IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 July 2024, accepted 5 August 2024, date of publication 8 August 2024, date of current version 20 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3440828

RESEARCH ARTICLE

Fast IEEE802.1Qbv Gate Scheduling Through Integer Linear Programming

ALITZEL GALILEA TORRES-MACÍAS[1,2], (Graduate Student Member, IEEE), JUAN SEGARRA FLOR[2,3], JOSÉ LUIS BRIZ VELASCO[2,3], ANTONIO RAMÍREZ-TREVIÑO[1], (Member, IEEE), AND HÉCTOR BLANCO-ALCAINE[4]

[1]CINVESTAV Unidad Guadalajara, Zapopan 45019, Mexico
[2]Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, 50009 Zaragoza, Spain
[3]I3A Research Institute, Universidad de Zaragoza, 50009 Zaragoza, Spain
[4]Intel Deutschland GmbH, 85579 Neubiberg, Germany

Corresponding author: Alitzel Galilea Torres-Macías (alitzel.torres@cinvestav.mx)

This work was supported in part by the Ministerio de Ciencia, Innovación y Universidades (MCIN)/Agencia Española de Investigación (AEI)/10.13039/501100011033 under Grant PID2022-136454NB-C22; in part by the Department of Science, University and Knowledge Society, Government of Aragón, Spain, under Grant T58_23R; and in part by the CINVESTAV, Mexico.

ABSTRACT Time-Sensitive Networking (TSN) is an in-development technology that enables predictability over Ethernet or wireless networks. Network interfaces compliant with the IEEE 802.1Qbv standard provide different queues/gates on each bridge egress port. In this way, a global network schedule can be set by defining the opening and closing times (Gate Control List, GCL) for each gate. In this paper, we propose a new method to schedule GCLs by dividing the problem into several subproblems. We use Weighted Fair Queuing (WFQ) to set the ordering of frames, and then generate an Integer Linear Programming (ILP) model to optimize the TSN scenario. Next, we assign gates to the scheduled windows, trying to ensure frame isolation whenever possible. Our results show that we can schedule GCLs around 2 times faster than previous studies and up to 5.5 orders of magnitude faster if we choose to obtain any valid solution instead of the optimal one. In addition, we are able to schedule systems with utilization up to 85%, whereas previous papers reach 65%. Moreover, our approach does not need to predefine the number of windows or gates, as required by other methods.

INDEX TERMS 802.1Qbv, delay, GCL, ILP, jitter, real-time, scheduling, TAS, time-sensitive networking, TSN.

I. INTRODUCTION

Time-Sensitive Networking (TSN) is a set of technologies to guarantee precise synchronization and timeliness across Ethernet and wireless networks. It is especially valuable for real-time control applications, across industrial, energy, or transportation domains, as it allows the deployment of ultra-low latency traffic relying on standard IEEE mechanisms. TSN provides traffic shaping and scheduling mechanisms to guarantee the Quality of Service (QoS) for different traffic types in a converged network.

The associate editor coordinating the review of this manuscript and approving it for publication was Shih-Wei Lin.

The Time-Aware Shaper (TAS), defined in IEEE 802.1Qbv, allows the scheduling of egress traffic for an end-station or bridge in different periodic slots. In order to accomplish that, the frames for each traffic class are queued in their egress queue. The transmission for each queue is governed by the so-called gates. The opening and closing of each gate follow a preconfigured periodic schedule known as the Gate Control List (GCL). When a gate is opened, frames from its respective queue are forwarded for transmission in first-in first-out (FIFO) order, resulting in time-predictable transmissions [1]. Fig. 1 illustrates an IEEE 802.1Qbv TAS, which holds up to eight different queues. The configuration of the TAS involves synchronization and priority assignment,

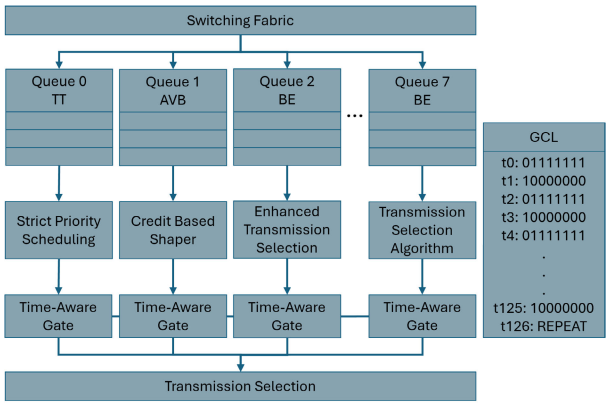

FIGURE 1. The TAS mechanism in a TSN bridge.

meeting real-time applications requirements such as ultra-low latency or jitter. The practical result of a TSN scheduler is the GCL entries to be deployed over the TAS of the network nodes.

Time-aware traffic in TSN consists of periodic streams with tight timing constraints regarding latency and jitter. Scheduling time-aware traffic and generating the GCL entries for the TAS is a pivotal aspect of TSN. However, computing the schedules that coordinate the transmission times for all frames along the network such that their real-time requirements are met is a challenging task [2]. Ensuring frame isolation can be all the more complex. Frame isolation guarantees that frames from different streams never share the same queue at the same time. Without it, the timing of frames from a single stream can be altered by frames from other streams, especially if those frames are larger, shorter than expected, or lost altogether. Thus, the TSN scheduling problem and its variations (such as joint routing, GCL synthesis, queueing, multicasting, integration of other traffic types, and dynamic reconfiguration) are an active area of research and development.

Optimal approaches face the GCL synthesis problem relying on Satisfiability Modulo Theories (SMT) [3], [4], which require rather long analysis times. Other approaches leverage heuristics to get valid schedules faster [5]. In this study, we propose to use a mixed heuristic-optimal approach: we divide the GCL synthesis problem into several sub-problems in order to reduce its complexity and yet obtain solutions as good as pure optimal approaches.

First, we calculate heuristically the order in which frames should be transmitted, based on a Weighted Fair Queuing (WFQ) policy [6], [7]. Then, we optimize the GCL synthesis complying with the precalculated frame transmission order, resorting to a Mixed Integer Linear Programming (MILP) solver, which is faster than an SMT solver as a rule. Also, our model automatically calculates parameters such as the maximum number of windows (open-close periods) in the GCLs, which, with other methods, must be known in advance. Finally, we allocate gates to windows to obtain the resulting schedule. In this last step, we ensure frame isolation whenever possible. Our approach inherently considers best-effort (BE) traffic in the network. We also allow multicast (multiple destinations over multiple paths), merge frames (frames go over different paths and then join together), and end-to-end delays longer than the stream period.

The contributions of this work are as follows:

1) A novel model for TSN scheduling problems with heuristics and Mixed Integer Linear Programming (MILP).
2) Outperforming similar methods [4] by a factor of about 2 times in finding the optimal schedule, and about 5.5 orders of magnitude faster if we choose to find a valid schedule, even if it is not optimal.
3) Automatic computation of the maximum number of windows to use in each gate.
4) Gate selection after scheduling, ensuring frame isolation if possible, or scheduling non-isolated frames otherwise.
5) BE traffic is considered inherently in the model.
6) Open optimization objectives.
7) Higher (more cases) and better (frame isolation) schedulability than previous studies [5].
8) Zero-jitter results with our proposed optimization objective.

We limit our approach in this paper to fixed paths, i.e., we do not optimize the path in case there are alternative paths.

This paper is structured as follows. Sec. II provides an overview of the most relevant related works in the context of this study. Sec. III explains the system model and the problem definition. Sec. IV presents our contribution: the solution to the GCL synthesis problem by means of heuristics and linear programming. Sec. V describes our experiments and compares our results with those of existing models. Finally, Sec. VI summarizes our conclusions.

II. RELATED WORK

In this section, we gather the contributions that are closer to our proposal. For recent and comprehensive surveys about TSN scheduling approaches, we refer to [2] and [8].

Scheduling algorithms for TSN streams on the IEEE 802.1Qbv TAS fall into two categories: optimal and heuristic. Optimal methods compute optimal schedules, if the schedule exists. Otherwise, they prove the problem instance is infeasible. Heuristic methods prioritize speed over optimality. They cannot deduce whether a problem instance is infeasible, nor are they guaranteed to find a solution if one exists.

In a seminal optimization approach, Satisfiability Modulo Theories (SMT) are used to schedule time-aware streams [3]. They define stream and frame isolation as properties of a schedule to prevent single-link failures. Nevertheless, the GCL configuration is not obtained. The latter problem is addressed in [4], mapping stream frames to transmission windows. Here, the number of windows and gates is fixed per egress port, limiting the number of GCL entries. Isolation is used to avoid unconsidered events for a frame (such as getting lost or being shorter than expected) to affect others (e.g., anticipated arrivals that increase jitter). The experimental

results show that solving time increases exponentially with the number of streams and predefined windows. Our proposal avoids some of the drawbacks of this study and provides faster scheduling times.

The process of transmitting packets as an SMT specification is also formulated in [9], relaxing the constraint that bridges must record the port controls for all packets. Since the execution time of SMT solvers increases exponentially with the problem size, the original specification is divided into multiple Optimization Modulo Theories (OMT) specifications, whose execution time increases linearly as the number of streams grows.

The authors in [10] present an ILP model and a heuristic algorithm for no-wait scheduling of time-aware streams. The model uses a binary variable for each combination of two frames. Since the solving time increases as the number of frames increases, they separate the problem into frame sorting, stream *tabulation* (getting the initial offset), and compaction to avoid gaps. They resolve each one using Tabu Search and obviate both the number of gates and the number of windows.

The heuristic approach in [10] is extended and outperformed in [11]. Here, the network is split into hierarchical sub-networks to improve the scalability of schedule planning, which consists of two parts. First, a cycle phase model is used to isolate different traffic types in time. Second, a two-stage approach: intra-level scheduling for communication within one level, and inter-level scheduling for communication between different levels of the hierarchy.

A different ILP model based on [3], is presented in [12]. They consider that the frame isolation constraint may be a burden in finding a feasible schedule, so they propose a hardware enhancement of a TSN bridge, which checks if the device is about to send the correct frame as scheduled. If this is not the case, the corresponding egress port remains idle instead of sending a different frame from the queue.

Also, related to [3], the SMT model in [13] integrates frame and stream isolation. They focus on the improvement of QoS for the BE traffic with soft real-time requirements while ensuring real-time guarantees for time-aware streams. The study introduces three alternative objective functions (*Maximization*, *Sparse*, and *Evenly Sparse*), a set of constraints on time-aware streams, and the notion of *slack*, which is a time interval between the transmission of time-aware windows. Since existing analytical schedulability analysis methods do not provide support for lower-priority traffic, the proposed solutions are evaluated using the NeSTiNg TSN simulation tool, which is based on the OMNeT++/INET framework. In our analytical proposal, we achieve a cleaner management of slack/gaps between windows without enforcing such gaps, although gap management itself is outside our scope.

Oftentimes there are time-aware streams with relatively small periods within large hyperperiods. Such streams must be scheduled multiple times per hyperperiod, requiring many windows, and yielding GCLs longer than the available hardware allows. This limitation is considered in [14] for industrial scenarios, with constraints also based on Satisfiability Modulo Theories (SMT). Instead of using the hyperperiod as the GCL cycle time, they use a base period less than or equal to the periods of streams, and then try to schedule each stream on several of these GCL cycles.

A Genetic Algorithm (GA) is considered in [15] to optimize a time-aware traffic schedule in TSN on a real autonomous driving vehicle network as the use case. A chromosome consists of messages (genes) to be scheduled (ordered). A schedule is the order of messages arranged in a chromosome. The fitness function considers end-to-end delay, jitter, and bandwidth utilization for guard bands. The chromosome length is the total number of frames transmitted during the hyperperiod for all bridges in the network. There is no evaluation data on rate-constrained and BE-related delays. Slightly increasing the number and type of streams or nodes would hamper the feasibility of the method due to the poor scalability of the GA.

More recently, authors in [16] introduce a heuristic algorithm to schedule large-scale problem instances (2 000 network nodes and more than 10 000 streams). They leverage the search in the space of partial schedules, guided by discovered conflicts. Frames are scheduled one by one, and if a conflict arises, all decisions are reverted up to the conflicting frame. The experimental evaluation shows that this algorithm outperforms existing schedulers on a production line and an avionic dataset.

Authors in [5] introduce a heuristic scheduler for time-aware traffic to be applied in runtime for online rescheduling. The paper details the experimental setup with the LETRA Evaluation Toolset, which includes network configurations, network generators, HERMES scheduler, Constraint Programming scheduler, and result comparison mechanisms. For the experiments, they use a single-bridge network with three nodes and a three-bridge network with nine nodes. Results show that the time to calculate a schedule for both network topologies is in the order of milliseconds. As far as we know, this is the only work that studies the schedulability of the proposed scheduler. Although our proposal is not designed to be used in runtime, we compare our analysis time and schedulability to this proposal.

III. SYSTEM MODEL AND PROBLEM DEFINITION

This section focuses on formalizing the problem addressed in this work which, in general terms, tries to synthesize the GCLs in a network so that the traffic meets the time requirements established a priori.

Definition 1: A TSN system encompasses two elements: a) Data traffic that must be transmitted subject to temporal constraints, and b) A communications network composed of end-stations (the sources and destinations of the traffic), bridges, and connections between them. The IEEE 802.1Qbv TAS must shape the traffic through the network honoring the aforementioned constraints.

The communications network is represented by a directed graph as follows.

***Definition** 2:* The network topology is represented by the digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ [3], [4], where each vertex represents a network element (either a bridge or an end-station), and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ represents the set of physical unidirectional links between each pair of connected nodes, e.g., the ordered pair $(a, b) \in \mathcal{E}$ corresponds to the physical connection from node $a \in \mathcal{V}$ (point-to-point source) to node $b \in \mathcal{V}$ (point-to-point destination). A bidirectional link can be represented as two ordered pairs (a, b) and (b, a).

For each link $(a, b) \in \mathcal{E}$, an egress port in element a leads to element b through this physical link. This means that b identifies both the destination node and the egress port in a used to reach b. In this paper, we indistinctly identify egress ports with the destination nodes they are connected to.

The traffic is modeled as a set of streams where each stream is a tuple with temporal attributes [3], [4], according to the following definition.

***Definition** 3:* The set of streams is $\mathcal{S} = \{s_i | s_i \text{ is a stream}\}$ and each stream is a 5-tuple $s_i = \langle \mathcal{R}_i, L_i, T_i, D_i, J_i \rangle \in \mathcal{S}$. $\mathcal{R}_i \subseteq \mathcal{E}$ is the set of links (the path) traversed by all the frames in this stream. L_i is the length of each frame in the stream. T_i is the period of the stream, that is, the time between the dispatch of each consecutive pair of frames. D_i is the deadline or maximum allowed end-to-end latency. J_i is the maximum allowed jitter.

$\mathcal{R}_i$ describes the path in the net that the frames of the s_i stream follow. There exists $(a, j) \in \mathcal{R}_i$ such that a is the source end-station (end-to-end source, or talker), and any other link $(r, s) \in \mathcal{R}_i$ is reachable from a through other links in $\mathcal{R}_i$. For unicast streams, $\mathcal{R}_i$ is a chain of links with a single end-to-end destination, or listener. In Multicast streams, $\mathcal{R}_i$ represents the branches of a tree with multiple leaves (multiple listeners). Multipath streams (which reach a given node by replicated frames from different paths to provide fault tolerance) are also supported.

The final goal of a TAS scheduler is to synthesize the GCL entries that shape network traffic, honoring the time constraints of time-aware streams, and considering the delays introduced by each node and link. The GCL entries determine the opening and closing times of bridge gates. Thus, the problem addressed in this work can be defined as follows.

***Definition** 4:* Let be the elements of a TSN described in Def. 1. The digraph in Def. 2 represents the network topology. The set of 5-tuples in Def. 3 represents the time-aware streams. The problem to solve consists of computing a schedule establishing the opening and closing times of the gates of the IEEE 802.1Qbv TAS in the network, subject to the time constraints of each time-aware stream.

We take into account the following considerations and assumptions in order to find a solution to this problem.

A. GCL PERIOD (HYPERPERIOD)

To calculate a schedule, we consider a hyperperiod that corresponds to the gate cycle time (i.e., the scheduling period of the GCL), computed as the least common multiple of the periods of the streams:

$$T = \text{l.c.m.}(T_i) \tag{1}$$

This means that each stream i will dispatch $j = \frac{T}{T_i}$ frames separated by T_i over the hyperperiod T. We define $\mathcal{F}_i = \left\{0, \ldots, \frac{T}{T_i} - 1\right\}$ as the set of frames for each stream.

We calculate the *transmission time* $TransT_i$ of a frame i as L_i divided by the transmission speed of the physical network. We also consider the *propagation delay* $PropT_{(a,b)}$ of a wave in the physical link between a and b as calculated by the Peer Delay Mechanism of the gPTP protocol [17]. Finally, we consider a *processing time* $ProcT_b$ at each node b which is device-specific and, therefore, must be measured or provided by the vendor.

***Assumption** 1:* For simplicity, we assume that all frames in a stream have the same length L_i and, accordingly, the same $TransT_i$.

***Assumption** 2:* When the transmission of a frame from a node a is completed, it will take a time $PropT_{(a,b)}$ to propagate the frame up to node b. Next, it will require a time $ProcT_b$ for the frame to enter the state *queued* in the corresponding egress port in node b.

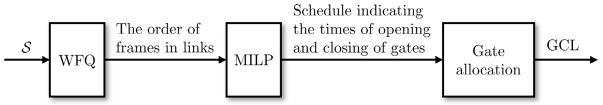

FIGURE 2. Proposed GCL synthesis scheme: WFQ (Sec. IV-A), MILP (Sec. IV-B to IV-G), Gate allocation (Sec. IV-H).

IV. PROBLEM SOLUTION

We leverage heuristics for ordering frame transmissions and for associating windows with gates. In this way, the remaining problem can be stated as a MILP problem, and the process of reaching a solution is simpler than in related approaches that solve the whole stream scheduling problem using SMT (e.g., [3], [4], [13]). Additionally, we do not restrict the solver to use a predefined number of windows or gates and do assume that BE traffic will be present in the system. Fig. 2 depicts our procedure for CGL synthesis.

A. FRAME ORDERING

One of the subproblems to solve when finding a schedule in a TSN system is to decide the order in which frames of different streams are transmitted through the same physical link.

We propose to order the frames heuristically, before computing a schedule of the system. Specifically, we order the frames according to a speculative application of Weighted Fair Queueing (WFQ) [6], [7]. That is, for each frame, we calculate the time it should reach each node in its path, and how long it should take to complete its corresponding transmission. Then, we use this completion value to order the frames traversing each egress port of each node.

Alg. 1 applies WFQ calculations to set the transmission order of frames. Lines 1 to 5 set the initial values for each bridge/port: empty for the ordered list of frames (line 2),

Algorithm 1 Calculate Speculative WFQ Completion Time for Each Frame in Each Bridge/Port

Ensure: Ordered set of frames $WFQ^{(a,b)}$ traversing each bridge/port $(a, b) \in \mathcal{E}$.

1: **for all** $(a, b) \in \mathcal{E}$ **do**
2: $\quad WFQ^{(a,b)} \leftarrow \emptyset$
3: $\quad weightSum_{(a,b)} \leftarrow 0$
4: $\quad$ **for all** $s_i \in \mathcal{S}, (a, b) \in \mathcal{R}_i$ **do**
5: $\quad\quad weightSum_{(a,b)} \leftarrow weightSum_{(a,b)} + weight_i$
6: **for all** $s_i \in \mathcal{S}$ **do**
7: $\quad$ **for all** $j \in \mathcal{F}_i$ **do**
8: $\quad\quad ready \leftarrow j \cdot T_i$
9: $\quad\quad$ **for all** $(a, b) \in \mathcal{R}_i$ **do**
10: $\quad\quad\quad hops \leftarrow hopsToNode((a, b), \mathcal{R}_i)$
11: $\quad\quad\quad transmission \leftarrow hops \cdot TransT_i \cdot \frac{weight_i}{weightSum_{(a,b)}}$
12: $\quad\quad\quad propagation \leftarrow hops \cdot PropT_{(a,b)}$
13: $\quad\quad\quad processing \leftarrow hops \cdot ProcT_b$
14: $\quad\quad\quad fin_{i,j}^{(a,b)} \leftarrow ready + transmission + propagation + processing$
15: $\quad\quad\quad WFQ^{(a,b)} \leftarrow WFQ^{(a,b)} \cup fin_{i,j}^{(a,b)}$
16: **for all** $(a, b) \in \mathcal{E}$ **do**
17: $\quad sort(WFQ^{(a,b)})$

and the sum of the weights of the streams traversing this bridge/port (lines 3 to 5). Then, for each stream (line 6) and frame (line 7), we set the dispatch time (line 8) and then calculate the time upon which perform the ordering. For each traversed bridge/port (line 9), several components of the final time are calculated. Line 10 calculates the number of hops from the talker to the current bridge (function not detailed). Line 11 computes the transmission time component, considering the weights of the streams. Lines 12 and 13 compute the propagation and processing time components. Line 14 computes the transmission completion time. Line 15 adds this time element to the list. This time element also contains the stream and frame identifiers. Once the time for all frames has been added for each bridge/port (line 16), the time list is sorted by completion time (line 17). If several frames have the same completion time, they are sorted by their stream identifier.

B. MILP AND MODEL CONSTRAINTS

In this section, we specify the main variables and constraints in our model.

1) STREAM OFFSET

The transmission start of a stream i can be delayed by an $offset_i$ up to its period. The following constraint allows the solver to accommodate the dispatch of the frames of the different streams during the stream period T_i:

$$\forall\, s_i \in \mathcal{S} \quad 0 \leq offset_i < T_i \tag{2}$$

2) FRAME QUEUED AT TALKER

A frame j of stream i is queued in the egress port of its source end-station (node $first_i$ and port/destination b, $(first_i, b) \in \mathcal{R}_i$) at its corresponding period plus the stream offset:

$$\forall\, s_i \in \mathcal{S},\ j \in \mathcal{F}_i \quad queued_{i,j}^{(first_i,b)} = j \cdot T_i + offset_i \tag{3}$$

The *queued* variables in bridges are defined in Sec. IV-B5.

3) FRAME TRANSMISSION START

Each frame starts its transmission after it has been queued:

$$\forall\, s_i \in \mathcal{S},\ j \in \mathcal{F}_i, (a, b) \in \mathcal{R}_i \quad start_{i,j}^{(a,b)} \geq queued_{i,j}^{(a,b)} \tag{4}$$

4) FRAME TRANSMISSION COMPLETION

Each frame in stream i completes its transmission after a time $TransT_i$ since its transmission starts:

$$\forall\, s_i \in \mathcal{S},\ j \in \mathcal{F}_i,\ (a, b) \in \mathcal{R}_i$$
$$complete_{i,j}^{(a,b)} = start_{i,j}^{(a,b)} + TransT_i \tag{5}$$

5) FRAME QUEUED AT BRIDGE / NODE ORDER IN PATH

This constraint provides the *queued* variables in bridges and also the transmission sequence of any particular frame along its path $\mathcal{R}_i$:

$$\forall\, s_i \in \mathcal{S},\ j \in \mathcal{F}_i,\ (a, b), (b, c) \in \mathcal{R}_i$$
$$queued_{i,j}^{(b,c)} = complete_{i,j}^{(a,b)} + PropT_{(a,b)} + ProcT_b \tag{6}$$

6) FRAME ORDER IN A STREAM

The next constraint sets the transmission order for the frames of a single stream i. At any one bridge/port (a, b), the transmission of a frame $j + 1$ may start after the transmission of frame j completes:

$$\forall\, s_i \in \mathcal{S},\ j, j+1 \in \mathcal{F}_i,\ (a, b) \in \mathcal{R}_i$$
$$complete_{i,j}^{(a,b)} \leq start_{i,j+1}^{(a,b)} \tag{7}$$

7) CALCULATION OF TRANSMISSION TIMES IN/BEYOND T

All frames are transmitted from their source end-station before T. However, they may traverse bridges and arrive at destinations beyond T, e.g., when the end-to-end delay is greater than the period, or when the stream is scheduled to start after a large enough offset. In the worst case, a source end-station may complete the transmission of its last frame at time T. Therefore, we need to compute the number of

Algorithm 2 Calculate the Number of Hyperperiods to Consider in the Analysis

Ensure: Number of hyperperiods N to consider

1: $maxDelay \leftarrow 0$
2: **for all** $s_i \in \mathcal{S}$ **do**
3: $\quad maxDelay \leftarrow \max(maxDelay, T + D_i)$
4: $N \leftarrow \left\lceil \frac{maxDelay}{T} \right\rceil$

hyperperiods N amenable to hold all the traffic sent during the first hyperperiod (Alg. 2).

A schedule repeats every T, therefore, any event beyond T must also be considered inside T. That is, an event occurring at, say, $T+2$, will be scheduled at $0+2$, $T+2$, $2T+2$, and so on, as shown in Fig. 3. In other words, it is not relevant the absolute time of the event, but its relative time, or offset, with respect to T.

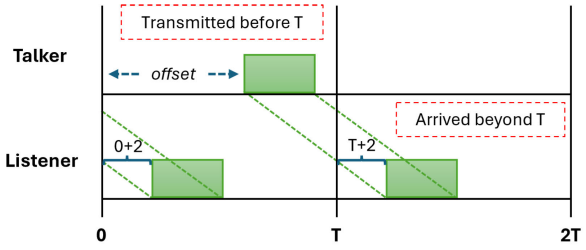

FIGURE 3. Transmission of a frame started after a large offset.

The constraints exposed so far can take values beyond T because we use the absolute time in their formulation. However, we also need relative values for our *start* variables. Essentially, we need to calculate the offset of each starting transmission with respect to the start of the hyperperiod:

$$startOffset_{i,j}^{(a,b)} = start_{i,j}^{(a,b)} \bmod T \tag{8}$$

Since the function modulo is not linear, it cannot be used in an ILP solver. Still, we can explicitly unroll the starting transmission time as an offset plus a number of Ts:

$$start = startOffset + \underbrace{T + \cdots + T}_{k},\ k \in \{0, \ldots, N-1\} \tag{9}$$

Upon the previous idea, we can now set the following constraints:

$$\begin{gathered} \forall\, s_i \in \mathcal{S},\ j \in \mathcal{F}_i,\ (a,b) \in \mathcal{R}_i,\ k \in \{0, \ldots, N-1\} \\ start_{i,j}^{(a,b)} = startOffset_{i,j}^{(a,b)} + \sum_{k=0}^{N-1} k \cdot T \cdot period_{k\,i,j}^{(a,b)} \end{gathered} \tag{10}$$

$period_{k\,i,j}^{(a,b)}$ are binary variables indicating the hyperperiod k during which frame j of stream i in bridge a and port b is transmitted: hyperperiod 0 ($period_0 = 1$), hyperperiod 1 ($period_1 = 1$), and so on. A given frame is transmitted just once through each link along its route. Therefore, one and only one of each $period_{k\,i,j}^{(a,b)}$ may be 1:

$$\begin{gathered} \forall\, s_i \in \mathcal{S},\ j \in \mathcal{F}_i,\ (a,b) \in \mathcal{R}_i,\ k \in \{0, \ldots, N-1\} \\ \sum_{k=0}^{N-1} period_{k\,i,j}^{(a,b)} = 1 \end{gathered} \tag{11}$$

C. FRAME TO WINDOW ASSOCIATION

A queue is served in IEEE 802.1Qbv by opening that queue's gate (see Fig. 1). Each egress port in a bridge holds a GCL. The GCL specifies when to open and close each gate during the schedulable hyperperiod T. Frames in a queue are dispatched as long as its corresponding gate window is open (i.e., between the open and close instants). In order to obtain a predictable/schedulable system, we constraint to 1 the number of open gates at a given time, following [3] and [4]. These studies predefine the number of windows per hyperperiod, introducing a constraint that makes the solver fail to schedule a system when the solution requires more windows than this predefined number. Besides, providing large window values makes the analysis take a very long time [4]. Unlike these approaches, we do not predefine the number of windows in a period but calculate the maximum number of potentially required windows. Thus, in addition to its short computation time, our approach provides the advantage of obtaining a bound for the number of windows in the hyperperiod.

For a given bridge/port, this bound matches the number of frames that traverse this bridge/port. Although some previous works allow several frames to use the same window [3], [4], we associate each frame with a single window. This simplifies the model, and the previous number of potentially required windows is now the exact number of required windows. This is not a restriction in practice, because the scheduler may deploy several windows consecutively to transmit several frames consecutively.

Authors in [3] propose to optionally include a constraint in their SMT formulation to enforce one frame per window and separate these consecutive windows by T_i to reduce jitter to zero for that stream. We achieve a similar result without constraining the open/close operations on windows, which adds flexibility and increases the chances of finding a solution.

Since we associate each frame with a particular window, we apply the same notation used so far for frames to the variables *open* and *close* associated with windows. Recall that these variables are in the range $[0, T]$, as our *startOffset* variables. Thus, a given window must open before transmitting its associated frame, which must complete its transmission before closing the window. However, it is no use opening a gate without traffic, so we can safely equal the opening time to the start of transmission and the closing time to the end of transmission to simplify our model further. We also consider possible time synchronization errors by taking into account the maximum clock skew *ClkErr*, which limits the theoretical transmission times to the range $[ClkErr, T - ClkErr]$.

$$\forall\, s_i \in \mathcal{S},\ j \in \mathcal{F}_i,\ (a,b) \in \mathcal{R}_i$$

$$open_{i,j}^{(a,b)} = startOffset_{i,j}^{(a,b)} - ClkErr \tag{12}$$

$$startOffset_{i,j}^{(a,b)} + TransT_{i,j} + ClkErr = close_{i,j}^{(a,b)} \tag{13}$$

D. WINDOW CONSTRAINTS

In this section, we describe all the elements related to the windows in the GCL. This includes the ordering and separation of windows.

1) WINDOWS AND GAPS

To include BE traffic in the resulting schedule, previous studies require either to schedule predefined BE-devoted windows as time-aware traffic windows [4], or to enforce time intervals without time-aware windows *(slack)* [13]. Either solution might compromise schedulability even if the actual time-aware traffic is schedulable. However, if neither of these solutions is taken, the scheduled windows will cover the whole hyperperiod even when there is no scheduled time-aware traffic, preventing other traffic from using the network. To address this problem, our model separates each pair of windows with a gap. This gap is equivalent to a window during which all the gates are closed. This construct allows us to identify time intervals with no scheduled time-aware traffic and inherently forces windows to be as tight as possible ((12) and (13)). These gaps could be used later to schedule new streams without rescheduling the whole system, or as windows for BE traffic, or they could simply be ignored to reduce the GCL length. The concepts *window*, *slack*, and *gap* are graphically differentiated in Fig. 4.

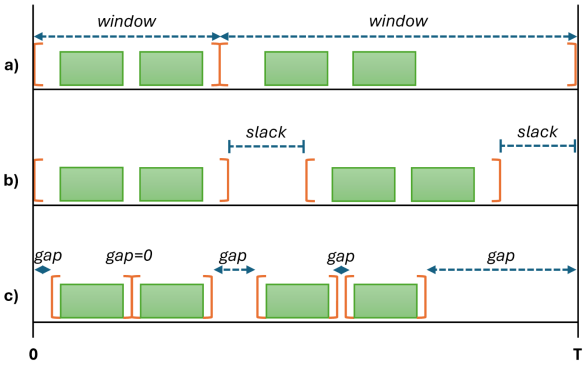

FIGURE 4. a) Time-aware *windows* covering the whole hyperperiod [4]. b) Fixed enforced *slack* [13]. c) Our proposed virtual *gaps*.

2) GENERAL GAP AND WINDOW CONSTRAINTS

We assume that each required window is followed by a gap, which may have duration zero.

$$\forall\, s_i \in \mathcal{S},\ j \in \mathcal{F}_i,\ (a,b) \in \mathcal{R}_i$$

$$open_{i,j}^{(a,b)} \leq close_{i,j}^{(a,b)} \tag{14}$$

$$close_{i,j}^{(a,b)} = gapOpen_{i,j}^{(a,b)} \tag{15}$$

$$gapOpen_{i,j}^{(a,b)} \leq gapClose_{i,j}^{(a,b)} \tag{16}$$

3) WINDOW ORDERING CONSTRAINTS

Following the order obtained in Alg. 1 to schedule our windows, each close instant of the predecessor ($\prec$) gap must equal the opening of the successor window.

$$\forall\,(a,b) \in \mathcal{E}, fin_{i,j}^{(a,b)}, fin_{k,l}^{(a,b)} \in WFQ^{(a,b)}, fin_{i,j}^{(a,b)} \prec fin_{k,l}^{(a,b)}$$

$$gapClose_{i,j}^{(a,b)} = open_{k,l}^{(a,b)} \tag{17}$$

4) INITIAL GAP CONSTRAINTS

We assume an initial gap at time 0, before the first window:

$$\forall\,(a,b) \in \mathcal{E}, fin_{i,j}^{(a,b)} \in WFQ^{(a,b)},\ \nexists\, first \in WFQ^{(a,b)}, fin_{i,j}^{(a,b)} \prec first$$

$$0 = initialGapOpen^{(a,b)} \tag{18}$$

$$initialGapOpen^{(a,b)} \leq initialGapClose^{(a,b)} \tag{19}$$

$$initialGapClose^{(a,b)} = open_{first}^{(a,b)} \tag{20}$$

5) LAST GAP CONSTRAINTS

Since the GCL restarts at time T, the last gap must be closed exactly at T:

$$\forall\,(a,b) \in \mathcal{E}, fin_{i,j}^{(a,b)} \in WFQ^{(a,b)},\ \nexists\, last \in WFQ^{(a,b)}, last \prec fin_{i,j}^{(a,b)}$$

$$gapClose_{last}^{(a,b)} = T \tag{21}$$

E. CALCULATION OF GLOBAL METRICS

We perform the following calculations to optimize and bound global metrics.

1) DELAY COMPUTATION

$$\forall\, s_i \in \mathcal{S}$$

$$avgDelay_i = \frac{1}{|\mathcal{F}_i|} \sum_{j \in \mathcal{F}_i} delay_{i,j} \tag{22}$$

$$systemMaxDelay \geq maxDelay_i \tag{23}$$

$$\forall\, s_i \in \mathcal{S},\ j \in \mathcal{F}_i$$

$$maxDelay_i \geq delay_{i,j} \tag{24}$$

$$delay_{i,j} = complete_{i,j}^{(a,last)} - ready_{i,j}^{(first,b)} + PropT_{(a,last)} + ProcT_{last} \tag{25}$$

2) JITTER COMPUTATION

$$\forall\, s_i \in \mathcal{S}$$

$$systemJitter \geq jitter_i \tag{26}$$

$$\forall\, s_i \in \mathcal{S},\ j \in \mathcal{F}_i$$

$$jitter_{i,j} \geq avgDelay_i - delay_{i,j} \tag{27}$$

$$jitter_{i,j} \geq delay_{i,j} - avgDelay_i \tag{28}$$

$$jitter_i \geq jitter_{i,j} \tag{29}$$

3) GAP COMPUTATION

$$systemGap = \sum_{(a,b) \in \mathcal{E}} gap^{(a,b)} \tag{30}$$

$$\forall\,(a,b) \in \mathcal{E}$$

$$gap^{(a,b)} = initialGapClose^{(a,b)} - initialGapOpen^{(a,b)} + \sum_{s_i \in \mathcal{S}} \sum_{j \in \mathcal{F}_i} gapClose_{i,j}^{(a,b)} - gapOpen_{i,j}^{(a,b)} \tag{31}$$

Other parameters could be computed in a similar way, such as the longest gap.

F. END-TO-END CONSTRAINTS

To limit the predefined delay and jitter in each stream, the following constraints are required:

$$\forall\, s_i \in \mathcal{S}$$
$$maxDelay_i \leq D_i \tag{32}$$
$$jitter_i \leq J_i \tag{33}$$

G. OBJECTIVE FUCTION

The objective function can be adjusted to specific problems using the previous constraints. As a generic objective function, we propose to minimize the average delay of all streams in the system. Accordingly, we formulate the following objective function and problem statement:

$$\text{min: } \sum_{s_i \in \mathcal{S}} avgDelay_i$$
$$\text{subject to: (2) to (7) and (10) to (33)} \tag{34}$$

We use this objective in our experiments.

H. GATE ALLOCATION

IEEE802.1Qbv defines 8 queues/gates for 8 traffic classes (see Fig. 1). As a rule, the number of gates should not be a problem, because a correct scheduling implies few queue requirements, and most of the queues often remain unoccupied. Thus, most previous studies, e.g., [3], [4], do not fix traffic classes to frames along the whole path but allow each frame to switch its associated traffic class identifier at bridges. In practice, this means that each stream can be isolated from the other streams, even when there are more streams than queues, as long as frames of different streams are not driven to the same queue at the same time.

The constraints formulated so far provide no limitation on the number of gates (queues) per bridge egress port. In this way, the problem becomes much simpler. Nevertheless, completing the schedule requires computing the number of required gates and obtaining a window-to-gate allocation [3], [4]. Recall that each window transmits a single frame according to our model. Thus, gate selection must consider the *queued* and *complete* times of frames along with the stream they belong to. Alg. 3 performs such computation.

For each bridge/port (line 1), we obtain the set of frames traversing this bridge/port (lines 2 to 6), along with their queue times and transmission completion times. The gate identifier is initially set to -1 (line 6). This set is, then, sorted by the *queued* time (line 7). Also, we initialize the *lastgate* variable to -1 to state that it is still invalid (line 8). Afterward, for all frames in the set of frames traversing this bridge/port (line 9), we analyze the frames that have been queued before (lines 13 to 21) in the same bridge/port. For each frame (lines 9 and 10) we take note (lines 17 and 21) of the frames previously queued but not yet transmitted (lines 13 to 15), that is, the frames which are queued at the same time. We also check whether any of these frames belong to the same stream (line 18) so that we can use the same queue (line 19).

Algorithm 3 Assign a Gate for Each Frame/Window

Ensure: $\forall\, (a, b) \in \mathcal{E}$, a set $frames^{(a,b)}$ whose elements are tuples $\langle stream, frame, queued, complete, gate\rangle$ containing the queue/gate assigned to the corresponding frame (from *queued* to *complete*). Recall that each frame is transmitted in its own window. A warning is printed if frame isolation is not possible.

1: **for all** $(a, b) \in \mathcal{E}$ **do**
2: $frames^{(a,b)} \leftarrow \emptyset$
3: **for all** $s_i \in \mathcal{S}$ **do**
4: **if** $(a, b) \in \mathcal{R}_i$ **then**
5: **for all** $j \in \mathcal{F}_i$ **do**
6: $frames^{(a,b)} \leftarrow frames^{(a,b)} \cup \{i, j, queued_{i,j}^{(a,b)}, complete_{i,j}^{(a,b)}, -1\}$
7: sort-by-queued($frames^{(a,b)}$)
8: $lastgate \leftarrow -1$
9: **for** $k \leftarrow 1$ to $|frames^{(a,b)}|$ **do**
10: $current \leftarrow frames_k^{(a,b)}$
11: $usedGates \leftarrow \emptyset$
12: $queuedStreams \leftarrow \emptyset$
13: **for** $l \leftarrow k-1$ to 1 step -1 **do**
14: $previous \leftarrow frames_l^{(a,b)}$
15: **if** $previous.complete > current.queued$ **then**
16: **if** $previous.stream \notin queuedStreams$ **then**
17: $queuedStreams \leftarrow queuedStreams \cup \{previous.stream\}$
18: **if** $previous.stream = current.stream$ **then**
19: $gateToReuse \leftarrow previous.gate$
20: **else**
21: $usedGates \leftarrow usedGates \cup \{previous.gate\}$
22: **if** $current.stream \in queuedStreams$ **then**
23: **if** $gateToReuse \in usedGates$ **then**
24: print warning: $current.stream$ not isolated
25: $current.gate \leftarrow gateToReuse$
26: $frames_k^{(a,b)} \leftarrow current$
27: **else**
28: **if** $|usedGates| < AVAILABLE\_GATES$ **then**
29: $newGate \leftarrow (lastGateUsed + 1) \bmod AVAILABLE\_GATES$
30: **while** $newGate \in usedGates$ **do**
31: $newGate \leftarrow (newGate + 1) \bmod AVAILABLE\_GATES$
32: **else**
33: print warning: $current.stream$ not isolated
34: $newGate \leftarrow (lastGateUsed + 1) \bmod AVAILABLE\_GATES$
35: $lastGateUsed \leftarrow newGate$
36: $current.gate \leftarrow newGate$
37: $frames_k^{(a,b)} \leftarrow current$

Once all the frames previously scheduled have been checked, we proceed to allocate a gate to the current frame/window. If the frame belongs to a stream already queued, it is allocated to the same gate (lines 22 and 25) and the frame information

is updated (line 26). If not, a new gate should be used. If there are free gates (line 28), we allocate the new gate according to a Round Robin policy (lines 29 to 31). If all gates are occupied, the new gate is also selected by Round Robin (line 34), but line 33 prints a message warning that isolation is not possible (this is also performed at lines 23 and 24). Then, the new gate is set as the gate allocated to the current frame (line 36) and its information is updated (line 37).

V. CASE STUDIES AND RESULTS

We rely on two metrics to test our approach: analysis time and schedulability. To test the analysis time, we replicate scenarios from [4], inspired by industrial use cases, and compare our results. Scheduling the proposed streams is costly, but schedulability is not a problem because the network utilization is rather low. Nevertheless, we must also verify that our approach does not fail to schedule high-utilization scenarios because part of our analysis is heuristic. To test the schedulability, we replicate scenarios from [5] and compare our results. All our experiments assume a weight of 1 for all streams when applying Alg. 1. Our algorithms are programmed in Python, and we use lp-solve version 5.5.2.5 as our MILP solver. The next sections extend on these evaluations.

A. ANALYSIS TIME

To evaluate the analysis time of our approach, we replicate the *medium-sized network* scenario in [4]. It consists of a line topology for 10 bridges with 5 end-stations connected to each bridge. All streams are unicast, with the sender (talker) and the receiver (listener) chosen randomly, and period T_i randomly set to either 10 or 20 ms. The transmission time of each frame is 13 μs, assuming the maximum Ethernet frame size on a 1 Gb/s network. In order to mimic the replicated scenario [4], we assume propagation and processing times of 0, and clocks with error 0. Our model is able to consider non-zero values, though. Experiments using this scenario in previous papers test 10, 20, 30, 40 and 50 streams, 2, 4, 8, 16 and 32 windows and 4 available gates, totaling 25 experiments [4]. Our model automatically calculates the required windows and gates, so our only experimental variable is the number of streams. We perform 100 experiments for each number of streams, totaling 500 experiments, to provide a wider analysis.

Fig. 5 shows the analysis time required for our proposal to optimize the system. Each mark in the plot represents a scenario. The x axis represents the number of frame transmissions required in T. For instance, a frame traversing 4 bridges would need its initial transmission from the corresponding end-station plus a transmission by each traversed bridge, totaling 5 transmissions. The y axis shows the analysis time required to optimize the system. In our experiments, we optimize (34), that is, minimize the average delay of all streams. Since this axis is logarithmic, we show horizontal dotted lines at 1 minute, 1 hour, and 1 day. Finally, color and shape represent the number of streams in the experiment.

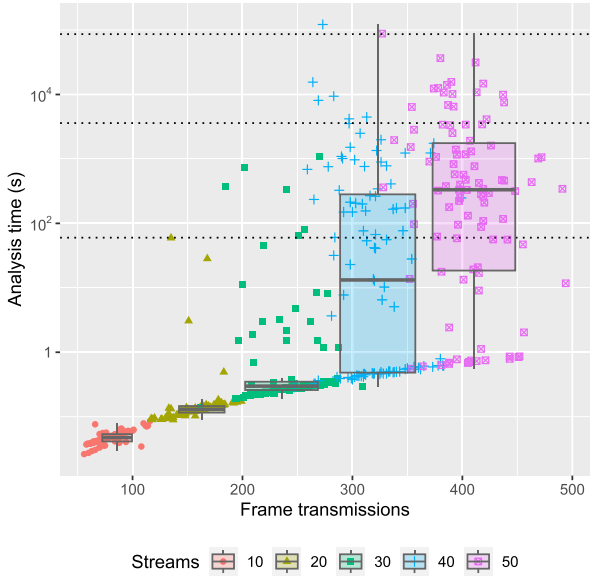

FIGURE 5. Analysis time.

To show how points are distributed, we overlay a boxplot for each number of streams. The scenarios in [4] do not specify the number of frame transmissions in T, but we estimate this value from their *frame instances*, resulting approximately in 81, 161, 233, 299, and 396 frame transmissions for their 10, 20, 30, 40, and 50 streams experiments, respectively. These values fit in the ranges shown in the x axis of Fig. 5 for the corresponding streams. The analysis times required by previous studies for these scenarios range from 100 ms to above 40 hours (set as timeout), running the z3 solver on an Intel(R) Core(TM) i7-2600 3.40GHz CPU [4]. On average, our experiments are completed in a much shorter time on an Intel(R) Core(TM) i7-10700 2.90GHz CPU. We use a single core and require a small amount of RAM (around 14 MB). If we set the same 40-hour timeout, only one out of our 500 experiments aborts, while one out of the 25 experiments aborted in [4]. Solving the model is the main contribution to the global analysis time. The application of our three algorithms, the generation of the model, and feeding the solver with the model always takes 38 ms on average, irrespective of the number of streams.

All our results achieve 0 jitter for the scheduled streams, whereas previous studies require specific optimizations (adding up much more analysis time) to reduce jitter [4]. Furthermore, the maximum number of gates required in egress ports is very low, as expected: just 1 gate for 75% of our experiments, 2 gates for 23%, and 3 gates for the remaining 2% of our experiments. According to these results, previous studies were unnecessarily over-dimensioning their predefined number of gates [4].

We can still improve our analysis times, although they are considerably lower than previous approaches. Often enough, it is sufficient to obtain any schedule that satisfies the requirements, instead of finding an optimal. Fig. 6 shows that

the analysis time plummets below 1.05 seconds if we stop the solver when finding the first valid solution, for the same scenarios considered in Fig. 5. The thickness of the layered clusters in Fig. 6 relates to the number of iterations it takes the solver to reach the solution.

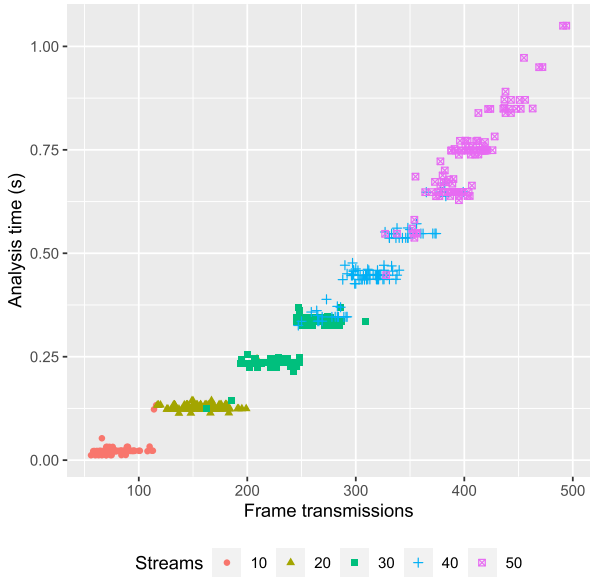

FIGURE 6. Analysis time until first suitable scheduling.

Fig. 7 plots the maximum number of windows required in the system, i.e., the length of the longest GCL, for the same scenarios. Recall that previous works require a predefined maximum number of windows to solve the system [4], whereas our method calculates such maximum automatically. The figure shows that some scenarios require more than 30 windows, which exceeds the predefined value used in previous papers [4]. Those scenarios might not be schedulable under such limitation.

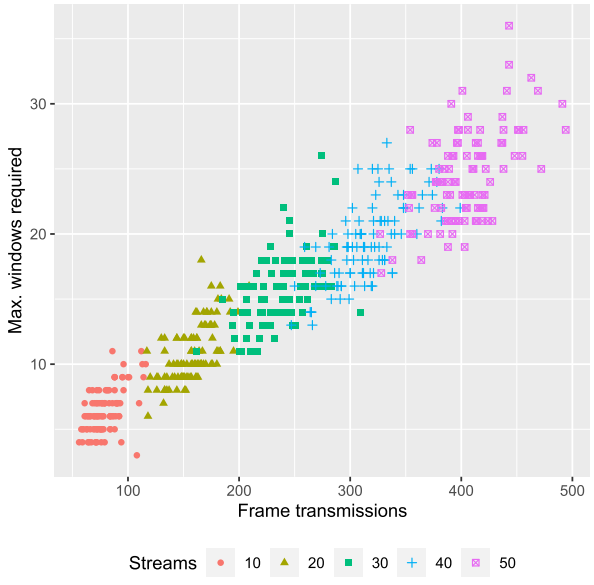

FIGURE 7. Maximum number of windows required in the system.

In summary, these experimental results show that the analysis times of our method are considerably shorter than previous approaches, providing the best solution around 2 times faster [4]. Also, our resulting schedules yield no jitter in the scenarios under test, whereas previous studies require far more time to minimize jitter. Additionally, we can provide a valid scheduling around 5.5 orders of magnitude faster if we select the first valid solution found instead of the optimal one. Further, we do not need to predefine a specific number of gates/windows to use.

B. SCHEDULABILITY

We evaluate schedulability by performing experiments akin to [5]. Their results are based on network utilization, generally defined in real-time systems as the fraction of time the system is used. They set a simple topology consisting of three end-stations linked through a single bridge. In this case, *utilization* is the fraction of time the 6 unidirectional 1 Gb/s links are in use. All streams are unicast, with the sender (talker) and the receiver (listener) chosen randomly. Periods are randomly set between 200 and 1000 μs, always multiple of 200 μs, and the same values are used for the maximum allowed end-to-end latency. The transmission time for the frames of each stream is also randomly set between 3 and 7 μs, which would correspond to frames between 500 and 1000 bytes approximately. We assume that inter-frame spaces are included in the frame transmission time, as in previous papers [4], [5]. The utilizations previously tested are in the range from 10% to 90%. To cover this utilization range, we use scenarios ranging from 30 to 240 streams; 5 scenarios with 30 randomized streams, 5 scenarios with 31 randomized streams, and so on, with a total of 1055 scenarios/experiments.

Before analyzing schedulability, let us consider the analysis times until a valid schedule is found, as displayed in Fig. 8[1] The x-axis shows the network utilization, covering a range wider than previous studies [5]. As expected, some scenarios with high utilization cannot be scheduled ($\times$ in the figure). For the rest, the analysis time until a valid scheduling (not necessarily the best one) is found is under 45 seconds. This is around four orders of magnitude longer than the HERMES results [5]. However, our goals are different. HERMES focuses on obtaining a very fast heuristic scheduling with zero or low jitter, whereas our method explores a much broader solution space, and we can optimize the scheduling for any desired metric. Also, our proposal provides frame isolation whenever possible, as discussed below.

Fig. 8 also shows colors for the scheduled scenarios. These colors indicate the number of gates required to achieve frame isolation, i.e., to ensure that a frame does not affect others. IEEE 802.1Qbv specifies 8 gates, shown in white in Fig. 8. Lower gate requirements present reddish tones. On the other hand, schedules requiring more than 8 gates/queues to provide frame isolation are shown in bluish tints. These

[1]These first solutions may not be the optimal ones, but we have verified that all obtained solutions provide a schedule without jitter.

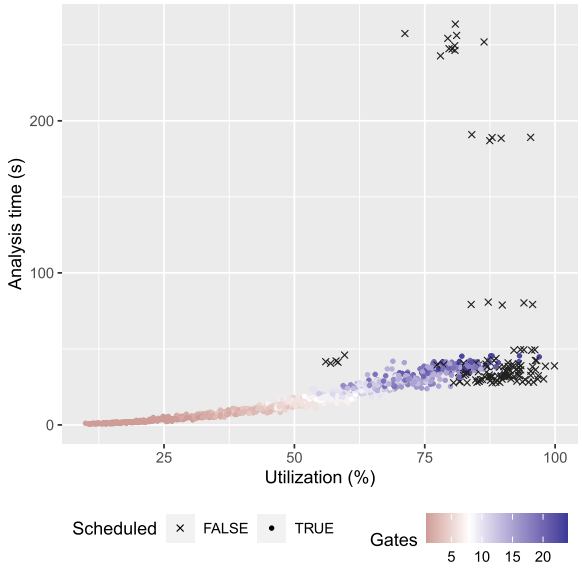

FIGURE 8. Analysis time until first suitable scheduling.

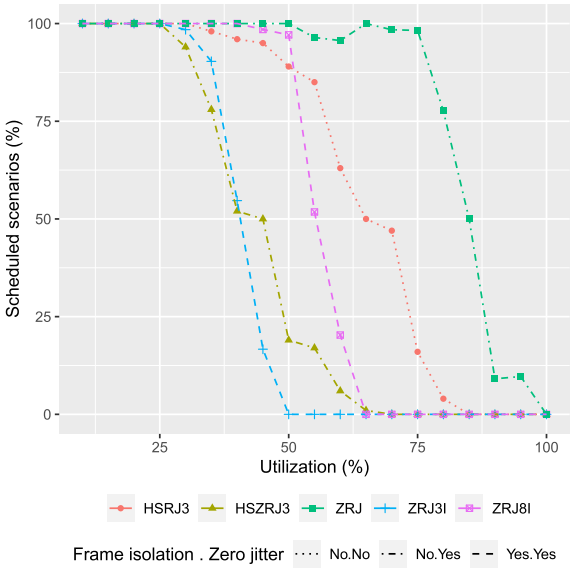

FIGURE 9. Schedulability.

schedules are also schedulable with just 8 gates if frame isolation is relaxed (i.e., not guaranteed). Alg. 3 provides a scheduling with frame isolation whenever possible. However, to clearly see the extent of gate-sharing (bluish tones in Fig. 8), we have run it with an AVAILABLE_GATES value much higher than 8. As far as we know, no other method yields a schedule with frame isolation when possible and without frame isolation when not. In previous approaches, frame isolation is either a requirement, and the method fails if frames cannot be isolated [4], or not [5], and it never ensures this property.

To quantify the scheduling capabilities of our proposal, we group our experiments in steps of 5% in utilization and test how many of these experiments have been effectively scheduled (Fig. 9), as previous papers [5]. We include in the figure results HSRJ3 and HSZRJ3 from the HERMES method [5]. These policies employ 3 gates with reception jitter and zero reception jitter, respectively, and they do not provide frame isolation. In contrast, our proposal guarantees frame isolation whenever possible, and our solutions always yield zero jitter. We show our schedulability results with 3 and 8 gates with no jitter and frame isolation (ZRJ3I and ZRJ8I) and our results with no jitter without frame isolation (ZRJ).

As shown in Fig. 9, we can schedule almost all scenarios until a 75% utilization, and 50% of them with utilization of 85%, without frame isolation. In contrast, HERMES presents a much lower scheduling factor, failing to schedule 50% of scenarios with a 65% utilization when it allows jitter and 40% utilization with zero jitter. With 3 gates and frame isolation, our proposal achieves similar results to HERMES without frame isolation. With 8 gates, frame isolation, and no jitter, our results appear between the HERMES policies with and without jitter (3 gates without frame isolation).

In summary, our schedulability results outperform previous works in the same conditions (frame isolation not guaranteed) [5]. Also, our approach generates schedules without jitter in all our tested scenarios and tries to guarantee frame isolation whenever possible.

VI. CONCLUSION

We propose a new scheduling method for TSN streams. Our method leverages WFQ to set the ordering of frames and then generates a MILP model to optimize the TSN scenario. Next, we assign gates to each scheduled frame, guaranteeing frame isolation whenever possible.

Our model is more versatile compared to previous approaches [4], [5] because it does not require predetermined windows/gates nor prefix frame isolation. We also encompass end-to-end delays longer than the hyperperiod. Further, our proposal inherently considers the existence of BE traffic and computes the GCLs so that the gates devoted to time-aware traffic remain open for just the indispensable time to forward this traffic. Compared in time, we obtained better results than equivalent proposals. Our method provides the best solution around 2 times faster than previous studies [4]. Additionally, our resulting schedules provide no jitter, whereas previous studies require much more time to minimize jitter. Also, we can provide a valid scheduling around 5.5 orders of magnitude faster if we choose to get any valid solution instead of the optimal one. Finally, our approach provides higher schedulability than previous studies. We are able to schedule systems with utilization up to 85%, whereas previous papers reach 65% [5].

The implementation of our proposal can be found at https://gitlab.com/uz-gaz/ilp-tsn-scheduler.

REFERENCES

[1] L. Zhao, P. Pop, and S. Steinhorst, "Quantitative performance comparison of various traffic shapers in time-sensitive networking," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 3, pp. 2899–2928, Sep. 2022, doi: 10.1109/TNSM.2022.3180160.

[2] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (TSN)," *IEEE Access*, vol. 11, pp. 61192–61233, 2023, doi: 10.1109/ACCESS.2023.3286370.
[3] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, Brest, France, Oct. 2016, pp. 183–192, doi: 10.1145/2997465.2997470.
[4] R. S. Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv gate control list synthesis using array theory encoding," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Porto, Portugal, Apr. 2018, pp. 13–24, doi: 10.1109/RTAS.2018.00008.
[5] D. Bujosa, M. Ashjaei, A. V. Papadopoulos, T. Nolte, and J. Proenza, "HERMES: Heuristic multi-queue scheduler for TSN time-triggered traffic with zero reception jitter capabilities," in *Proc. RTNS 30th Int. Conf. Real-Time Netw. Syst.*, Y. Abdeddaïm, L. Cucu-Grosjean, G. Nelissen, and L. Pautet, Eds., Paris, France, Jun. 2022, pp. 70–80, doi: 10.1145/3534879.3534906.
[6] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 4, pp. 1–12, Aug. 1989, doi: 10.1145/75247.75248.
[7] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, Jun. 1993, doi: 10.1109/90.234856.
[8] H. Chahed and A. Kassler, "TSN network scheduling—Challenges and approaches," *Network*, vol. 3, no. 4, pp. 585–624, Dec. 2023, doi: 10.3390/network3040026.
[9] X. Jin, C. Xia, N. Guan, C. Xu, D. Li, Y. Yin, and P. Zeng, "Real-time scheduling of massive data in time sensitive networks with a limited number of schedule entries," *IEEE Access*, vol. 8, pp. 6751–6767, 2020, doi: 10.1109/ACCESS.2020.2964690.
[10] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, Brest, France, Oct. 2016, pp. 203–212, doi: 10.1145/2997465.2997494.
[11] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr, "Scaling TSN scheduling for factory automation networks," in *Proc. 16th IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, Porto, Portugal, Apr. 2020, pp. 1–8, doi: 10.1109/WFCS47810.2020.9114415.
[12] M. Vlk, Z. Hanzálek, K. Brejchová, S. Tang, S. Bhattacharjee, and S. Fu, "Enhancing schedulability and throughput of time-triggered traffic in IEEE 802.1Qbv time-sensitive networks," *IEEE Trans. Commun.*, vol. 68, no. 11, pp. 7023–7038, Nov. 2020, doi: 10.1109/TCOMM.2020.3014105.
[13] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, "Synthesising schedules to improve QoS of best-effort traffic in TSN networks," in *Proc. 29th Int. Conf. Real-Time Netw. Syst.*, Nantes, France, Apr. 2021, pp. 68–77, doi: 10.1145/3453417.3453423.
[14] Q. Li, D. Li, X. Jin, Q. Wang, and P. Zeng, "A simple and efficient time-sensitive networking traffic scheduling method for industrial scenarios," *Electronics*, vol. 9, no. 12, p. 2131, Dec. 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/12/2131
[15] H. J. Kim, K. C. Lee, and S. Lee, "A genetic algorithm based scheduling method for automotive Ethernet," in *Proc. IECON 47th Annu. Conf. IEEE Ind. Electron. Soc.*, Toronto, ON, Canada, Oct. 2021, pp. 1–5, doi: 10.1109/IECON48115.2021.9589998.
[16] M. Vlk, K. Brejchová, Z. Hanzálek, and S. Tang, "Large-scale periodic scheduling in time-sensitive networks," *Comput. Oper. Res.*, vol. 137, Jan. 2022, Art. no. 105512, doi: 10.1016/j.cor.2021.105512.
[17] *IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications*, Standard Std 802.1AS-2020 (Revision IEEE Std 802.1AS-2011), 2020, pp. 1–421.

ALITZEL GALILEA TORRES-MACÍAS (Graduate Student Member, IEEE) received the B.Sc. degree in mechatronics engineering from the University of Guadalajara, Mexico, in 2021, and the M.Sc. degree in electrical engineering from CINVESTAV Unidad Guadalajara, Mexico, in 2024. She is currently pursuing the joint Ph.D. degree with CINVESTAV Unidad Guadalajara and the University of Zaragoza.

JUAN SEGARRA FLOR received the degree in computer science and the Ph.D. degree from Universitat Jaume I (Spain), in 2003. In 2003, he joined the University of Zaragoza, where he is currently with the Department of Computer and Systems Engineering. He is a member with the Computer Architecture Group (gaZ), University of Zaragoza. His research interests include time-sensitive networking, worst-case execution time, and worst-case memory performance in hard real-time systems.

JOSÉ LUIS BRIZ VELASCO received the combined B.Sc./M.Sc. degree in geology, the M.Sc. degree in computer science, and the Ph.D. degree in computer engineering from the University of Zaragoza (UZ), Spain, in 1996. He is currently a tenured Associate Professor with the Department of Computer and Systems Engineering and a Researcher with the I3A Research Institute, UZ. His research interests include memory hierarchy, processor microarchitecture, and real-time systems. He is a member with the gaZ Group and an Affiliate with the HiPEAC European Network of Excellence. He is also a member of the Spanish Society of Computer Architecture (SARTECO).

ANTONIO RAMÍREZ-TREVIÑO (Member, IEEE) received the B.Sc. degree in electrical engineering from Universidad Autónoma Metropolitana, Mexico City, Mexico, in 1986, the M.Sc. degree from CINVESTAV Unidad Guadalajara, Mexico, in 1990, and the Ph.D. degree from the University of Zaragoza, Spain, in 1993. He is currently an Active Professor of automation with CINVESTAV Unidad Guadalajara. His research interests include scheduling, analysis, and control of discrete event systems, including controllability, observability, and stability.

HÉCTOR BLANCO-ALCAINE received the M.Sc. degree (Hons.) in computer engineering from the University of Zaragoza. He is currently a Software Enabling and Optimization Architect with Intel Deutschland GmbH, Germany. He bridges application requirements from customers to specific software components and hardware technologies. He has twenty years of experience in software engineering and customer-facing roles. His research interests include time-sensitive systems, involving end-to-end determinism and synchronization requirements for networking and SoCs.