Real-Time Traffic Guarantees in Heterogeneous Time-sensitive Networks

Mohammadreza Barzegaran
Technical University of Denmark
Kongens Lyngby, Denmark
mohba@dtu.dk

Niklas Reusch
Technical University of Denmark
Kongens Lyngby, Denmark
nikre@dtu.dk

Luxi Zhao
Beihang University
Beijing, China
zhaoluxi@buaa.edu.cn

Silviu S. Craciunas
TTTech Computertechnik AG
Vienna, Austria
silviu.craciunas@tttech.com

Paul Pop
Technical University of Denmark
Kongens Lyngby, Denmark
paupo@dtu.dk

ABSTRACT

Time-Sensitive Networks (TSN) enhance standard IEEE 802.1Q Ethernet devices with real-time and time-aware capabilities. The forwarding of time-critical frames is done according to a so-called Gate Control List (GCL) schedule via the timed-gate mechanism introduced in IEEE 802.1Qbv. Most TSN scheduling mechanisms impose that all devices in the network must have the TSN capabilities related to scheduled gates and time synchronization. However, this is often an unrealistic assumption since many distributed applications use heterogeneous TSN networks with legacy or off-the-shelf end systems that are unscheduled and/or unsynchronized.

We proposes a novel, more flexible TSN scheduling algorithm that intertwines a worst-case delay analysis within the scheduling synthesis step. Through this, we leverage the solution's optimality to support heterogeneous TSN networks featuring unscheduled and/or unsynchronized end-systems while still guaranteeing the timeliness of critical communication. We evaluate the performance of our approach using both synthetic and real-world use cases, comparing it with existing TSN scheduling mechanisms. Furthermore, we use OMNET++ to validate the generated GCL schedules.

CCS CONCEPTS

• **Computer systems organization → Dependable and fault-tolerant systems and networks**.

KEYWORDS

Time sensitive networking, Scheduled traffic, Network calculus.

ACM Reference Format:
Mohammadreza Barzegaran, Niklas Reusch, Luxi Zhao, Silviu S. Craciunas, and Paul Pop. 2022. Real-Time Traffic Guarantees in Heterogeneous Time-sensitive Networks. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems (RTNS '22), June 7–8, 2022, Paris, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3534879.3534921

RTNS '22, June 7–8, 2022, Paris, France

1 INTRODUCTION

Standardized communication protocols allowing safety-critical communication with real-time guarantees are becoming increasingly relevant in application domains beyond aerospace, e.g., in industrial automation and even in the automotive sector for advanced driver assistance functions or fully autonomous driving [1]. Time-Sensitive Networking (TSN) [24] amends the standard Ethernet protocol with real-time capabilities ranging from clock synchronization and frame preemption to redundancy management and schedule-based traffic shaping [9]. These novel mechanisms allow standard best-effort (BE) Ethernet traffic to coexist with isolated and guaranteed scheduled traffic (ST) within the same multi-hop switched Ethernet network. The main enablers of this coexistence are a network-wide clock synchronization protocol (802.1ASrev [25]) defining a global network time, known and bounded device latencies (e.g., switch forwarding delays), and a Time-Aware Shaper (TAS) mechanism [23] with a global communication schedule implemented in so-called Gate Control Lists (GCLs), facilitating ST traffic with bounded latency and jitter in isolation from BE communication. The TAS mechanism is implemented as a gate for each transmission queue that either allows or denies the sending of frames according to the configured GCL schedule.

Most approaches that guarantee real-time temporal properties of critical traffic (e.g., [9, 33, 37]) assume a homogeneous TSN network in which all devices have the time-aware shaper 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 [36], 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 [29]. Furthermore, converged IT/OT networks in, e.g., fog and edge use-cases [36], interconnection of TSN networks with, e.g., 5G domains [27], or multi-domain TSN networks with different sync mechanisms cannot readily communicate isochronous (fully periodic) traffic [6]. 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 [45]), 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 in this paper heterogeneous TSN networks where the end-systems are unscheduled and/or unsynchronized (i.e., they do not have the TSN capabilities related to 802.1Qbv and 802.1AS), leading to a sporadic arrival of critical traffic at the TSN-capable switches in the network. Classical schedule generation methods for GCLs enforce either a fully deterministic 0-jitter forwarding of critical frames using either exact SMT/ILP-based solvers [9] or heuristics [33], or a more flexible window-based approach that allows some (bounded) degree of interference between critical frames [37]. However, both methods require that end-systems send the respective critical frames in a scheduled and synchronized way that matches the forwarding schedule defined in the switches, thus requiring TSN capabilities on both end systems and switches. Other work, c.f. [20, 34], 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, furthermore, do not use safe formal verification methods like network calculus for the schedule creation.

In this paper, 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 [50] with the scheduling step in order to generate correct schedules where the end-to-end requirements of ST flows 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 (FWND). 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 flows 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 flow to check its schedulability, considering arbitrary arrival times of flows and the open GCL window placements.
- We propose a proxy function as an extension for the analysis in [50] and implement it in our problem formulation.
- We formulate and solve a window optimization problem that uses the proxy function and provides timing guarantees for real-time flows even in heterogeneous TSN networks.
- 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 with a review of related research, focusing on the existing scheduling mechanisms that we compare our work to, in Sect. 2. We introduce the system, network, and application models, as well as a description of the main TSN standards, in Sect. 3, and outline the problem formulation in Sect. 3.4. In Sect. 4, we present the novel scheduling mechanism and the optimization strategy based on the Constraint Programming (CP) for FWND followed by the comparison and evaluation results in Sect. 5. We conclude the paper in Sect. 6.

2 RELATED WORK

Scheduling homogeneous TSN networks in which all devices are scheduled and synchronized has been solved in various forms using heuristics [28, 30–32, 42] and optimal ILP- or SMT-based approaches [9, 13, 15, 37, 44, 54, 55]. The most relevant results for providing real-time communication properties in TSN networks, to which we compare our approach, have been presented in [9, 14, 33, 37] (summarized in Table 3).

Originally, the TSN scheduling problem was addressed in [9] for fully deterministic ST traffic temporal behavior and temporal isolation between ST and non-ST (e.g., AVB, BE) streams, similar to TTEthernet [7, 40]. 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 [33] 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 [37], which we call *Frame-to-Window-based*, the 0-jitter constraint of [9] 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 [37] 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 [14] 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 flows 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 [9, 33] or are uniquely associated with predefined subsets of frames [37]. However, the above property is a significant limitation. In many use cases, especially in the industrial and automotive

domains (c.f. [36]), the end-systems are usually off-the-shelf sensors, MCUs, industrial PCs, and edge devices that do not have TSN capabilities.

The work in [34] 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, [34] uses a less advanced analysis step (c.f. [49]) in the scheduling decisions and a more naive heuristic approach. The work in [21] 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 [20] 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 [38], 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, [38] cannot guarantee the schedulability of traffic arriving from unscheduled or unsynchronized end-systems.

Classical approaches like strict priority (SP) and AVB [22] 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 [11, 35] or Compositional Performance Analysis (CPA) [12] are used. In [4, 51, 53], the rate-constrained (RC) flows of TTEthernet [26, 41] are analyzed using network calculus. Other works, such as [10, 47], study the response-time analysis for TDMA-based networks under the strict priority (SP) and weighted round-robin (WRR) queuing policies. Zhao et al. [50] 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.

In [43], 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 [9]. Another hardware adaptation for TSN bridges, which has been proposed by Heilmann et al. [19] is to increase the number of non-critical queues in order to improve the bandwidth utilization without impacting the guarantees for critical messages.

3 SYSTEM MODEL

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

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 flow $f \in \mathcal{F}$
$f.P, f.D$	Priority, and deadline of a flow $f \in \mathcal{F}$
$f.r$	Route for a flow $f \in \mathcal{F}$

3.1 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 [9]. 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.

3.2 Switch Model

Fig. 1 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. Within a queue, frames are transmitted in first-in-first-out (FIFO) order. Similar to [9], a subset ($p.Q_{ST}$) of the queues are 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

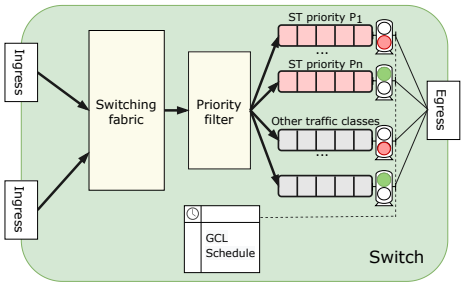


Figure 1: TSN Switch Internals

a closed gate will not allow transmission, even if the queue is not empty. When multiple gates are open simultaneously, the highest priority queue has preference, 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 [26]) acts on the level of traffic-classes instead of on an individual frame level [8]. Hence, an imperfect time synchronization, frame loss, ingress policing (c.f. [9]), 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 Fig. 2, 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 Fig. 2. 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

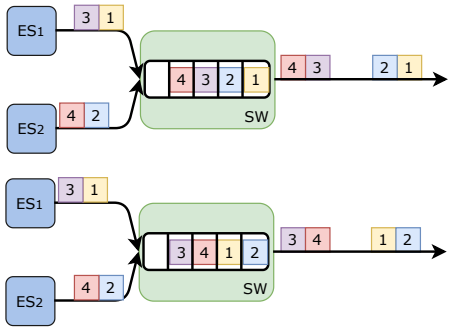


Figure 2: Queue states (inspired by [43]).

(e.g. [9, 14, 33, 37]). We refer the reader to [9] 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 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.

3.3 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 [9]. Communication requirements of ST traffic itself are modeled with the concept of flows (also called streams), representing a communication from one sender (talker) to one or multiple receivers (listeners). We define the set of ST flows in the network as $\mathcal{F}$. A flow $f \in \mathcal{F}$ is expressed as the tuple $\langle l, T, P, D \rangle_f$, including the frame size, the flow period in the source ES, the priority of the flow, and the required deadline representing the upper bound on the end-to-end delay of the flow.

The route for each flow is statically defined as an ordered sequence of directed links,e.g., a flow $f \in F$ 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 flow with additional constraints between them on the common path. In this way, a multicast flow consumes the bandwidth as all sender-receiver pairs of the stand-alone flow do. Furthermore, it is more likely that the multicast flow misses deadline for some sender-receiver pairs.

3.4 Problem Formulation

Given (1) a set of flows $\mathcal{F}$ 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 flows are satisfied and the overall bandwidth utilization (c.f. Sect. 4.1) 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 flows 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. To better understand the importance of determining optimized windows we refer the reader to the motivational example described in Section 4 of the technical report version of this work [2].

4 OPTIMIZATION STRATEGY

The problem presented in the previous section is intractable. The decision problem associated with the scheduling problem has been proved to be (NP)-complete in the strong sense [39]. Hence, we

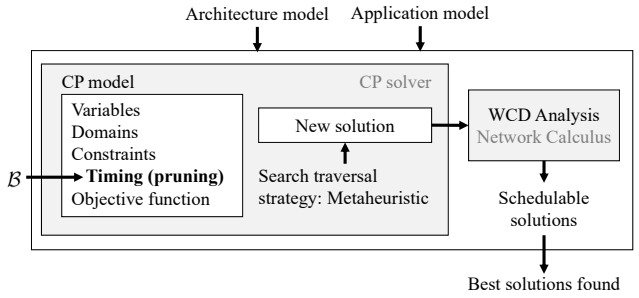


Figure 3: Overview of our CPWO optimization strategy

use an optimization strategy, called Constraint Programming-based Window Optimization (CPWO), which is more suitable to find good solutions in a reasonable time.

CPWO takes as the inputs the architecture and application models and outputs a set of the best solutions found during search (see Fig. 3). We use Constraint Programming (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 Sect. 4.2, the constraints in Sect. 4.3 and the objective function in Sect. 4.1. 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 [50] to determine the WCDs, see the "WCD Analysis" box in Fig. 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.

Although CP can perform an exhaustive search and find the optimal solution, this is infeasible for large networks. Instead, CPWO employs two strategies to speed up the search to find optimized solutions in a reasonable time, at the expense of optimality.

(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 [5] inspired from [3].

(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 Fig. 3), explained in Sect. 4.4.

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 of frames of the same stream. A parameter $\mathcal{B}$ is used to adjust the number of frames in the backlog, 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 Sect. 4.4 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.

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 flows assigned to q
$\mathcal{GB}(q)$	Maximum transmission time of ST frames competing in q
$\mathcal{R}(q)$	All flows assigned to the queue q
$\mathcal{X}(q)$	All flows arriving from a switch and assigned to the queue q

4.1 Objective Function

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

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

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. Note that solutions found by a CP solver are guaranteed to satisfy the constraints defined in Sect. 4.3. In addition, the schedulability is checked with the NC-based WCD Analysis [50].

4.2 Variables

The model variables are the offset, length, and period of each window, see Sect. 3.2. 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 flow 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 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 flows 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.

4.3 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) windows' bandwidth is not exceeded. The last 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.

(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 flows:

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

where $\mathcal{F}(q)$ is the set of flows 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.

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

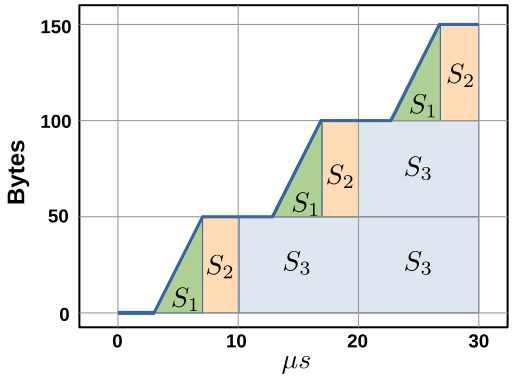


Figure 4: Example capacity for a window.

(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):

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

4.4 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. A too aggressive pruning may eliminate

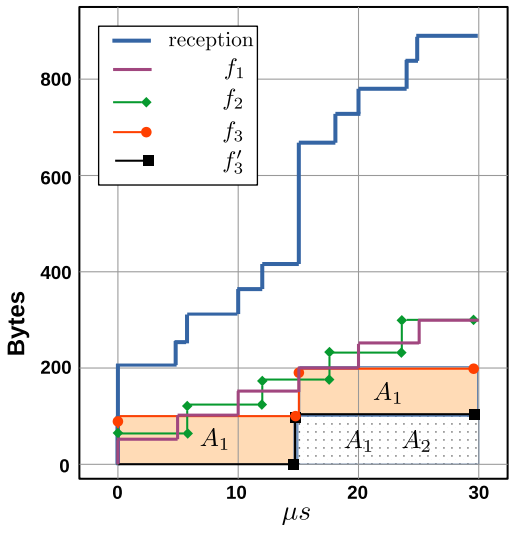


Figure 5: Example capacity and transmission demand for a window.

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 first-order formulation of CP. However, the NC formulation from [50] has inspired us in defining the CP timing constraints. The *Timing Constraint* is defined in Eq. (8) 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 high chances to be schedulable:

$$\forall p \in P, \forall q \in p.Q : \mathcal{W}_D \leq \mathcal{W}_C. \tag{8}$$

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 [46] 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 Fig. 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. (9), 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 $\mathcal{W}_C$ value of the example in Fig. 4 is 2,250 $Bytes \times \mu s$, where the S terms are shown.

$$\begin{aligned}
&\forall p \in P, \forall q \in p.Q : \\
&I = \frac{\mathcal{K}(p)}{q.T}, \quad J = \frac{(q.w - \mathcal{GB}(q)) \times \mathcal{Y}([v_a, v_b])}{[v_a, v_b].C}, \\
&S_1 = I \times \frac{q.w \times J}{2}, \quad S_2 = I \times (q.T - q.w. - q.\phi) \times J, \\
&S_3 = \frac{I \times (I-1)}{2} \times q.T \times J, \quad \mathcal{W}_C = S_1 + S_2 + S_3
\end{aligned} \tag{9}$$

Secondly, we calculate the transmission demand $\mathcal{W}_D$ using Eq. (10), where $\mathcal{R}(q)$ captures all the flows 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 flows pass via switches and may change their arrival patterns [50]. In our case, we have made the following simplifying assumptions to be able to express the "transmission demand" in CP. We assume that all flows 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 flows have a bursty behavior when they reach our window. To compensate for this, we consider that those flows that arrive from a switch may be bursty and thus have a backlog $\mathcal{B}$ of frames that have accumulated; flows that arrive from ESs do not accumulate a backlog. Fig. 5 shows three flows, f_1 to f_3, and only f_3 arrives from a switch and hence will have a backlog of frames captured by the flow 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 flow 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 flow, bursts of frames coming from SWs and an additional frame for each flow 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 flow forwarded from another SW. Note that as explained in the overview at the beginning of Sect. 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.

We give an example in Fig. 5 where the flows $f_1 < 50, 5, 0, 5 >$ and $f_2 < 60, 6, 0, 6 >$ have been received from an ES and the flow $f_3 < 100, 15, 0, 15 >$ has been received from a SW. For the flow f_3 forwarded from a previous switch, we consider that one instance of the flow (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 flows in the current node. The reception curve in Fig. 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 flows and backlogs of the flows 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. The term A_1 calculates the area under the curve for the accumulated data of all flows assigned to the queue q captured by $\mathcal{R}(q)$, in a hyperperiod. Any frames of all flows $\mathcal{R}(q)$ have arrived at the beginning of their period. The term A_2 calculates the area under the curve for the accumulated backlog data of the flows arrived from a switch captured by $\mathcal{X}(q)$. The backlog data of the flows $\mathcal{X}(q)$ are delayed for a period and controlled by $\mathcal{B}$, which captures the number of backlogs. The function $\mathcal{W}_D$ returns 16,650 $Bytes \times \mu s$ in our example, see also Fig. 5 for the values of the terms A_1 and A_2.

$$\begin{aligned}
&\forall p \in P, \forall q \in p.Q, \forall f \in \mathcal{R}(q), \forall f' \in \mathcal{X}(q) : \\
&I = \frac{\mathcal{K}(p)}{f.T}, \quad I' = \frac{\mathcal{K}(p)}{f'.T} \\
&A_1 = \frac{I \times (I+1)}{2} \times f.T \times f.l, \\
&A_2 = \frac{I' \times (I' + 1 - 2 \times \mathcal{B})}{2} \times f'.T \times f'.l, \\
&\mathcal{W}_D = A_1 + A_2
\end{aligned} \tag{10}$$

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 [9, 37]) while other constraints (Eq. (6), (7)) are used to limit the placement of GCL windows and are not related to the fundamental schedule correctness, but are used to improve the runtime of the search by limiting the window placements. 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 main scheduling step.

5 EVALUATION

In this section, we give details of our setup and test cases in Sect. 5.1 and evaluate our windows optimization solution CPWO for our Flexible Window-based approach (*FWND*) on synthetic and real-world test cases on Sect. 5.2 and Sect. 5.3, respectively. We also compare the CPWO results for FWND with the related work and validate the generated GCLs with OMNET++ in Sect. 5.4.

The related work on ST scheduling using 802.1Qbv consists of: (i) zero-jitter GCL (0GCL) [9, 33], (ii) Frame-to-Window-based GCL (FGCL) [37], and (iii) Window-based GCL (WND) [34].

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 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. The *Link Constraint* specifies that frames routed on the same physical link cannot overlap in the time domain, also named "Ordered Windows Constraint" in [37] and the *Flow Transmission Constraint* specifies that the propagation of frames of a flow follows the sequential order along the path of the flow, also named "Stream Constraint" in [37]. We refer the reader to the supplementary material included in the technical report version of this work [2] for a reiteration of the relevant scheduling constraints for creating correct TSN schedules when using frame- and window-based methods.

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 flows and scheduled devices from previous work, thereby reducing the complexity of the schedule synthesis.

5.1 Test Cases and Setup

We implemented our *FWND* approach using the Java version of the Google OR-Tools [18] and the Java kernel of the RTC toolbox [48]. The tests were run on an i9 CPU (3.6 GHz) with 32 GB of memory. 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.

We have generated 15 synthetic test cases that have different network topologies (three test cases for each topology in Fig. 6) inspired by industrial and automotive application requirements. Similar to [52], 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 flows 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 for 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 flows are shown in columns 3 to 6.

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 flows 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 flows, 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 for 1000 Mbps. More information can be found in the corresponding columns in Table 6. The test cases use the same topologies as in [17] and [50], and we consider that all flows belong to the ST traffic class.

5.2 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 [9] and [37], respectively. The *WND* method has been implemented with the heuristic presented in [34], but instead of using the WCD analysis from [49], we extend it to use the analysis from [50] 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 flows 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. Please note that *NA* in the results means an out of memory error and Ω values are multiplied by 1000.

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

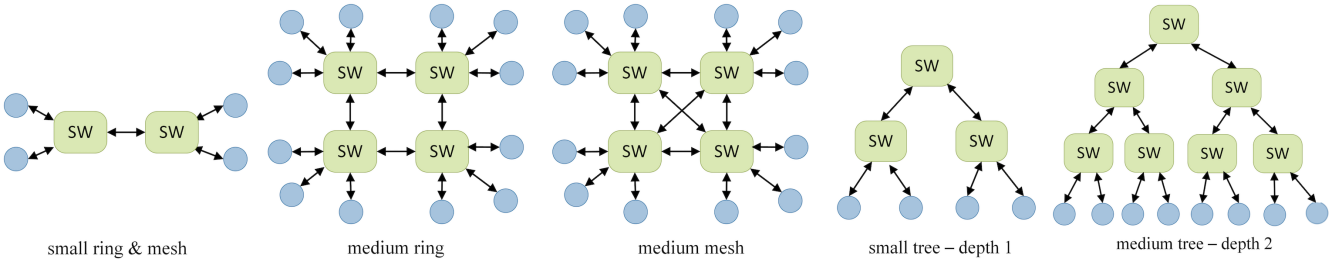


Figure 6: Network topologies used in the test cases [52]

Table 3: Scheduling Approaches in TSN

Requirements	0GCL [9][33]	FGCL [37]	WND [34]	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	Yes	Yes	No	No
Bandwidth Constraint	No	No	No	Yes
Flow Transmission Constraint	Yes	Yes	No	No
Frame-to-Window Assignment	No	Yes	No	No
Window Size Constraint	No	Yes	Yes	Yes
Flow/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

best results in terms of bandwidth usage (Ω) and WCDs, noting that *0GCL* may further reduce the WCDs compared to *FGCL*.

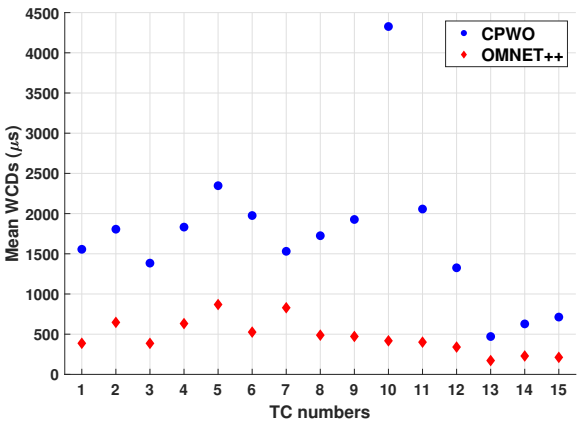
The only other approach that has similar assumptions to our *FWND* is *WND* from [34]. As we can see from Table 5, in comparison to *WND*, our CPWO solution can slightly reduce the bandwidth usage. The most important result is that CPWO significantly reduces the WCDs compared to *WND*, with an average 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 [34]. Also, when comparing the WCDs obtained by our *FWND* 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 *FWND*, *FWND* reduces the runtime significantly. The reason for reduced runtime with *FWND* is that the CP model has to determine values for fewer variables compared to *0GCL*. As shown in the table, the runtime increases with the size of test cases in all four solutions and grows exponentially since the problem is proved to be NP-complete (c.f. [9, 15]). *FWND* introduces 3 variables (offset, period and length) for each window (queue) in the network, whereas *0GCL* introduces a variable for each frame of each flow. The number of variables in the *0GCL* model depends on the hyperperiod, the number of flows, and the flow periods, whereas the number of variables in the *FWND* model depends on the number of switches and used queues.

Table 4: Details of the synthetic test cases

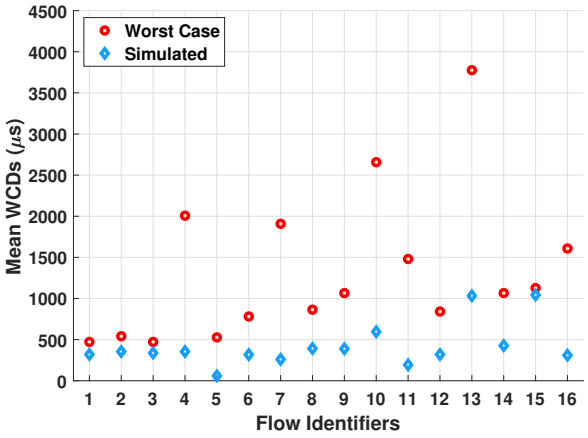
No.	Network Topology	Total No. of SWs	Total No. of ESs	Total No. of Flows	Hyperperiod (μs)
1	SRM	2	3	9	15,000
2	SRM	3	3	11	70,000
3	SRM	3	4	15	70,000
4	MR	4	6	15	30,000
5	MR	4	8	21	210,000
6	MR	5	11	27	210,000
7	MM	4	5	13	15,000
8	MM	6	12	30	210,000
9	MM	7	13	35	210,000
10	ST	3	4	7	15,000
11	ST	3	6	12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

Table 5: Evaluation results on synthetic test cases

No.	Ω for 0GCL	Ω for FGCL	Ω for WND	Ω for FWND	Mean worst-case e2e-delay for 0GCL (μs)	Mean worst-case e2e-delay for FGCL(μs)	Mean worst-case e2e-delay for WND (μs)	Mean worst-case e2e-delay for FWND (μs)	Mean Runtime for 0GCL (ms)	Mean Runtime for FWND (ms)
1	35	35	614	510	192	126	1838	1556	215	8
2	25	22	640	528	246	151	2461	1806	895	12
3	15	15	549	495	175	486	1964	1384	1518	22
4	13	13	330	285	160	776	2925	1832	525	16
5	14	NA	295	285	131	NA	2838	2347	5187	16
6	13	NA	275	205	129	NA	2953	1976	6291	17
7	12	12	238	204	125	764	2913	1561	1152	35
8	13	NA	238	202	114	NA	2878	1725	7603	36
9	12	NA	217	191	122	NA	3074	1927	9171	56
10	8	8	329	265	136	2284	4397	4327	2611	165
11	10	10	381	302	159	984	3047	2057	2840	231
12	11	NA	516	321	187	NA	2543	1326	4650	260
13	10	10	401	302	101	561	2529	471	978	130
14	9	9	611	402	120	785	2254	628	1256	162
15	9	NA	544	413	114	NA	2680	713	6116	163



(a) Mean WCDs vs. simulated delays for FWND



(b) Mean WCDs comparison for the flows of TC12

Figure 7: WCD vs. simulated delays

5.3 Evaluation on realistic test 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 flows in both test cases. Note that once all flows 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 of runtime, the CEV test case takes longer since it has 864 variables, whereas GM has only 102 variables in the CP models.

Table 6: CPWO for FWND results on realistic test cases

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

5.4 Evaluating the solutions with OMNET++

We have used the OMNET++ simulator with the TSN NeSTiNg extension [16] to validate the generated GCLs. Thus, 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. The mean WCDs of CPWO for the synthetic test cases and the worst-case latency observed during multiple OMNET++ simulations (with the windows from CPWO) are depicted in Fig. 7a. As expected, the latency values reported by OMNET++ are smaller than the WCDs, as reported by the WCD Analysis from [50]. 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

Table 7: Scalability evaluation of CPWO

No.	Total No. of Flows	Total No. of ESs	Total No. of SWs	Mean WCDs μs	Largest deadline μs	Ω (×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

(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 *FWND* 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 Fig. 7a and Fig. 7b. We also show in Fig. 7b the simulated delays and WCDs for all flows of TC12. All the flows 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 [50].

5.5 Scalability Evaluation

We have investigated the scalability of CPWO on 6 larger test cases (TC1 to TC6) inspired from industrial applications, that have up to 120 devices (75 ESs and 45 SWs) and 500 flows. The results and the details of the test cases are presented in Table 7, where columns 2, 3, and 4 show the number of flows, end-systems, and switches, respectively. Columns 5, 6, and 7 show the mean WCD of flows in μs, the largest deadline of all flows in μs, and the objective value Ω, related to bandwidth, see Eq. (1). Although CPWO was able to generate schedulable solutions in all cases, CPWO can not guarantee finding solutions for larger test cases with the problem being NP-complete. Furthermore, CPWO can optimize 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.

6 CONCLUSIONS

We have presented a novel, more flexible heuristic schedule synthesis approach for TSN networks, which decouples the frame scheduling from the generation of time-aware shaper (TAS) windows. Our method eliminates the often-unrealistic constraint that end systems are scheduled and synchronized (i.e., they have TSN capabilities) required by previous methods to provide real-time guarantees for critical traffic in IEEE 802.1Qbv TSN networks. Our approach intertwines an existing worst-case delay analysis method with a CP-solver into a novel and scalable heuristic approach that uses a Tabu Search metaheuristic search strategy in the CP-solver. Furthermore, to improve scalability, we have proposed a novel *proxy function* which can be parametrized to trade-off runtime performance for search-space pruning in the CP-model. The tuning of the proxy function's parameters is left for the future work. We evaluated our approach using synthetic and real-world test cases, comparing it with existing mechanisms and validated the generated schedules using OMNET++.

REFERENCES

[1] Mohammad Ashjaei, Lucia Lo Bello, Masoud Daneshtalab, Gaetano Patti, Sergio Saponara, and Saad Mubeen. 2021. Time-Sensitive Networking in automotive embedded systems: State of the art and research opportunities. *JSA* 117 (2021), 102137. https://doi.org/10.1016/j.sysarc.2021.102137

[2] Mohammadreza Barzegaran, Niklas Reusch, Luxi Zhao, Silviu S. Craciunas, and Paul Pop. 2021. Real-Time Guarantees for Critical Traffic in IEEE 802.1Qbv TSN Networks with Unscheduled and Unsynchronized End-Systems. *CoRR* abs/2105.01641 (2021). https://arxiv.org/pdf/2105.01641.

[3] Mohammadreza Barzegaran, Bahram Zarrin, and Paul Pop. 2020. Quality-of-control-aware scheduling of communication in TSN-based fog computing platforms using constraint programming. In *2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[4] Marc Boyer, Hugo Daigmorte, N. Navet, and Jorn Migge. 2016. Performance impact of the interactions between time-triggered and rate-constrained transmissions in TTEthernet. In *Proc. ERTS*.

[5] Edmund K. Burke and Graham Kendall. 2005. *Search methodologies*. Springer.

[6] Martin Böhm and Diederich Wermser. 2021. Multi-Domain Time-Sensitive Networks—Control Plane Mechanisms for Dynamic Inter-Domain Stream Configuration. *Electronics* 10, 20 (2021). https://doi.org/10.3390/electronics10202477

[7] Silviu S. Craciunas and Ramon Serna Oliver. 2016. Combined Task- and Network-level Scheduling for Distributed Time-Triggered Systems. *Journal of Real-Time Systems* 52, 2 (2016), 161–200.

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

[9] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelik, and Wilfried Steiner. 2016. Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks. In *Proc. RTNS*. 183–192.

[10] Dinh-Khanh Dang and Ahlem Mifdaoui. 2014. Timing Analysis of TDMA-based Networks using Network Calculus and Integer Linear Programming. In *Proc. MASCOTS*. 21–30.

[11] Joan Adrià Ruiz De Azua and Marc Boyer. 2014. Complete Modelling of AVB in Network Calculus Framework. In *Proc. RTNS*. 55–64.

[12] Jonas Diemer, Daniel Thiele, and Rolf Ernst. 2012. Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching. In *Proc. SIES*. 1–10.

[13] Aellison Cassimiro T. dos Santos, Ben Schneider, and Vivek Nigam. 2019. TSNSCHED: Automated Schedule Generation for Time Sensitive Networking. In *Proc. FMCAD*. https://doi.org/10.23919/FMCAD.2019.8894249

[14] Frank Dürr and Naresh Ganesh Nayak. 2016. No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN). In *Proc. RTNS*. ACM.

[15] Jonathan Falk, Frank Dürr, and Kurt Rothermel. 2018. Exploring Practical Limitations of Joint Routing and Scheduling for TSN with ILP. In *Proc. RTCSA*.

[16] Jonathan Falk, David Hellmanns, Ben Carabelli, Naresh Nayak, Frank Dürr, Stephan Kehrer, and Kurt Rothermel. 2019. NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++. In *Proc. NetSys*. 1–8.

[17] Voica Gavrilut, Bahram Zarrin, Paul Pop, and Soheil Samii. 2017. Fault-Tolerant Topology and Routing Synthesis for IEEE Time-Sensitive Networking. In *Proc. RTNS*. ACM.

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

[19] Florian Heilmann and Gerhard Fohler. 2019. Size-Based Queuing: An Approach to Improve Bandwidth Utilization in TSN Networks. *SIGBED Rev.* 16, 1 (2019), 9–14.

[20] David Hellmanns, Jonathan Falk, Alexander Glavackij, René Hummen, Stephan Kehrer, and Frank ü. 2020. On the Performance of Stream-based, Class-based Time-aware Shaping and Frame Preemption in TSN. In *Proc. ICIT*. 298–303.

[21] D. Hellmanns, A. Glavackij, J. Falk, F. Duerr, R. Hummen, and S. Kehrer. 2020. Scaling TSN Scheduling for Factory Automation Networks. In *Proc. WFCS*. 1–8.

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

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

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

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

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

[27] Ana Larrañaga, M. Carmen Lucas-Estañ, Imanol Martinez, Iñaki Val, and Javier Gozalvez. 2020. Analysis of 5G-TSN Integration to Support Industry 4.0. In *Proc.*

ETFA. https://doi.org/10.1109/ETFA46521.2020.9212141
[28] Rouhollah Mahfouzi, Amir Aminifar, Soheil Samii, Ahmed Rezine, Petru Eles, and Zebo Peng. 2018. Stability-aware integrated routing and scheduling for control applications in Ethernet networks. In *Proc. DATE*.
[29] Daniel Bujosa Mateu, Mohammad Ashjaei, Alessandro V. Papadopoulos, Julian Proenza, and Thomas Nolte. 2021. LETRA: Mapping Legacy Ethernet-Based Traffic into TSN Traffic Classes. In *Proc. ETFA*. https://doi.org/10.1109/ETFA45728.2021.9613637
[30] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. 2018. Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks. *IEEE Trans Industr Inform* 14, 5 (2018).
[31] Maryam Pahlevan and Roman Obermaisser. 2018. Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks. In *Proc. ETFA*. https://doi.org/10.1109/ETFA.2018.8502515
[32] Maryam Pahlevan, Nadra Tabassam, and Roman Obermaisser. 2019. Heuristic List Scheduler for Time Triggered Traffic in Time Sensitive Networks. *SIGBED Rev.* 16, 1 (2019), 15–20.
[33] Paul Pop, Michael L. Raagaard, Silviu S. Craciunas, and Wilfried Steiner. 2016. Design Optimization of Cyber-Physical Distributed Systems using IEEE Time-Sensitive Networks (TSN). *IET Cyber-Physical Systems: Theory and Applications* 1, 1 (2016), 86–94.
[34] Niklas Reusch, Luxi Zhao, Silviu S. Craciunas, and Paul Pop. 2020. Window-Based Schedule Synthesis for Industrial IEEE 802.1Qbv TSN Networks. In *Proc. WFCS*.
[35] Jens Schmitt, Paul Hurley, Matthias Hollick, and Ralf Steinmetz. 2003. Per-flow guarantees under class-based priority queueing. In *IEEE Global Telecommunications Conference*. 4169–4174.
[36] Sebastian Schriegel, Thomas Kobzan, and Jürgen Jasperneite. 2018. Investigation on a distributed SDN control plane architecture for heterogeneous time sensitive networks. In *Proc. WFCS*. https://doi.org/10.1109/WFCS.2018.8402356
[37] Ramon Serna Oliver, Silviu S. Craciunas, and Wilfried Steiner. 2018. IEEE 802.1Qbv Gate Control List Synthesis using Array Theory Encoding. In *Proc. RTAS*.
[38] Khaled M. Shalghum, Nor Kamariah Noordin, Aduwati Sali, and Fazirulhisyam Hashim. 2021. 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).
[39] Oliver Sinnen. 2007. *Task scheduling for parallel systems*. Vol. 60. John Wiley & Sons.
[40] Wilfried Steiner. 2010. An Evaluation of SMT-based Schedule Synthesis For Time-Triggered Multi-Hop Networks. In *Proc. RTSS*. IEEE.
[41] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch. 2011. TTEthernet: Time-Triggered Ethernet. In *Time-Triggered Communication*, Roman Obermaisser (Ed.). CRC Press.
[42] Marek Vlk, Kateřina Brejchová, Zdeněk Hanzálek, and Siyu Tang. 2022. Large-scale periodic scheduling in time-sensitive networks. *Computers & Operations Research* 137 (2022), 105512. https://doi.org/10.1016/j.cor.2021.105512
[43] Marek Vlk, Zdeněk Hanzálek, Kateřina Brejchová, Siyu Tang, Sushmit Bhattacharjee, and Songwei Fu. 2020. Enhancing Schedulability and Throughput of Time-Triggered Traffic in IEEE 802.1Qbv Time-Sensitive Networks. *IEEE Transactions on Communications* 68, 11 (2020), 7023–7038.
[44] Marek Vlk, Zdeněk Hanzálek, and Siyu Tang. 2021. Constraint programming approaches to joint routing and scheduling in time-sensitive networks. *Computers & Industrial Engineering* 157 (2021), 107317. https://doi.org/10.1016/j.cie.2021.107317
[45] Christian von Arnim, Mihai Drăgan, Florian Frick, Armin Lechler, Oliver Riedel, and Alexander Verl. 2020. TSN-based Converged Industrial Networks: Evolutionary Steps and Migration Paths. In *Proc. ETFA*, Vol. 1. 294–301. https://doi.org/10.1109/ETFA46521.2020.9212057
[46] Ernesto Wandeler. 2006. *Modular performance analysis and interface-based design for embedded real-time systems*. Shaker.
[47] Ernesto Wandeler and Lothar Thiele. 2006. Optimal TDMA time slot and cycle length allocation for hard real-time systems. In *Proc. ASP-DAC*.
[48] Ernesto Wandeler and Lothar Thiele. 2006. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`. http://www.mpa.ethz.ch/Rtctoolbox Accessed: 23.10.2020.
[49] Luxi Zhao, Paul Pop, and Silviu S. Craciunas. 2018. Worst-Case Latency Analysis for IEEE 802.1Qbv Time Sensitive Networks Using Network Calculus. *IEEE Access* 6 (2018), 41803–41815. https://doi.org/10.1109/ACCESS.2018.2858767
[50] Luxi Zhao, Paul Pop, Zijie Gong, and Bingwu Fang. 2021. Improving Latency Analysis for Flexible Window-Based GCL Scheduling in TSN Networks by Integration of Consecutive Nodes Offsets. *IEEE Internet of Things Journal* 8, 7 (2021), 5574–5584. https://doi.org/10.1109/JIOT.2020.3031932
[51] Luxi Zhao, Paul Pop, Qiao Li, Junyan Chen, and Huagang Xiong. 2017. Timing analysis of rate-constrained traffic in TTEthernet using network calculus. *Real-Time Systems* 52, 2 (2017), 254–287.
[52] Luxi Zhao, Paul Pop, and Sebastian Steinhorst. 2017. Quantitative Performance Comparison of Various Traffic Shapers in Time-Sensitive Networking. *CoRR* abs/2103.13424 (2017). arXiv:2103.13424 https://arxiv.org/abs/2103.13424.
[53] LuXi Zhao, Huagang Xiong, Zhong Zheng, and Qiao Li. 2014. Improving worst-case latency analysis for rate-constrained traffic in the Time-Triggered Ethernet network. *IEEE Communications Letters* 18, 11 (2014), 1927–1930.
[54] Yuanbin Zhou, Soheil Samii, Petru Eles, and Zebo Peng. 2021. ASIL-Decomposition Based Routing and Scheduling in Safety-Critical Time-Sensitive Networking. In *Proc. RTAS*. https://doi.org/10.1109/RTAS52030.2021.00023
[55] Yuanbin Zhou, Soheil Samii, Petru Eles, and Zebo Peng. 2021. Reliability-Aware Scheduling and Routing for Messages in Time-Sensitive Networking. *ACM Trans. Embed. Comput. Syst.* 20, 5, Article 41 (2021), 24 pages. https://doi.org/10.1145/3458768