Link Between Real-Time Scheduling and Time-Triggered Networks

Richard Garreau, Matheus Ladeira, Emmanuel Grolleau, Henri Bauer, Frédéric Ridouard, Pascal Richard

Link between real-time scheduling and time-triggered networks

Richard Garreau*, Matheus Ladeira†, Emmanuel Grolleau†, Henri Bauer†, Frédéric Ridouard†, Pascal Richard*

LIAS, Université de Poitiers*
LIAS, ISAE-ENSMA†
Poitiers, France
{richard.garreau, pascal.richard}@univ-poitiers.fr*, {matheus.ladeira, grolleau, bauer, ridouard}@ensma.fr †

***Abstract*—We demonstrate that, for periodic systems with offsets (tasks or flows of messages within a time-triggered network), the simulation cycle can be confined to the range of $[0, \text{hyperperiod})$ only under the condition that an idle point exists at the hyper-period. Furthermore, we establish that ensuring both (1) contention-freedom and (2) that no offset exceeds the value of the period minus duration, is a sufficient condition to guarantee the presence of an idle point at the hyper-period. Most contemporary methods aiming to eliminate latency are implicitly based on these properties and fail to propose a schedule if the input system does not allow a contention-free solution. Consequently, we propose a heuristic approach to scheduling periodic flows of frames within time-triggered networks. Our method focuses on minimizing latency, without necessitating a solution where every frame at every output port is contention-free, and it effectively manages the cyclicity problem.**

I. Introduction

This paper is a product of extensive deliberations between researchers with expertise in real-time scheduling and those in real-time networks. Since the advent of time-triggered networks, an increasing number of papers address what real-time schedulers call an offset-free problem (where the designer chooses the offset). We have periodic flows of frames to be sent over a network, and the question to be addressed is how to assign an offset to the flows in order to be able to schedule them with low or no latency over the output ports of the encountered switches. While offset-free systems in real-time scheduling have been mainly studied from the late 1990s to the early 2000s, in works such as [1], offset-free studies in networks are more recent [2], but are more focused on recent standards, especially regarding Scheduled Traffic in Time-Sensitive Networking (TSN).

One of the questions that emerged was related to the cyclicity problem: in real-time scheduling, as established by the seminal work of [3], the maximum simulation duration for a task system with offsets is $O_{\max} + 2H$, where H represents the hyper-period, given by the Least Common Multiple (LCM) of the periods of the tasks, and $O_{\max}$ the largest offset. The schedule over $[O_{\max} + H, O_{\max} + 2H)$ is guaranteed to repeat infinitely. However, certain reservations arose when it was observed that every offset-free method applied to time-triggered network scheduling only considers the window $[0, H)$. To the best of our knowledge, there is no formal proof showing that the study can be confined to this window. We, therefore, decided to investigate the validity and applicability of this assumption. The methodologies adopted and our resulting conclusions are elaborated in this paper.

We formally demonstrate that forcing an idle point (a point where there is no pending work). to occur at time H constrains the studied window of periodic tasks (or periodic flows) with offsets to $[0, H)$. Additionally, we establish that a sufficient condition to obtain this idle point is to (1) ensure that, at any time in the schedule, at most one frame is ready to be sent (reffered to as *Contention-freedom*), and (2) have offsets not greater than the period minus the duration of the message (referred to as the *Frame constraint* in subsequent sections).

Since constraint (1) may be hard, or even unachievable for some systems, we additionally propose a novel heuristic able to reduce latency, even if constraint (1) cannot be met. This heuristic does not invariably comply with the Contention-freedom constraint, but is not affected by the problem of cyclicity of the schedule.

The ensuing sections are organized as follows: Sec. II provides a comprehensive review of published work focused on the generation of a feasible schedule through the identification of appropriate offsets for Time-Triggered (TT) Networks. Sec. III uses straightforward examples to explain why the assumption of $[0, H)$ as the cycle of a schedule is incorrect when fulfilling just one of the two conditions (Contention-freedom OR Frame constraint). Sec. IV explores the essence of the problem with identifying the beginning of the cyclic part of a schedule with offsets, by first showing the main result of the paper – Contention-freedom AND Frame constraints are a sufficient condition for the cycle to act on $[0, H)$ –, and then exploring the way the start of the cyclic part of a schedule can be computed in a general context. In Sec. V, we propose a novel heuristic method to find a feasible schedule for a given network and set of periodic flows even when the cycle is not the first hyper-period. Lastly, Sec. VI offers a description of the heuristic performance in comparison with the seminal optimal method.

II. State of the Art

A. Notation

A switched network, where flows of non-preemptive frames can be transmitted, is represented as a set of nodes $\mathcal{N} = \{\mathcal{N}_k\}$. Each node, $\mathcal{N}_k$, represents an output port of either an End Station (ES) or a Switch (SW). Every link of the network is operating at a constant transmission rate of r.

The network is used by flows denoted S_i with $i \in \mathbb{N}^*$. A flow is defined by the tuple:

$$S_i = \{C_i, T_i, \{O_{i,k}\}, \{\mathcal{P}_{i,m}\}\} \tag{1}$$

where C_i is the transmission duration of each periodic frame of S_i (given by its frame size divided by r), T_i is the period of the flow, $\{\mathcal{P}_{i,m}\}$ is a set of paths for S_i. A path is an ordered set of nodes. $\{O_{i,k}\}$ constitutes an offset for the flow S_i in each node k of $\{\mathcal{P}_{i,m}\}$. For the sake of simplicity, we consider single frame flows only, i.e., there is only one frame to be sent each period of the flow. For a given flow S_i, every path $\{\mathcal{P}_{i,m}\}$ starts with the same first node, called *source node*, but has a different destination node (concept of *multicast* flows). Paths from a given flow cannot rejoin once they have diverged.

B. Schedule synthesis in time-triggered networks

Approaches developed for the generation of schedules can be primarily categorized into two groups: Satisfiability Modulo Theories (SMT) or Optimization Modulo Theories (OMT) constitute a category of optimal methods, while Heuristics form the other.

1) SMT/OMT methods: The seminal paper that used this methodology for TTEthernet networks is [2]. This method produces schedules for network flows that respect SMT constraints. The constraints proposed have later been extended to other protocols, such as TSN standards [4] [5] [6] [7] [8].

SMT methods use constraints to synthesize schedules. We will use [2] as a reference paper for the constraints with the additions given by [9] and [10]:

- Path-dependent constraints: The offset of a frame at an output port must exceed the offset given on any previous port of the path plus a user-defined variable delay, representing the maximum switching fabric processing time;
- Bounded switch memory constraints: A set of constraints limits the number of frames that can be stored in queues at an output port;
- End-to-end Transmission constraints: The offsets should be placed in a way that end-to-end latency is bounded by a given value;
- Application Level constraints: Transcribes precedence constraints between messages;
- Contention-free constraints: At any time, only one frame at a time is ready to be transmitted in an output port;
- Frame constraints: Introduced with [9] in 2014 for TTEthernet and with [10] in 2016 for TSN: constraining the offsets to be in the interval $0 \leq O_{i,k} \leq T_i - C_i$.

An analogous formulation as an Integer Linear Programming (ILP) has been proposed in [11].

The constraints are formulated on the frames of the time interval $[0, H)$ because the schedule on this window is assumed to repeat.

Note that, while optimal, these methods all require the contention-free constraint, this implies that they cannot propose a solution if this constraint cannot be met (e.g., if messages passing by the same ports have co-prime periods).

2) Heuristics: Heuristic methods for schedule synthesis have been designed for schedule generation within a time-triggered network. These methods have mostly been studied in the context of TSN standards.

Various heuristic methods have been proposed since the seminal paper [2]. The Heuristic List Scheduler [12] minimizes the time used to find valid schedules within generic time-triggered networks. In [13], the authors have proposed strategies to guide the SMT schedule generation method defined in [2]. In [14], the same was done with the help of ILP. A method based on genetic algorithms has also been proposed in [15].

Methods based on the Tabu search algorithm have been proposed for both TTEthernet [16] [17] [18] and TSN [19]. Another method [20] for TTEthernet has been proposed to reduce the communication schedule time while maximizing the uninterrupted gap for non-scheduled traffic. Scheduling in software-defined networks has also been addressed in [21]. The authors in [22] describe a method that tolerates assignment conflicts, with an emphasis on in-vehicle systems. A heuristic algorithm has been proposed [23] to schedule time-triggered networks on large scale networks with numerous flows. [24] uses a window-based scheduling algorithm to produce schedules for TSN networks. The algorithms described in [25] manage topology, routing and schedules synthesis for TSN, with an emphasis on fault-resilience.

All these methods consider frames over the window $[0, H)$. Some require their solution to meet the (1) Contention-freedom constraints at each output ports, and some require their offsets to meet the (2) Frame constraint. We show in the sequel that considering $[0, H)$ without both constraints can produce flawed schedules. We note that these heuristics methods cannot produce an offset assignment if the Contention-free constraint cannot be met.

C. Cyclicity of schedules on uniprocessor systems

In time-triggered networks; each output port of a switch is often regarded as a processor queue, scheduling tasks which represent message flows, each job representing a frame. Therefore, while the literature on the problem of cyclicity is extensive, we can limit ourselves to the papers considering uniprocessor scheduling.

The problem of finding a cycle in a schedule of periodic tasks with offsets was initially addressed in [3], [26], where two fundamental results concerning task systems with offset on uniprocessor systems are presented. First, it is demonstrated that an infinite schedule obtained by a priority assignment (note that the proofs are still valid for any deterministic and memoryless schedule) for a system of periodic tasks

with offsets and implicit deadlines is valid if, and only if, it is valid over the interval $[0, O_{\max} + 2H)$, and that the schedule given over the time interval $[O_{\max} + H, O_{\max} + 2H)$ repeats indefinitely. We will refer to this interval that repeats as the cyclic part of a schedule. This has been extended to arbitrary deadlines in [27]. Secondly, in [3], [26] the problem of determining if a system of asynchronous tasks is schedulable on a single processor is shown as NP-hard in the strong sense. In other words, nothing simpler than a simulation or equivalent, operating over at least a hyper-period, can be employed to prove schedulability. The problem is that a hyper-period, unless periods are harmonic, is exponential, as $\mathrm{LCM}\{1, 2, 3, ...n + 1\} \geq 2^n$ (see a proof in [28]).

Then, a specific feasibility interval for preemptive fixed-task priority scheduling algorithms is presented in [1]. The proposed interval, which takes priorities into account, includes the cycle if, and only if, the system is feasible by the chosen priority assignment.

An exact way to compute the acyclic and cyclic parts of a schedule, which depends on offsets, periods, and durations only, is presented in [29]. The cyclic part of a valid schedule produced by any conservative and memoryless scheduler[1] for periodic tasks with offsets is identified as the first interval of size H following the last extra idle time. It could be conceptualized as the first interval of length H without any extra idle time. Moreover, the latest possible extra idle time appears before $O_{max} + H$. In this paper, our objective is to provide the theoretical groundwork required for methods trying to assign offsets to eliminate or reduce the interference on a node, be it a processor scheduling periodic tasks, or an output port of a switch scheduling periodic message flows. For this purpose, we revisit [29] and expand upon the findings in that paper to show what works and what does not work in the state-of-the-art papers that try to assign offsets to message flows. To the best of our knowledge, all the proposed methods limit themselves to the interval $[0, H)$. While in general, considering only the frames in this window is insufficient to assign offsets to flows of messages, we identify specific scenarios where this interval is indeed sufficient.

III. The problems with the hyper-periods

A. Cyclicity vs. Contention-freedom

In an asynchronous environment, frames from multiple flows can be competing for the same switch output port. This competition can cause fluctuation in buffer occupancy and transmission delays. Such an issue may be mitigated by scheduling strategies in synchronous networks.

To the best of our knowledge, every proposed method for scheduling flows within switched networks involves assigning an offset to each flow in each output port. To achieve this, periodic flows of frames are conceptualized as frames over a hyper-period of the periods of the flows. However, there is a missing link between observations made in scheduling theory and the assumption that a schedule in an output port will repeat itself post the initial hyper-period. Without giving special attention to the properties of the schedule of flows with offsets, we could encounter schedules where the cycle fails to repeat itself after the first hyper-period. We illustrate this phenomenon using straightforward examples.

[1]Any scheduler based only on the current state of the system, and not its history, to make a decision. A conservative scheduler does not let the processor idle if there is pending work. All classic schedulers are conservative and memoryless.

TABLE I
Traffic parameters for the examples from Fig. 1

	Flow	Period	Transmission Time	Offset
Example Case 1	1	12	8	0
	2	18	5	8
Example Case 2	1	12	8	5
	2	18	5	0
Example Case 3	1	7	2	0
	2	7	4	4

We introduce a primary example featuring two flows crossing a node, detailed as Case 1 in Tab. I. In this instance, it is impossible for the two flows to entirely avoid interference. An overlap is inevitable at some point in the schedule, because $C_1 + C_2 = 13 > \mathrm{GCD}\{T_1, T_2\} = 6$, where GCD stands for the Greatest Common Divisor (the method is explained in Lem. V.2). The simulation of this example is depicted in Fig. 1(Case 1).

In this example, we can notice that the first hyper-period, $[0, \mathrm{LCM}\{12, 18\}] = 36$, is not a cycle that repeats. The schedule necessitates construction after the latest extra *idle time*, occurring at time $[21, 22)$, to yield a cyclic hyper-period. Given that the processor utilization of the system is $U = \sum_i \frac{C_i}{T_i} = \frac{34}{36}$, in a cycle of length equal to the hyper-period, we should find exactly two *idle times*: $H(1 - U) = 2$. If we consider the first hyper-period $[0, 36)$ (see Fig. 1(Case 1)), there are not only two but three *idle times*, forming an idle slot on the time interval $[21, 24)$. The first of these three *idle times* is deemed an *extra idle time*, because if a window of size $H = 36$ is presented within the interval $[22, 58)$, it emerges as the first window of size equivalent to the hyper-period with precisely two *idle times*. As a result, to study the whole behavior of the schedule, we have to consider five frames for flow 1, and three frames for flow 2, while the cyclic part contains only three and two frames respectively.

One of the problems that methods assigning offsets face is that they all consider frames (resp. jobs), while the number of frames to consider depends on the offsets. Therefore, in general, for example in SMT, there is a cyclic dependency between the solution and the input variables. To illustrate this, if we consider the same periodic tasks as in Case 1, but change the offsets as $O_1 = 5$ and $O_2 = 0$ (Case 2), the schedule does not reach its cycle at the same time, and a different number of frames for each flow should be considered. As depicted in Fig. 1(Case 2), we see that the latest extra *idle time* occurs at time $[14, 15)$, and the schedule repeats on the interval $[15, 51)$. Therefore, if a method requires considering every frame in a schedule, it should consider four frames for flow 1 (instead of five for Case 1), and three frames for flow 2.

Consequently, when Contention-freedom is not met (i.e.,

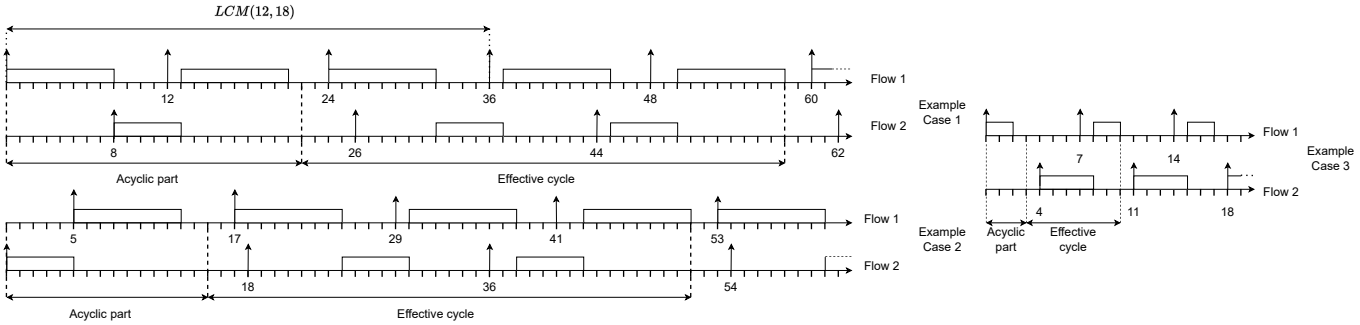

Fig. 1. Acyclic and cyclic parts of a schedule, based on parameters from Tab. I

several frames can be concurrently ready to be sent), considering the first hyper-period of a schedule of tasks or frames with offsets is not sufficient.

B. Importance of the Frame constraint

We show in this section that meeting the Contention-freedom constraint is not sufficient to ensure that the schedule $[0, H)$ repeats itself. We use the Contention-free constraint as proposed in [10] because it has been simplified to reduce its complexity compared to previous works.

$$\begin{aligned}
&\forall \mathcal{N}_k \in \mathcal{N}, \forall a, b \in \mathcal{N}_{k,Flows}, a \neq b \\
&\forall \alpha \in [0, \mathrm{LCM}\{T_a, T_b\}/T_a], \forall \beta \in [0, \mathrm{LCM}\{T_a, T_b\}/T_b] : \\
&(O_{a,k} + \alpha \times T_a \geq O_{b,k} + \beta \times T_b + C_b) \vee \\
&(O_{b,k} + \beta \times T_b \geq O_{a,k} + \alpha \times T_a + C_a)
\end{aligned} \tag{2}$$

where $\mathcal{N}_{k,Flows}$ is the set of flows crossing the node $\mathcal{N}_k$. With this set of constraints, the number of frames to take into account is limited by the number of frames that we can find in the first hyper-period of length $\mathrm{LCM}\{T_a, T_b\}$. This constraint necessitates sequential transmission of frames within a busy period: a frame can be released only when no other frame is pending. However, if we do not include the Frame constraint, the set of constraints given in Eq. (2) is not sufficient to consider all the frames that appear in the schedule.

This phenomenon is illustrated in a simple example shown in Fig. 1, with two flows crossing a given node (their characteristics are described as Case 3 in Tab. I).

In this case, the schedule starts to behave cyclically after the latest extra idle slot that occurs during $[2, 3)$. The schedule behaves cyclically on $[3, 10)$, and the number of frames to consider as representing the whole schedule (both acyclic and cyclic parts) includes two frames of flow 1 and one frame of flow 2. This implies that, if we limit ourselves to one frame per flow, in accordance with constraints Eq. (2), we then neglect the second frame of flow 1, and the delay it incurs due to the first frame of flow 2. We would therefore conclude erroneously that, in this system, there is no contention at all. However, this example does not comply with the Frame constraint. In fact, the Frame constraint, imposing that $O_i \leq T_i - C_i$, would forbid the first frame of flow 2 to finish after $H = 7$.

This example shows that when the Frame constraint added in [9] and [10] is not respected, schedules that do not behave cyclically from the start can be found. Another risk would be the generation of a schedule from $[0, H)$ that includes extra *idle times*. Because of the extra *idle times* present in the schedule, there would be insufficient time dedicated to frames to be able to schedule every frame in each hyper-period. Frames would accumulate at the output port until after a certain number of hyper-periods, queues would reach full capacity, resulting in frames being dropped.

As a conclusion for this section, if Frame constraints are met, but some frames have the potential to interfere with each other, the system may not behave cyclically over $[0, H)$. Moreover, if we only consider the frames released in the first hyper-period, but do not respect the Frame constraint, then we may be led to the erroneous conclusion that no interference occurs.

In the subsequent section, we demonstrate that the conjunction of Contention-freedom (i.e., busy periods are limited to one frame) and Frame-constraint forms a sufficient condition to guarantee that $[0, H)$ is the cyclic part of a schedule. For this purpose, we delve into the problem of the cyclicity of schedules of tasks with offsets in the next section.

IV. Cyclicity and acyclicity in schedules with offsets

A. Fundamental cyclicity results

To analyze schedulers, it is necessary to establish a comprehensive definition of a scheduler in relation to a task system that requires scheduling. These definitions are derived from [30]. At any given time t within a schedule, we represent $C_i(t)$ as the pending work required to complete the ongoing active job of τ_i. It assumes a value of zero when the respective task has no incomplete active job. The cumulative pending work for processing is defined as $C(t) \stackrel{\text{def}}{=\!=} \sum_{i=1}^{n} C_i(t)$.

Definition IV.1 (Local clock of a periodic task)**.** The local clock of a task τ_i, denoted $\Theta_i(t)$, is a non-negative, real-valued, continuous periodic function modulo T_i, such that it varies between 0 and $T_i - \epsilon$ for any arbitrary small non-negative ϵ. It is defined on the open interval $(O_i - T_i, +\infty)$, and undefined before. If $O_i < T_i$, then $\Theta_i(0) = (O_i - T_i) \bmod T_i = T_i - O_i$, such that the local clock reaches zero (i.e., T_i modulo T_i) at O_i. If $O_i \geq T_i$, then it is undefined at

and before $O_i - T_i$, and takes the value $\epsilon > 0$ at $O_i - T_i + \epsilon$, such that it is reset to zero at O_i.

It can be observed that, according to basic algebra [31, Section 4.5.2], the set of local clocks is periodic with a period H, commencing from zero if no offset is greater than or equal to the period, and commencing from the maximum value of $O_i - T_i + \epsilon$ if there exist tasks τ_i with $O_i \geq T_i$.

As tasks are released, the amount of work to be processed increases. We denote $r_i(t)$ as the characteristic release function of τ_i, defined as zero everywhere except when its local clock is zero, i.e., when t satisfies $\Theta_i(t) = 0$. In this case, $r_i(t) = C_i$. The total released work at time t is $r(t) = \sum_{i=1}^{n} r_i(t)$. At any given time t, the individual workload to be processed for a task is $w_i(t) = C_i(t) + r_i(t)$, and this workload decreases by δ whenever τ_i is executed in the interval $[t, t + \delta)$. The sum of the individual workloads is denoted $w(t)$. All of these functions are non-negative since a processor cannot execute non-existent work.

Definition IV.2 (State of a system, deterministic and memoryless scheduler)**.** The state of a system at time t is determined by the values of the individual workloads $w_i(t)$ of the tasks and the local clocks $\Theta_i(t)$. When making a decision, a deterministic and memoryless scheduler takes the same action for the same state.

Definition IV.3 (Idle point, idle time, idle slot, conservative scheduler)**.** An *idle point* is a moment in time during which the processor (or the link in a network) is not busy. Utilizing our notation, an idle point t is characterized by $\forall i$, $C_i(t) = 0$, which is equivalently defined as $C(t) = 0$, and due to the non-negative nature of the functions, further equivalent to $w(t) = r(t)$. During an idle point, the workload is solely determined by the release of new jobs, if any are present.

The initial idle point following a busy period signifies the conclusion of said busy period. An *idle slot* is initiated by an idle point and denotes a continuous period of inactivity extending from this idle point until the subsequent job (or frame) is released. It is defined as a window of size $\delta > 0$ that ensues an idle point at t, satisfying $\forall i$, $\forall \theta \in [t \dots t + \delta)$, $r_i(\theta) = 0$.

We establish the concept of an *idle time*, representing the shortest possible idle slot. In cases where task parameters are integers, an idle time takes the form of an idle slot confined to a unit of time. For rational parameters, the minimum duration of an idle slot, referred to as an idle time, can be articulated as $\frac{1}{\mathrm{LCM}_d}$, where LCM_d denotes the least common multiple of all denominators appearing within the values of task offsets, durations, and periods.

A *conservative scheduler* does not leave the processor idle unless $w = 0$.

The general cyclicity theorem, as presented in [29], which is shown for integer parameters, can be generalized to rational numbers. It can be expressed as follows:

Theorem IV.1 (CYCLICITY)**.** *[29] When relative deadlines are not greater than the periods, any feasible schedule produced by a conservative, deterministic and memoryless scheduler behaves cyclically with a period H after the latest extra idle time. This extra idle time occurs in the time interval $[-1, O_{\max} + H)$ where $O_{\max} = \max\{O_i \ \forall i\}$ and $H = \mathrm{LCM}\{T_i \ \forall i\}$.*

This theorem gains clarity when examined in the context of Fig. 1 (Case 3). With $H = 7$ and a processor utilization of $U = \frac{6}{7}$, a feasible schedule must encompass precisely $H \cdot (1-U) = 1$ idle time within a cycle. Without this, the schedule would incur increasing delays across hyperperiods. The interval $[0, 7)$ encompasses two instances of idle time. Consequently, one of them becomes an extra idle time.

The first window with exactly one idle time is $[3, 10)$, constituting the cyclic portion of the schedule. The window $[0, 3)$ represents the acyclic segment of the schedule, culminating in the final extra idle time $[2, 3)$. This segment is termed "acyclic" due to its execution only once at the start of the schedule.

It should be noted that the specific case where the latest extra idle time occurs on $[-1, 0)$ aligns to the case where the cycle starts at time 0, and the window $[0, H)$ is repeated. Furthermore, when the latest extra idle time ends at $O_{\max} + H$, which is, given Th. IV.1, its latest possible position in a schedule, we find the upper bound on cyclicity $O_{\max} + 2H$ presented in [3].

Theorem IV.2. *Both acyclic and cyclic parts of a feasible schedule start and end at an idle point.*

Proof. Referring to Def. IV.3, at time 0, $w(0) = r(0)$, which is an idle point. Now, let's consider two scenarios: (1) the absence of an acyclic segment in the schedule. In this instance, it is defined over the interval $[-1, 0)$ where the workload is null. Thus, it both commences and concludes at an idle point, forming an idle time from inception to termination. The cyclic part of the schedule operates over $[0, H)$: time 0 is an idle point, and in accordance with Th. IV.1, the same state is attained at H, also representing an idle point. (2) The presence of an acyclic idle time, denoted as t_x, which marks its termination. Consequently, the acyclic segment of the schedule is $[0, t_x)$, starting at 0 as an idle point and concluding at t_x. Since it remains idle and the scheduler is conservative, $C(t_x) = 0$, affirming it as an idle point. The cyclic part of the schedule operates over $[t_x, t_x + H)$, both instances representing the same state according to Th. IV.1. Notably, since t_x is an idle point, so is $t_x + H$. □

Corollary 1. *There is an integer number of jobs (resp. frames) in the acyclic part of a schedule, and an integer number of jobs (resp. frames) in the cyclic part of a schedule.*

Proof. Following Th. IV.2, since the cyclic part starts and ends with an idle point, there is no pending job at the end of the cycle, and every job that is released in this interval is also ended in this interval. □

Cor. 1 bears positive implications for methods working on flows with offsets, because it implies that frames do not

overlap between acyclic and cyclic parts of a schedule. It is not sufficient for time-triggered networks where methods limit the window to $[0, H)$. This is why we need the next theorem and its derived corollary.

Theorem IV.3. *Given the hypothesis that no offset is greater or equal than the period, a schedule behaves cyclically over $[0, H)$ if, and only if, there is an idle point at H.*

Proof. For the "if": if there is an idle point at H, since $\forall i, O_i < T_i$, from Def. IV.1, local clocks behave cyclically with a period H from time 0. There is an idle point at 0. Therefore the state of the system (see Def. IV.2), given by the values of the individual workloads and local clocks, is the same at time 0 and H. Since the scheduler is deterministic and memoryless, it makes the same decision, and the same states will be reached, making the system repeat from time H what happened from 0 to H. For the "only if": if there is no idle point at H, then states at 0 and H are different, and the system cannot behave cyclically on $[0, H)$. □

Th. IV.3 plays a pivotal role in time-triggered networks, since we can derive the main result allowing some methods able to guarantee Contention-freedom on one hand, and no frame spilling after H on the other hand, to generate schedules that are guaranteed to behave cyclically on $[0, H)$:

Corollary 2. *If a schedule of flows with offsets guarantees (1) Contention-freedom on $[0, H)$, and (2) Frame constraint, i.e., $0 \leq O_i \leq T_i - C_i$, then the schedule over $[0, H)$ is repeated infinitely.*

Proof. Contention-freedom in a conservative schedule means that the workload to be treated at the output is only given by one task (or frame), which is ended at or before the next release. This means that for every t such that $\exists i, r_i(t) > 0$, the next time where there is a j such that $r_j(t + \delta) > 0$ is such that $\delta \geq C_i$. As a result, each busy period corresponds to only one request, which is executed as soon as it is released, and immediately ended by an idle point. Since $O_i \leq T_i - C_i$, for every task, the latest possible release in $[0, H)$ can occur at $H - C_i$, and has to be executed alone (contention-freedom) and end at or before H, making it an idle point. As a result, 0 and H are two idle points, the Corollary is obtained by applying Th. IV.3.. □

Corollary 2 holds paramount importance for the integrity of numerous papers related to offset assignment in switched networks. The Frame constraint was initially introduced in 2014 for TTEthernet [9], and later for TSN in 2016 [10]. Despite this, several papers not employing the Frame constraint claim that considering frames sent over $[0, H)$ is sufficient. However, our findings in Section III-A indicate that ensuring Contention-freedom without the Frame constraint does not guarantee a cycle over $[0, H)$. Hence, it is prudent for readers to be aware that some papers may inadvertently overlook this crucial aspect, potentially affecting the accuracy of their methods.

For the next section, we showcase a method that can cope with acyclic idle times and provide schedules even when the two constraints of Cor. 2 are not respected. To the best of our knowledge, this is the only method allowing that, since its calculations are based on flows instead of individual frames.

V. A NON CONTENTION-FREE HEURISTIC

In our review of the state-of-the-art, we highlighted that every method addressing the offset assignment problem, whether exact or heuristic, requires a solution that ensures contention-freedom. However, some systems do not permit such solutions, posing unsolvable challenges for current methods.

In this section, we introduce a novel heuristic approach suitable for time-triggered networks, even when frame contention is inevitable. By operating at the flow level, this strategy focuses on direct interaction with flows instead of frames, freeing us from the constraints of the time window $[0, H)$.

Consequently, this approach can manage systems with flows that prohibit interference-free solutions, where both SMT-based methods and heuristics that require contention-freedom fail to provide a feasible solution. This heuristic, adapted from GCD+ [32], is denoted as GCD#. We start with an overview of the main concepts, which closely align with GCD+, focusing on a single node before expanding the discussion to the network level.

All parameters described in the subsequent sections are positive integers. While we could consider rational numbers, we opt to express every value as an integer for the sake of simplicity.

A. Scheduling a single node

GCD+ [32] is designed to generate offsets for uniprocessor offset-free systems. It accepts as input a set of tuples $\tau_i \in \mathbb{N}^2 \mid \tau_i = (T_i, C_i)$, where T_i denotes the period of the task (or flow) and C_i represents its duration (be it the Worst-Case Execution Time for tasks or the transmission duration over a link for flows). For simplicity in our discussion, we will refer to each tuple as a 'task'.

GCD+ strives to achieve (but not guarantee) contention-freedom, while also aiming for the response-time of each task to be equal to its execution time. To accomplish this, the algorithm leverages the semi-harmonicity of offset-free systems., i.e., it focuses on systems where all the periods T_i are a multiple of a common factor. Semi-harmonicity is sought to reduce the exploration space of potential solutions [33]–[37]. This is based on a technique originally used to analyze two tasks at a time, which is itself based on Lem. V.1, rewritten here using the modulo operator.

Lemma V.1. *[38, Lemma 2] Given two tasks τ_i and τ_j, the minimum time distance between any release time of task τ_i and the successive release time of τ_j is equal to:*

$$\Delta_{ij} = (O_j - O_i) \mod \mathrm{GCD}\{T_i, T_j\} \tag{3}$$

In a system comprised solely of two tasks τ_1 and τ_2, it is possible to represent its execution around the $\mathrm{GCD}\{T_1, T_2\}$ circle, such as in Fig. 2. In the figure, if there is no overlap in execution, then the conditions in Eq. (4) are being met.

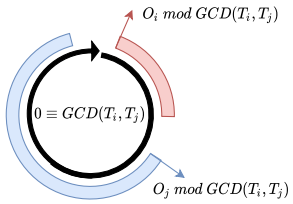

Fig. 2. $\mathrm{GCD}\{T_i, T_j\}$ circle, with a representation of τ_i and τ_j

Therefore, τ_1 and τ_2 will never interfere with each other, and the response time of each task will be equal to its execution time. Thus, we can assert that if the system of two tasks respects the following set of inequalities where $i = 1$ and $j = 2$, we can guarantee no interference.

$$\begin{aligned} \Delta_{i,j} &\geq C_i \\ \Delta_{j,i} &\geq C_j \end{aligned} \tag{4}$$

We can extend this technique to assess more than two tasks simultaneously. This is achieved by employing the concepts of sections and cycles.

Lemma V.2 (Pairwise non-interference)**.** *If every task pair in a system respects Eq. (4), no interference ever occurs in the execution of the system, and the response time of each task equals its execution time.*

Observe that Lem. V.2 can effectively replace the Contention-free constraint as expressed in Eq. (2), because they are equivalent, yet the former can be computed in polynomial time, while the latter requires an exponential time.

Definition V.1 (The overall GCD, Ω)**.** The overall GCD, also referred to as Ω, is the greatest common divisor of all the periods (Eq. (5)).

$$\Omega = \mathrm{GCD}\{T_i \ \forall i\} \tag{5}$$

Our method operates better on sets of tasks that present execution times no larger than Ω. Otherwise, it still provides results, but they will reduce contention heuristically, without guaranteeing Contention-freedom. In networks, a large Ω is common: for example, in the switched Ethernet-based network ARINC 664 part 7 used in civil avionics, every allowed period for network flows is a power of two milliseconds, and all the flows have harmonic periods.

Definition V.2 (Cycle)**.** The cycle O_C (for $O_C \in \mathbb{N}$) is the time window such that $O_C\,\Omega \leq t < (O_C + 1)\,\Omega$.

Definition V.3 (Subperiod, T_S)**.** The subperiod T_{S_i} of a task τ_i is the ratio between its period T_i and Ω (Eq. (6)).

$$T_{S_i} = \frac{T_i}{\Omega} \tag{6}$$

The offset O_i assigned to any task τ_i can be related to a specific cycle O_{C_i} according to Eq. (7).

$$O_{C_i} = \left\lfloor \frac{O_i}{\Omega} \right\rfloor \tag{7}$$

From the definition of Ω, we know that $T_{S_i} \in \mathbb{N}\ \forall i$. Moreover, considering that τ_i was assigned to cycle O_{C_i}, it is trivial to show that every release time of τ_i will happen in cycles k such that $k \equiv O_{C_i} \mod T_{S_i}$.

Lemma V.3. *For any given pair of tasks τ_1 and τ_2, if $\mathrm{GCD}\{T_{S_1}, T_{S_2}\} = 1$, they must be released in the same cycle at some point in the hyper-period of the system, and if $\mathrm{GCD}\{T_{S_1}, T_{S_2}\} > 1$, they can be arranged so that they are never released in the same cycle.*

Proof. This proof is related to the Generalized Chinese Remainder Theorem (GCRM). Let $\{T_i \in \mathbb{N} \mid i \in [1, n]\}$ and P be the least common multiple of all T_i. Let $a, O_1, O_2, \ldots, O_n$ be any integers. If $O_i \equiv O_j \mod \mathrm{GCD}\{T_i, T_j\}$ for $1 \leq i < j \leq n$, the GCRM states that there is exactly one integer k that satisfies the conditions:

$$a \leq k < (a + P), k \equiv O_i \mod T_i\ \forall i \in [1, n]$$

Otherwise, no such integer exists. If $\mathrm{GCD}\{T_{S_1}, T_{S_2}\} = 1$, regardless of the first cycle (O_{C_1} and O_{C_2}) where they are released, $O_{C_1} \equiv O_{C_2} \equiv 0 \mod 1$, and there must exist an integer $k \equiv O_{C_1} \mod T_{S_1} \equiv O_{C_2} \mod T_{S_2}$. Otherwise, if $\mathrm{GCD}\{T_{S_1}, T_{S_2}\} > 1$, O_1 and O_2 might be chosen so that $O_1 \not\equiv O_2 \mod \mathrm{GCD}\{T_1, T_2\}$, making it impossible for such k to exist. □

In light of Lem. V.3, the concept of a section can emerge. Tasks that have coprime sub-periods must be scheduled considering they will eventually be released in the same cycle, and therefore a certain portion of the cycle must be reserved for each task if we are looking for a Contention-free offset assignment. However, non-coprime sub-periods will allow tasks to use the same reserved portion of the cycle, since they can be assigned to alternating cycles. These reserved portions are the sections to which the tasks are assigned.

Definition V.4 (Section, Ψ)**.** A section Ψ is a subset of tasks such that their sub-periods are either all equal to one, or they must share a common prime factor. Thus, each section is related to a number, the common factor in the factorization of the subperiods of the section members (or 1, in the case where the subperiods are unitary), denoted here as $p \in \mathbb{P} \cup \{1\}$, where $\mathbb{P}$ is the set of all prime numbers. In order to create a reserved partition in every cycle for each section, each section Ψ_p is given a start time O_S relative to the beginning of the cycle and has a finite size δ, given by Eq. (8).

$$\begin{aligned} \Psi_p.\delta = &\max_{\forall i \mid \tau_i \in \Psi_p} ((O_i + C_i) \mod \Omega) \\ &- \min_{\forall i \mid \tau_i \in \Psi_p} (O_i \mod \Omega) \end{aligned} \tag{8}$$

Then, for every cycle $k \in \mathbb{N}$, the interval $[k\,\Omega + \Psi_p.O_S, k\,\Omega + \Psi_p.O_S + \Psi_p.\delta)$ is reserved for the execution of tasks in the section Ψ_p. In other words, every job of a task in Ψ_p is only released at an instant r_{ij} such that $r_{ij} \geq k\,\Omega + \Psi_p.O_S$ and $r_{ij} + C_i \leq k\,\Omega + \Psi_p.O_S + \Psi_p.\delta$.

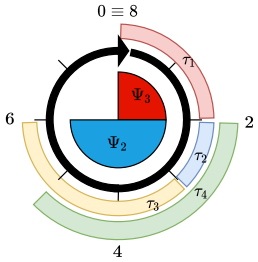

Fig. 3. Modular circle of size Ω with example tasks

In each section, O_S is assigned as a partial offset to every task in the section. Inside each section, however, some tasks might need to occupy the same cycle as well. In these cases, one of the tasks has to be set to be released only after the other finishes its execution. For this, an internal offset O_{I_i} is added to the final offset. Hence, the final offset can be obtained from Eq. (9).

$$O_i = \Omega\, O_{C_i} + O_{S_i} + O_{I_i} \tag{9}$$

In addition, sections must begin after previous sections end in order to guarantee the isolation between sections. Furthermore, to optimally fit every section in the overall GCD, a section should begin at the instant the previous section ends, avoiding the creation of small gaps where no section would fit. This means that, if $\Psi_1.O_S = 0$, then $\Psi_2.O_S = \Psi_1.\delta$, $\Psi_3.O_S = \Psi_1.\delta + \Psi_2.\delta$, and so forth.

To better understand the concepts, we can work with the four-task system containing the following tasks:

$$\tau_1 = (T_1 = 24, C_1 = 2)$$
$$\tau_2 = (T_2 = 16, C_2 = 1)$$
$$\tau_3 = (T_3 = 16, C_3 = 3)$$
$$\tau_4 = (T_4 = 16, C_4 = 3)$$

In this system, $\Omega = 8$. Therefore, $T_{S_1} = 3$ and $T_{S_2} = T_{S_3} = T_{S_4} = 2$. We need then to work with two sections: one for $p = 2$ (Ψ_2) and one for $p = 3$ (Ψ_3). Ψ_3 only contains τ_1, and the other tasks are contained in Ψ_2. Choosing the cycle 0 for τ_1 and τ_2, we can choose cycle 1 for τ_4 and again cycle 0 for τ_3. Since τ_2 and τ_3 share the same cycle in the same section, τ_3 needs to have an internal offset $O_{I_3} = C_2$.

Finally, it is now feasible to compute the sizes of the sections. The size of Ψ_3 is 2, and the size of Ψ_2 is 4. We can arbitrarily choose $\Psi_3.O_S$ to be 0, and then $\Psi_2.O_S = \Psi_3.\delta = 2$. Hence, every offset is calculated according to Eq. (9): $O_1 = 0$, $O_2 = 2$, $O_3 = 3$ and $O_4 = 10$.

The visual representation of these calculations can be seen in Fig. 3. Also, we can observe from Fig. 4 the execution of the task system divided in cycles, based on Fig. 3. Task τ_1 will have to share a cycle with every other task, but τ_2 and τ_4, for example, are never in the same cycle.

Lemma V.4. *If $C_i \leq \Omega\ \forall i$ and tasks are assigned to sections such that $\sum \Psi_p.\delta \leq \Omega$ for all sections (considering $\Psi_p.\delta = 0$ for empty sections), then the tasks are contention-free, and every task's response time equals its execution time C_i.*

Proof. The proof relies on analyzing each possible task pair in a system, proving they can never cause any delay in the execution of one another. Therefore, by design, and taking into consideration Th. V.2, the whole system of n tasks cannot present any execution delay.

Considering any task τ_i, there are three possibilities to analyze a second task τ_j:

1) τ_j is assigned to a different section than τ_i;
2) τ_j is assigned to the same section but in a non-congruent cycle than τ_i;
3) τ_j is assigned to the same section and a congruent cycle to τ_i.

In the first case, based on the definition of sections, the execution of tasks inside a section is entirely contained in that section. Considering the sum of the section sizes is at most Ω, knowing that $\mathrm{GCD}\{T_i, T_j\}$ is at least equal to Ω, then it is easy to see that the conditions in Eq. (4) will be respected.

For the second case, knowing that $\sum \Psi_p.\delta \leq \Omega$, that each task execution is contained in the limits of its section, and that τ_i and τ_j do not execute in the same cycle, then each task will always be executed inside a different cycle. Therefore, every job of τ_i will have finished its execution before τ_j starts, and vice versa.

For the third case, thanks to the internal offset O_I and the fact that a section size is not greater than Ω, then every job of τ_i will have finished its execution before τ_j starts, and vice versa. □

If the conditions in Lem. V.4 are not satisfied, GCD+ cannot guarantee contention-freedom. However, it can still provide an offset assignment. For instance, if we add τ_5 to the example such that $T_5 = 40$ and $C_5 = 3$, GCD+ would need to assign τ_5 to a new section, with $p = 5$. Therefore, Ψ_5 would start after Ψ_2 (hence $\Psi_5.O_S = 6$) and would have a size of 3 units, creating an overlap with the beginning of Ψ_3.

In light of Lem. V.4, it becomes clear that the algorithm choices must seek to maintain section sizes as small as possible.

B. GCD#

Using the concepts for a single node, GCD# adapts the method in order to account for flows in a network. For this adaptation, we considered some properties, which are commonly done in the context of a Time-Triggered network, to define a context in which GCD# should perform well. These are not constraints, since GCD# can be run without these properties to be true.

1) The maximum transmission time should be smaller than Ω, the GCD of all the periods of the flows.
2) All nodes operate at the same speed, rendering the C_i value constant across the network.
3) Since we focus only on scheduling points, only the output ports of switches and end-stations are considered, and will be designated as nodes of the network. Flows passing through the same node can interfere with each other, while flows that never share a common node

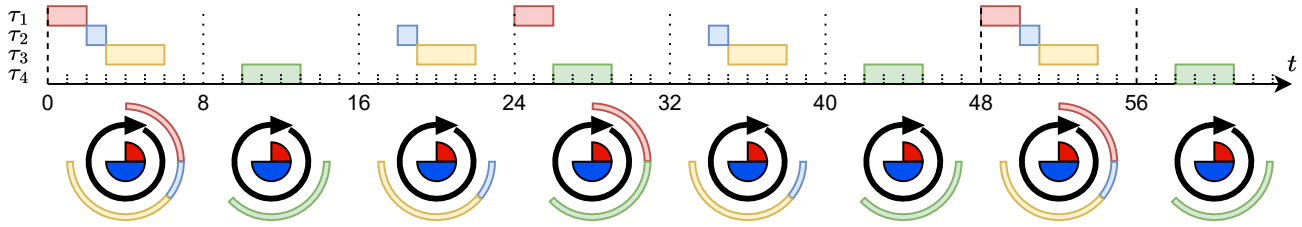

Fig. 4. Execution of the example tasks divided in cycles

cannot directly interfere with each other (even if they pass through the same switch).

4) The time taken for any frame to be received and resent by any node is constant ("Store & Forward", or $S\&F$ cost). This time is dependent on the time required by the largest frame to be processed by a node and is constant across all the network.
5) The time taken for any frame to be transferred between any two nodes is zero, but is accounted for in the Store & Forward delay.
6) The end-to-end delay, i.e., the time it takes for any frame to arrive at its destination, regarding the time it was sent from its source node, is smaller than Ω.

Assumptions 2, 3, 5, and 6 are common in the literature. Assumption 1, coupled with Assumption 4, is due to the fact that GCD# works globally on the flows by assigning a unique offset to each flow at its source node. This offset will be incremented by the $S\&F$ delay with every hop of the network. Since there is a unique GCD circle used to compute the offsets, if a message cannot fit in this circle, then contention cannot be avoided by GCD#. In this case, as shown Sec. VI-A, GCD# still provides offsets, but the existence of contention is highly probable. Note that if such event occurs locally on any output port of a node, it is not possible to avoid contention, even for optimal methods. We can note that in real TT networks, Assumption 1 is generally met.

The algorithm consists of five phases: choosing the section for each flow; choosing a cycle for each flow in a given section; choosing an internal offset for every flow that shares a section and a cycle with another; calculating section sizes; and calculating the final offset.

1) Choosing the section: The task of selecting sections can be characterized as a bin-packing problem. We aim to determine the appropriate "bin" for each flow such that the cumulative size of all the "bins" is minimized. It is important to note that each flow will carry a specific weight, contingent upon its placement within the section. To address this, GCD# employs a Best Fit Decreasing algorithm, as elucidated in the subsequent paragraphs.

Every flow with $T_S = 1$ must be assigned to Ψ_1, and every flow for which T_S is composed of a single prime number p must be placed in the corresponding section Ψ_p. Among the other flows, the one with the highest value for C is the first to be analyzed.

If none of the potential sections have flows assigned to them, preference is given to the smallest prime p_L. since it has the highest probability of also composing other sub-periods (considering the probability of a prime number to compose a random natural number). By doing so, we enhance the probability of sharing the section with other flows. This strategy avoids the creation of superfluous sections, subsequently minimizing the overall section sizes (Lem. V.4).

If a potential section is already occupied by a flow, the preference is given to the occupied section. This might reduce the need to increase section sizes, since adding the flow to a new section would definitely increase the section's size from 0 to C_i, while only in the worst case the already occupied section would need to have its size increased by C_i. Yet, in more favorable cases, the ability to alternate cycles could result in a more modest (or even nonexistent) increment in section size.

Amongst the occupied sections, the preference is given according to a score function. The score represents an estimation about how many occupied cycles there are in the section (since no cycles were defined) with respect to a candidate S_i, and is given by Eq. (10). This equation is based on the principle that each flow $S_j \in \Psi_p$ will have a chance of $1/\operatorname{GCD}\{T_{S_i}, T_{S_j}\}$ to occupy a cycle congruent to the cycle assigned to S_i. However, if a score is above the value of 1, we consider that it loses meaning, since there exists a chance that every cycle is already occupied, and the best choices for assigning S_i in this case would require information we do not have yet. Therefore, the score is limited to 1.

$$\Psi_p.\text{score}(S_i) = \min\left(1, \sum_{\forall i | S_j \in \Psi_p} \frac{1}{\operatorname{GCD}\{T_{S_i}, T_{S_j}\}}\right) \quad (10)$$

2) Choosing a cycle: Once every flow has been assigned to a section, the next step is to determine the cycles for each flow. For each section, its flows are analyzed according to a decreasing order of their message sizes, as large messages have greater impact. Since we consider that the end-to-end delay is smaller than Ω, each flow S_i will occupy only cycles congruent to a single value $k \mod T_{S_i}$.

To achieve this, a vector of possibilities is constructed, with size T_{S_i}, populated with zeros. Then, by analyzing every other flow S_j in the same section which has already been assigned, and checking that they do cross with the analyzed flow S_i (sharing the same node at some point), C_j is added to every position in the vector of possibilities that are congruent to $O_{C_j} \mod \operatorname{GCD}\{T_{S_i}, T_{S_j}\}$. Assigning S_i to any of these positions

guarantees that S_i and S_j will eventually share the same cycle (Lem. V.3). Therefore, the best choice is the position in the vector with the lowest sum – ideally, 0.

3) Choosing an internal offset: Once each flow has been assigned a cycle, the internal offsets are analyzed for every flow. To assign an internal offset to a flow S_i, we only need to consider flows S_j that cross S_i, are in the same section, have a congruent cycle (mod $\mathrm{GCD}\{T_{S_i}, T_{S_j}\}$) and have already been assigned an internal offset O_I. If no such S_j exists, then $O_{I_i} = 0$.

The "Store & Forward" ($S\&F$) constant needs to be considered. If S_j shares a certain node $\mathcal{N}_k$ with S_i, and $\mathcal{N}_k$ is the n^{th} node for both the respective paths, then both flows will experience the same $S\&F$ delay. In this case, the flows must be analyzed as if there was no $S\&F$, and the internal offset is chosen such that the following conditions are true $\forall j$:

$$\begin{aligned} O_{I_i} \leq O_{I_j} &\Rightarrow O_{I_i} + C_i \leq O_{I_j} \\ O_{I_i} > O_{I_j} &\Rightarrow O_{I_i} \geq O_{I_j} + C_j \end{aligned} \tag{11}$$

However, if $\mathcal{N}_k$ is the n^{th} node for a path of S_i but the $(n+1)^{\text{th}}$ for S_j, this means that, when their frames arrive in $\mathcal{N}_k$, S_j will be late by one $S\&F$ delay relatively to S_i. Therefore, for every flow S_j that complies to the aforementioned conditions, there needs to be a correction. If S_j is in advance with respect to S_i, we must subtract the proper number of $S\&F$ delays from O_{I_j} in the conditions in Eq. (11). If it is late, we must add the number of $S\&F$ delays to O_{I_j}.

4) Calculating section sizes and the final offset: To calculate each section size, the execution order between sections must be defined. GCD# arbitrarily chooses the order based on the number p related to the section: if $p_1 < p_2$, then Ψ_{p_1} precedes Ψ_{p_2}.

The size of any section Ψ_{p_i} can be calculated in two parts, illustrated in Eq. (12):

1) For each flow in the section, the transmission times are added to the respective internal offset. Amongst the sums for each flow, the maximum value is the first part of the section size (equivalent to Eq. (8)).
2) A margin is calculated to account for possible differences in $S\&F$ times. If we are analyzing Ψ_{p_i} which precedes Ψ_{p_j}, we must take into account the maximum possible amount of $S\&F$ delay a flow $S_i \in \Psi_{p_i}$ will have relative to any other flow $S_j \in \Psi_{p_j}$. Then, add that value to the size of the section such that, for every node, no flow of the following section will be transmitted before any flow of the preceding section.

$$\Psi.\delta = \max_{\forall i | S_i \in \Psi} (O_{I_i} + C_i) + \mathrm{margin}_{\mathrm{L}} \tag{12}$$

After this is done for every section, including the last one (which must be considered as preceding the first section due to the cyclicity), then every O_S can be calculated. For every section, O_S is the sum of the sizes of preceding sections (disregarding the cyclicity). In other words, the O_S for the first section is 0; for the second, it equals the size of the first section; for the third, it equals the sum of the sizes of the two first sections; and so on.

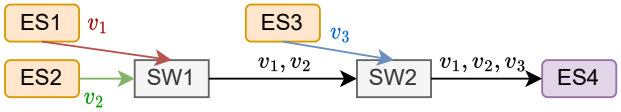

Fig. 5. Topology used in the synthetic case for evaluating GCD# collisions

TABLE II
PARAMETERS OF THE DIFFERENT FLOWS IN GCD#'S TEST CASE

Test case	Flow	Transmission Time	Period	GCD#'s offsets
Test case 1	1	2	4	0
	2	1	8	6
	3	3	8	2
Test case 2	1	1	8	0
	2	1	8	2
	3	3	6	1

Therefore, the offsets for each flow can finally be calculated according to Eq. (9).

VI. EXPERIMENTS

In this section, we will firstly work on a synthetic network in order to have a better glance into GCD# in the case of heavily loaded switches or configurations that do not respect the properties highlighted in Sec. V-B. Then, we will compare the performance of GCD# against the SMT-based formal method delineated in [4] (using the Z3 solver). We specifically selected this method as, to the best of our knowledge, there is no available heuristic method and SMT is used in several heuristics as a basis for solution exploration.

All experiments are coded in monothreaded Python 3.11 running under Fedora Linux 38 on an AMD Ryzen 9 5900HS CPU (up to 4.6 Ghz on one core) with 24 GB of RAM.

All networks used hereinafter will use the following specification: 1 Gb/s links, resulting in a unit-time of 1 ns.

A. Pushing GCD# to its limits

We introduce hereafter two test cases to demonstrate the performance of GCD# under extreme conditions. The objectives are to study the outcomes of GCD# when the load reaches 100% in a node and to understand its behavior when a transmission time exceeds the GCD of all periods.

These test cases use the topology depicted in Fig. 5, with three flows following a partly shared path. Their parameters are detailed in Tab. II. In every test case, the traffic scheduling policy of each node is non-preemptive rate-monotonic and we assume a $S\&F$ delay equal to the transmission time of the largest frame (3 time units in both test cases). Due to parameter choices, a SMT solver would not be able to produce results.

1) Highly loaded switch: Scheduling becomes challenging with high traffic loads. As GCD# cannot always produce optimal solutions, it sometimes produces a solution that is not collision-free. The simulation result for test case 1 is shown in Fig. 6. The following end-to-end delays can be infered: 10 for v_1, 7 for v_2, and 6 for v_3. The cycle that will recur within each node is highlighted with a gray background. A white part preceding a gray section corresponds to an acyclic idle time.

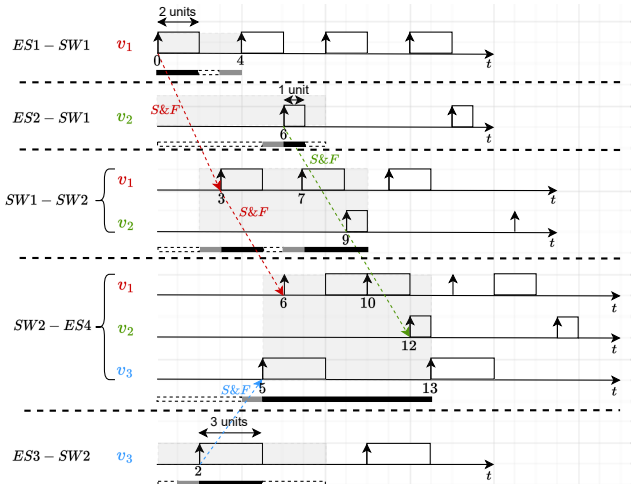

Fig. 6. Simulation of test case 1 described in Tab. II

Let n be the number of hops. The end-end Delay in a Collision-Free system (DCF) is (n-1) times the duration of $S\&F$ plus the transmission time of the analyzed frame. For our scenario, this would have been respectively 8, 7, 6 units of time for v_1, v_2 and v_3. Collision leads to a 25% increase of end-to-end latency for v_1 compared to DCF.

From this simulation, we can infer the scheduling windows as anticipated in a TSN IEEE 802.1Q network [39]. Such a mechanism, based on "doors", will allow the addition of unscheduled traffic to the existing flows, without jeopardazing their scheduling. In our diagram, a white line means that unscheduled traffic transmission can begin: the door is open. A gray line corresponds to a "guard band", during which, ongoing transmissions can continue but new ones can't start. A black line means that unscheduled transmissions are blocked: the door is closed. The size of the guard band should be at least the maximum transmission time of an unscheduled frame (1 in the example of Fig. 6).

2) Frame size superior to the GCD of the periods: In this second test case, our objective is to observe how GCD# responds to configurations where the Ω value is less than the transmission time of the largest frame. The corresponding simulation is depicted in Fig. 7. Based on the parameters provided in Tab. II, the GCD is determined to be 2. In contrast, the transmission time for the largest frame (from v_3) is 3.

For this simulation, we have chosen to exclusively display the output port of SW2. From the diagram, we can see that the largest frame (v_3) collides with both v_1 and v_2. Such collisions are inevitable since the maximum available space within each GCD circle is 2. In that case the worst case end-to-end delays for v_1 and v_2 are impacted and their respective delays are both 10. Meaning that the increase of the worst case end-to-end delays is equal to 42% compared to DCF.

B. GCD# compared to SMT

1) Synthetic configuration: The objective of this case study is to compare GCD# to SMT when applied to a compact

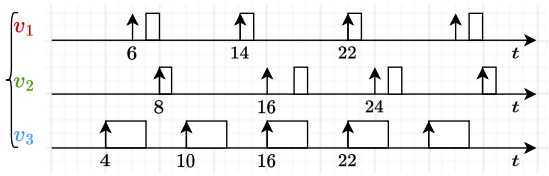

Fig. 7. Simulation of test case 2 described in Tab. II

synthetic configuration. For this study, we choose to use a configuration sourced from [40] (both topology and flows).

All the flows in this network originate from End Systems (ES)[1,2,3] transiting trough two Switches, and ultimately reach ES[4,5,6]. Every flow is dispatched from ES[1,2,3] and may arrive at one or several of the other End Systems.

The output from the SMT solver can fluctuate based on the ID attributed to the constraints (i.e., the sequence in which the constraints are added to the solver or the effect of removing and reinserting the constraints), whereas the results from GCD# remain constant. As the same configuration could yield multiple outcomes, each experimentation on SMT is repeated 10000 times with different IDs. On average, GCD# is 14.63% better than SMT for end-to-end delays, running in 53 ms on average for SMT, against less than 1 ms for GCD#. Both manage to guarantee Contention-freedom, but GCD# performs better than SMT, because SMT stops at the first feasible offset assignment, without trying to minimize end-to-end delays. Yet, GCD# runs with a handicap since it takes the maximum $S\&F$ delay into consideration for each frame.

2) Orion test case: Orion is a spacecraft developed by NASA. Its network topology is described in [41], and a flow path was proposed in [40] for end-to-end delay analysis using network calculus. We generated flows with the following random parameters: frame size (72 to 1542 bytes, in accordance with the Ethernet and IEEE802.1Q standards), period, deadline (higher than period) and path (in a set of predefined paths).

The topology of the network is depicted in Fig. 8. Periods were chosen as multiples of 7500, following a real-world random distribution, to considerably reduce SMT computation time. As a result, we obtained 625 μs as the gcd. $S\&F$ delay were capped at 12,336 ns (max Eth. frame with Interframe gap), respecting the execution time condition of GCD# (no frame shall exceed the gcd). The network is loaded with 100 flows, having one sender and one or more destinations.

Conducting 100 evaluations led to the following results:

- End-to-end delays are on average 72.82% better with GCD# compared to SMT;
- On average SMT end-to-end delays reach 61.32% of their dealdline, against 0.19% for GCD#;
- The average computing time of GCD# is about 0.02 s, whereas SMT takes 188 s to build and 47 s to solve.

In Fig. 9, we showcase key results from the comparison of SMT and GCD# for a representative test case. On the left-hand graph, for most of the destinations (rather than flows, because a flow can have multiple destinations), GCD# has better results in terms of end-to-end latencies, whereas SMT is not efficient for most of the destinations). On the right,

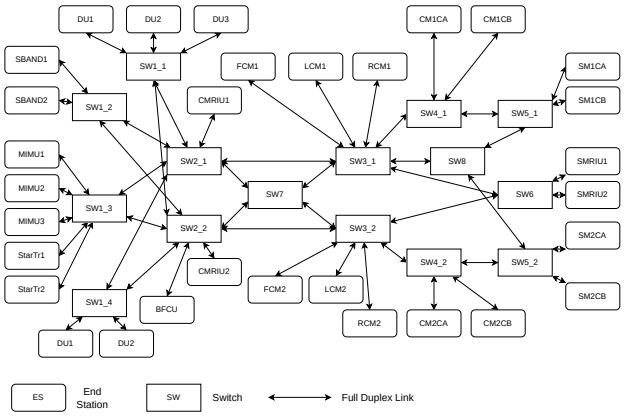

Fig. 8. Orion's topology from [40]

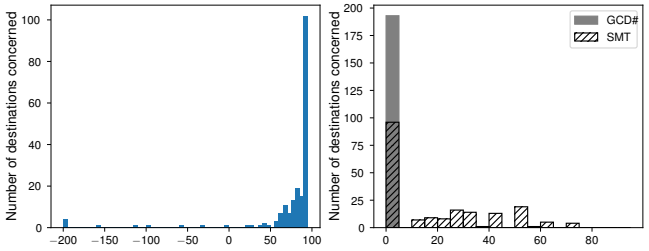

Fig. 9. Key values of the comparison of SMT and GCD# with 100 TT flows (left: Relative improvement of GCD# over SMT in terms of end-to-end latencies; right: End to end delay relative to the deadline (%))

the end-to-end delays are plotted relatively to their deadlines. We can see that GCD# delays are stacked under 5% of the deadline, while SMT delays can go up to 90% of the deadline.

The main difference between SMT and GCD# regarding end-to-end delay is due to SMT assigning arbitrary offsets to each node, while GCD# assigns aligned offsets for each following node in the network.

After executing 10,000 test runs, we found GCD# delivering non-collision-free solutions 1.34% of the time. By construction, the SMT method does not yield such invalid results.

We conducted another experiment involving 200 flows and ten executions. Increasing the number of flows in SMT poses a considerable challenge, as the frame constraint causes an exponential increase in the number of constraints to be formulated. On average, it took 680 s to generate the constraints and 348 s to solve them in the case of SMT. In contrast, GCD# only required 63 ms to identify a solution, while providing an end-to-end delay reduction of 90%.

The values from Fig. 10 are even more favorable towards GCD# than in the previous case. On the left side, only a handful of delays tend to be better with SMT than GCD#, whereas the vast majority of destination show superior performance with GCD#. Relatively to the deadlines, GCD# consistently positions all end-to-end latencies within a 0-5% range, whereas SMT results scatter broadly across the percentage spectrum.

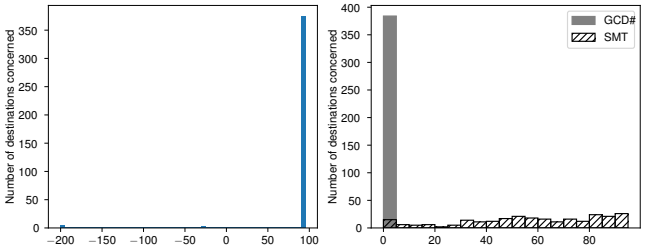

Fig. 10. Key values of the comparison of SMT and GCD# with 200 TT flows (left: Relative improvement of GCD# over SMT in terms of end-to-end latencies; right: End to end delay relative to the deadline (%))

VII. Conclusion and Perspectives

We have formally shown that the most recent approaches to address the problem of finding offsets in time-triggered networks by (1) avoiding any frame interference, and (2) meeting the Frame constraint, can limit the study to the time interval $[0, H)$. We illustrated on examples why both of the constraints taken independently are not sufficient alone to guarantee that the first hyper-period is repeating. To address this, we revisited some classic cyclicity results, which we extended to show several interesting results that can be useful for studies of tasks or flows of frames with offset. Particularly, we showed that both cyclic and acyclic parts of a schedule start and end at an idle point, implying that there is an integer number of jobs or frames in both parts of a schedule. This can be useful for optimization methods that require working on jobs or frames rather than tasks or flows. Moreover, we showed that the necessary and sufficient condition for a schedule to behave cyclically over $[0, H)$ is that there is an idle point at H. This is what allowed the main result to be shown.

Finally, we proposed a heuristic method called GCD#, which seeks to (but does not require to) completely avoid interference, and therefore is able to cope with a schedule that does not necessarily cycle over $[0, H)$. It is based on the Generalized Chinese Remainder Theorem, but decomposes the time in cycles, sections and offsets within sections in order to take into account the semi-harmonicity of the periods. Using the offsets given by the heuristic and the algorithms to find the cycle, one can create a correct schedule for a TT Network. Note that if some frames can be concurrent, each switch should have enough waiting queues to allow controlling each concurrent flow individually. To the best of our knowledge, this is the first heuristic method that does not have to respect: (1) avoid frame interference and (2) meet the Frame constraint.

In the future, we plan to include in GCD# a more realistic approach to account for the "Store & Forward" delay, with a value that depends on the size of the message and possible previous messages. Also, improvements can be made in order to account for limits in the number of queues available per switch. Heterogeneous networks, where the transmission rate is not uniform, can also be a target for future improvements. Finally, the complexity of finding the exact starting point of the cyclic part of a schedule is still an open problem.

Acknowledgment

We would like to thank the Shepherding team appointed by the conference chairs for their pertinent reviews and guidance.

References

[1] J. Goossens, R. Devillers, The non-optimality of the monotonic assignments for hard real-time offset free systems, Real-Time Systems: The International Journal of Time-Critical Computing 13 (2) (1997) 107–126. doi:10.1023/A:1007980022314.

[2] W. Steiner, An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks, in: 31st IEEE Real-Time Systems Symposium 2010, IEEE, 2010. doi:10.1109/RTSS.2010.25.

[3] J. Y.-T. Leung, J. Whitehead, On the complexity of fixed-priority scheduling of periodic, real-time tasks, Performance Evaluation 2 (4) (1982) 237–250. doi:10.1016/0166-5316(82)90024-4.

[4] S. S. Craciunas, R. S. Oliver, M. Chmelík, W. Steiner, Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks, in: Proceedings of the 24th International Conference on Real-Time Networks and Systems, 2016, pp. 183–192. doi:10.1145/2997465.2997470.

[5] R. S. Oliver, S. S. Craciunas, W. Steiner, IEEE 802.1 Qbv gate control list synthesis using array theory encoding, in: 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE, 2018, pp. 13–24. doi:10.1109/RTAS.2018.00008.

[6] M. Vlk, Z. Hanzálek, S. Tang, Constraint programming approaches to joint routing and scheduling in time-sensitive networks, Computers & Industrial Engineering 157 (2021) 107317. doi:10.1016/j.cie.2021.107317.

[7] Q. Yu, M. Gu, Adaptive group routing and scheduling in multicast time-sensitive networks, IEEE Access 8 (2020) 37855–37865. doi:10.1109/ACCESS.2020.2974580.

[8] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Mubeen, Synthesising schedules to improve QoS of best-effort traffic in TSN networks, in: 29th International Conference on Real-Time Networks and Systems, 2021, pp. 68–77. doi:10.1145/3453417.3453423.

[9] S. Beji, S. Hamadou, A. Gherbi, J. Mullins, SMT-based cost optimization approach for the integration of avionic functions in IMA and TTEthernet architectures, in: 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications, IEEE, 2014, pp. 165–174. doi:10.1109/DS-RT.2014.28.

[10] S. S. Craciunas, R. S. Oliver, Combined task-and network-level scheduling for distributed time-triggered systems, Real-Time Systems 52 (2) (2016) 161–200. doi:10.1007/s11241-015-9244-x.

[11] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, G. Mühl, ILP-based joint routing and scheduling for Time-Triggered Networks, in: Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 8–17. doi:10.1145/3139258.3139289.

[12] M. Pahlevan, N. Tabassam, R. Obermaisser, Heuristic list scheduler for time triggered traffic in Time Sensitive Networks, SIGBED Rev. 16 (1) (2019) 15–20. doi:10.1145/3314206.3314208.

[13] F. Pozo, G. Rodriguez-Navas, W. Steiner, H. Hansson, Period-aware segmented synthesis of schedules for multi-hop time-triggered networks, in: 2016 IEEE 22nd international conference on embedded and real-time computing systems and applications (RTCSA), IEEE, 2016, pp. 170–175. doi:10.1109/RTCSA.2016.42.

[14] Z. Pang, X. Huang, Z. Li, S. Zhang, Y. Xu, H. Wan, X. Zhao, Flow scheduling for conflict-free network updates in time-sensitive software-defined networks, IEEE Transactions on Industrial Informatics 17 (3) (2020) 1668–1678. doi:10.1109/TII.2020.2998224.

[15] M. Pahlevan, R. Obermaisser, Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks, in: 2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA), Vol. 1, IEEE, 2018, pp. 337–344. doi:10.1109/ETFA.2018.8502515.

[16] D. Tamas-Selicean, P. Pop, W. Steiner, Synthesis of communication schedules for TTEthernet-based mixed-criticality systems, in: Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, 2012, pp. 473–482. doi:{10.1145/2380445.2380518}.

[17] D. Tămaş-Selicean, P. Pop, W. Steiner, Design optimization of TTEthernet-based distributed real-time systems, Real-Time Systems 51 (1) (2015) 1–35. doi:10.1007/s11241-014-9214-8.

[18] D. Tămaş-Selicean, P. Pop, Optimization of ttethernet networks to support best-effort traffic, in: Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), IEEE, 2014, pp. 1–4. doi:10.1109/ETFA.2014.7005256.

[19] F. Dürr, N. G. Nayak, No-wait packet scheduling for IEEE time-sensitive networks (TSN), in: Proceedings of the 24th International Conference on Real-Time Networks and Systems, 2016, pp. 203–212. doi:10.1145/2997465.2997494.

[20] J. Dvořák, M. Heller, Z. Hanzálek, Makespan minimization of time-triggered traffic on a TTEthernet network, in: 2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS), IEEE, 2017, pp. 1–10. doi:10.1109/WFCS.2017.7991955.

[21] N. G. Nayak, F. Dürr, K. Rothermel, Incremental flow scheduling and routing in time-sensitive software-defined networks, IEEE Transactions on Industrial Informatics 14 (5) (2017) 2066–2075. doi:10.1109/TII.2017.2782235.

[22] M. Hu, J. Luo, Y. Wang, M. Lukasiewycz, Z. Zeng, Holistic scheduling of real-time applications in time-triggered in-vehicle networks, IEEE Transactions on Industrial Informatics 10 (3) (2014) 1817–1828. doi:10.1109/TII.2014.2327389.

[23] M. Vlk, K. Brejchová, Z. Hanzálek, S. Tang, Large-scale periodic scheduling in time-sensitive networks, Computers & Operations Research 137 (2022) 105512. doi:10.1016/j.cor.2021.105512.

[24] N. Reusch, L. Zhao, S. S. Craciunas, P. Pop, Window-based schedule synthesis for industrial IEEE 802.1 qbv TSN networks, in: 2020 16th IEEE International Conference on Factory Communication Systems (WFCS), IEEE, 2020, pp. 1–4. doi:10.1109/WFCS47810.2020.9114414.

[25] A. A. Atallah, G. B. Hamad, O. A. Mohamed, Fault-resilient topology planning and traffic configuration for IEEE 802.1 qbv TSN networks, in: 2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS), IEEE, 2018, pp. 151–156. doi:10.1109/IOLTS.2018.8474201.

[26] J. Y.-T. Leung, M. Merrill, A note on preemptive scheduling of periodic, real-time tasks, Information Processing Letters 11 (3) (1980) 115–118. doi:10.1016/0020-0190(80)90123-4.

[27] J. Goossens, R. Devillers, Feasibility intervals for the deadline driven scheduler with arbitrary deadlines, in: Proceedings of the 6th IEEE International Conference on Real-time Computing Systems and Applications, 1999, pp. 54–61. doi:10.1109/RTCSA.1999.811193.

[28] H.-L. Chan, M. Norrish, Proof pearl: Bounding least common multiples with triangles, in: J. C. Blanchette, S. Merz (Eds.), Interactive Theorem Proving, Springer International Publishing, Cham, 2016, pp. 140–150.

[29] A. Choquet-Geniet, E. Grolleau, Minimal schedulability interval for real-time systems of periodic tasks with offsets, Theoretical computer science 310 (1-3) (2004) 117–134. doi:10.1016/S0304-3975(03)00362-1.

[30] J. Goossens, E. Grolleau, L. Cucu-Grosjean, Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms, Real-time systems 52 (2016) 808–832. doi:10.1007/s11241-016-9256-1.

[31] D. E. Knuth, Art of computer programming, volume 2: Seminumerical algorithms, Addison-Wesley Professional, 2014.

[32] M. Ladeira, E. Grolleau, F. Bonneval, G. Hattenberger, Y. Ouhammou, Y. Hérouard, Scheduling offset-free systems under FIFO priority protocol, in: 34th Euromicro Conference on Real-Time Systems (ECRTS 2022), Vol. 231, 2022. doi:10.4230/LIPIcs.ECRTS.2022.11.

[33] C. Belwal, A. M. Cheng, Generating bounded task periods for experimental schedulability analysis, in: 2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing, IEEE, 2011, pp. 249–254. doi:10.1109/EUC.2011.39.

[34] J. Xu, A method for adjusting the periods of periodic processes to reduce the least common multiple of the period lengths in real-time embedded systems, in: Proceedings of 2010 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, IEEE, 2010, pp. 288–294. doi:10.1109/MESA.2010.5552058.

[35] V. Brocal, P. Balbastre, R. Ballester, I. Ripoll, Task period selection to minimize hyperperiod, in: ETFA2011, IEEE, 2011, pp. 1–4. doi:10.1109/ETFA.2011.6059178.

[36] I. Ripoll, R. Ballester-Ripoll, Period selection for minimal hyperperiod in periodic task systems, IEEE Transactions on Computers 62 (9) (2012) 1813–1822. doi:10.1109/TC.2012.243.

[37] M. Nasri, G. Fohler, An efficient method for assigning harmonic periods to hard real-time tasks with period ranges, in: 2015 27th Euromicro Conference on Real-Time Systems, IEEE, 2015, pp. 149–159. doi:10.1109/ECRTS.2015.21.

[38] R. Pellizzoni, G. Lipari, Feasibility analysis of real-time periodic tasks with offsets, Real-Time Systems 30 (1-2) (2005) 105–128. doi:10.1007/s11241-005-0506-x.

[39] IEEE standard for local and metropolitan area network–bridges and bridged networks, IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014) (2018) 1–1993doi:10.1109/IEEESTD.2018.8403927.

[40] L. Zhao, P. Pop, Z. Zheng, H. Daigmorte, M. Boyer, Latency analysis of multiple classes of AVB traffic in TSN with standard credit behavior using network calculus, IEEE Transactions on Industrial Electronics 68 (10) (2020) 10291–10302. doi:10.48550/arXiv.2005.08256.

[41] R. Obermaisser, Time-Triggered Communication, 1st Edition, CRC Press, Inc., USA, 2011. doi:10.1109/MIE.2011.943033.