flow $f_i \in \mathcal{F}$. On another front, src_i and dst_i are used to determine the path length PL_i of flow f_i, i.e., the number of links traversed by f_i from src_i to dst_i. Therefore, the number of features employed for the clustering problem can be reduced to the tuple $\langle \text{PL}_i, T_i, D_i, S_i \rangle$.

Note that these selected features are not on the same unit scale. As a matter of fact, flow periods and deadlines can range from a few microseconds to hundreds of thousands of microseconds. At the other end of the table, flow sizes can only take values from 64 bytes to 1500 bytes (i.e., the minimum and maximum valid Ethernet frame sizes, respectively). Finally, the range of PL_i depends largely on the network topology. In such a scenario, features with large values might dominate those with lower values, thereby impacting the actual performance of the k-means algorithm negatively. To ensure that this is not the case and make sure that each feature has an impact on the learning process, a process called "normalisation" is usually carried out on the selected feature set [17]. In this work, this normalization process is carried out on each of the features as follows. Regarding the flow length, all values are modified to obtain features in the interval $[-1, 0]$. To this end, we divide all flow lengths by the negative value of the longest flow path. Similarly, both T_i and D_i are normalized by dividing all values by the largest values of T_i and D_i, respectively, to obtain features in the interval $[0, 1]$. Finally, the values of S_i are normalized with the maximum frame size in the flowset to obtain features in the interval $[0, 1]$. The rationale behind the normalization of flow paths with the negative value of the longest flow path is to make flows with shorter paths have higher feature values. This is further explained in the following section.

3.1.2 Clustering and priority assignment. In this section, we describe our strategy for assigning the priorities to flows through partitioning the n flows in $\mathcal{F}$ into k clusters (with $1 \leq k \leq 8$) by using the k-means algorithm. The overall objective is to assign priorities to flows in a manner that allows as many as possible flows to be schedulable, i.e., meet their deadlines. Figure 4 presents the adopted flowchart. From the figure, the process begins with the feature engineering processes described in Section 3.1.1. After this step, two important questions need to be addressed:

Q_1: How to determine the value of k? In other words, how to determine the number of clusters (priorities) in which the flows should be divided into?

Q_2: How to determine the relative order among the clusters during the priority assignment process?

▹ **About** Q_1. Since we do not know which value of k will yield the maximum number of schedulable flows, we initialize k to 1 and iterate through all possible values ($1 \leq k \leq 8$).

▹ **About** Q_2. At each iteration, we perform k-means clustering on the normalized features and obtain the centroid of each of the k clusters. Each centroid is a vector of the means of the features of items in the cluster. Afterwards, we compute the mean of each centroid. Recall that from the normalization process, flows with longer paths, smaller periods, deadlines, and sizes are assigned smaller feature values. As such, the smaller the final value of the centroid of a given cluster, the longer the path and the smaller the period; deadline and size of each of its members in general. In practice, flows with longer paths are more at risk of missing their deadlines as they may experience delays on the links they traverse. Also, flows with shorter periods and deadlines are usually assigned higher priorities in fixed priority scheduling theory because they are more at risk of missing their deadlines (think about Rate Monotonic and Deadline Monotonic policies). Finally, the sizes of non time-critical flows are typically larger than those of the time-critical ones (see [26, 34] for examples inspired by real-life use-cases from the automotive domain). With the aforementioned observations, we opted for assigning priorities to the clusters in ascending order of their centroid means. Specifically, *the lower the centroid mean, the higher the priority.* In this work, we assume that all features

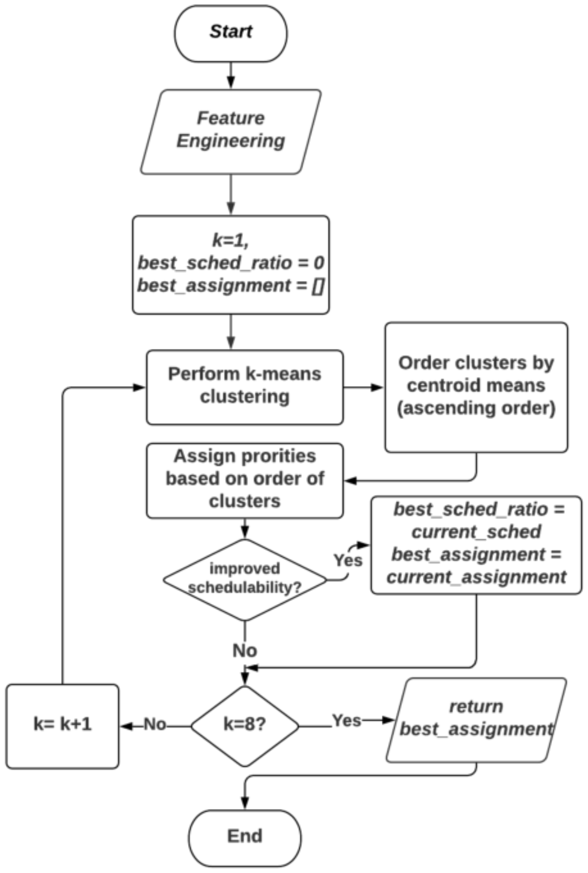

Start
Feature Engineering
k=1,
best_sched_ratio = 0
best_assignment = []
Perform k-means clustering
Order clusters by centroid means (ascending order)
Assign prorities based on order of clusters
improved schedulability?
Yes
best_sched_ratio = current_sched
best_assignment = current_assignment
No
k= k+1
No
k=8?
Yes
return best_assignment
End

Figure 4: Flowchart: k-means priority assignment algorithm.

have equal importance in the priority assignment process. Obviously, other feature engineering approaches can be adopted, where specific characteristics of the network are exploited. For example, the path length may be given a higher weight in a line topology network. We also note that the schedulability tests in the priority assignment process (see Figure 4) depend necessarily on the preemption level assignment, which has not been defined at this point. To resolve this dependency, the schedulability is evaluated by assuming a fully-preemptive scheme, i.e, every flow can preempt any other flow with a lower priority.

3.2 Preemption class assignment

3.2.1 Synopsis. Our preemption class assignment solution builds on the priority assignment scheme. It begins from a non-preemptive setup and proceeds by using a guided exhaustive search approach described as follows. At each step, we introduce an additional preemption level and test all possible flow-to-preemption-class configurations to check if all timing requirements are met. Our algorithm terminates as soon as a solution is found. Otherwise, another preemption level is added to the flow transmission scheme and a new test is carried out. Note that we are interested in determining the schedulability of a flow set only, and not on whether there are multiple solutions for a given flow set. This process is performed in an iterative manner up until a valid configuration is found at a given preemption level; or the maximum number of preemption levels (i.e., 7) is exceeded. In the latter case, a valid configuration could not be found and the flow set is deemed "unschedulable". Below, we provide the step-by-step details of our proposed solution, which consists of two algorithms, namely (1) the *flow-to-preemption-class assignment*; and (2) the *valid configuration search.*

3.2.2 Flow-to-preemption-class assignment. Algorithm 1 presents the pseudo-code for this step. Two arguments are required as inputs, namely: (1) the network topology G; and (2) the flow set $\mathcal{F}$. In the description, notation "P. Len" refers to the length of list P and "$[y]*\eta$" refers to a list of length η filled up with "y", e.g., $[0]*3 = [0, 0, 0]$. The algorithm proceeds as follows.

First, the set P of all unique flow priorities in $\mathcal{F}$ is stored in variable P (see line 1). Thereafter, a systematic search is carried out through all preemption levels m (with $1 \leq m \leq$ P. Len −1) for a solution (see lines 2 to 18). At each preemption level m, after the initialization phase (see lines 3 to 8), a recursive function VALID_CONFIGS() is invoked (see line 9). This function returns the set of all valid configurations for preemption level m, i.e., C^m. Then, a search through this set is conducted to find a solution (see lines 10 to 17). If a solution is found, then Algorithm 1 terminates with this solution (see line 15), otherwise m is incremented and the process is repeated for the next preemption level, $(m+1)$. In case all iterations are exhausted and no solution is found, then a NULL value is returned (see line 19), which implies that the flow set is not schedulable. We note that for the schedulability test (see line 14), we employ the worst-case traversal time (WCTT) analysis for multi-level preemption schemes presented by Ojewale et al. [34]. Here, the authors computed the WCTT of each flow by using a Compositional Performance Analysis (CPA) framework.

3.2.3 Valid configuration search. Algorithm 2 presents the pseudo-code. VALID_CONFIGS() is a recursive function that computes the set of all valid configurations for a preemption level m. It takes 3 inputs: (1) an initialization of the set C^m (which is usually an empty set at the start); (2) a leftPart list (which is also usually empty at the start); and finally (3) a rightPart which is a configuration initialization for the preemption level m. The notations in Algorithm 2 have the same meanings as those of Algorithm 1. Additionally, the notation ListA + ListB implies a concatenation operation of two lists. In summary, VALID_CONFIGS solves the multisets generation problem described in Section 2. Since the algorithm is recursive, the first step is to test for the base case (see lines 2 to 5). This is when the items in the rightPart are either all the same or all unique. In this case, the algorithm terminates and returns a merged (and sorted) list of leftPart and rightPart. If the base condition is not met, then the recursive step (see lines 6 to 20) is executed.

3.2.4 Computational Complexity. We recall that the problem of searching for all valid configurations (Algorithm 2) was mapped to the multi-set problem which is known to have exponential complexity. Stanley [42] already showed that the number of multisets

Algorithm 1: ASSIGN_PREEMPTION_CLASS(G,$\mathcal{F}$)

```
Data: Network topology $G$; Flow set $\mathcal{F}$.
Result: A valid flow-to-preemption-class configuration.
1  P ← [$Set\ of\ all\ unique\ flow\ priorities\ in\ \mathcal{F}$]
2  for $m = 0$ to P. Len −1 do
3      $C^m \leftarrow \emptyset$
4      leftPart ← []
5      rightPart ← $[m] *$ P. Len
6      for $j = 0$ to $m$ do
7          rightPart[$j$] = $j$
8      end
9      $C^m$ = VALID_CONFIGS($C^m$, leftPart, rightPart)
10     foreach $C^m_x \in C^m$ do
11         foreach $f_i \in \mathcal{F}$ do
12             PC$_i$ = $c^m_{x,P_i}$
13         end
14         if SCHEDULABLE($\mathcal{F}$) then
15             return $C^m_x$
16         end
17     end
18 end
19 return NULL
```

of cardinality k to be taken from m-element set is given by:

$$\frac{(k+m-1)!}{k!(m-1)!}$$

We recall that every element of $\{1, 2, \ldots, m\}$ must appear at least once in each multiset (see Section 2). This constraint reduces the number of elements order to be decided from k to $k-m$. By replacing k with $k-m$, the total number of valid configurations $|C^m|$ for preemption level m with k unique priorities, is given by Equation 1.

$$|C^m| = \frac{(k-1)!}{(k-m)!(m-1)!} \quad (1)$$

Algorithm 1 invokes function VALID_CONFIGS() continuously until a solution is found or the maximum number of preemption levels is exceeded. Consequently, the maximum number of valid configurations to be tested is given by Equation 2.

$$\sum_{m=1}^{P.Len-1} |C^m| \quad (2)$$

Now that we have provided the specifics of our proposed framework for determining the priority assignment and a valid configuration for a given flow set, we can proceed with demonstrating its applicability and evaluate its performance in the next section. To this end purpose, we consider two different settings: (1) a synthetic use-case; and (2) two real-life use-cases from the literature – SAE [15] and Orion [46].

Algorithm 2: VALID_CONFIGS(C^m, leftPart, rightPart)

```
Data: Set of generated configurations $C^m$; a left partition;
      and an initial configuration for preemption level $m$
Result: Set of generated configurations $C^m$
1  rightPartSet = $sorted(set($rightPart$))$
2  if rightPart. Len == rightPartSet. Len
     OR rightPartSet. Len == 1 then
3      $C^m.add(sorted($leftPart + rightPart$))$
4      return $C^m$
5  end
6  diff = rightPart. Len – rightPartSet. Len
7  diff + = 1
8  currentLeft = rightPartSet. getFirstElem
9  rightPartSet. removeFirstElem
10 for $x = 1$ to $diff$ do
11     tempLeft = [currentLeft] $* x$
12     newLeftPart = leftPart + tempLeft
13     rightSize = rightPart. Len $-x$
14     newRightPart = $list($rightPartSet$)$
15     if rightSize > newRightPart. Len then
16         lenDiff = rightSize – rightPartSet. Len
17         newRightPart = [rightPartSet. getFirstElem] *
             lenDiff +$list($rightPartSet$)$
18     end
19     $C^m$ = VALID_CONFIGS($C^m$, $newLeftPart$,
         $newRightPart$)
20 end
21 return $C^m$
```

4 EVALUATION

4.1 Evaluation on a synthetic network.

▹ **Setup.** We consider a quad-star topology consisting of six EPs and three TSN switches connected as shown in Figure 5. Link speeds

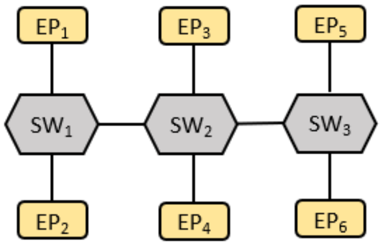


Figure 5: Synthetic network with quad-star topology.

are assumed constant and set to 100MBits/s. In each batch of experiments, we randomly generated 1000 flowsets of equal sizes, i.e., each flowset has exactly the same number of flows as the others. We varied the sizes between 100 and 250 flows per flowset. Each generated flow is characterised by a source, destination, period, deadline, and size. The sources and destinations are chosen randomly among the EPs. The values of the periods and deadlines range from 500μs to 100000μs. The values for the flow sizes range from 64bytes to 1500 bytes. We then assign priorities to the flows using our proposed approach. The priority assignment framework was implemented in Python 3.6, and the `scikit-learn`[5] – a free software machine learning library for the Python programming language – was used for the k-means clustering. We benchmark our solution against DMPO because it is known to dominate most of the other priority assignment schemes in the literature[14, 25]. Furthermore, DMPO has been specifically recommended for fixed-priority preemptive systems with preemption thresholds [14], which is a flow transmission model close to the one considered in this paper. Since DMPO provides a fully ordered priority list for the flowset and Ethernet only supports up to eight priority levels, we perform a greedy bin packing of the ordered flows by assigning equal number of flows to each priority level. Also, as the exact number of priority levels that gives the best performance is not known under DMPO, we tried all possible numbers of priority levels k (with $1 \leq k \leq 8$) and report the best performance.

Our major evaluation metrics is the schedulability ratio, i.e, the percentage (%) of flows that meet their timing requirement under the priority assignment scheme. To evaluate the schedulability of each scheme, we employ the worst-case traversal time (WCTT) analysis for preemptive TSN presented by Ojewale et al. [34]. Here, the authors computed the WCTT of each flow by using a Compositional Performance Analysis (CPA) framework. For both our k-means approach and the DMPO, we ran the experiments several times each and then report the average observed schedulability performances. In the first batch of experiments, we assign priorities to the flows using DMPO and k-means, and evaluated the schedulability of flows by assuming a fully-preemptive scheme, i.e, a flow can preempt any other flow of lower priority than itself. In the second batch of experiments, we applied our preemption class assignment algorithm to determine the appropriate preemption level assignment for each flowset.

▹ **Results and discussion.** Figure 6 shows the schedulability results under the k-means and DMPO schemes. From the Figure, k-means is able to schedule a higher number of flows than DMPO. Specifically, the average number of scedulable flowsets ranges from 999.5 to 981 for the k-means scheme, and from 998 to 974 for DMPO. Figure 7 shows the average runtime per flowset for both k-means and DMPO. The reported execution times were obtained from a commodity hardware (Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, 16GB Memory). From Figure 7, DMPO is much faster than the k-means algorithm. This is due to its complexity. At the core of DMPO is a sorting, with a complexity of $O(nlog(n))$ whereas k-means is known to have a complexity of $O(n^2)$ [36]. We note, however, that k-means is still reasonably fast for the priority assignment task with an average runtime of 1.6s compared to DMPO's 0.79s for flowsets with 250 flows each.

Figure 8 shows the schedulability results for the preemption level assignment (Algorithm 1). From the Figure, the average number of schedulable flowsets increases with increasing preemption levels. Specifically, the schedulability ratio jumps from 45.5% under the non-preemptive scheme to 85.8% under the 1-level preemption scheme and the trend continues, in a non-linear manner though, with each additional preemption level to peak at 98.1% under a

[5]https://scikit-learn.org/stable/modules/clustering.html

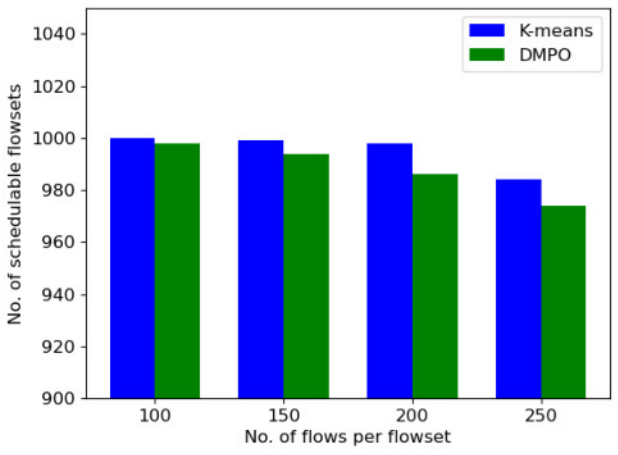


Figure 6: Schedulability: K-means vs. DMPO.

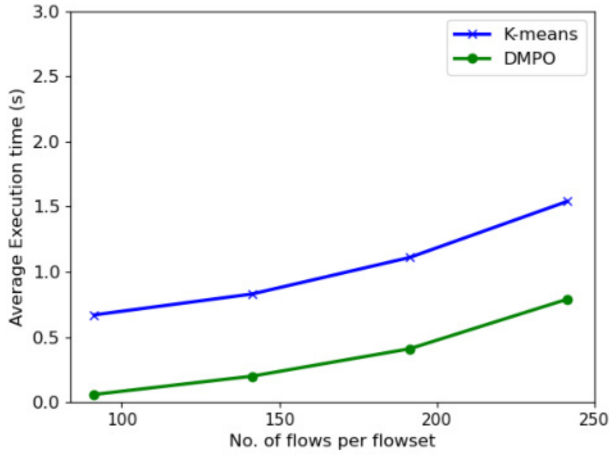


Figure 7: Runtime: K-means vs. DMPO.

fully preemptive scheme. From these observations, it follows that the limitations of the 1-level preemption scheme may still cause a non-negligible number of flowsets (here 13%) to be unschedulable, despite the huge improvements it brings about over the non-preemptive scheme. This issue is circumvented by the multi-level preemption scheme. Note that the increase over the 4-level preemption scheme is not significant and/or offers limited gain in this case. In addition, it is important to mention that each preemption level involves extra hardware implementation overheads and the framework presented in this paper is useful to evaluate the trade-off in terms of preemption-level scheme to adopt and the performance gain brought about by enabling each extra-preemption level.

Figure 9 shows the average runtime per flowset for each preemption level configuration. From the figure, it takes an average of 1.6s to compute the preemption configuration and assess the schedulability of a flowset of 250 flows, under a non-preemptive

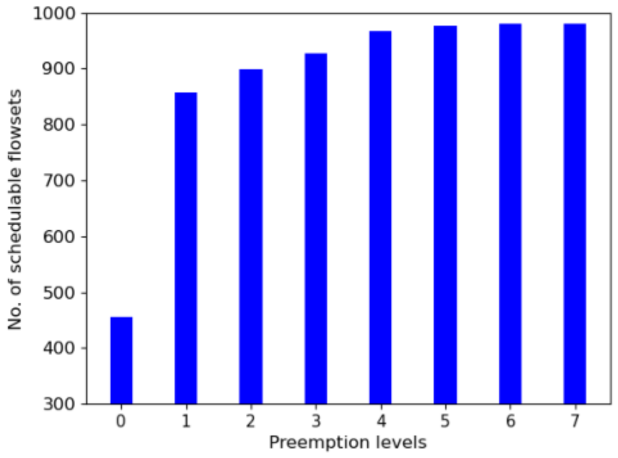


Figure 8: Schedulability results w.r.t. increasing preemption levels

scheme. The execution time grows slowly as new preemption levels are added, and peaks at an average of 6.21s per flowset for a fully preemptive scheme. This shows that despite the fact that the preemption level algorithm is a guided exhaustive search approach, its execution time is not prohibitive, and consequently, making the proposed scheme applicable in real-life scenarios.

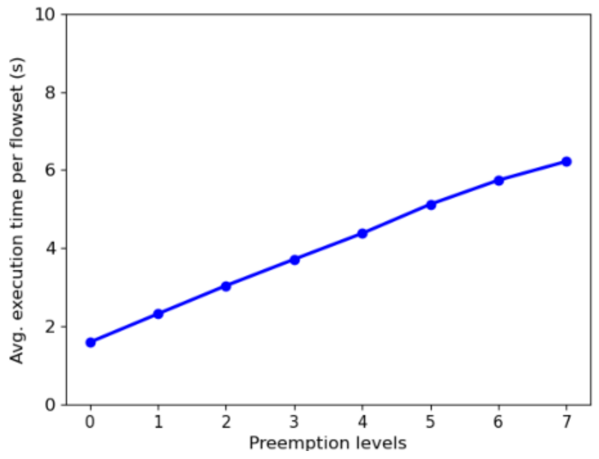


Figure 9: Average runtime w.r.t. increasing preemption levels

4.2 Evaluaton on real-life use-cases

▹ **Setup.** SAE is the "SAE automotive communication benchmark" and Orion is the embedded communication network of the Orion Crew Exploration Vehicle (CEV). The network topologies of the SAE and Orion use-cases are depicted in Figures 10, and 11, respectively. In the figures, the EPs are represented with rectangles and the TSN switches are represented with hexagons. The flow parameters are provided by Gavriluţ and Pop in [15][6]. Link speeds are assumed

[6]The files for all test cases are available at https://bit.ly/2kpLrKj.

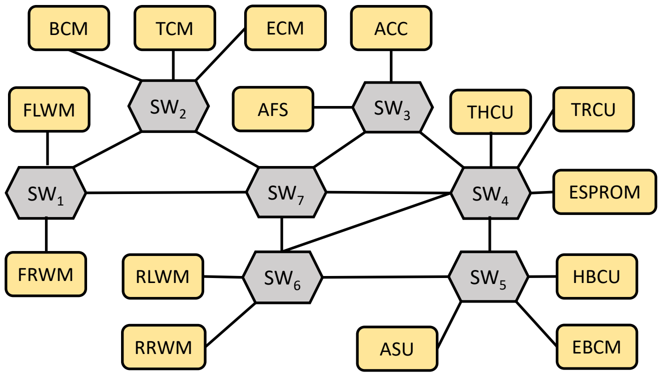


Figure 10: SAE automotive communication benchmark topology [15]

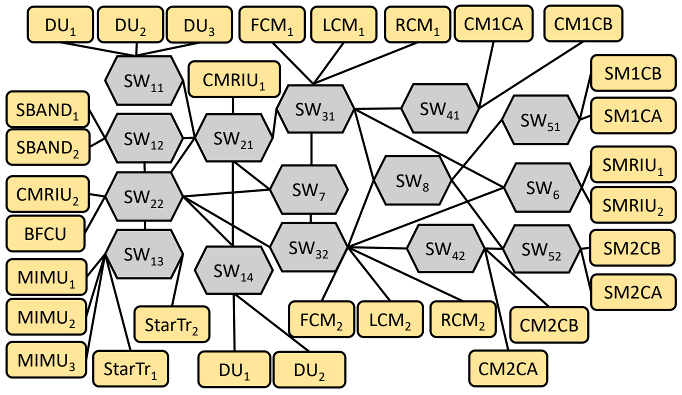


Figure 11: Orion CEV network topology [46].

constant and set to 100MBits/s. In the first batch of experiments, we assign priorities to the flows using the two priority assignment schemes, and evaluated the schedulability of flows by assuming a fully-preemptive scheme. In the second batch of experiments, we applied our preemption class assignment algorithm to determine the preemption level in which the number of schedulable flows is highest.

▹ **Results and discussion.** Table 1 shows the schedulability performance of the two priority assignment schemes evaluated. From the table, the k-means priority assignment scheme again outperforms DMPO in both use-cases. Precisely, k-means was able to schedule 97.46% and 86.9% compared to DMPO's 93.67% and 80.4%, for the SAE and Orion networks, respectively. For the SAE benchmark network, the schedulability performance of the k-means algorithm matches the one reported by Gavriluţ and Pop [15] who employed a traffic-type assignment (TTA) approach to find a feasible schedule. To the best of our knowledge, this is the best reported performance in the literature.

Figure 12 shows the behavior of the SAE and Oreon setups with increasing number of preemption levels. From the Figure, frame preemption clearly increases the number of schedulable flows for both SAE and Oreon. We also note that the number of scedulable flows continue to increase in the Oreon use-case until it peaks at

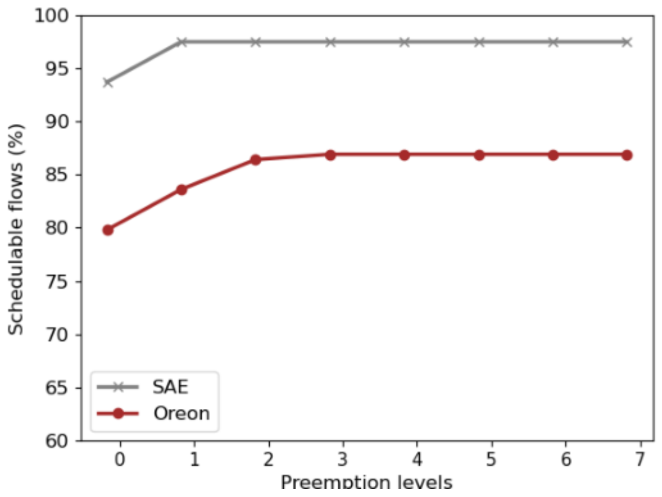


Figure 12: % of schedulable flows w.r.t. increasing preemption levels for SAE and Oreon.

86.9% under a 3-level preemption scheme. On the other hand, the flows schedulability in the SAE use-case already peaked under 1-level preemption. Observations like this can help system designers to decide the best number of preemption levels to enable for a system at design time, and ensure that just the needed preemption levels are implemented.

5 RELATED WORK

An appropriate priority assignment policy plays a central role in the resulting performance of real-time systems [14]. For example, in the Controller Area Network (CAN) [8], the maximum reliable utilization level was intially believed to be at 35% due to the legacy practice of assigning message IDs (which corresponds to messages priorities) in a random or an ad-hoc manner [9]. Davis et al. [14] later showed that a reliable CAN utilization over 80% can be achieved using Audley's Optimal Priority Assignment (AOPA) algorithm [3]. AOPA has become the reference priority assignment scheme in many real-time systems and has been proved to be optimal when there are no priority inversions [38]. Davis et al. [12] introduced an improvement of the AOPA – the Robust Priority Assignment algorithm (RPA) – which, in addition to being optimal, maximizes the number of tolerable transmission errors in CAN. We note that both AOPA and RPA are not applicable to IEEE802.1 Qbu networks because priority inversion can occur in these networks [33].

For systems where priority inversions can occur, the Deadline Monotonic Priority Order and Deadline minus Execution time Monotonic Priority Order (D-CMPO), and the so-called "DkC" heuristics are often employed and these heursitics are known to dominate most of the other priority assignment heuristics in the literature in terms of schedulability [14, 25]. Specifically, DMPO is the recommended priority assignment scheme in scenarios where preemption overheads are taken into account and/or fixed-priority scheduling with preemption thresholds. These two conditions also apply to the IEEE802.1 Qbu networks as each preemption results in significant overhead and thresholds are set for each preemption class [33].

Name	EPs	SWs	No. of flows	K-means Schedulability	DMPO Schedulability
SAE	15	7	79	97.46 %	93.67%
Orion CEV	31	15	184	86.9%	80.4%

Table 1: Experimental results from real-life use-cases

On another front, DMPO is not suited for priority assignment in preemptive TSN because it provides a fully ordered priority list for the flowset but Ethernet only supports up to eight priority levels. This will consequently lead to another bin packing problem which is known to be strongly NP-Complete.

For TSN, most work in the literature focused on the configuration synthesis of time-triggered Ethernet networks where flows are transmitted by following a pre-computed schedule. In this regard, Beji et al. [5] proposed a Satisfiability Modulo Theories (SMT) approach and Tamas-Selicean et al. [44] proposed a heuristics for the communication synthesis of TTEthernet [43]. On another front, Oliver et al. [35] and recently Reusch et al. [40] proposed different frameworks for the synthesis of the so-called Gate Control Lists for the TSN IEEE802.1 Qbv [20]. For event-triggered time-sensitive Ethernet networks, Specht et al. [41] considered a TSN network with the Urgency Based Scheduler (UBS) and used an SMT approach to assign hard real-time data flows to queues and priority levels to these queues on a per hop basis. This work differs from our work in that it addresses the priority assignment for UBS only and the work assumes a non-preemptive frame transmission scheme. Gavriluţ and Pop [15] provided a method to assign traffic classes to frames in TSN-based mixed-criticality systems. However, none of these works addressed the priority assignment and configuration synthesis of preemptive TSN networks.

Several works have applied ML techniques to different problems in the real-time domain [1, 10, 11, 25]. Among these, the most relevant to this work is the recent work of Lee et al. [25] where the authors proposed a Priority Assignment Learning (PAL) framework for multi-core real-time systems. PAL was found to be more effective than existing approaches but suffers severe scalability challenges as the number of tasks grows. There are other works in the literature that have applied ML techniques to Ethernet TSN [28–31]. Mai et. al. [30] and Navet et al. [31] employed ML techniques to search for feasible TSN configurations. We note that both works do not address the priority assignment problem as part of the configuration. The authors also presented a so-called "hybrid" approach that combines ML techniques with theoretical performance analysis to control the false prediction rate of the ML models [29]. Furthermore, Mai et al. recently provided a Generative Neural Networks (GNN)-based technique to predict a feasible TSN configuration [28]. But the work stops short of defining any priority assignment scheme.

Park et al. [37] showed that both the priority and the preemption class assignment schemes employed in a preemptive TSN network have a significant impact on the ability of the frames to satisfy their timing constraints. In this regards, proposed a framework to compute efficient priority assignments for flows and an efficient eMAC/pMAC queue boundary at each switch port. But, the work was conducted by assuming a 1-level preemption scheme, thus leaving open the question of finding the appropriate priority assignment policy, preemption levels, and a flow-to-preemption-class assignment. These gaps are closed in this contribution.

6 CONCLUSION AND FUTURE DIRECTIONS

In this work, we address the synthesis problem for multi-level frame preemption in TSN. Given a set of flows and the network topology, we present a framework to assign priorities to the flows, determine the optimal preemption level to enable, and a flow-to-preemption-class assignment. We evaluate the performance of the proposed framework experimentally and our results show that the proposed scheme outperforms the DMPO scheme which is known to dominate most of the other priority assignment schemes in the literature. We also show that the proposed framework provides the minimum number of preemption levels to guarantee schedulability. This is especially important as each additional preemption class comes along with significant hardware overheads that can increase the cost of switch manufacturing. In addition, our proposed framework demonstrates acceptable scalability for practical use-cases. As future work, studies into how to further explore the search space for possible improvements of the priority assignment scheme would be of great benefit to our proposed approach. Specifically, the effect of each of the selected flow characteristics on the outcome of the clustering algorithm could be explored. Another interesting research direction would be the comparison of the performance of multi-level preemption scheme against other TSN traffic control mechanisms from WCTT and QoS standpoints. Routing is fixed and is always the same between any pair of nodes in this work. Thus, the impact of a static per flow routing on schedulability and runtime for the mutli-level preemption scheme could be considered. Last but not least, a comparison between an optimal priority assignment without preemption and a sub-optimal priority assignment with preemption would also be interesting to evaluate the trade-off between priority assignment and preemption.

ACKNOWLEDGMENTS

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDP/UIDB/04234/2020); also by FCT through the European Social Fund (ESF) and the Regional Operational Programme (ROP) Norte 2020, under grant 2020.09636.BD.

REFERENCES

[1] Tadashi Ae and Reiji Aibara. 1990. Programmable real-time scheduler using a neurocomputer. *Real-Time Systems* 1, 4 (1990), 351–363.
[2] Mohammad Ashjaei, Mikael Sjödin, and Saad Mubeen. 2021. A novel frame preemption model in TSN networks. *Journal of Systems Architecture* 116 (2021), 102037.
[3] Neil C Audsley. 2001. On priority assignment in fixed priority scheduling. *Inform. Process. Lett.* 79, 1 (2001), 39–44.
[4] AVB Task Group. 2009. IEEE 802.1Qav - Forwarding and Queuing Enhancements for Time-Sensitive Streams.
[5] Sofiene Beji, Sardaouna Hamadou, Abdelouahed Gherbi, and John Mullins. 2014. SMT-based cost optimization approach for the integration of avionic functions in IMA and TTEthernet architectures. In *18th Int. Symposium on Distributed Simulation and Real Time Applications.* 165–174.
[6] Lucia Lo Bello and Wilfried Steiner. 2019. A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems. *Proc. IEEE* 107, 6 (2019), 1094–1120.

[7] Wayne D. Blizard et al. 1989. Multiset theory. *Notre Dame Journal of formal logic* 30, 1 (1989), 36–66.

[8] Bosch. 1991. CAN Specification. *Robert Bosch GmbH, Postfach* 50 (1991).

[9] D Buttle. 2012. Real-time in the prime-time, ETAS GmbH, Germany. In *Keynote talk at 24th Euromicro Conference on Real-Time Systems, Pisa, Italy.*

[10] Carlos Cardeira and Zoubir Mammeri. 1994. Neural networks for multiprocessor real-time scheduling. In *Proceedings Sixth Euromicro Workshop on Real-Time Systems*. IEEE, Vaesteraas, Sweden, 59–64.

[11] Carlos Cardeira and Zoubir Mammeri. 1995. Preemptive and non-preemptive real-time scheduling based on neural networks. In *Distributed Computer Control Systems 1995*. Elsevier, Toulouse-Blagnac, France, 67–72.

[12] Robert I Davis and Alan Burns. 2009. Robust priority assignment for messages on Controller Area Network (CAN). *Real-Time Systems* 41, 2 (2009), 152–180.

[13] Robert I Davis, Alan Burns, Reinder J Bril, and Johan J Lukkien. 2007. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems* 35, 3 (2007), 239–272.

[14] Robert I. Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns. 2016. A Review of Priority Assignment in Real-Time Systems. *J. Syst. Archit.* 65, C (2016), 64–82. https://doi.org/10.1016/j.sysarc.2016.04.002

[15] Voica Gavriluţ and Paul Pop. 2020. Traffic-type Assignment for TSN-based Mixed-criticality Cyber-physical Systems. *ACM Transactions on Cyber-physical Systems* 4, 2 (2020), 1–27.

[16] A. Gogolev and P. Bauer. 2020. A Simpler TSN? Traffic Scheduling vs. Preemption.. In *25th IEEE Int. Conference on Emerging Technologies and Factory Automation (ETFA)*, Vol. 1. 183–189. https://doi.org/10.1109/ETFA46521.2020.9211987

[17] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The elements of statistical learning*. Springer, New York, USA.

[18] D. Hellmanns, J. Falk, A. Glavackij, R. Hummen, S. Kehrer, and F. Dürr. 2020. On the Performance of Stream-based, Class-based Time-aware Shaping and Frame Preemption in TSN. In *IEEE Int. Conference on Industrial Technology (ICIT)*. 298–303. https://doi.org/10.1109/ICIT45562.2020.9067122

[19] IEEE. [n.d.]. Time-Sensitive Networking Task Group. http://www.IEEE802.org/1/pages/tsn.html

[20] IEEE. 2016. IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic. *IEEE Std 802.1Qbv-2015* (2016), 1–57.

[21] IEEE. 2016. Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic. *Std 802.3br-2016* (2016), 1–58.

[22] IEEE. 2016. Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption. *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)* (2016), 1–52.

[23] IEEE. 2017. IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 29: Cyclic Queuing and Forwarding. *IEEE 802.1Qch-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd(TM)-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, IEEE Std 802.1Qbz-2016, and IEEE Std 802.1Qci-2017)* (2017), 1–30.

[24] IEEE. 2020. IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks - Amendment 34:Asynchronous Traffic Shaping. *IEEE Std 802.1Qcr-2020 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018, IEEE Std 802.1Qcc-2018, IEEE Std 802.1Qcy-2019, and IEEE Std 802.1Qcx-2020)* (2020), 1–151. https://doi.org/10.1109/IEEESTD.2020.9253013

[25] Seunghoon Lee, Hyeongboo Baek, Honguk Woo, Kang G Shin, and Jinkyu Lee. 2021. ML for RT: Priority Assignment Using Machine Learning. In *27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, Online, 118–130. https://doi.org/10.1109/RTAS52030.2021.00018

[26] Lucia Lo Bello, Mohammad Ashjaei, Gaetano Patti, and Moris Behnam. 2020. Schedulability analysis of Time-Sensitive Networks with scheduled traffic and preemption support. *J. Parallel and Distrib. Comput.* 144 (2020), 153–171. https://doi.org/10.1016/j.jpdc.2020.06.001

[27] Andrea Lodi, Silvano Martello, and Daniele Vigo. 2002. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics* 123, 1-3 (2002), 379–396.

[28] Tieu Long Mai and Nicolas Navet. 2021. Deep learning to predict the feasibility of priority-based Ethernet network configurations. *Transactions on Cyber-Physical Systems (TCPS)* 5, 4 (2021), 1–26.

[29] Tieu Long Mai, Nicolas Navet, and Jörn 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)*. IEEE, Sundsvall, Sweden, 1–8.

[30] 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 the 27th International Conference on Real-Time Networks and Systems*. ACM, Paris, France, 143–153.

[31] Nicolas Navet, Tieu Long Mai, and Jörn Migge. 2019. *Using machine learning to speed up the design space exploration of Ethernet TSN networks*. Technical Report. University of Luxembourg.

[32] Mubarak Adetunji Ojewale, Patrick Meumeu Yomsi, and Geoffrey Nelissen. 2018. On Multi-Level Preemption in Ethernet. In *WiP Session (ECRTS)*. 16–18.

[33] Mubarak Adetunji Ojewale, Patrick Meumeu Yomsi, and Borislav Nicolić. 2020. Multi-level preemption in TSN: feasibility and requirements analysis. In *23rd IEEE Int. Symposium on Real-Time Distributed Computing*. 1–9.

[34] Mubarak Adetunji Ojewale, Patrick Meumeu Yomsi, and Borislav Nikolić. 2021. Worst-case traversal time analysis of TSN with multi-level preemption. *Journal of Systems Architecture* 116 (2021), 102079. https://doi.org/10.1016/j.sysarc.2021.102079

[35] Ramon Serna Oliver, Silviu S Craciunas, and Wilfried Steiner. 2018. IEEE 802.1 Qbv gate control list synthesis using array theory encoding. In *Real-Time and Embedded Technology and Applications Symposium*. 13–24.

[36] Malay K. Pakhira. 2014. A Linear Time-Complexity k-Means Algorithm Using Cluster Shifting. In *International Conference on Computational Intelligence and Communication Networks*. IEEE, Bhopal, India, 1047–1051. https://doi.org/10.1109/CICN.2014.220

[37] Taeju Park, Soheil Samii, and Kang G Shin. 2019. Design optimization of frame preemption in real-time switched Ethernet. In *DATE*. 420–425.

[38] Taeju Park and Kang G. Shin. 2019. Optimal Priority Assignment for Scheduling Mixed CAN and CAN-FD Frames. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, Montreal, Canada, 192–203. https://doi.org/10.1109/RTAS.2019.00024

[39] Aleksander Pruski, Mubarak Adetunji Ojewale, Voica Gavriluţ, Patrick Meumeu Yomsi, Michael Stübert Berger, and Luis Almeida. 2021. Implementation Cost Comparison of TSN Traffic Control Mechanisms. In *26th IEEE ETFA*.

[40] Niklas Reusch, Luxi Zhao, Silviu S Craciunas, and Paul Pop. 2020. Window-based schedule synthesis for industrial IEEE 802.1 Qbv TSN networks. In *IEEE Int. Conference on Factory Communication Systems*. 1–4.

[41] Johannes Specht and Soheil Samii. 2017. Synthesis of queue and priority assignment for asynchronous traffic shaping in switched ethernet. In *IEEE Real-Time Systems Symposium (RTSS)*. 178–187.

[42] Richard P Stanley. 2011. Enumerative Combinatorics Volume 1 second edition. *Cambridge studies in advanced mathematics* (2011).

[43] Wilfried Steiner, Günther Bauer, Brendan Hall, Michael Paulitsch, and Srivatsan Varadarajan. 2009. TTEthernet dataflow concept. In *8th IEEE Int. Symposium on Network Computing and Applications*. 319–322.

[44] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. 2012. Synthesis of communication schedules for TTEthernet-based mixed-criticality systems. In *8th IEEE/ACM/IFIP Int. conference on Hardware/software codesign and system synthesis*. 473–482.

[45] D. Thiele and R. Ernst. 2016. Formal Worst-Case Performance Analysis of Time-Sensitive Ethernet with Frame Preemption. In *21st IEEE Int. Conf. on Emerging Technologies and Factory Automation*. 1–9.

[46] Luxi Zhao, Paul Pop, Qiao Li, Junyan Chen, and Huagang Xiong. 2017. Timing analysis of rate-constrained traffic in TTEthernet using network calculus. *Real-Time Systems* 53, 2 (2017), 254–287.