Configuration Optimization for Heterogeneous Time-Sensitive Networks

Niklas Reusch[1], Mohammadreza Barzegaran[2], Luxi Zhao[3], Silviu S. Craciunas[4] and Paul Pop[1]

[1]Technical University of Denmark, Kongens Lyngby, Denmark.
[2]Univerity of California Irvine, Irvine, CA, USA.
[3]Beihang University, Beijing, China.
[4]TTTech Computertechnik AG, Vienna, Austria.

Contributing authors: nikre@dtu.dk; barzegm1@uci.edu; zhaoluxi@buaa.edu.cn; silviu.craciunas@tttech.com; paupo@dtu.dk;

Abstract

Time-Sensitive Networking (TSN) collectively defines a set of protocols and standard amendments that enhance IEEE 802.1Q Ethernet nodes with time-aware and fault-tolerant capabilities. Specifically, the IEEE 802.1Qbv amendment defines a timed-gate mechanism that governs the real-time transmission of critical traffic via a so-called Gate Control List (GCL) schedule encoded in each TSN-capable network device. Most TSN scheduling mechanisms are designed for homogeneous TSN networks in which all network devices must have at least the TSN capabilities related to scheduled gates and time synchronization. However, this assumption is often unrealistic since many distributed applications use heterogeneous TSN networks with legacy or off-the-shelf end systems that are unscheduled and/or unsynchronized. We propose a new scheduling paradigm for heterogeneous TSN networks that intertwines a network calculus worst-case interference analysis within the scheduling step. Through this, we compromise on the solution's optimality to be able to support heterogeneous TSN networks featuring unscheduled and/or unsynchronized end-systems while guaranteeing the real-time properties of critical communication. Within this new paradigm, we propose two solutions to solve the problem, one based on a Constraint Programming formulation and one based on a Simulated Annealing metaheuristic, that provide different trade-offs and

scalability properties. We compare and evaluate our flexible window-based scheduling methods using both synthetic and real-world test cases, validating the correctness and scalability of our implementation. Furthermore, we use OMNET++ to validate the generated GCL schedules.

Keywords: Time-Sensitive Networking (TSN), Scheduled Traffic, Metaheuristics, Constraint Programming.

1 Introduction

Real-time communication with safety guarantees over standardized protocols is becoming increasingly necessary in various application domains, such as industrial automation and advanced driver assistance in automotive applications [1]. Time-Sensitive Networking (TSN) [2] is a collection of amendments and protocols that enhances standard 802.1 Ethernet to provide real-time capabilities such as time synchronization, preemption, redundancy management and scheduled traffic [3]. Through TSN, safety-critical scheduled traffic (ST) can be isolated and guaranteed in the presence of best-effort (BE) traffic within the same multi-hop switched Ethernet network. The main TSN mechanisms that enable ST traffic with bounded latency and jitter to coexist with BE communication are a network-wide clock synchronization protocol (802.1ASrev [4]) and a Time-Aware Shaper (TAS) mechanism [5] with a global communication schedule implemented in Gate Control Lists (GCLs). The TAS mechanism operates as a gate for each transmission queue, allowing or denying the transmission of frames as configured in the GCL schedule.

Most previous work guaranteeing real-time communication behavior through TSN networks (e.g., [3, 6, 7]) assume that the network devices are homogeneous, i.e., they all have the time-aware gating mechanism and are synchronized to a global network time. However, many brownfield deployments in industrial systems require end-to-end guarantees in heterogeneous TSN networks that connect TSN-capable switches with legacy resource-constrained end-points (e.g., PLC, sensors, actuators) that are not easily retrofitted with TSN capabilities. Moreover, in industrial systems that have a long life-cycle and which are dependent on legacy technology [8], customers are more likely to accept the replacement of switches but not of customized end-points; hence it is more beneficial to transition gradually to new technologies making the integration of legacy systems into TSN networks essential [9]. Furthermore, converged IT/OT networks in, e.g., fog and edge use-cases [8], interconnection of TSN networks with, e.g., 5G domains [10], or multi-domain TSN networks with different sync mechanisms cannot readily communicate isochronous (fully periodic) traffic [11]. Here, the region outside the TSN domain can be viewed as an unscheduled and unsynchronized end-point sending sporadic critical traffic. Moreover, even if the end-points do have some form of TSN capability

(e.g., via switched end-points [12]), the software layers on top of the TSN hardware mechanism can suffer from non-deterministic jitter and delays, leading to missed transmission slots and ultimately resulting in a sporadic, rather than periodic frame transmission from the end-points.

Hence, we investigate heterogeneous TSN networks in which end-systems are unscheduled and/or unsynchronized, meaning that they do not have TSN capabilities (such as 802.1Qbv and 802.1AS) and thus lead to sporadic arrivals of critical traffic at the TSN-capable switches or domains. Classical schedule generation methods for GCLs typically enforce either a fully deterministic, 0-jitter forwarding of critical frames with exact SMT/ILP-based solvers [3] or heuristics [7], or a more flexible window-based approach that allows a bounded interference between critical frames [6]. The drawback of these methods is that they require end-systems to send critical frames in a scheduled and synchronized way that matches the forwarding schedule of switches, and thus necessitate TSN capabilities in both end systems and switches. Other work, c.f. [13, 14], introduce scheduling approaches that do not impose synchronization on the end-systems level but constrain all forwarding GCL windows on switches to be aligned and, in the case of [14], do not use safe formal verification methods like network calculus for the interference calculation used for the schedule creation.

In this paper, which is an extended version of [15], we consider heterogeneous TSN networks, relaxing the requirement that end-systems need to be synchronized and/or scheduled and, furthermore, take into account relative offsets of windows on different nodes. We intertwine the worst-case delay analysis from [16] with the scheduling step in order to generate correct schedules where the end-to-end requirements of ST streams (also called flows in previous work) are met. Furthermore, we compare different TSN scheduling approaches that have been proposed in the literature (see Table 3 for an overview) to our flexible window-based approach. We define the analysis-driven window optimization problem resulting from our more flexible approach with the goal to be able to enlarge the solution space, reduce computational complexity, and apply it to end-systems without TSN mechanisms. Depending on industrial applications' requirements, our evaluation can help system designers choose the most appropriate combination of configurations for their use-case. The main contributions of the paper are:

- We propose a novel flexible window-based scheduling method that does not individually schedule ST frames and streams, but rather schedules open gate windows for individually scheduled queues. Hence, we can support non-deterministic queue states and thus networks with unscheduled and/or unsynchronized end-systems by integrating the WCD Network Calculus (NC) analysis into the scheduling step. The NC analysis is used to construct a worst-case scenario for each stream to check its schedulability, considering arbitrary arrival times of these streams and the given open GCL window placements.

Table 1: Summary of notations

Symbol	System model
$G = (\boldsymbol{V}, \boldsymbol{E})$	Network graph with nodes ($\boldsymbol{V}$) and links ($\boldsymbol{E}$)
$[v_a, v_b] \in \boldsymbol{E}$	Link
$[v_a, v_b].C$,	Link speed
$[v_a, v_b].mt$	Link macrotick
$p \in P$	Output port
$p.Q$	Eight priority queues in an output port p
$q \in p.Q_{ST}$	A queue used for ST traffic in p
$\langle \phi, w, T \rangle_q$	GCL configuration for a queue $q \in p.Q_{ST}$, where $q.\phi$, $q.w$, and $q.T$ are the window offset, length, and period for queue q, respectively.
$f.l, f.T$	Payload size and period of a stream $f \in \mathcal{F}$
$f.P, f.D$	Priority, and deadline of a stream $f \in \mathcal{F}$
$f.r$	Route for a stream $f \in \mathcal{F}$

- We formulate the "window optimization problem" and provide timing guarantees for real-time streams even in systems with unscheduled and unsynchronized end systems.
- We propose two solutions to solve the problem, one based on a Constraint Programming formulation and one based on a Simulated Annealing metaheuristic.
- For the CP formulation, we propose a proxy function as an alternative to the network calculus analysis in [16] and use it to provide timing guarantees inside the CP search.
- We compare and evaluate our flexible window-based scheduling method with existing scheduling methods for TSN networks. The evaluation is based on both synthetic and real-world test cases, validating the correctness and the scalability of our implementation. Furthermore, we use the OMNET++ simulator to validate the generated solutions.

We start by introducing the system, network, and application models in Section 2 and outline the problem formulation, including a motivational example, in Section 3. In Section 4, we present the novel scheduling mechanism and the optimization strategy based on Constraint Programming (CP), followed by our heuristic solution in Section 5. We review related research in Section 6, focusing on the existing scheduling mechanisms that we compare our work to. In Section 7, we present the comparison and evaluation results of our methods and conclude the paper in Section 8.

2 System Model

This section defines our system model for which we summarize the notation in Table. 1.

2.1 Network Model

We represent the network as a directed graph $G = (\boldsymbol{V}, \boldsymbol{E})$ where $\boldsymbol{V} = \boldsymbol{ES} \bigcup \boldsymbol{SW}$ is the set of end systems (ES) and switches (SW) (also called

nodes), and $\boldsymbol{E}$ is the set of bi-directional full-duplex physical links. An ES can receive and send network traffic while SWs are forwarding nodes through which the traffic is routed. The edges $\boldsymbol{E}$ of the graph represent the full-duplex physical links between two nodes, $\boldsymbol{E} \subseteq \boldsymbol{V} \times \boldsymbol{V}$. If there is a physical link between two nodes $v_a, v_b \in \boldsymbol{V}$, then there exist two ordered tuples $[v_a, v_b], [v_b, v_a] \in \boldsymbol{E}$. An equivalence between output ports $p \in P$ and links $[v_a, v_b] \in \boldsymbol{E}$ can be drawn as each output port is connected to exactly one link. A link $[v_a, v_b] \in \boldsymbol{E}$ is defined by the link speed C (Mbps), propagation delay d_p (which is a function of the physical medium and the link length), and the macrotick mt. The macrotick is the length of a discrete time unit in the network, defining the granularity of the scheduling timeline [3]. Without loss of generality, we assume $d_p = 0$ in this paper.

As opposed to previous work, we do not require that end-system are either synchronized or scheduled. Since ESs can be unsynchronized and unscheduled, they transmit frames according to a strict priority (SP) mechanism. Switches still need to be synchronized and scheduled using the 802.1ASrev and 802.1Qbv, respectively. Please note that our method also works for systems where only a subset of the ESs are unsynchronized or unscheduled, and the rest have TSN capabilities. However, in such mixed systems, the response times of critical traffic coming from synchronized/scheduled ESs may be unnecessarily pessimistic, and the pessimism may lead to reduced schedulability in some use cases. If schedulability needs to be increased and the pessimism decreased, we need to isolate critical traffic that arrives at a switch from TSN-capable ESs from any other traffic. This can be achieved by placing the scheduled critical traffic from scheduled and synchronized ESs into different queues than the sporadic traffic arriving from unscheduled/unsynchronized ESS in order to eliminate interference. While this will guarantee a deterministic timely behavior as in the classical models (e.g. [3, 6]), it will mean that the reserved queues will not be available for the sporadic critical traffic arriving from unscheduled/unsynchronized ESs.

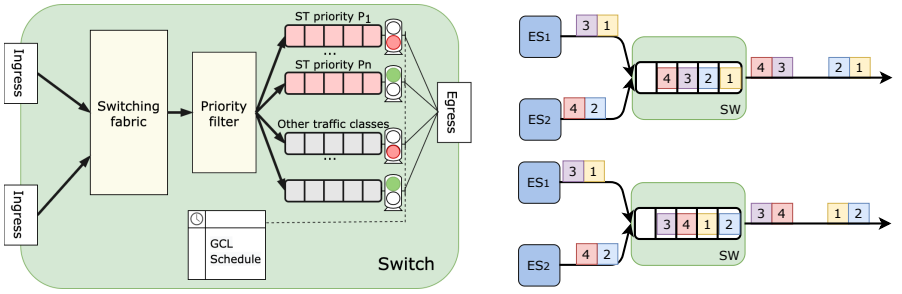
2.2 Switch Model

Figure 1a depicts the internals of a TSN switch. The switching fabric decides, based on the internal routing table to which output port p a received frame will be forwarded. Each egress port has a priority filter that determines in which of the available 8 queues/traffic-classes $q \in p.Q$ of that port a frame will be put. We assume that all 8 may (but don't have to) be used. Within a queue, frames are transmitted in first-in-first-out (FIFO) order. Similar to [3], a subset ($p.Q_{ST}$) of the queues is reserved for ST traffic, while the rest ($p.\overline{Q}$) are used for non-critical communication. As opposed to regular 802.1Q bridges, where enqueued frames are sent out according to their respective priority, in 802.1Qbv bridges, there is a Time-Aware Shaper (TAS), also called timed-gate, associated with each queue and positioned behind it. A timed-gate can be either in an *open* (O) or *closed* (C) state. When the gate is open, traffic from the respective queue is allowed to be transmitted, while a closed gate will

not allow transmission, even if the queue is not empty. When multiple gates are open simultaneously, the highest non-empty priority queue gets selected first, blocking others until it is empty or the corresponding gate is closed. The 802.1Qbv standard includes a mechanism to ensure that no frames can be transmitted beyond the respective gate's closing point. This look-ahead checks whether the entire frame present in the queue can be fully transmitted before the gate closes and, if not, it will not start the transmission.

The state of the queues is encoded in a GCL, which (contrary to e.g., TTEthernet [18]) acts on the level of traffic-classes instead of on an individual frame level [19]. Hence, an imperfect time synchronization, frame loss, ingress policing (c.f. [3]), or the variance in the arrival of frames from unscheduled and/or unsynchronized ESs may lead to non-determinism in the state of the egress queues and, as a consequence, in the whole network. If the state of the queue is not deterministic at runtime, the order and timing of the sending of ST frames can vary dynamically. In Figure 1b, the schedule for the queue of the (simplified) switch SW, opens for two frames and then, sometime later, for the duration of another two frames. The arrival of frames from unscheduled and/or unsynchronized end systems may lead to a different pattern in the egress queue of the switch, as illustrated in the top and bottom figures of Figure 1b. Note that we do not actually know the arrival times of the frames, and what we depict in the figure are just two scenarios to illustrate the non-determinism. There may be scenarios where one of the frames, e.g., frame "2", arrives much later. This variance makes it impossible to isolate frames in windows and obtain deterministic queue states, and, as a consequence, deterministic egress transmission patterns, as required by previous methods for TSN scheduling (e.g. [3, 6, 7, 20]). We refer the reader to [3] for an in-depth explanation of the TSN non-determinism problem.

The queue configuration is expressed by $q = \langle Q_{ST}, \overline{Q} \rangle$. The decision in which queue to place frames is taken either according to the priority code point (PCP) of the VLAN tag or according to the priority assignment of the IEEE 802.1Qci mechanism. In order to formulate the scheduling problem, the



(a) TSN Switch Internals. (b) Queue states (inspired by [17]).

Fig. 1: TSN Switch model and queue interference.

GCL configuration is defined as a tuple $\langle \phi, w, T \rangle_q$ for each queue $q \in p.Q_{ST}$ in an output port p, with the window offset ϕ, window length w and window period T.

2.3 Application Model

The traffic class we focus on in this paper is scheduled traffic (ST), also called time-sensitive traffic. ST traffic is defined as having requirements on the bounded end-to-end latency and/or minimal jitter [3]. Communication requirements of ST traffic itself are modeled with the concept of streams (also called flows), representing a communication from one sender (talker) to one or multiple receivers (listeners). We define the set of ST streams in the network as $\mathcal{F}$. A stream $f \in \mathcal{F}$ is expressed as the tuple $\langle l, T, P, D \rangle_f$, including the frame size, the stream period in the source ES, the priority of the stream, and the required deadline representing the upper bound on the end-to-end delay of the stream.

The route for each stream is statically defined as an ordered sequence of directed links,e.g., a stream $f \in F$ sending from a source ES v_1 to another destination ES v_n has the route $r = [[v_1, v_2], ..., [v_{n-1}, v_n]]$. Without loss of generality, the notation is simplified by limiting the number of destination ES to one, i.e., unicast communication. Please note that the model can be easily extended to multicast communication by adding each sender-receiver pair as a stand-alone stream with additional constraints between them on the common path.

We assume that streams arrive sporadically, meaning at a random time, but with a minimum interarrival time of the stream's period.

3 Problem Formulation

Given (1) a set of streams $\mathcal{F}$ with statically defined routes $\mathcal{R}$, and (2) a network graph G, we are interested in determining GCLs, which is equivalent to determining (i) the offset of windows $q.\phi$, (ii) the length of windows $q.w$, and (iii) the period of windows $q.T$ such that the deadlines of all streams are satisfied and the overall bandwidth utilization (c.f. Section 4.2) is minimized.

We remind the reader that with flexible window-based scheduling, we do not know the arrival times of frames, and frames of different ST streams may interfere with each other. Frames that arrive earlier will delay frames that arrive later; also, a frame may need to wait until a gate is open, or arrive at a time just before a gate closure and cannot fit in the interval that remains for transmission.

Once the problem is unschedulable (some stream deadlines are missed), we determine the solution in which the number of missed stream deadlines is minimized. In this paper, we use Network Calculus [21] to calculate the worst-case delay of streams.

Our problem is intractable, i.e., the decision problem associated with the scheduling problem has been proved to be (NP)-complete in the strong

sense [22]. We present two solutions for this problem. Our first solution is based on Constraint Programming (CP), see Section 4. CP is an exact mathematical programming approach that attempts to find an optimal solution. However, as our experiments in Section 7 will show, CP cannot handle realistic test cases. Hence, we also propose a second solution, based on a Simulated Annealing (SA) metaheuristic, see Section 5. Metaheuristics have been used as an alternative to exact optimization methods such as CP [23].

3.1 Motivational Example

Let us illustrate the importance of determining optimized windows. Recall that with flexible window-based scheduling, we do not know the arrival times of frames, and frames of different ST streams may interfere with each other. Frames that arrive earlier will delay frames that arrive later; also, a frame may need to wait until a gate is open, or arrive at a time just before a gate closure and cannot fit in the interval that remains for transmission.

We illustrate in Figure 2 three window configurations (a), (b), and (c), motivating the need to optimize the windows. The vertical axis represents each egress port in the network, and the horizontal axis represents the timeline. The tall grey rectangles give the gate open time for a priority queue. As mentioned, we do not know the arrival times of the frames, thus it is necessary to provide a formal analysis method to ensure real-time performance. In this paper, we use the Network Calculus (NC)-based approach from [16] to determine the worst-case end-to-end delay bounds (WCDs) for each stream. In the motivational example, the WCDs are determined by constructing a worst-case scenario for

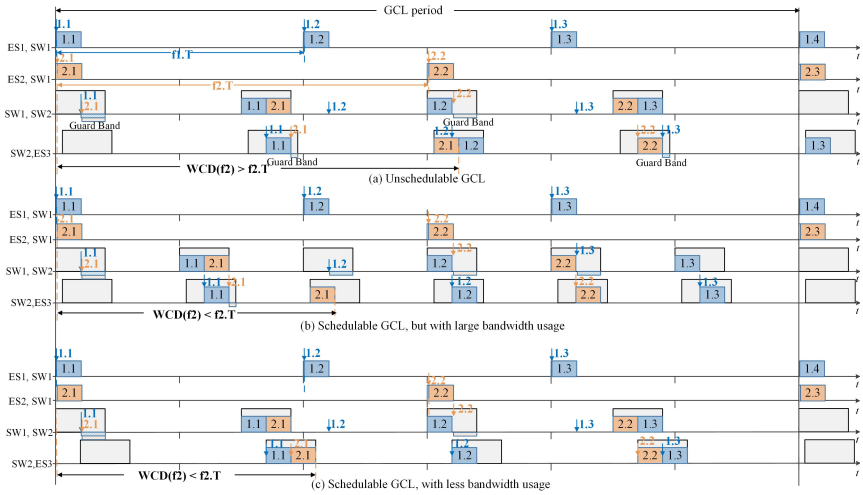


Fig. 2: Motivational example showing the importance of optimizing the windows. Note that the arrival times of frames are not known beforehand, hence we decided to illustrate in each configuration (a) to (c) an arrival scenario that leads to the worst-case delay for frame f_2.

each stream. Hence, in Figure 2 we show worst-case scenarios. The red and blue rectangles in the figure represent ST frames' transmission. There are two periodic streams f_1 (blue rectangles) and f_2 (red rectangles) with the same frame size and priority. We use the arrows pointing down to mark the arrival time for ST frames creating the worst-case case for f_2. Let us assume that the deadline of each stream equals its period $f_i.T$. In each configuration in Figure 2 we show that arrival scenario which would lead to the worst-case situation for frame f_2, i.e., the largest WCD for f_2.

Since the ESs are unscheduled (without TAS) and/or unsynchronized, the frames can arrive and be transmitted by the ES at any time (the offset of a periodic stream on the ES is in an arbitrary relationship with the offsets of the windows in the SWs). However, on the switches, frame transmissions are allowed to be forwarded only during the scheduled window. The worst-case for f_2 happens when the frame (1.1) of f_1 arrives on $[ES_1, SW_1]$ slightly earlier than the frame (2.1) of f_2 arrives on $[ES_2, SW_1]$, and at the same time, they arrive on the subsequent egress port $[SW_1, SW_2]$ at a time when the remaining time during the current window is smaller than the frame transmission time. In this case, the guard band delays the frames until the next window slot.

With the window configuration in Figure 2(a), the WCD of f_2 is larger than its deadline, i.e., $WCD(f_2) > f_2.T$, hence, f_2 is not schedulable. If the window period is narrowed down, as shown in Figure 2(b), the WCD of f_2 satisfies its deadline. However, there is a large bandwidth usage occupied by the windows. Figure 2(c) uses the same window period as in Figure 2(a) but changes the window offset. As can be seen in the figure, the WCD of the stream f_2 is also smaller than its deadline $f_2.T$, and compared with Figure 2(b), the bandwidth usage of the windows is reduced. With the increasing complexity of the network, e.g., multiple streams joining and leaving at any switch in the network and/or an increased number of ST streams and priorities, an optimized window configuration cannot be done manually; therefore, optimization algorithms are needed to solve this problem.

4 Constraint Programming Window Optimization (CPWO)

In this section we present a solution based on a Constraint Programming formulation. Although CP can perform an exhaustive search and find the optimal solution, this is infeasible for large networks. Hence, we propose a strategy called Constraint Programming-based Window Optimization (CPWO) that is able to "prune" the search to find optimized solutions in a reasonable time, at the expense of optimality. CPWO has two features intended to speed up the search:

(i) A metaheuristic search traversal strategy: CP solvers can be configured with user-defined search strategies, which enforce a custom order for selecting variables for assignment and for selecting the values from the variable's domain. Here, we use a *metaheuristic* strategy based on Tabu Search [23].

(ii) A timing constraint specified in the CP model that prunes the search space: Ideally, the WCD Analysis would be called for each new solution. However, an NC-based analysis is time-consuming, and it would slow down the search considerably if called each time the CP solver visits a new valid solution. Hence, we have introduced "search pruning" constraints in the CP model (the "Timing (pruning)" constraints in the "CP model" box in Figure 3), explained in Section 4.5.

4.1 Overview

CPWO takes as the inputs the architecture and application models and outputs a set of the best solutions found during search (see Figure 3). We use CP to search for solutions (the "CP solver" box). CP performs a systematic search to assign the values of variables to satisfy a set of constraints and optimize an objective function, see the "CP model" box: the sets of variables are defined in Section 4.3, the constraints in Section 4.4 and the objective function in Section 4.2. A feasible solution is a valid solution that is schedulable, i.e., the worst-case delays (WCDs) of streams are within their deadlines. Since it is impractical to check for schedulability within a CP formulation, we employ instead the Network Calculus (NC)-based approach from [16] to determine the WCDs, see the "WCD Analysis" box in Figure 3. The WCD Analysis is called every time the CP solver finds a "new solution" which is valid with respect to the CP constraints. The "new solution" is not schedulable if the calculated latency upper bounds are larger than the deadlines of some critical streams.

These timing constraints implement a crude analysis that indicates if a solution may be schedulable and are solely used by the CP solver to eliminate solutions from the search space. These constraints may lead to both "relaxed-pruning" scenarios that are actually unschedulable or "aggressive-pruning" scenarios that eliminate solutions that are schedulable. The proxy function (pruning constraint) can thus be parameterized to trade-off runtime performance for search-space pruning in the CP-model.

The timing constraints assume that for a given stream, its frames in a queue will be delayed by other frames in the same queue, including a backlog

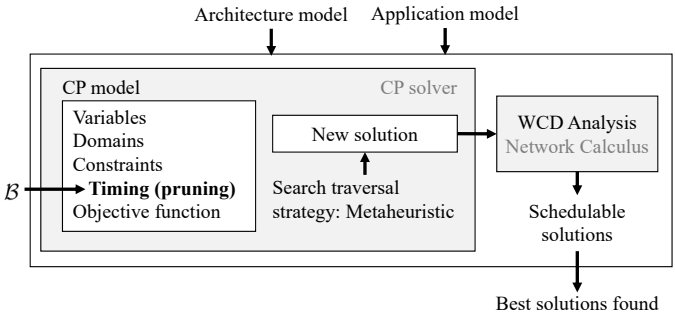


Fig. 3: Overview of our CPWO optimization strategy

Table 2: Definition of terms used in CP model formulation

Term	Definition
$\mathcal{N}(P)$	Total number of windows assigned to priority queues
$\mathcal{K}(p)$	Hyperperiod of the port p
$\mathcal{L}(q)$	Maximum size of any frame from all streams assigned to q
$\mathcal{GB}(q)$	Maximum transmission time of ST frames competing in q
$\mathcal{R}(q)$	All streams assigned to the queue q
$\mathcal{X}(q)$	All streams arriving from a switch and assigned to the queue q

of frames of the same stream. A parameter $\mathcal{B}$ is used to adjust the number of frames in the backlog which is an integer number greater than zero, as preemption is not allowed, tuning the pruning level of the CP model's timing constraints. Note that NC still checks the actual schedulability, so it does not matter if the CP analysis is too relaxed—this will only prune fewer solutions, slowing down the search. However, using overly aggressive pruning runs the risk of eliminating schedulable solutions of good quality. We consider that $\mathcal{B}$ is given by the user, controlling how fast to explore the search space. In the experiments, we adjusted $\mathcal{B}$ based on the feedback from the WCD Analysis and the pruning constraint. If, during a CPWO run, the pruning constraint from Section 4.5 was invoked too often, we decreased $\mathcal{B}$, as it was pruning too aggressively; otherwise, if the WCD analysis was invoked too often and was reporting that the solutions were schedulable, we increased $\mathcal{B}$.

We first define the terms needed for the CP model in Table. 2. Then, we continue with the definition of the objective function, model variables, and constraints of the CP model.

4.2 CP Objective Function

The CP solver uses the objective function Ω, which minimizes the average bandwidth usage:

$$\forall p \in P : \Omega = \frac{\sum_{q \in p.Q} \frac{q.w}{q.T}}{\mathcal{N}(P)}. \tag{1}$$

Eq. 1 defines the objective function Ω as the average bandwidth usage in the network. The average bandwidth usage is calculated as the sum of each window's utilization, i.e., the window length over its period, divided by the total number of windows in the CP model. For each port in each device of the network ($\forall p \in P$), the sum is computed over all windows of the port schedule ($\forall q \in p.Q$). This objective function might be chosen differently depending on the use-case. Other possible formulations could involve minimizing the maximum load on any link or the average end-to-end delay of streams. Note that solutions found by a CP solver are guaranteed to satisfy the constraints defined in Section 4.4. In addition, the schedulability is checked with the NC-based WCD Analysis [16].

4.3 Variables

The model variables are the offset, length, and period of each window, see Section 2.2. For each variable, we define a domain which is a set of finite values that can be assigned to the variable. CP decides the values of the variables as an integer from their domain in each visited solution during the search. The domains of offset $q.\phi$, length $q.w$, and period $q.T$ variables are defined, respectively, by

$$\begin{aligned} &\forall p \in P, \forall q \in p.Q: \\ &0 < q.T \leq \frac{\mathcal{K}(p)}{[v_a, v_b].mt}, \quad 0 \leq q.\phi \leq \frac{\mathcal{K}(p)}{[v_a, v_b].mt}, \\ &\frac{\mathcal{L}(q)}{[v_a, v_b].mt \times [v_a, v_b].C} + \mathcal{GB}(q) \leq q.w \leq \frac{\mathcal{K}(p)}{[v_a, v_b].mt}. \end{aligned} \tag{2}$$

The domain of the window period is defined in the range from 0 to the hyperperiod of the respective port p, i.e., the Least Common Multiple (LCM) of all the stream periods forwarded via the port. The window period is an integer and cannot be zero. The domain of the window offset is defined in the range from 0 to the hyperperiod of the respective port p. Finally, the domain of the window length is defined in the range from the minimum accepted window length to the hyperperiod of the respective port p. The minimum accepted window length is the length required to transfer the largest frame from all streams assigned to the queue q, protected by the guard band $\mathcal{GB}(q)$ of the queue. A port p is attached to only one link $[v_a, v_b]$; and values and domains are scaled by the macrotick mt of the respective link.

4.4 Constraints

The first three constraints need to be satisfied by a valid solution: (1) the window is valid, (2) two windows in the same port do not overlap, and (3) the window bandwidth is not exceeded. The next two constraints reduce the search space by restricting the periods of (4) queues and (5) windows to harmonic values in relation to the hyperperiod. Harmonicity may eliminate some feasible solutions but we use this heuristic strategy to speed up the search. Finally, we introduce a constraint that limits the size of the resulting GCL configuration to be smaller than or equal to a given maximum (Eq. (8)). This is necessary since some resource-constrained TSN devices may only allow a limited GCL size per port.

(1) The **Window Validity Constraint** (Eq. (3)) states that the offset plus the length of a window should be smaller or equal to the window's period:

$$\forall p \in P, \forall q \in p.Q: \quad (q.w + q.\phi) \leq q.T. \tag{3}$$

(2) **Non-overlapping Constraint** (Eq. (4)). Since we search for solutions in which windows of the same port do not overlap, the opening or closing of each

window on the same port (defined by its offset and the sum of its offset and length, respectively) is not in the range of another window, over all period instances:

$$\begin{aligned}&\forall p \in P, \forall q \in p.Q, \forall q' \in p.Q, T_{q,q'} = max(q.T, q'T),\\&\forall a \in [0, T_{q,q'}/q.T), \forall b \in [0, T_{q,q'}/q'.T) :\\&(q.\phi + q.w + a \times q.T) \leq (q'.\phi + b \times q'.T) \vee\\&(q'.\phi + q'.w + b \times q'.T) \leq (q.\phi + a \times q.T)\end{aligned} \tag{4}$$

(3) The **Bandwidth Constraint** (Eq. (5)) ensures that all the windows have enough bandwidth for the assigned streams:

$$\forall p \in P, \forall q \in p.Q : \frac{q.w}{q.T} \geq \sum_{f \in \mathcal{F}(q)} \frac{f.l}{f.T}. \tag{5}$$

where $\mathcal{F}(q)$ is the set of streams assigned to the queue q.
(4) The **Port Period Constraint** (Eq. (6)) imposes that the periods of all the queues in a port should be harmonic. This constraint is used to avoid window overlapping and to reduce the search space. We note that this constraint has no influence on the period design of the streams in the system. The *period of the given streams* and the *period of the windows* that our solutions determine are two different entities that affect each other but are not restricted by one another. A stream may have any period, and we only constrain the period the resulting *GCL windows* in the schedule. It is necessary to constrain the GCL window period since it is an optimization variable, and if left unconstrained, there would be infinite possible solutions.

$$\forall p \in P, \forall q \in p.Q, \forall q' \in p.Q : (q.T \% q'.T = 0) \vee (q'.T \% q.T = 0). \tag{6}$$

(5) The **Period Limit Constraint** (Eq. (7)) reduces the search space by considering window periods $q.T$ that are harmonic with the hyperperiod of the port $\mathcal{K}(p)$ (divide it):

$$\forall p \in P, \forall q \in p.Q : \quad \mathcal{K}(p) \% q.T = 0. \tag{7}$$

(6) The **GCL size Constraint** (Eq. (8)) mandates that the number of open/close entries in the resulting GCL is below a given maximum defined by GCL_{max}. Since in the previous constraint we imposed that any window period $q.T$ is harmonic with the hyperperiod of the port $\mathcal{K}(p)$, we need to make sure that the resulting number of open/close windows until the hyperperiod of the port is smaller than or equal to the given maximum. Since we do not know if windows on different queues are back-to-back, the state transition from the closing event of one window may not coincide with the opening event of another window. Hence, each window can potentially introduce two entries in the GCL

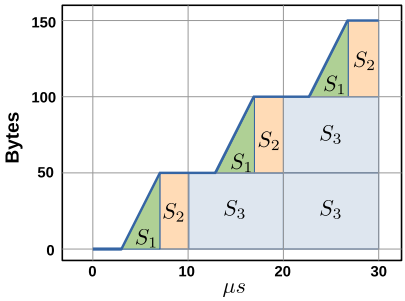


Fig. 4: Example capacity for a window.

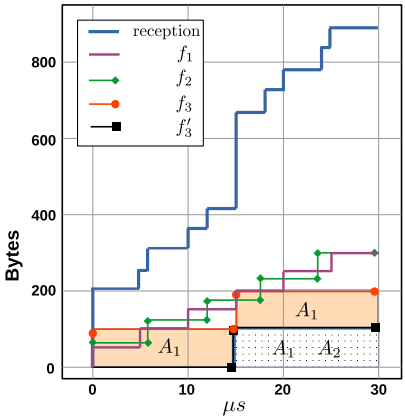


Fig. 5: Example capacity and transmission demand for a window.

table, one for opening the window and another for closing it. Therefore the maximum number of GCL entries can be bounded as follows:

$$\forall p \in P: \quad \sum_{q \in p.Q} 2 \times \left\lceil \frac{\mathcal{K}(p)}{q.T} \right\rceil \leq GCL_{max}. \tag{8}$$

4.5 Timing Constraints

As mentioned, it is infeasible to use a Network Calculus-based worst-case delay analysis to check the schedulability of *each* solution visited. Thus, we have defined a *Timing Constraint* as a way to prune the search space. Every solution that is not eliminated via this timing constraint is evaluated for schedulability with the NC WCD analysis. The timing constraint is a heuristic that prunes the search space of (potentially unschedulable) solutions; it is not a sufficient nor a necessary schedulability test. The timing constraint is related to the optimality of the solution, not to its correctness in terms of schedulability. An overly aggressive pruning may eliminate good quality solutions, and too little

pruning will slow down the search because the NC WCD analysis is invoked too often.

The challenge is that the min+ algebra used by NC cannot be directly expressed in the first-order formulation of CP. However, the NC formulation from [16] has inspired us to define the CP timing constraints. The *Timing Constraint* is defined in Eq. (9) and uses the concepts of *window capacity* $\mathcal{W}_C$ and *transmission demand* $\mathcal{W}_D$ to direct the CP solver to visit only those solutions where the *capacity* of each window, i.e., the amount of time available to transmit frames assigned to its queue, is greater than or equal to its transmission *demand*, i.e., the amount of transmission time required by the frames in the queue. A window capacity larger than the transmission demand indicates that a solution has a high chance to be schedulable:

$$\forall p \in P, \forall q \in p.Q : \mathcal{W}_D \leq \mathcal{W}_C. \tag{9}$$

Thus, we first calculate the capacity $\mathcal{W}_C$ of each window within the hyperperiod. This capacity is similar to the NC concept of a service curve, and its calculation is similar to the service curves proposed in the literature [24] for resources that use Time-Division Multiple Access (TDMA), which is how our windows behave. For e.g., a window with a period of 10 μs, a length of 4 μs, and an offset of 3 μs; forwards 150 bytes over a 100 Mbps link in a hyperperiod of 30 μs. In Figure 4, the capacity of such a window is depicted where the blue line shows the throughput of the window for transferring data. The capacity increases when the window opens (the rising slopes of the curve). The effect of window offset on the capacity (the area under the curve) can be observed in the figure. The function $\mathcal{W}_C$ calculates the area under the curve to characterize the amount of capacity for a window in a hyperperiod, defined in Eq. (10), where the link $[v_a, v_b]$ is attached to the port p and assigned to the queue q; and function $\mathcal{Y}$ captures the transmission time of a single byte through link $[v_a, v_b]$.

To calculate the area under the curve, we consider 3 terms that are S_1, S_2, and S_3. They represent the total area under the curve caused by the window length, the window closure in the remainder of the window period, and the window period, respectively. The value for S_1 equals to the area of three triangles, each with the base of 4 μs and the height of 50 $Bytes$. Similarly, the value for S_2 equals to the three rectangles, each with a length of 3 μs and a width of 50 $Bytes$. Lastly, the value for S_3 equals to the area of three triangles with a base of 10 μs and a height of 50 $Bytes$. The $\mathcal{W}_C$ value of the example

in Figure 4 is 2,250 $Bytes \times \mu s$, where the S terms are shown.

$$\begin{aligned}
&\forall p \in P, \forall q \in p.Q: \\
&I = \frac{\mathcal{K}(p)}{q.T}, \quad J = \frac{(q.w - \mathcal{GB}(q)) \times \mathcal{Y}([v_a, v_b])}{[v_a, v_b].C}, \\
&S_1 = I \times \frac{q.w \times J}{2}, \quad S_2 = I \times (q.T - q.w. - q.\phi) \times J, \\
&S_3 = \frac{I \times (I-1)}{2} \times q.T \times J, \quad \mathcal{W}_C = S_1 + S_2 + S_3
\end{aligned} \tag{10}$$

Secondly, we calculate the transmission demand $\mathcal{W}_D$ using Eq. (11), where $\mathcal{R}(q)$ captures all the streams that are assigned to the queue q. The transmission demand is inspired by the *arrival curves* of NC. These are carefully determined in NC considering that the streams pass via switches and may change their arrival patterns [16]. In our case, we have made the following simplifying assumptions to be able to express the "transmission demand" in CP. We assume that all streams are strictly periodic and arrive at the beginning of their respective periods. This is "optimistic" with respect to NC in the sense that NC may determine that some of the streams have a bursty behavior when they reach our window. To compensate for this, we consider that those streams that arrive from a switch may be bursty and thus have a backlog $\mathcal{B}$ of frames that have accumulated; streams that arrive from ESs do not accumulate a backlog. It is also noteworthy that the value of $\mathcal{B}$ being an integer or decimal depends on whether preemption (IEEE 802.1Qbu) is supported/enabled for the critical flows in the network, i.e., if critical flows are marked as express, and the devices support preemption. When frames can be preempted, the value of $\mathcal{B}$ can be chosen from $\mathbb{R}^+$, which implies that also fragments of frames are allowed to be backlogged. If preemption is not supported or enabled for the critical frames, the value of $\mathcal{B}$ can only take integer values greater than or equal to 1, specifying how many full frames can be backlogged. In this work, we consider that critical flows are not preemptable; hence, we consider only integers greater than zero for $\mathcal{B}$. Figure 5 shows three streams, f_1 to f_3, and only f_3 arrives from a switch and hence will have a backlog of frames captured by the stream denoted with f_3' (we consider a $\mathcal{B}$ of 1 in the example). We also assume that the backlog f_3' will not arrive at the same time as the original stream f_3, and instead, it is delayed by a period. Again, this is a heuristic used for pruning, and the actual schedulability check is done with the NC analysis. So, the definition of the "transmission demand" does not impact correctness, but, as discussed, it will impact our algorithm's ability to search for solutions.

Since, in our case, the deadlines can be larger than the periods, we also need to consider, for each stream, bursts of frames coming from SWs and an additional frame for each stream coming from ESs (the ES periods are not synchronized with the SWs GCLs). Since we do not perform a worst-case analysis, we instead use a backlog parameter $\mathcal{B}$, capturing the possible number of delayed frames in a burst within a stream forwarded from another SW. Note

that as explained in the overview at the beginning of Section 4, $\mathcal{B}$ is a user-defined parameter that controls the "pruning level" of our timing constraint, i.e., how aggressively it eliminates candidates from the search space depending on the preemption.

We give an example in Figure 5 where the streams $f_1 < 50, 5, 0, 5 >$ and $f_2 < 60, 6, 0, 6 >$ have been received from an ES and the stream $f_3 < 100, 15, 0, 15 >$ has been received from a SW. For the stream f_3 forwarded from a previous switch, we consider that one instance of the stream (determined by the backlog parameter $\mathcal{B} = 1$), let us call it f_3', may have been delayed and received together with the current instance f_3. This would cause a delay in the reception of the streams in the current node. The reception curve in Figure 5 is the sum of curves for each stream separately in a hyperperiod of 30 μs.

We give the general definition of the transmission demand value $\mathcal{W}_D$ as the area under the curve for the accumulated data amount of received streams and backlogs of the streams arrived from switches in a hyperperiod. For calculating the transmission demand $\mathcal{W}_D$, we consider 2 terms that are A_1 and A_2 for each stream. The term A_1 calculates the area under the curve for the accumulated data of all streams assigned to the queue q captured by $\mathcal{R}(q)$, in a hyperperiod. Any frames of all streams $\mathcal{R}(q)$ have arrived at the beginning of their period. Considering the hyperperiod of the received streams which is denoted by $\mathcal{K}(p)$, each stream will have I instances in this queue. For each instance i, the area under the curve consists of i rectangles with the length of the stream period and the width of the stream size. Thus for all I instances of the stream the area under the curve is equal to $\frac{I\times(I+1)}{2}$ rectangles.

The term A_2 calculates the area under the curve for the accumulated backlog data of the streams arrived from a switch captured by $\mathcal{X}(q)$. The backlog data of the streams $\mathcal{X}(q)$ are delayed for a period and controlled by $\mathcal{B}$, which captures the number of backlogs. Similarly, the area under the curve for the stream backlogs consists of the number of rectangles with the length of the stream period and the width of the stream size. The only difference is that the stream has $\mathcal{B}$ instances less. Thus, the number of rectangles equals to $\frac{I\times(I+1-2\times\mathcal{B})}{2}$. The function $\mathcal{W}_D$ returns 16,650 $Bytes \times \mu s$ in our example, see also Figure 5 for the values of the terms A_1 and A_2.

$$
\begin{aligned}
&\forall p \in P, \forall q \in p.Q, \forall f \in \mathcal{R}(q), \forall f' \in \mathcal{X}(q) : \\
&I = \frac{\mathcal{K}(p)}{f.T}, \quad I' = \frac{\mathcal{K}(p)}{f'.T} \\
&A_1 = \frac{I \times (I+1)}{2} \times f.T \times f.l, \\
&A_2 = \frac{I' \times (I'+1-2\times\mathcal{B})}{2} \times f'.T \times f'.l, \\
&\mathcal{W}_D = A_1 + A_2
\end{aligned}
\tag{11}
$$

Please note that the correctness of the constraints (Eq. (3), (4), (5)) follows from the implicit hardware constraints of 802.1Q(bv) (see the discussion in [3,

6]) while other constraints (Eq. (6), (7)) are used to limit the placement of GCL windows and are not related to correctness, just to optimality. Since the transmission of frames is decoupled from the GCL windows, the schedule's correctness concerning the end-to-end latency of streams is always guaranteed due to the NC analysis, which is intertwined in the schedule step.

5 Simulated Annealing Window Optimization (SAWO)

The previously described CPWO delivers good results in a reasonable time for small problem sizes. However, for larger problem sizes, the method either becomes intractable, or the search space pruning has to be done very aggressively, leading to a degradation in the quality of the results. For such intractable problems, optimal algorithms such as Branch & Bound and Integer Linear Programming require an exponentially increasing time with increasing input size. Therefore we propose a metaheuristic algorithm that is aimed to be scalable for large problem sizes while still offering good quality solutions. Metaheuristics are designed to find good quality solutions while still being scalable for large problem sizes but are not guaranteed to find an optimal (or any) solution [23]. A plethora of metaheuristic approaches have been proposed in the literature for intractable problems [23, 25]. In this paper, we have developed a Simulated Annealing (SA)-based metaheuristic solution.

Our Simulated Annealing Window Optimization (SAWO) algorithm is shown in Alg. 1. The key feature of SA is that it avoids getting stuck in a local optimum by accepting worse intermediate solutions with a certain probability, which decreases throughout the search [23, 26]. The likelihood of considering a worse solution (compared to the current solution) depends on the worsening of the objective function and a temperature parameter t [27]. In SA, the temperature starts from an initial temperature T_{start} (line 3) and is decreased in every iteration with a factor $0 < \alpha < 1$ (line 16).

SA starts from an initial solution Φ (line 1, see Section 5.1) which is evaluated using the objective function Ω^{SA} (line 2), see Section 5.2 for details. When there are no infeasible streams the objective functions for CPWO and SAWO are identical and thus comparable. If there are infeasible streams, we penalize this in the SAWO solution in order to guide the search (see Section 5.2).

SA iterates until a stopping criterion, like a timeout or iteration limit, is satisfied (lines 4–17). In every iteration, we generate a random "neighbor" of the current solution (line 5) and calculate the difference δ between its objective value Ω^{SA}_{new} and the objective value Ω^{SA} of the current solution (line 7). Section 5.3 presents how we generate a neighbor solution. If the new objective value is smaller, we accept the new solution as the current (and possibly best, see lines 11–14). However, we also sometimes accept a worse neighbor solution. This is the case if a random value (between 0 and 1) is smaller than an "acceptance probability function" $e^{\frac{\delta}{t}}$ (see line 8). This acceptance probability

Algorithm 1 Simulated Annealing Window Optimization (SAWO)

Require: Streams $\mathcal{F}$, Switch Ports P
1: $\Phi_{best} = \Phi = \text{InitialSolution}(\mathcal{F}, P)$
2: $\Omega^{SA}_{best} = \Omega^{SA} = \text{Objective}(\Phi)$
3: $t = T_{start}$
4: **while** stopping-criterion not True **do**
5: $\quad \Phi_{new} = \text{RandomNeighbor}(\Phi, p_{mv})$
6: $\quad \Omega^{SA}_{new} = \text{Objective}(\Phi_{new})$
7: $\quad \delta = \Omega^{SA}_{new} - \Omega^{SA}$
8: $\quad$ **if** $\delta < 0$ **or** random[0,1) $< e^{\frac{\delta}{t}}$ **then**
9: $\quad\quad \Phi = \Phi_{new}$
10: $\quad\quad \Omega^{SA} = \Omega^{SA}_{new}$
11: $\quad\quad$ **if** $\Omega^{SA}_{new} < \Omega^{SA}_{best}$ **then**
12: $\quad\quad\quad \Phi_{best} = \Phi_{new}$
13: $\quad\quad\quad \Omega^{SA}_{best} = \Omega^{SA}_{new}$
14: $\quad\quad$ **end if**
15: $\quad$ **end if**
16: $\quad$ t $= t * \alpha$
17: **end while**
18: **return** Φ_{best}

function decreases with a larger δ, i.e., we are less likely to accept worse neighbors if they are further from the current solution, or a smaller temperature, i.e., the probability of accepting worse neighbors decreases during the search. We use a time limit as the stopping criterion.

5.1 Initial Solution

The goal of the InitialSolution function, shown in Alg. 2, is to find a good starting point for SA. We start out by choosing a common period for all windows in the port, which helps speed up overlap and worst-case latency calculations. This also means that there is a maximum of two GCL entries per queue, which means that a $GCL_{max} \geq 16$ will suffice for this solution. The choice of this period is important: A larger period means longer worst-case latencies but less bandwidth occupation since the distance between two consecutive windows is longer (in the worst-case a stream arrives right at the moment when it can't fit into the current window anymore, requiring it to wait a full window period). We choose the minimum period that is larger than the combined length of all streams in the port from a set containing all stream periods, their greatest common divisor (GCD), and half of that value (line 2-5). We found empirically that choosing a period from this set of values provides a good balance between minimizing worst-case delays and bandwidth occupation. The half GCD value is often useful for cases in which we have a large GCD of the given stream periods but tight deadlines (e.g. equal to the stream period). In the worst-case a stream gets delayed by approximately one window period per hop, so having

Algorithm 2 SA Initial Solution

Require: Streams $\mathcal{F}$, Switch Ports P

1: **for all** $p \in P$ **do**
2: $T^p = \{f.T \mid q \in p.Q_{ST}, f \in \mathcal{F}(q)\}$
3: **for all** $q \in p.Q_{ST}$ **do**
4: $q.T = MinPossiblePeriod(T^p \cup gcd(T^p) \cup \frac{gcd(T^p)}{2})$
5: **end for**
6: $\phi_{cur} = 0$
7: **for all** $q \in p.Q_{ST}$ **do**
8: $tl = sum(\{f.l \mid f \in \mathcal{F}(q)\})$
9: $pp = sum(\{\frac{f.l}{f.T} \mid f \in \mathcal{F}(q)\})$
10: $q.w = max(tl, pp * q.T) + max(F_l^q)$
11: $q.\phi = \phi_{cur}$
12: $\phi_{cur} = \phi_{cur} + q.w$
13: **end for**
14: **end for**
15: **return** P

those small window periods becomes very important. For topologies with very long routes, it could be worthwhile to consider even smaller fractions of the GCD in the set. Then we decide the length for each window (line 10). It has to be at least as long as the total sending time of all streams in that queue (line 8), and its size relative to the period of the window has to be at least as big as the stream sizes relative to their period (line 9). Finally, we align the windows in the different queues so they do not overlap (line 11-12), since, in the worst-case, an overlapping part of a window has to be considered occupied.

5.2 SA Objective Function

The objective function Ω^{SA} used inside SA is shown in Eq. (12). The difference between Ω^{SA} and Ω used by CP (Eq. (1), Section 4.2) is that Ω minimizes the average bandwidth usage and then uses timing constraints and network calculus to check that a solution is schedulable.

Ω^{SA} has two components: The first component Ω^{bw} is measuring the bandwidth consumed by all windows across all ports on average, and is equivalent Ω. The second component Ω^{inf} is the number of streams that miss their deadline, also called infeasible streams. Thus, instead of using the schedulability as a constraint as in the CP formulation, we allow SA to visit unschedulable solutions in the hope of driving the search towards schedulable solutions. The amount of infeasible streams is determined by running the NC-based worst-case delay analysis of [16] with the given set of windows. Since the average consumed bandwidth is at most 1 (equals to 100%), any missed deadlines will increase the objective value and thus drive the search to schedulable solutions that decrease Ω^{inf}, which dominates the objective function when a solution is not schedulable. The weights w_a and w_b in Eq. (12) can be used to control

the relative importance of the two components, e.g., when a system engineer prefers lower bandwidth at the expense of schedulability, w_a can be increased and w_b decreased. In our experiments, we have used $w_a = w_b = 1$, which we found is a good choice when searching for schedulable solutions.

$$\Omega^{SA} = w_a \times \Omega^{bw} + w_b \times \Omega^{inf}, \text{ where} \tag{12}$$

$$\Omega^{bw} = \frac{\sum_{q \in p.Q} \frac{q.w}{q.T}}{\mathcal{N}(P)}, \forall p \in P \tag{13}$$

$$\Omega^{inf} = |\{f \mid f \in \mathcal{F} \wedge m_i^{dest(f)}.\phi + m_i^{dest(f)}.L - m_i^{src(f)}.\phi > f.D - \delta\}|$$

5.3 Neighbor Function

The purpose of the RandomNeighbor(Φ, p_{mv}) function in Alg. 1 is to select a close neighbor of the current solution Φ. A neighbor is generated by performing a transformation (also called "moves") on the current solution. This transformation function is designed such that the SA search will have good coverage of the solution space. The solution space, in our case, includes all solutions that have one window per queue, with a minimum length and without overlap of windows in the same port. Our neighbor function uses two different moves to change the current solution:

- **MoveWindow**: Selects a random occupied queue. Changes the window offset to a random value within the range of all offsets where the window will not overlap with windows in other queues on the same port. This move occurs with a given probability of p_{mv}.
- **ChangeWindowSize**: Selects a random occupied queue. Changes the window size to a random value in the range between the minimum window size and the maximum size before an overlap with another window in the same port would occur. This move is applied with a probability of $1 - p_{mv}$.

We have used a value of $p_{mv} = 0.8$ in the experiments. This makes it more likely for a MoveWindow move to occur, which is usually more impactful, since the window sizes are already set to reasonable initial values by the initial solution, while the window alignment across ports is not very good yet.

6 Related Work

Scheduling homogeneous TSN networks in which all devices are scheduled and synchronized has been solved in various forms using heuristics [28–33] and optimal ILP- or SMT-based approaches [3, 6, 34–37]. The most relevant results for providing real-time communication properties in TSN networks, to which we compare our approach, have been presented in [3, 6, 7, 20] (summarized in Table 3). Originally, the TSN scheduling problem was addressed in [3] for fully deterministic ST traffic temporal behavior and temporal isolation between ST and non-ST (e.g., AVB, BE) streams/flows, similar to TTEthernet [38, 39]. In our comparison, we call this method *0GCL*, since, besides enforcing the

required end-to-end latency of ST streams, the scheduling constraints also impose a strictly periodic frame transmission resulting in 0 jitter forwarding of critical traffic. The work in [7] uses heuristics instead of SMT-solvers to solve the 0-jitter scheduling problem in order to improve scalability while also minimizing the end-to-end latency of AVB streams. In [6], which we call *Frame-to-Window-based*, the 0-jitter constraint of [3] is relaxed by allowing more variance in the transmission times of frames over the hops of their routed paths. This increases the solution space at the expense of increased complexity in the correctness constraints. The method in [6] can be viewed as window-based scheduling, but, unlike our approach, it requires a unique mapping between GCL windows and frames in order to avoid non-determinism in the queues. In [20] the TSN scheduling problem is reduced to having one single queue for ST traffic and solving it using Tabu Search that optimizes the number of guard-bands in order to optimize bandwidth usage.

The main goal of the aforementioned works is similar to ours, namely to allow temporal isolation and compositional system design for ST streams with end-to-end guarantees and deterministic communication behavior. However, all previous methods impose that the end-systems from which the ST traffic originates are synchronized to the rest of the network and have the IEEE 802.1Qbv timed-gate mechanism (i.e., they are scheduled). The open gate windows are then either a result of the frame transmission schedule [3, 7] or are uniquely associated with predefined subsets of frames [6]. However, the above property is a significant limitation. In many use cases, especially in the industrial and automotive domains (c.f. [8]), the end-systems are usually off-the-shelf sensors, microcontrollers, industrial PCs, and edge devices that do not have TSN capabilities.

The work in [13] proposed a more naive window-based approach (WND) in which the GCL window offsets on different network nodes are not included, thereby essentially limiting the mechanisms by requiring all GCL windows to be lined up between bridges. Moreover, [13] uses a less advanced analysis step (c.f. [40]) in the scheduling decisions and a more naive heuristic approach. These limiting assumptions were relaxed in [15], which has proposed a Constraint Programming solution to the window optimization problem.

The work in [41] proposes a scheduling model for TSN networks in industrial automation with different traffic types and a hierarchical scheduling procedure for isochronous traffic. The method proposed in [14] adopts a so-called stream batching approach, which can be classified as window-based in that it can assign multiple frames to the same GCL window. However, the end-points still need to be synchronized and scheduled, and, additionally, the worst-case delay bounds within the batch windows may lead to deadline misses since they are not based on formal methods like the network calculus framework in our approach. In [42], the authors present an NC-based analysis for overlapping GCL windows with less pessimistic latency bounds and a scheduling algorithm (FWOS) that focuses on maximizing the allowable overlap of GCL windows to increase the bandwidth of unscheduled traffic without

jeopardizing the schedulability of ST traffic. As opposed to our method, [42] cannot guarantee the schedulability of traffic arriving from unscheduled or unsynchronized end-systems.

Classical approaches like strict priority (SP) and AVB [43] do not require a time-gate mechanism and also work with unscheduled end-systems. In order to provide response-time guarantees, a worst-case end-to-end timing analysis through methods like network calculus [44, 45] or Compositional Performance Analysis (CPA) [46] are used. In [47–49], the rate-constrained (RC) streams of TTEthernet [18, 50] are analyzed using network calculus. Other works, such as [51, 52], study the response-time analysis for TDMA-based networks under the strict priority (SP) and weighted round-robin (WRR) queuing policies. Zhao et al. [16] present a worst-case delay analysis, which we use in this paper, for determining the interference delay between ST traffic on the level of flexible GCL windows. Using SP only or leaving all ST windows open for the entire hyperperiod duration (which amounts to SP for ST traffic) will not result in the same response-time bounds and schedulability as our method. Our method can delay specific high-priority ST streams when needed to allow a timely transmission of lower-priority ST streams with a much tighter deadline. Unlike SP, our method uses the IEEE 802.1Qbv timed gates to open and close queues as needed to enforce isolation between traffic classes. With pure SP (or when leaving all gates open at all times), misbehaving end-systems (e.g., babbling-idiot failures) will disrupt all (lower-priority) traffic classes, potentially leading to a loss of all real-time properties of the network.

In [17], the authors present hardware enhancements to standard IEEE 802.1Qbv bridges (along with correctness constraints for the schedule generation) that remove the need for the isolation constraints between frames scheduled in the same egress queue defined in [3]. Another hardware adaptation for TSN bridges, which has been proposed by Heilmann et al. [53] is to increase the number of non-critical queues in order to improve the bandwidth utilization without impacting the guarantees for critical messages.

Lastly, we look at our CP optimization strategy to improve scalability and performance. In our CPWO method, we combine an exact CP solver with a metaheuristic search within a loop to improve scalability. A similar direction and aim, but using a very different method, is given in [54], where the authors combine an SMT-based method with a discrete event simulation within a counterexample-guided abstraction refinement loop and apply it to railway design.

6.1 In-depth Formal Comparison

In this section, we compare *FWND* with the related work in terms of the objectives and constraints. The related work on ST scheduling using 802.1Qbv consists of: (i) zero-jitter GCL (0GCL) [3, 7], (ii) Frame-to-Window-based GCL (FGCL) [6], and (iii) Window-based GCL (WND) [13].

We summarize the requirements of the ST scheduling approaches from the related work and our *FWND* approach in the first column of Table 3. The

Table 3: Scheduling Approaches in TSN

Requirements	**0GCL** [3][7]	**FGCL** [6]	**WND** [13]	**FWND**
Device Capabilities	802.1Qbv	802.1Qbv	802.1Qbv	802.1Qbv
ES Capabilities	scheduled	scheduled	non-scheduled	non-scheduled
SW Capabilities	scheduled	scheduled	scheduled	scheduled
Frame Constraint	Yes	Yes	Yes	Yes
Link Constraint[1]	Yes	Yes	No	No
Bandwidth Constraint[2]	Implicit	Yes	Yes	Yes
Stream Transmission Constraint[3]	Yes	Yes	No	No
Frame-to-Window Assignment	Implicit	Yes	No	No
Stream/Frame Isolation	Yes	Yes	No	No
End-to-end Constraint	Yes	Yes	Yes	Yes
Schedule synthesis	Yes (intractable)	Yes (intractable)	No (only windows)	No (only windows)
Timing analysis required	No	No	Yes	Yes

[1] This constraint refers to windows on different queues of the same port not allowing to overlap in the time domain. This constraint is called the "ordered window constraint" in [6].
[2] This constraint is called "window size constraint" in [6].
[3] This constraint is called "Stream Constraint" in [6].

first three requirements refer to the device capabilities needed for the different approaches, and the next seven rows summarize which constraints and isolation requirements are needed by which approach. The last two rows present the requirements of the complexity of the optimization problem that needs to be solved to provide a solution for the respective approach.

To better understand the fundamental differences and the similarities (in terms of the imposed correctness constraints and schedulability parameters) between our work and the approaches that require synchronized and scheduled end systems, we briefly reiterate the formal constraints of previous work. We describe, based on [3, 6, 19], the relevant scheduling constraints for creating correct TSN schedules when using frame- and window-based methods. Table 3 shows which of these are needed by which approach.

We adapt some notations from [3, 19] to describe the constraints and assume certain simplifications without loss of generality, e.g. the macrotick is the same in all devices, all streams have only one frame per period, the propagation delay d_p is 0. We refer the reader to [3, 6] for a complete and generalized formal definition of the correctness constraints. We denote the messages (frames) of a stream f_i on a link $[v_a, v_b]$ as $m_i^{[v_a,v_b]}$. A message $m_i^{[v_a,v_b]}$ is defined by the tuple $\langle m_i^{[v_a,v_b]}.\phi, m_i^{[v_a,v_b]}.l\rangle$, denoting the transmission time and duration of the frame on the respective link [3, 19].

Frame Constraint. Any frame belonging to a critical stream has to be transmitted between time 0 and its period T_i. To enforce this, we have the frame constraint from [3]:

$$\forall f_i \in \mathcal{F}, \forall [v_a, v_b] \in f_i.r : \\ \left(m_i^{[v_a,v_b]}.\phi \geq 0\right) \wedge \left(m_i^{[v_a,v_b]}.\phi \leq f_i.T - m_i^{[v_a,v_b]}.l\right).$$

Link Constraint. A physical constraint of Ethernet-based networks is that only one frame can be on the wire from one port to another at a time. In [3] and [6] this constraint is expressed as windows on two different queues of the same egress port not being able to overlap. In [13] and the FWND method presented in this paper, windows on different queues may overlap, leading to added interference delays since naturally, only one frame can be sent on the physical link at a time. Still, in both [13] and the FWND, solutions where windows overlap are excluded since there is no added improvement from such schedules.

The link constraint adapted from [3] is hence:

$$\begin{aligned}
&\forall [v_a, v_b] \in \boldsymbol{E}, \forall m_i^{[v_a,v_b]}, m_j^{[v_a,v_b]} (i \neq j), \\
&\forall a \in [0, hp_i^j / f_i.T - 1], \forall b \in [0, hp_i^j / f_j.T - 1] : \\
&\Big(m_i^{[v_a,v_b]}.\phi + a \times f_i.T \geq m_j^{[v_a,v_b]}.\phi + b \times f_j.T + m_j^{[v_a,v_b]}.l\Big) \vee \\
&\Big(m_j^{[v_a,v_b]}.\phi + b \times f_j.T \geq m_i^{[v_a,v_b]}.\phi + a \times f_i.T + m_i^{[v_a,v_b]}.l\Big),
\end{aligned}$$

where $hp_i^j = lcm(f_i.T, f_j.T)$ is the hyperperiod of f_i and f_j.

Bandwidth Constraint. The bandwidth constraint expressed explicitly in our method ensures that there is no infinite backlog, i.e., the windows for the streams are large enough that the frames of the streams can be transmitted at some point. In 0GL [3] this constraint is implicit since the schedule is created without the separation of streams and windows, meaning that each window is large enough to transmit the respective frames. In [6] there is no one-to-one assignment between frames and windows; however, the window size constraint is equivalent to the bandwidth constraint. Using this constraint the length of the gate open window is required to be equal to the sum of the frame lengths that have been assigned to it. In [13] the bandwidth constraint is explicit in the conditions for the correctness of the schedule generation.

Stream Transmission Constraint. The stream transmission constraint expresses that the propagation of frames of a stream follows the sequential order along the path of the stream. This (optional) constraint enforces that a frame is forwarded by a device only after it has been received at that device also taking into account the network precision, denoted with δ:

$$\begin{aligned}
&\forall f_i \in \mathcal{F}, \forall [v_a, v_x], [v_x, v_b] \in f_i.r, \forall m_i^{[v_a,v_x]}, \forall m_i^{[v_x,v_b]} : \\
&m_i^{[v_x,v_b]}.\phi - \delta \geq m_i^{[v_a,v_x]}.\phi + m_i^{[v_a,v_x]}.l.
\end{aligned}$$

In FWND (and also in [13]) this constraint is not explicitly needed since there is no predefined assignment of frames to windows and hence, there is no explicit ordering needed in sequential hops along the route of a stream, i.e., the transmission GCL window which is used at a certain time will depend on the enqueueing order at that time in the egress queue.

End-to-End Constraint. The maximum end-to-end latency constraint (expressed by the deadline $f_i.D$) enforces a maximum time between the sending and the reception of a stream. We denote the sending link of stream f_i with $src(f_i)$ and the last link before the receiving node with $dest(f_i)$. The maximum end-to-end latency constraint [3] is hence

$$\forall f_i \in \mathcal{F} : m_i^{dest(f_i)}.\phi + m_i^{dest(f_i)}.L - m_i^{src(f_i)}.\phi \leq f_i.D - \delta.$$

Here again, the network precision δ needs to be taken into account since the local times of the sending and receiving devices can deviate by at most δ.

802.1Qbv Stream/Frame Isolation. Due to the non-determinism problem in TSN (c.f. Section 2.2), previous solutions (e.g., [3, 6]) need an isolation constraint that maintains queue determinism. We refer the reader to [3] for an in-depth explanation and only summarize here the stream and frame isolation constraint adapted from [3]. Let $m_i^{[v_a,v_b]}$ and $m_j^{[v_a,v_b]}$ be, respectively, the frame instances of $f_i \in \mathcal{F}$ and $f_j \in \mathcal{F}$ scheduled on the outgoing link $[v_a, v_b]$ of device v_a. Stream f_i arrives at the device v_a from some device v_x on link $[v_x, v_a]$. Similarly, stream f_j arrives from another device v_y on incoming link $[v_y, v_a]$. The simplified stream isolation constraint adapted from [3], under the assumption that the macrotick of the involved devices is the same, is as follows:

$$\begin{aligned}
&\forall [v_a, v_b] \in \boldsymbol{E}, \forall m_i^{[v_a,v_b]}, m_j^{[v_a,v_b]} (i \neq j),\\
&\forall a \in [0, hp_i^j / f_i.T - 1], \forall b \in [0, hp_i^j / f_j.T - 1] :\\
&\Big(m_i^{[v_a,v_b]}.\phi + a \times f_i.T + \delta \leq m_j^{[v_y,v_a]}.\phi + b \times f_j.T\Big) \vee\\
&\Big(m_j^{[v_a,v_b]}.\phi + b \times f_j.T + \delta \leq m_i^{[v_x,v_a]}.\phi + a \times f_i.T\Big).
\end{aligned}$$

Here again $hp_i^j = lcm(f_i.T, f_j.T)$ is the hyperperiod of f_i and f_j. The constraint ensures that once a stream arrives at a device, no other stream can enter the device until the first stream has been sent.

The above constraints apply to frames that are placed in the same queue on the egress port. However, the scheduler may choose (if possible) to place streams in different queues, isolating them in the space domain. Hence, the complete constraint [3] for frame/stream isolation for two streams f_i and f_j scheduled on the same link $[v_a, v_b]$ can be expressed as

$$\Big(\Phi_{[v_a,v_b]}(f_i, f_j)\Big) \vee \Big(m_i^{[v_a,v_b]}.q \neq m_j^{[v_a,v_b]}.q\Big),$$

with $m_i^{[v_a,v_b]}.q \leq N_{tt}$ and $m_j^{[v_a,v_b]}.q \leq N_{tt}$ and where $\Phi_{[v_a,v_b]}(f_i, f_j)$ denotes either the stream or frame isolation constraint from before.

Decoupling of frames. So far, the constraints were applicable on the level of frames, and the open windows of the GCLs were constructed from the resulting frame schedule. The approach in [6] decouples the frame transmission from the respective open gate windows defined in the GCLs, similar

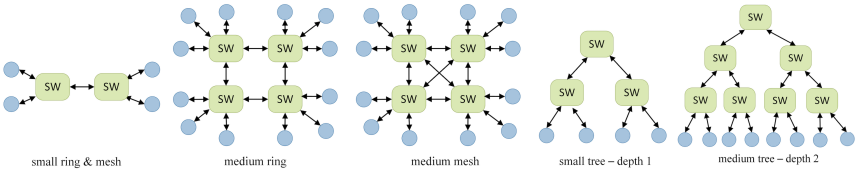


Fig. 6: Network topologies used in the test cases [55]

to our approach[1]. However, in [6] the requirement is that there is a unique assignment of which frames are transmitted in which windows, although also multiple frames can be assigned to be sent in the same window. Hence, the assignment of frames and, consequently, the length of each gate open window are, therefore, an output of the scheduler. Therefore, we have to construct additional constraints when (partially) decoupling frames from windows. For a more in-depth description and formalization of these constraints, we refer the reader to [6].

Frame-to-Window Assignment. The frame-to-window assignment restricts a frame to be assigned to a specific window, although multiple frames can be assigned to the same window. In [3] each frame is assigned implicitly to exactly one GCL window.

Comparing the existing approaches with the one proposed in this paper, we see that the choice of scheduling mechanism is, on the one hand, highly use-case specific and, on the other hand, is constrained by the available TSN hardware capabilities in the network nodes. While the frame- and window-based methods from related work result in precise schedules that emulate either a 0- or constrained-jitter approach (e.g., like in TTEthernet), they require end systems to not only be synchronized to the network time but also the end devices to have 802.1Qbv capabilities, i.e., to be scheduled. This limitation might be too restrictive for many real-world systems relying on off-the-shelf sensors, processing, and actuating nodes. While our *FWND* method overcomes this limitation, it does require a worst-case end-to-end analysis that introduces a level of pessimism into the timing bounds, thereby reducing the schedulability space for some use cases. However, as seen in Table 3, our method does not require many of the constraints imposed on the streams and scheduled devices from previous work, thereby reducing the complexity of the schedule synthesis.

7 Evaluation

In this section, we evaluate our optimization solutions CPWO and SAWO for *FWND* on synthetic and real-world test cases in Section 7.1 and Section 7.2, respectively.

[1]Note that [3, 6, 7] cannot be used in our context because they require scheduled and synchronized ESs.

7.1 CPWO Evaluation

7.1.1 Test Cases and Setup

We implemented our *CPWO* approach in Java, utilizing the Java kernel of the RTC toolbox [56]. *CPWO* is designed as a command-line program, employing XML for input and output data, and it utilizes Google OR-Tools [57] as the CP solver. Being open-source, the codebase comprises more than 7500 lines and is accessible via a public repository[2]. The tests were run on an i9 CPU (3.6 GHz) with 32 GB of memory running the Windows operating system. The timeout is set to 10 to 90 minutes, depending on the size of the test case. The macrotick and $\mathcal{B}$ parameters are set to 1 μs and 1, respectively, in all the test cases.

Table 4: Details of the synthetic test cases

No.	Network Topology	Total No. of SWs	Total No. of ESs	Total No. of Streams	Hyperperiod (μs)
1	SRM	2	3	9	15,000
2	SRM	3	3	11	70,000
3	SRM	3	4	15	70,000
4	MR	4	6	15	30,000
5	MR	4	8	21	210,000
6	MR	5	11	27	210,000
7	MM	4	5	13	15,000
8	MM	6	12	30	210,000
9	MM	7	13	35	210,000
10	ST	3	4	7	15,000
11	ST	3	6	12	15,000
12	ST	3	7	16	105,000
13	MT	7	8	18	105,000
14	MT	7	8	25	105,000
15	MT	7	12	32	210,000

We have generated 15 synthetic test cases that have different network topologies (three test cases for each topology in Figure 6) inspired by industrial and automotive application requirements. Similar to [55], the network topologies are small ring & mesh (SRM), medium ring (MR), medium mesh (MM), small tree-depth 1 (ST), and medium tree-depth 2 (MT). The message sizes of streams are randomly chosen between 64 bytes and 1518 bytes, while their periods are selected from the set $P = \{1{,}500, 2{,}500, 3{,}500, 5{,}000, 7{,}500, 10{,}000\}\mu$s. The physical link speed is set to 100 Mbps. The details of the synthetic test cases are in Table 4 where the second column shows the topology of the test cases, and the number of switches, end systems, and streams are shown in columns 3 to 6.

[2]You can find *CPWO* on GitHub at https://github.com/rezabarzegaran/TSN.

We have also used two realistic test cases: an automotive case from General Motors (GM) and an aerospace case, the Orion Crew Exploration Vehicle (CEV). The GM case consists of 27 streams varying in size between 100 and 1,500 bytes, with periods between 1 ms and 40 ms and deadlines smaller or equal to the respective periods. The CEV case is larger, consisting of 137 streams, with sizes ranging from 87 to 1,527 bytes, periods between 4 ms and 375 ms, and deadlines smaller or equal to the respective periods. The physical link speed is set to 1000 Mbps. More information can be found in the corresponding columns in Table 6. Use cases use the same topologies as in [58] and [16], and we consider that all streams are ST.

As already mentioned, we opted for $\mathcal{B} = 1$ in all the test cases as it represents the minimum feasible value, assuming preemption is not allowed, even though it results in aggressive pruning. A less "relaxed" $\mathcal{B}$, e.g., $\mathcal{B} = 0$, does not generate feasible solutions as it does not consider frame backlog. A value of $\mathcal{B}$ in the interval $[0, 1)$ is only possible if preemption is enabled and supported in the network since a fractional value of $\mathcal{B}$ denotes the fragment length of frames that can be backlogged. Since we focus on non-preemptable critical flows, this is not an option in this work. A more "relaxed" $\mathcal{B}$ will result in CPWO running for several days without returning a solution. Due to the aggressive pruning, the solution returned by CPWO is not guaranteed to find the optimal solution and will, in fact, as the next section will show, miss good quality solutions.

However, we assign the value $\mathcal{B} = 1$ and the less "relaxed" value $\mathcal{B} = 0$ to the *CPWO* and assess its performance on the test cases. When $\mathcal{B} = 0$, *CPWO* generates solutions that do not comply with the WCD analysis. For instance, the WCD analysis reveals that in TC1, one out of 9 streams, and in TC15, four out of 32 streams fail to meet their deadlines. Conversely, all solutions generated by *CPWO* with $\mathcal{B} = 1$ are feasible,i.e. no missed deadlines, according to the WCD analysis.

7.1.2 CPWO Evaluation on Synthetic Test Cases

We have evaluated our CPWO solution for *FWND* on synthetic test cases. The results are depicted in Table 5 where we show the objective function value (average bandwidth Ω from Eq. (1)) and the mean WCDs. For a quantitative comparison, we have also reported the results for the three other ST scheduling approaches: *0GCL*, *FGCL*, *WND*. *0GCL* and *FGCL* were implemented by us with a CP formulation using the constraints from [3] and [6], respectively. The *WND* method has been implemented with the heuristic presented in [13], but instead of using the WCD analysis from [40], we extend it to use the analysis from [16] instead, in order not to unfairly disadvantage WND over our CPWO solution. Note that the respective mean worst-case end-to-end delays in the table are obtained over all the streams in a test case, from a single run of the algorithms, since the output of the algorithms is deterministic based on worst-case analyses, not based on simulations.

Table 5: CPWO evaluation results on synthetic test cases

No.	Ω^1 for 0GCL	Ω^1 for FGCL	Ω^1 for WND	Ω^1 for CPWO	ξ^2 for 0GCL (μs)	ξ^2 for FGCL(μs)	ξ^2 for WND (μs)	ξ^2 for CPWO (μs)	Γ^3 for 0GCL (ms)	Γ^3 for CPWO (ms)
1	35	35	614	510	192	126	1838	1556	215	8/164
2	25	22	640	528	246	151	2461	1806	895	12/249
3	15	15	549	495	175	486	1964	1384	1518	22/203
4	13	13	330	285	160	776	2925	1832	525	16/338
5	14	NA[4]	295	285	131	NA[4]	2838	2347	5187	16/423
6	13	NA[4]	275	205	129	NA[4]	2953	1976	6291	17/721
7	12	12	238	204	125	764	2913	1561	1152	35/3235
8	13	NA[4]	238	202	114	NA[4]	2878	1725	7603	36/2075
9	12	NA[4]	217	191	122	NA[4]	3074	1927	9171	56/12084
10	8	8	329	265	136	2284	4397	4327	2611	165/325
11	10	10	381	302	159	984	3047	2057	2840	231/1552
12	11	NA[4]	516	321	187	NA[4]	2543	1326	4650	260/3393
13	10	10	401	302	101	561	2529	471	978	130/914
14	9	9	611	402	120	785	2254	628	1256	162/1077
15	9	NA[4]	544	413	114	NA[4]	2680	713	6116	163/1433

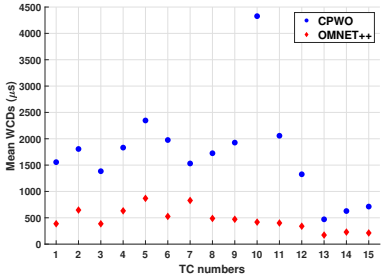
[1] Mean bandwidth usage. Values are multiplied by 1000
[2] Mean worst-case e2e-delay
[3] Mean runtime
[4] Ran out of memory

It is important to note that *0GCL* and *FGCL* are presented here as a means to evaluate CPWO; however, they are *not producing valid solutions* for our problem, which considers unscheduled end systems, see Table 3 for the requirements of each method. As expected, when end systems are scheduled and synchronized with the rest of the network as is considered in *0GCL* and *FGCL*, we obtain the best results in terms of bandwidth usage (Ω) and WCDs, noting that *0GCL* may further reduce the WCDs compared to *FGCL*.

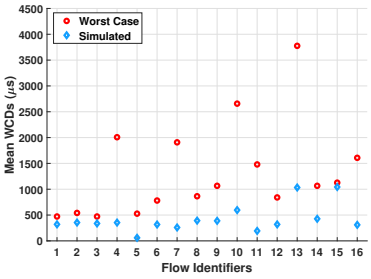
The only other approach that has similar assumptions to our *CPWO* is *WND* from [13]. As we can see from Table 5, in comparison to *WND*, our CPWO solution can slightly reduce the bandwidth usage. The most important result is that CPWO significantly reduces the WCDs compared to *WND*, with an average of 104% and up to 437% for some test cases such as TC13. Hence, we are able to obtain schedulable solutions in more cases compared to the work in [13]. Also, when comparing the WCDs obtained by our *CPWO* approach with the case when the end systems are scheduled, i.e., *0GCL* and *FGCL*, we can see that the increase in WCDs is not dramatic. This means that for many classes of applications, which can tolerate a slight increase in latency, we can use our CPWO approach to provide solutions for more types of network implementations, including those that have unscheduled and/or unsynchronized end systems. In addition, due to the complexity of their CP model, it takes a long runtime to obtain solutions for *0GCL* and *FGCL*, and the CP-model for *FGCL* run out of memory for some of the test cases (the NA in the table). As shown in the last two columns of Table 5, where we present the runtimes of *0GCL* and *CPWO*, *CPWO* reduces the runtime significantly. The two numbers in the runtime column represent the runtime for obtaining the last solution and the runtime for the whole CPWO run, respectively. The reason for reduced runtime with *CPWO* is that the CP model has to determine values for fewer variables compared to *0GCL*. *CPWO* introduces 3 variables (offset, period, and length) for each window (queue) in the network, whereas *0GCL* introduces a variable for each frame of each stream. The number of variables in the *0GCL* model depends on the hyperperiod, the number of streams,

Table 6: CPWO results on realistic test cases

	ORION (CEV)	GM
ES	31	20
SW	15	20
Streams	137	27
Mean WCDs (μs)	10,376	1,981
Ω ($\times$1000)	435	84
Runtime (s)	891	17



(a) Mean WCDs vs. simulated delays for CPWO



(b) Mean WCDs comparison for the streams of TC12

Fig. 7: WCD vs. simulated delays

and the stream periods, whereas the number of variables in the *CPWO* model depends on the number of switches and used queues.

7.1.3 CPWO Evaluation on Realistic Rest Cases

We have used two realistic test cases to investigate the scalability of CPWO and its ability to produce schedulable solutions for real-life applications. The results of the evaluation are presented in Table 6 where the mean WCDs, objective value Ω, and runtime for the two test cases are given. As we can see, CPWO has successfully scheduled all the streams in both test cases. Note that once all streams are schedulable, CPWO aims at minimizing the bandwidth. This means that CPWO may be able to achieve even smaller WCD values at the expense of bandwidth usage. In terms o runtime, the CEV test case takes longer since it has 864 variables, whereas GM has only 102 variables in the CP models.

7.1.4 Validating the CPWO Solutions with OMNET++

We have used the OMNET++ simulator with the TSN NeSTiNg extension [59] to evaluate the generated GCLs and the calculated worst-case delays. We have synthesized the GCLs for all approaches on all synthetic test cases, and we have observed that the GCLs are correct and the simulation behaves as expected based on the simulation. The mean WCDs of CPWO for the synthetic test

Table 7: Scalability evaluation of CPWO

No.	Total No. of Streams	Total No. of ESs	Total No. of SWs	Mean WCDs μs	Largest deadline μs	Ω ($\times$1000)
TC1	100	50	35	3,226	4,000	249
TC2	150	55	40	3,521	4,000	366
TC3	200	60	40	4,387	5,000	396
TC4	300	65	40	4,911	6,000	468
TC5	400	70	45	5,210	6,000	498
TC6	500	75	45	4,399	5,000	511

cases and the worst-case latency observed during multiple OMNET++ simulations (with the windows from CPWO) are depicted in Figure 7a. As expected, the latency values reported by OMNET++ are smaller than the WCDs, as reported by the WCD Analysis from [16]. This is because a simulation cannot easily uncover the worst-case behavior. However, the simulation indicates the average behavior and small delays mean that even for unscheduled/unsynchronized end systems, we are able to obtain solutions that are not only schedulable (WCDs are smaller than the deadlines) but also have good average behavior, where most of the time the delays are reasonable, even smaller than the static schedules obtained by *0GCL* and *CPWO* for scheduled and synchronized ESs. The pessimism result of the WCD analysis is unavoidable in systems with un-synchronized and/or unscheduled end-systems; in practice, however, simulated delays are much smaller, as can be seen in Figure 7a and Figure 7b. We also show in Figure 7b the simulated delays and WCDs for all streams of TC12. All the streams are schedulable, and, as expected, the simulated delays are smaller than the WCDs, calculated with the worst-case delay analysis derived in the work from [16].

7.1.5 CPWO Scalability Evaluation

We have investigated the scalability of CPWO on 6 larger test cases (TC1 to TC6), that have up to 120 devices (75 ESs and 45 SWs) and 500 streams. The results and the details of the test cases are presented in Table 7, where columns 2, 3, and 4 show the number of streams, end-systems, and switches, respectively. Columns 5, 6, and 7 show the mean WCD of streams in μs, the largest deadline of all streams in μs, and the objective value Ω, related to bandwidth, see Eq. (1). CPWO was able to generate schedulable solutions in all cases. Furthermore, CPWO optimizes the schedules for minimum bandwidth usage and has generated solutions that, besides being schedulable, have mean WCDs on average 14% smaller than the respective deadlines in all test cases.

7.2 SAWO Evaluation

7.2.1 Test Cases and Setup

For the SAWO vs. CPWO comparison in Section 7.2.2 and the SAWO evaluation using realistic test cases in Section 7.2.3, we used the same test cases defined in Section 7.1.1. For the test cases in Section 7.2.4 we have taken the

topology sizes proposed in [39] as a reference. We implemented our SAWO solution in Python and configured it for all experiments with $w_a = w_b = 1, p_{mv} = 0.8$ on an i7-8565U CPU with 16GB memory and using Python 3. The solution communicates with the worst-case delay analysis of [16] via sockets, eliminating the unnecessary delay of file I/O.s

7.2.2 SAWO Comparison to CPWO

We compare the SA-based approach (SAWO) to the CP-based solution (CPWO) but also to the classical zero-jitter GCL (0GCL) [3, 7], Frame-to-Window-based GCL (FGCL) [6], and Window-based GCL (WND) [13] solutions, in terms of the objective value (i.e., quality of the solution) and the mean worst-case end-to-end latency for the streams. We do not show the runtime figures since the SA-based solution was always set to a runtime of 2 and 10 min. We note that the runtime for CPWO is small due to the aggressive pruning parameter, thus trading off the quality of the solution for algorithm runtime. For the experiments, we use the same synthetic test cases described in Section 7.1.1. The details of the synthetic test cases can be found in Table 4.

Table 8: SAWO evaluation results on synthetic test cases

No.	Ω^1 for 0GCL	Ω^1 for FGCL	Ω^1 for WND	Ω^1 for CPWO	Ω^1 for SAWO	ξ^2 for 0GCL (μs)	ξ^2 for FGCL(μs)	ξ^2 for WND (μs)	ξ^2 for CPWO (μs)	ξ^2 for SAWO (μs)
1	35	35	614	510	452/455	192	126	1838	1556	1412/1424
2	25	22	640	528	346/343	246	151	2461	1806	1648/1797
3	15	15	549	495	300/293	175	486	1964	1384	1684/1752
4	13	13	330	285	236/222	160	776	2925	1832	1608/1488
5	14	NA[3]	295	285	250/225	131	NA[3]	2838	2347	1380/1526
6	13	NA[3]	275	205	254/208	129	NA[3]	2953	1976	1250/1407
7	12	12	238	204	136/92	125	764	2913	1561	848/784
8	13	NA[3]	238	202	225/144	114	NA[3]	2878	1725	787/981
9	12	NA[3]	217	191	218/146	122	NA[3]	3074	1927	774/1138
10	8	8	329	265	85/84	136	2284	4397	4327	4787/4509
11	10	10	381	302	104/97	159	984	3047	2057	2472/2518
12	11	NA[3]	516	321	254/251	187	NA[3]	2543	1326	1367/1188
13	10	10	401	302	157/166	101	561	2529	471	1149/898
14	9	9	611	402	196/201	120	785	2254	628	1219/1057
15	9	NA[3]	544	413	210/206	114	NA[3]	2680	713	1045/1081

[1] Mean bandwidth usage. Values are multiplied by 1000
[2] Mean worst-case e2e-delay
[3] Ran out of memory

Table 8 presents the results for SAWO compared to the aforementioned solutions. For SAWO, the columns showing the objective value Ω and the mean worst-case e2e delay present two numbers obtained with 2 and 10 minute runtime, respectively.

We can see that SAWO can achieve significantly better results than CPWO in terms of the objective value Ω. While the runtime of 2 min is sufficient to get a good result, this result can be further improved by a longer runtime (see Section 7.2.4 for a more detailed analysis of the runtime impact). The objective function does not include the mean e2e-delay but the number of infeasible streams. That means that it is beneficial for the solutions to accept a higher mean e2e-delay for a lower objective value, e.g., by decreasing the size of a window. That can be seen, for example, in test case 2. However, sometimes

there are also solutions that have both a lower objective value and e2e-delay, e.g., test case 4. This can happen through a window being moved to a better offset, which would decrease the e2e-delay without increasing the objective value. Please note that the objective function Ω is the same for both CPWO and SAWO since SAWO can schedule all streams and thus there is no penalty term for infeasible streams in the SAWO objective function.

As mentioned before in Section 7.1.2, *0GCL* and *FGCL* are included as a means to evaluate SAWO; however, they are *not producing valid solutions* for our problem, as they require scheduled end-systems. As expected, when end systems are scheduled and synchronized with the rest of the network, as is considered in *0GCL* and *FGCL*, we obtain the best results in terms of bandwidth usage (Ω) and WCDs.

7.2.3 SAWO Evaluation on Realistic Test Cases

Table 9: SAWO results on realistic test cases

	ORION (CEV)	**GM**
ES	31	20
SW	15	20
Streams	137	27
Mean WCDs (μs)	341	992
Ω ($\times$1000)	374	15
Runtime (s)	600	600

As with CPWO, we have used two realistic test cases from [58] and [16], an automotive case from General Motors (GM) and an aerospace case, the Orion Crew Exploration Vehicle (CEV), where we consider that all streams are critical and scheduled. For the details of the test cases, please see the description in Section 7.1.1. We show the scalability of SAWO and its ability to produce schedulable solutions for real-life applications. The results of the evaluation are presented in Table 9 where the mean WCDs, objective value Ω, and runtime for the two test cases are given. As a comparison to CPWO, we refer the reader to the results presented in Table 6. As we can see, SAWO has successfully scheduled all the streams in both test cases and produces better results than CPWO in terms of mean WCD and quality of the solution (objective value Ω). The runtime for SAWO was set to 10 minutes for the two realistic test cases.

Table 10: Parameters of SAWO test batches

	Topology	Number of testcases	SW	ES	Avg. number of streams
medium	mesh	50	4	16	46
large	mesh	50	8	48	148
huge	mesh	50	16	96	416

7.2.4 SAWO Evaluation on Large Synthetic Test Cases

As previously described, CPWO delivers good results in a reasonable time for small problem sizes, but does not scale well for large inputs unless the search space pruning is done very aggressively, which leads to low-quality solutions. Therefore, we show that our SAWO heuristic algorithm scales well with the network and problem size while still offering good-quality solutions.

To evaluate the impact of the heuristic runtime and the test case size on the resulting solution quality, we have created three test batches as described in Table 10. Each batch consists of 50 test cases with a mesh topology (see Figure 6) with sizes medium, large, and huge as described in [39]. For each test case, we generated streams with random routes, priorities, and sizes under the constraint that no link utilization may exceed 50% until an average link utilization threshold of 15% was reached. For the medium test cases, there were between 31 and 67 streams with an average of around 46 streams per test case. For the large batch, the 50 test cases had between 127 and 178 streams averaging 147 streams. The huge test batches averaged 416 streams per test case with a minimum of 364 and a maximum of 475. Each stream has a random size between 64 and 1500 Bytes and a random period from the set $\{1, 2, 5, 10\}$ ms, as defined for the use cases in [60]. The stream deadline is set to ten times the stream's period.

We ran each test batch with a 2-minute and a 10-minute timeout and measured the best objective function value Ω^{SA} obtained within the timeout. Figure 8 shows the results as box plots for the medium and large test batches (y-axis), with all objective values multiplied by 1000 for clarity (x-axis). We set the upper and lower whisker bounds to depict outliers above $1.5 \times IQR$ of the 3^{rd} quartile and under $1.5 \times IQR$ of the 1^{st} quartile. Additionally, we show all data points within the figure. The median for the medium-sized size test-cases with 2 minutes and 10 minutes timeout was 172.5 and 140, respectively. The median for the large test cases with 2 minutes and 10 minutes timeout was 180 and 179, respectively.

We can see that the test cases are consistently completely schedulable (objective value below 1000) with good solution quality. Furthermore, we can

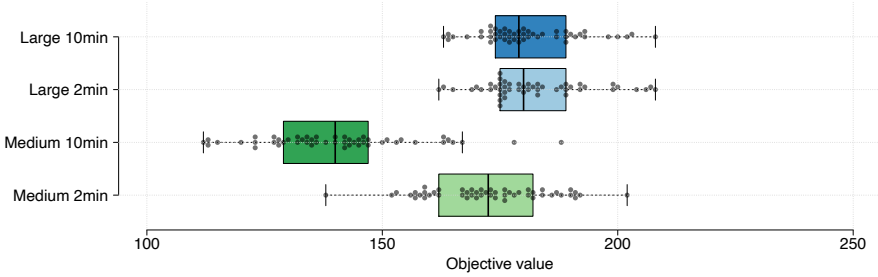


Fig. 8: Objective value boxplots for medium and large SAWO test batches

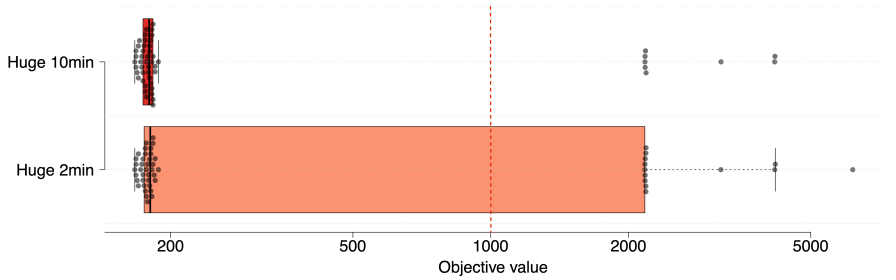


Fig. 9: Objective value boxplots for huge SAWO test batches

see that the heuristic quickly can find good quality solutions. A longer run-time has a positive impact on the solution quality, but this impact depends on the size of the test case. The time needed to achieve significant improvement increases with the size of the test case, as the amount of possible moves increases in parallel with the worst-case analysis taking more time per iteration.

Figure 9 shows the result for the huge test batches as a box plot with the same whisker boundaries and outlier setting as before but with a logarithmic x-axis showing the objective value. The median for the huge test cases were 180.5 and 179.5 for the 2 and 10 minute timeout, respectively. With a 2 minute timeout, 13 out of 50 test cases had at least one unschedulable stream (objective value over 1000) and overall low solution quality. With a 10 min timeout, the solution quality improved, and only 8 out of 50 test cases had at least one unschedulable stream. From the total of 20810 streams in the 50 test cases, a total of 35 streams were unschedulable with a 2 minute timeout, while a total of 21 were unschedulable with the 10 minute timeout.

8 Conclusions

We have addressed the problem of guaranteeing real-time communication behavior in heterogeneous TSN networks, introducing a more flexible heuristic schedule synthesis approach (FWND) which decouples the frame transmission from the scheduled time-aware shaper (TAS) windows. Using this approach, we have proposed two solutions to solve the problem, one based on a Constraint-Programming formulation within a Tabu Search metaheuristic (CPWO) and one based on a Simulated Annealing metaheuristic (SAWO). The CPWO solution uses a novel proxy function that can be parametrized to trade off run-time performance for search-space pruning in the CP-model. We have shown that for large use cases, CPWO has to either aggressively prune the search space, leading to low-quality solutions, or is intractable. Therefore, we have introduced SAWO, which scales better for large test cases while still offering good-quality solutions. We evaluated our approaches using synthetic and real-world test

cases, comparing them with existing mechanisms, and validated the generated schedules using OMNET++.

References

[1] Ashjaei, M., Lo Bello, L., Daneshtalab, M., Patti, G., Saponara, S., Mubeen, S.: Time-sensitive networking in automotive embedded systems: State of the art and research opportunities. JSA **117**, 102137 (2021). https://doi.org/10.1016/j.sysarc.2021.102137

[2] Institute of Electrical and Electronics Engineers, Inc: Official Website of the 802.1 Time-Sensitive Networking Task Group. http://www.ieee802.org/1/pages/tsn.html. Accessed: 23.10.2020 (2016)

[3] Craciunas, S.S., Serna Oliver, R., Chmelik, M., Steiner, W.: Scheduling real-time communication in IEEE 802.1Qbv Time Sensitive Networks. In: Proc. RTNS, pp. 183–192 (2016)

[4] Institute of Electrical and Electronics Engineers, Inc: 802.1AS-Rev - Timing and Synchronization for Time-Sensitive Applications. http://www.ieee802.org/1/pages/802.1AS-rev.html. Accessed: 23.10.2020 (2017)

[5] Institute of Electrical and Electronics Engineers, Inc: 802.1Qbv - Enhancements for Scheduled Traffic. http://www.ieee802.org/1/pages/802.1bv.html. Draft 3.1, Accessed: 23.10.2020 (2016)

[6] Serna Oliver, R., Craciunas, S.S., Steiner, W.: IEEE 802.1Qbv Gate Control List Synthesis using Array Theory Encoding. In: Proc. RTAS (2018)

[7] Pop, P., L. Raagaard, M., Craciunas, S.S., Steiner, W.: Design optimization of cyber-physical distributed systems using IEEE Time-Sensitive networks (TSN). IET Cyber-Physical Systems: Theory and Applications **1**(1), 86–94 (2016)

[8] Schriegel, S., Kobzan, T., Jasperneite, J.: Investigation on a distributed sdn control plane architecture for heterogeneous time sensitive networks. In: Proc. WFCS (2018). https://doi.org/10.1109/WFCS.2018.8402356

[9] Mateu, D.B., Ashjaei, M., Papadopoulos, A.V., Proenza, J., Nolte, T.: LETRA: mapping legacy ethernet-based traffic into TSN traffic classes. In: Proc. ETFA (2021). https://doi.org/10.1109/ETFA45728.2021.9613637

[10] Larrañaga, A., Lucas-Estañ, M.C., Martinez, I., Val, I., Gozalvez, J.: Analysis of 5G-TSN integration to support industry 4.0. In: Proc. ETFA (2020). https://doi.org/10.1109/ETFA46521.2020.9212141

[11] Böhm, M., Wermser, D.: Multi-domain time-sensitive networks—control plane mechanisms for dynamic inter-domain stream configuration. Electronics **10**(20) (2021). https://doi.org/10.3390/electronics10202477

[12] von Arnim, C., Drăgan, M., Frick, F., Lechler, A., Riedel, O., Verl, A.: Tsn-based converged industrial networks: Evolutionary steps and migration paths. In: Proc. ETFA, vol. 1, pp. 294–301 (2020). https://doi.org/10.1109/ETFA46521.2020.9212057

[13] Reusch, N., Zhao, L., Craciunas, S.S., Pop, P.: Window-based schedule synthesis for industrial IEEE 802.1Qbv TSN networks. In: Proc. WFCS (2020)

[14] Hellmanns, D., Falk, J., Glavackij, A., Hummen, R., Kehrer, S., Dürr, F.: On the performance of stream-based, class-based time-aware shaping and frame preemption in TSN. In: Proc. ICIT, pp. 298–303 (2020)

[15] Barzegaran, M., Reusch, N., Zhao, L., Craciunas, S.S., Pop, P.: Real-time traffic guarantees in heterogeneous time-sensitive networks. In: Proc. RTNS, pp. 46–57. ACM, New York, NY, USA (2022). https://doi.org/10.1145/3534879.3534921

[16] Zhao, L., Pop, P., Gong, Z.J., Fang, B.W.: Improving latency analysis for flexible window-based GCL scheduling in TSN networks by integration of consecutive nodes offsets. IEEE Internet of Things (2020, Early Access Article, https://doi.org/10.1109/JIOT.2020.3031932)

[17] Vlk, M., Hanzálek, Z., Brejchová, K., Tang, S., Bhattacharjee, S., Fu, S.: Enhancing schedulability and throughput of time-triggered traffic in IEEE 802.1Qbv time-sensitive networks. IEEE Transactions on Communications **68**(11), 7023–7038 (2020)

[18] Issuing Committee: As-2d2 Deterministic Ethernet And Unified Networking: SAE AS6802 Time-Triggered Ethernet. http://standards.sae.org/as6802/. Accessed: 23.10.2020 (2011)

[19] Craciunas, S.S., Serna Oliver, R.: An Overview of Scheduling Mechanisms for Time-sensitive Networks. Technical report, Real-time summer school L'École d'Été Temps Réel (ETR) (2017)

[20] Dürr, F., Nayak, N.G.: No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN). In: Proceedings of the 24th International Conference on Real-Time Networks and Systems, pp. 203–212 (2016). https://doi.org/10.1145/2997465.2997494

[21] Jean-Yves, L.B., Patrick, T.: Network calculus: a theory of deterministic queuing systems for the internet. Real-Time Syst. **51** (2001)

[22] Sinnen, O.: Task Scheduling for Parallel Systems vol. 60. John Wiley & Sons, USA (2007)

[23] Burke, E.K., Kendall, G. (eds.): Search Methodologies, 2nd edn. Springer, New York, NY (2014)

[24] Wandeler, E.: Modular Performance Analysis and Interface-based Design for Embedded Real-time Systems. Shaker, Germany (2006)

[25] Campelo, F., Aranha, C.: Lessons from the evolutionary computation bestiary. Artificial Life (2022)

[26] Kirkpatrick, S., Gelatt, C.D., Jr., Vecchi, M.P.: Optimization by simulated annealing. Science **220**, 671–680 (1983)

[27] Voß, S., Woodruff, D.L.: Optimization Software Class Libraries. Springer, Boston, MA (2002)

[28] Nayak, N.G., Dürr, F., Rothermel, K.: Incremental flow scheduling and routing in time-sensitive software-defined networks. IEEE Trans Industr Inform **14**(5) (2018)

[29] Mahfouzi, R., Aminifar, A., Samii, S., Rezine, A., Eles, P., Peng, Z.: Stability-aware integrated routing and scheduling for control applications in ethernet networks. In: Proc. DATE (2018)

[30] Pahlevan, M., Obermaisser, R.: Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks. In: Proc. ETFA (2018). https://doi.org/10.1109/ETFA.2018.8502515

[31] Pahlevan, M., Tabassam, N., Obermaisser, R.: Heuristic list scheduler for time triggered traffic in time sensitive networks. SIGBED Rev. **16**(1), 15–20 (2019)

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

[33] Berisa, A., Zhao, L., Craciunas, S.S., Ashjaei, M., Mubeen, S., Daneshtalab, M., Sjödin, M.: Avb-aware routing and scheduling for critical traffic in time-sensitive networks with preemption. In: Proc. RTNS. ACM, New York, NY, USA (2022)

[34] Falk, J., Dürr, F., Rothermel, K.: Exploring practical limitations of joint routing and scheduling for TSN with ILP. In: Proc. RTCSA (2018)

[35] Vlk, M., Hanzálek, Z., Tang, S.: Constraint programming approaches to joint routing and scheduling in time-sensitive networks. Computers &

Industrial Engineering **157**, 107317 (2021). https://doi.org/10.1016/j.cie.2021.107317

[36] Zhou, Y., Samii, S., Eles, P., Peng, Z.: Reliability-aware scheduling and routing for messages in time-sensitive networking. ACM Trans. Embed. Comput. Syst. **20**(5) (2021). https://doi.org/10.1145/3458768

[37] Zhou, Y., Samii, S., Eles, P., Peng, Z.: Asil-decomposition based routing and scheduling in safety-critical time-sensitive networking. In: Proc. RTAS (2021). https://doi.org/10.1109/RTAS52030.2021.00023

[38] Steiner, W.: An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks. In: Proc. RTSS. IEEE, USA (2010)

[39] Craciunas, S.S., Serna Oliver, R.: Combined task- and network-level scheduling for distributed time-triggered systems. Journal of Real-Time Systems **52**(2), 161–200 (2016)

[40] Zhao, L., Pop, P., Craciunas, S.S.: Worst-case latency analysis for IEEE 802.1Qbv time sensitive networks using network calculus. IEEE Access **6**, 41803–41815 (2018). https://doi.org/10.1109/ACCESS.2018.2858767

[41] Hellmanns, D., Glavackij, A., Falk, J., Duerr, F., Hummen, R., Kehrer, S.: Scaling TSN scheduling for factory automation networks. In: Proc. WFCS, pp. 1–8 (2020)

[42] Shalghum, K.M., Noordin, N.K., Sali, A., Hashim, F.: Network calculus-based latency for time-triggered traffic under flexible window-overlapping scheduling (FWOS) in a time-sensitive network (TSN). Applied Sciences **11**(9) (2021)

[43] Institute of Electrical and Electronics Engineers, Inc: 802.1BA—Audio Video Bridging (AVB) Systems. http://www:ieee802:org/1/pages/802:1ba:html. Accessed: 23.10.2020 (2011)

[44] Schmitt, J., Hurley, P., Hollick, M., Steinmetz, R.: Per-flow guarantees under class-based priority queueing. In: IEEE Global Telecommunications Conference, pp. 4169–4174 (2003)

[45] De Azua, J.A.R., Boyer, M.: Complete modelling of AVB in network calculus framework. In: Proc. RTNS, pp. 55–64 (2014)

[46] Diemer, J., Thiele, D., Ernst, R.: Formal worst-case timing analysis of ethernet topologies with strict-priority and AVB switching. In: Proc. SIES, pp. 1–10 (2012)

[47] Zhao, L., Pop, P., Li, Q., Chen, J., Xiong, H.: Timing analysis of rate-constrained traffic in TTEthernet using network calculus. Real-Time Systems **52**(2), 254–287 (2017)

[48] Zhao, L.X., Xiong, H.G., Zheng, Z., Li, Q.: Improving worst-case latency analysis for rate-constrained traffic in the Time-Triggered Ethernet network. IEEE Communications Letters **18**(11), 1927–1930 (2014)

[49] Boyer, M., Daigmorte, H., Navet, N., Migge, J.: Performance impact of the interactions between time-triggered and rate-constrained transmissions in TTEthernet. In: Proc. ERTS (2016)

[50] Steiner, W., Bauer, G., Hall, B., Paulitsch, M.: TTEthernet: Time-Triggered Ethernet. In: Obermaisser, R. (ed.) Time-Triggered Communication. CRC Press, USA (2011)

[51] Wandeler, E., Thiele, L.: Optimal TDMA time slot and cycle length allocation for hard real-time systems. In: Proc. ASP-DAC (2006)

[52] Khanh, D.D., Mifdaoui, A.: Timing Analysis of TDMA-based Networks using Network Calculus and Integer Linear Programming. In: Proc. MASCOTS, pp. 21–30 (2014)

[53] Heilmann, F., Fohler, G.: Size-based queuing: An approach to improve bandwidth utilization in TSN networks. SIGBED Rev. **16**(1), 9–14 (2019)

[54] Luteberget, B., Claessen, K., Johansen, C., Steffen, M.: SAT modulo discrete event simulation applied to railway design capacity analysis. Formal Methods Syst. Des. **57**(2), 211–245 (2021). https://doi.org/10.1007/s10703-021-00368-2

[55] Zhao, L., Pop, P., Steinhorst, S.: Quantitative performance comparison of various traffic shapers in time-sensitive networking. CoRR **abs/2103.13424** (2017) https://arxiv.org/abs/2103.13424. https://arxiv.org/abs/2103.13424

[56] Wandeler, E., Thiele, L.: Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`. Accessed: 23.10.2020 (2006). http://www.mpa.ethz.ch/Rtctoolbox

[57] Google: Google OR-Tools. https://developers.google.com/optimization (Accessed on Oct 2020)

[58] Gavrilut, V., Zarrin, B., Pop, P., Samii, S.: Fault-tolerant topology and routing synthesis for IEEE time-sensitive networking. In: Proc. RTNS. ACM, New York, NY, USA (2017)

[59] Falk, J., Hellmanns, D., Carabelli, B., Nayak, N., Dürr, F., Kehrer, S., Rothermel, K.: NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++. In: Proc. NetSys, pp. 1–8 (2019)

[60] Kramer, S., Ziegenbein, D., Hamann, A.: Real world automotive benchmarks for free. In: Proc. WATERS (2015)