Weakly-Hard Real-Time Flow Scheduling in Time-Sensitive Networks

Tamim Ahmed[1], Zain A. H. Hammadeh[2], Daniel Lüdtke[2], and Monowar Hasan[1]
[1]Washington State University, Pullman, USA
[2]Institute of Software Technology, German Aerospace Center (DLR), Braunschweig, Germany
Email: {tamim.ahmed, monowar.hasan}@wsu.edu, {zain.hajhammadeh, daniel.luedtke}@dlr.de

***Abstract*—Time-Sensitive Networking (TSN) offers deterministic transmission through mechanisms such as the Time-Aware Shaper (TAS), but existing scheduling approaches assume hard real-time semantics where all packets must meet deadlines, leading to inefficiency under high utilization or overload. In contrast, numerous cyber-physical systems operate under "weakly-hard" timing models that permit bounded deadline violations. This paper introduces a design-time scheduling framework that integrates weakly-hard requirements into TSN by categorizing packets as mandatory or optional and synthesizing Gate Control Lists (GCLs) that guarantee timing constraints for all mandatory packets while maximizing admission of optional traffic. We propose both a computationally efficient heuristic (named Lazy Search) and an Integer Linear Programming (ILP) formulation for GCL construction. We further systematically analyze the trade-off between serving optional packets through a dedicated priority queue versus multiplexing them within their designer-provided traffic class. Through synthetic workload evaluation and hardware validation on a commodity TSN switch, we demonstrate that the proposed framework substantially improves optional packet service rates without compromising timing guarantees of mandatory packets.**

I. Introduction

Time-Sensitive Networking (TSN) [1] extends standard Ethernet with deterministic transmission capabilities, which are crucial for latency-sensitive communications in autonomous vehicles, industrial automation, and other critical cyber-physical domains. Existing TSN mechanisms—such as the Time-Aware Shaper (TAS) [2]—offer strict timing guarantees but are typically designed under the assumption that *all* packets from traffic flows must meet their deadlines. However, in many safety-critical applications, not all messages are equally critical or must be delivered within *hard* deadlines. As a motivational example, we consider applications from the space domain. The Scalable On-board Computing for Space Avionics (ScOSA) [3] is a distributed, heterogeneous on-board computer architecture that combines radiation-hardened processors (GRE712RC LEON3FT) with multiple commercial off-the-shelf (COTS) systems-on-chip (eight Xilinx Zynq 7000 devices) interconnected via a SpaceWire network. A high-level schematic of ScOSA is illustrated in Fig. 1. The ScOSA middleware detects failures and migrates tasks from failing nodes to other nodes to maintain system reliability, which is particularly important given the radiation sensitivity of the COTS components. If a failure is transient, the recovered nodes can be automatically reintegrated into the network. ScOSA allows multiple applications to execute concurrently [4], exchanging messages such as heartbeat monitoring and reconfiguration with each other and with a middleware. The RCN (Reliable Computing Node) does not run any applications—it acts as the coordinator of the network and executes the telemetry task. Three missing heartbeat acknowledgments in a row from an HPN (High-Performance Node) trigger a reconfiguration process. However, missing two consecutive heartbeats is tolerable if followed by three consecutive successful (on-time) heartbeats—that is, the heartbeat service can tolerate up to two deadline misses within any five consecutive messages. In scenarios where more than one HPN node fails, the load temporarily increases on the remaining nodes until the failed nodes are rebooted and reintegrated. In such cases, the flows routed to a node can exploit these *tolerable* deadline misses to maintain the availability of as many applications as possible.

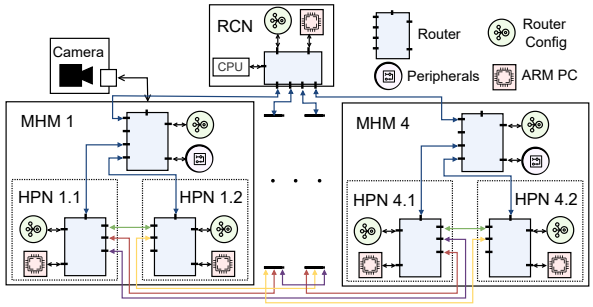


Fig. 1. Basic building blocks of the ScOSA architecture [3]. The system comprises one radiation-hardened CPU and eight COTS CPUs. Acronyms: RCN—Reliable Computing Node; HPN—High-Performance Node; MHM—Multi-HPN Module.

Many real-time communication tasks exhibit *weakly-hard* timing semantics [5], wherein bounded deadline misses are permissible as long as each flow satisfies a service constraint of the form: "at most m deadline misses are acceptable in any window of k consecutive packets." This controlled tolerance to deadline misses provides additional flexibility under overloaded conditions, where carefully regulated packet dropping can improve overall flow admissibility. These observations expose an important yet underexplored design opportunity for TSN: *how can we incorporate weakly-hard guarantees into TSN flow scheduling?*

In this work, **we investigate weakly-hard real-time flow**

scheduling in TSN. We consider real-time flows that must satisfy an (m, k) temporal constraint, where at most m out of every k consecutive packets are "optional" and *may* miss their deadlines. The remaining $k - m$ packets are "mandatory" and *must* meet deadlines. However, instead of simply discarding the optional packets, we investigate scheduling techniques that *increase their admissibility*, resulting in improved overall service guarantees. We do so by reserving a dedicated switch queue for servicing the optional packets.

TSN shapers (e.g., TAS) use *gates* that open and close according to a time-triggered schedule, allowing packets to be transmitted through an egress port of the TSN switch. A Gate Control List (GCL) specifies which switch queues are permitted to transmit at any given time. The challenge, then, is to determine: *what is the GCL for each queue that ensures all mandatory packets meet their deadlines while maximizing the admissibility of optional packets?* To address this problem, we introduce a heuristic search technique that determines the GCLs. However, given the inherent suboptimality of heuristic methods, we further develop an optimization-based GCL construction technique.

In addition to determining the GCLs, we further explore the design space and pose the following question: *Is it better to allocate a dedicated queue for serving optional packets in a best-effort manner, or to serve them alongside their own queue while attempting to optimize (i.e., reduce) their response times with the goal of preserving deadlines for most—if not all—flows?* Through extensive synthetic evaluation and a proof-of-concept demonstration on a real TSN switch, we find that reserving a dedicated queue leads to better flow admissibility.

In this paper, we make the following contributions:

- We explore the problem of scheduling TSN flows under weakly-hard timing constraints and characterize the distinction between mandatory and optional packets within TAS-enabled switch architectures (Section II).
- We develop two flow scheduling (GCL construction) methods: a heuristic search procedure and an optimization-based formulation and study their scalability (Sections III–V).
- We investigate the design choice of reserving a dedicated queue for optional packets and analyze how this affects their admissibility and the schedulability of mandatory traffic (Section V).

We evaluate our proposed techniques through extensive synthetic experiments (Section V-B) and a proof-of-concept implementation on an off-the-shelf TSN switch (Section V-C). We make our implementation **publicly available** for community use [6].

II. System Model

TSN extends Ethernet with scheduling and resource-isolation mechanisms that provide deterministic transmission guarantees at the link layer. As shown in Fig. 2, a TSN switch maintains a set of queues to transmit flows from various traffic classes. In this work, we focus on the

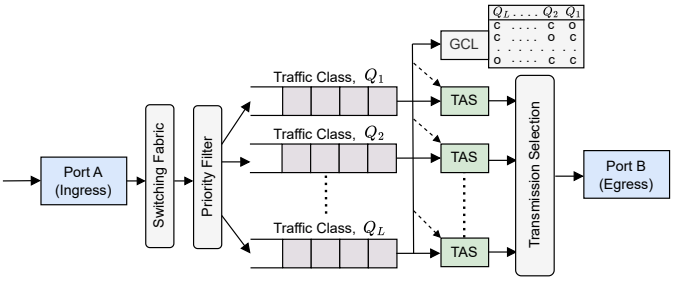


Fig. 2. High-level illustration of a TSN switch with queues regulated by the shaper (TAS). The "O" and "C" entries in GCL represent the gates are open and close, respectively.

IEEE 802.1Qbv (TAS), which enforces time-triggered access to each egress port of a switch. TAS associates every egress queue with a gate whose open/close state is controlled by a list (called GCL). The GCL specifies a cyclic sequence of time intervals, each defining which queues are permitted to transmit during that interval. When a queue's gate is closed, packets in that queue are prevented from transmission (thus eliminating contention). By carefully configuring GCLs, we can ensure conflict-free transmission windows.

A. Traffic Flows

We consider a TSN switch that is overloaded or has high utilization, and *all* packets of each flow may not meet their timing constraints (deadlines). However, the flows have "weakly-hard" requirements, meaning that a (bounded) deadline miss is acceptable. Consider a set of N flows, denoted by $\mathcal{F}$, routed through a TSN switch π_s. Each flow $F_i \in \mathcal{F}$ is characterized by the following tuple:

$$F_i = (e_i, D_i, T_i, (m_i, k_i), cls_i), \tag{1}$$

where e_i is the worst-case transmission time at the TSN egress port which includes transmitting an entire frame (payload and headers), D_i is the relative deadline, T_i is the inter-arrival time (period), (m_i, k_i) is the weakly-hard requirement, and cls_i is the flow traffic class. We consider a stringent weakly-hard constraint that considers the distributions of mandatory and optional instances [7]. Specifically, we define two parameters $(w_i,\ h_i)$, where w_i is the maximum number of consecutive packet drops permitted and h_i denotes the minimum number of consecutive successful transmissions that must follow before another packet drop can occur. The conversion from $(m_i,\ k_i)$ to $(w_i,\ h_i)$ is defined by [7]: $w_i = \max\left(\left\lfloor \frac{m_i}{k_i - m_i} \right\rfloor, 1\right)$ and $h_i = \left\lceil \frac{k_i - m_i}{m_i} \right\rceil$.

The transmission time of F_i at the egress port of switch π_s is defined as $e_i = \frac{\hat{l}_i}{\hat{R}_{\pi_s}}$, where $\hat{l}_i$ is the maximal frame size (payload plus header), and $\hat{R}_{\pi_s}$ denotes the transmission rate at the egress port of switch π_s. The flows generate packets periodically. We use p_{ij} to denote the j^{th} instance of a flow F_i. Hence, $e_{p_{ij}}$ represents the worst-case transmission time of the packet p_{ij}. The arrival time of packet p_{ij} from flow F_i is denoted by $a_{p_{ij}} = (j-1)T_i,\ \forall j \in \mathbb{N}$ and must be processed before its absolute deadline $d_{p_{ij}} = a_{p_{ij}} + D_i$.

We divide the set of all packets into two categories: *(a) mandatory* packets—these packets must meet deadline requirements and *(b) optional* packets—those that are served in a "best-effort" manner only if timing constraints of other mandatory packets are not violated. The condition $m_i = 0$ and $k_i = 1$ implies F_i has *hard* requirements (i.e., all packets must meet deadlines).

B. Switch Queues and Flow Priorities

A TSN egress port comprises a set of hardware traffic classes [2]. Each traffic class is assigned to a dedicated FIFO queue in the switch denoted by the set $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_L\}$. The TSN shaper (TAS in our context) controls packet transmission by opening and closing the corresponding queue gates. Packets generated by flows are mapped to these queues using the IEEE 802.1Q Priority Code Point (PCP) field based on their assigned traffic class [8]. The transmission of the packet is *non-preemptive*, that is, a packet can start its emission only if it can be fully transmitted before the next gate closing.

By assumption, each flow F_i is assigned a unique traffic class cls_i. We define the traffic class of packet p_{ij} as $\{Q_{p_{ij}} = p_{ij} | \, cls_i = Q_k\}$, which implies that each packet of F_i is routed through the same queue. We assume that the traffic class assigned to each flow is given. We make no assumptions about how this mapping is performed, as flow classes are typically determined by application-level requirements and are often not subject to change. For example, avionics data flows (e.g., flight-state sensing and monitoring streams) are commonly assigned to higher-priority traffic classes, whereas non-critical traffic (e.g., in-cabin infotainment or lighting) is assigned to lower-priority classes.

We split the queues into two disjoint sets: $\mathcal{Q}_M = \{Q_1, Q_2, \ldots, Q_{L-1}\}$ and $\{Q_L\}$, where $\mathcal{Q} = \mathcal{Q}_M \cup \{Q_L\}$. The first set of queues (i.e., $\mathcal{Q}_M$) is used for transmitting mandatory packets. One dedicated queue, Q_L, is reserved for optional packets. As all optional packets are assigned to Q_L, we additionally assign a weight, λ_i, to each optional packet of flow F_i. This allows the designers to prioritize the scheduling of optional packets based on application requirements.

Given the (w_i, h_i) requirements, the set of mandatory packets of flow F_i is given by: $\mathcal{P}_{M_i} = \{p_{ij} \mid ((j-1) \bmod x_i) + 1 \leq h_i\}$, where $x_i = w_i + h_i$. Packets are assigned to a queue $Q_k \in \mathcal{Q}_M$ based on flow class (i.e., $Q_k = cls_i$). Likewise, we define the optional packets of F_i as follows: $\mathcal{P}_{O_i} = \{p_{ij} \mid ((j-1) \bmod x_i) + 1 > h_i\}$. Hence $\mathcal{P}_M = \cup_{\forall F_i}\{\mathcal{P}_{M_i}\}$ and $\mathcal{P}_O = \cup_{\forall F_i}\{\mathcal{P}_{O_i}\}$ denote the set of all mandatory and optional packets from all flows, respectively. Optional packets that cannot be accommodated will be dropped using one of the available switch-level mechanisms. The possible options are discussed in Section VI. Recall that *all* optional packets are assigned to a dedicated queue, Q_L. As mentioned earlier, TSN switches use the PCP field to map the packets to a traffic class (queue) at the egress port. When a flow transmits h_i packets, the PCP value is set to the class for Q_L. Hence, the remaining (optional) packets can be forwarded to Q_L. After w_i optional packets are generated and processed, the original PCP value will be restored to serve the next h_i mandatory instances.

C. Analysis Window

For periodic real-time scheduling, it is often sufficient to analyze packet transmissions over a fixed time interval, called the *hyperperiod*. The hyperperiod, denoted by H, is the least common multiple (LCM) of all flow periods. A flow F_i generates $\frac{H}{T_i}$ packets in the hyperperiod. However, to properly evaluate (w_i, h_i) relations, we must analyze at least $x_i = w_i + h_i$ packets per flow, which may be higher than $\frac{H}{T_i}$. Hence, we define a new length of time, called *analysis window*, A, which is defined as:

$$A = \begin{cases} \text{LCM}(T_1, \ldots, T_N), & \text{if } \forall i : (\frac{H}{T_i}) \bmod x_i = 0 \\ \text{LCM}(x_1 T_1, \ldots, x_N T_N), & \text{otherwise.} \end{cases} \quad (2)$$

In Eq. (2), LCM$(\cdot)$ operator denotes the LCM of input arguments. When $\frac{H}{T_i} \bmod x_i = 0$, it is sufficient to analyze the schedule for the hyperperiod duration. Otherwise, the time length used for the schedulability analysis should be the LCM of all $x_i T_i$ values. As an example, consider two flows F_1 and F_2 with $T_1 = 3$, $T_2 = 5$, $w_1 = w_2 = h_1 = h_2 = 1$, and $x_1 = x_2 = 1 + 1 = 2$. In this example, 1st, 3rd, 5th, 7th (and so on) packets from F_1 and F_2 are mandatory. Here, $H = 15$ and if we check the condition: $\frac{H}{T_1} \bmod x_1 \neq 0$ and $\frac{H}{T_2} \bmod x_2 \neq 0$. Since, in this case, an optional packet is followed by a mandatory one, we must consider mandatory and optional packets as a pair. For instance, if we stop our analysis at $t = 15$, the flow F_1 generates three mandatory packets and two optional packets, which misses the mandatory packet from the last pair. Hence, as $A \neq H$ for this case, we should check *until* $A = \text{LCM}(x_1 T_1, x_2 T_2) = 30$. At $t = 30$, the 11th packet of F_1 and the 7th packet of F_2 are released, where both represent mandatory packets. Hence, the packets that arrive before $t = 30$ complete the first cycle, and $t = 30$ marks the start of the next cycle. Appendix A presents a formal proof for the analysis window.

III. Problem Overview

We develop a design-time scheduling technique for weakly-hard real-time flows routed through a TSN switch. At the output of each queue, packet transmission is determined by opening and closing gates for each traffic class. The TSN shaper (TAS) is responsible for controlling transmission using GCL. We consider the problem of determining the GCLs for TSN switch queues such that all mandatory packets meet their deadlines while maximizing the number of optional packet service rates. The packets in each queue are served in FIFO order. However, when multiple packets are ready for scheduling across several queues, we prioritize them based on their deadlines (i.e., following deadline-monotonic order). For example, if p_{ia}, p_{jb}, and p_{kc} are ready at Q_1, Q_2, and Q_3, respectively, at the same time and $d_{p_{jb}} < d_{p_{ia}} < d_{p_{kc}}$, then the gate of Q_2 will open first, then Q_1, and finally, Q_3. Following existing TAS design [9]–[12], we assume that gate openings

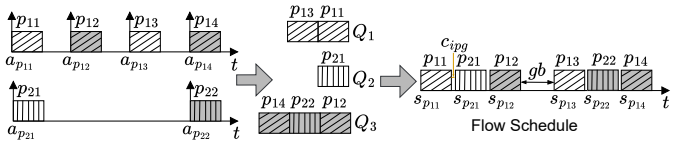


Fig. 3. High-level switch operation. Flows from different traffic classes are placed on their respective queues. We place all optional packets (darker shades) in a separate queue (Q_3 in this illustration). Packet transmissions are separated by inter-packet gap duration (denoted by c_{ipg}). If an optional packet is scheduled before a mandatory one, a guard band (gb) is used to ensure non-interference.

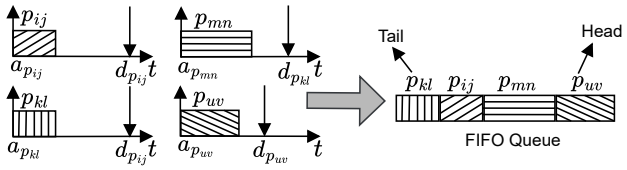


Fig. 4. Illustration of tie-breaking for exact flow arrival cases. Four packets p_{ij}, p_{kl}, p_{mn}, and p_{uv} arrive at the same time. The packet p_{uv} is enqueued first (earliest deadline), then p_{mn} (longest execution requirement), followed by p_{ij} (lower ID among remaining), and finally p_{kl}.

are non-overlapping; that is, only a single queue has an open gate at any given time.

The GCL specifies a periodic, time-triggered schedule that determines the intervals during which the gate of a particular flow class (queue) is open or closed. Hence, for each queue, the GCL represents a collection of non-overlapping time intervals during which the gate is open. We define the GCL of Q_k as $\mathcal{G}_{Q_k} = \bigcup_{l=1}^{n_k} \left[go^l_{Q_k},\ gc^l_{Q_k} \right)$. The interval $\left[go^l_{Q_k},\ gc^l_{Q_k} \right)$ specifies the l^{th} instance in which the gate of Q_k is open at time $go^l_{Q_k}$ and remains open until $gc^l_{Q_k}$. The variable n_k denotes the total number of intervals within the analysis window A. Hence, our goal is to determine $\mathcal{G}_{Q_k}$ for each queue Q_k such that the deadlines for all mandatory packets from $\mathcal{Q}_M$ are retained and the number of optional packets served in Q_L is maximized.

A. Weakly-Hard TSN Flow Scheduling

We begin with a simple example to illustrate the scheduling of real-time flows with weakly-hard constraints. Consider two flows F_1 and F_2 with $(w_i, h_i) = (1, 1)$ as shown in Fig. 3. Packets p_{11}, p_{12}, p_{13}, and p_{14} are from F_1. Flow F_2 generates packets p_{21} and p_{22}. Let the switch have three queues Q_1, Q_2, and Q_3, where Q_3 is reserved for holding optional packets. The mandatory packets from each flow are assigned to their respective queues (i.e., $p_{11}, p_{13} \in Q_1$ and $p_{21} \in Q_2$). The optional packets are assigned to the other queue (i.e., $p_{12}, p_{14}, p_{22} \in Q_3$). The packets are ordered in a FIFO queue by arrival time. The idle interval between the transmission of two successive packets is called the *inter-packet gap*, which is required for the physical-layer switch operation [13]. We use c_{ipg} to denote the inter-packet gap. Since mandatory packets must finish transmission before their deadlines, all packets in Q_1 and Q_2 are scheduled first. The packet p_{11} is scheduled before p_{21} as $d_{p_{21}} > d_{p_{11}}$. Once all the mandatory packets are scheduled (e.g., gate open durations are identified), we calculate the available intervals (slacks) for transmitting optional packets. If we want to schedule an optional packet before a mandatory one, we need to place a buffer (called *guard band*) to protect the mandatory packets from being delayed by the optional packets [8]. Guard band—equivalent to the Maximum Transmission Unit (MTU)—is a TSN feature that prevents lower-priority (or best-effort) traffic from overrunning into time slots reserved for time-critical packets. For instance, as we have an available slack between packet p_{21} and p_{13}, an optional packet p_{12} is scheduled in that interval, including a guard band (gb). After transmitting the mandatory packet p_{13}, the optional packets p_{22} and p_{14} can be scheduled without a guard band, since no mandatory packets remain to be transmitted. Note that an inter-packet gap is not required after the guard band, as the gates are closed for the gb duration, and we do not need any further padding [8]. To determine the flow scheduling order, we need to calculate the GCL for each queue. In the following section, we introduce an initial (non-optimal, yet effective) solution for determining the GCLs.

B. Handling Simultaneous Flow Arrivals

When multiple packets from the same traffic class arrive simultaneously (as can occur with periodic flows), it introduces nondeterminism in the GCL construction. Hence, we use three tie-breaking rules, which provide more fine-grained control.[1] The following rules, as depicted in Fig. 4, are applied sequentially, in the exact order in which they are written below.

- **Rule 1 (Deadline-based order).** When arrival times are the same, a packet with an earlier (absolute) deadline has higher priority (e.g., selected in a deadline-monotonic order).
- **Rule 2 (Execution-based order).** If two packets p_{ij} and p_{kl} have the same deadline, the packet requiring the longest transmission time (larger payload) is selected first.
- **Rule 3 (ID-based order).** If the above two conditions end up in a tie, the packet with a numerically smaller flow identifier receives higher priority (e.g., for F_i and F_k, if $i < k$, the packet from flow F_i receives priority).

Formally, for any two packets p_{ij} and $p_{kl} \in \mathcal{P}_M$, we consider $p_{ij} \prec p_{kl}$ (i.e., packet p_{ij} is chosen over packet p_{kl}) if the following conditions hold:

$$
\begin{aligned}
&\forall\, p_{ij},\, p_{kl} \in \mathcal{P}_M :\ p_{ij} \prec p_{kl} \Leftrightarrow \\
&(a_{p_{ij}} < a_{p_{kl}}) \ \vee \\
&(a_{p_{ij}} = a_{p_{kl}} \wedge d_{p_{ij}} < d_{p_{kl}}) \ \vee \ \text{/* Rule 1 */} \\
&(a_{p_{ij}} = a_{p_{kl}} \wedge d_{p_{ij}} = d_{p_{kl}} \wedge e_{p_{ij}} > e_{p_{kl}}) \ \vee \ \text{/* Rule 2 */} \\
&(a_{p_{ij}} = a_{p_{kl}} \wedge d_{p_{ij}} = d_{p_{kl}} \wedge e_{p_{ij}} = e_{p_{kl}} \wedge i < k) \ \text{/* Rule 3 */}
\end{aligned}
\tag{3}
$$

The above equation states that packets are sorted based on their arrival or, in the case of a tie (same arrivals), Rule 1–Rule 3 will be applied to decide the order in which they are considered for transmission.

[1] Other tie-breaking rules can be integrated without loss of generality.

C. Determining GCLs: Initial Attempt

Our primary objective is to guarantee deadlines for all the mandatory packets and *opportunistically* schedule the optional packets during slack times. This heuristic, termed the *Lazy Search* algorithm, incrementally identifies feasible gate-open intervals without exhaustively analyzing all packets within the analysis window.

Lazy Search works in three phases as follows. In the first phase (PHASE I), we order the packets into queues based on their traffic class and arrival times. If multiple flows arrive at the same time, we adopt a tie-breaking rule based on deadline, runtime, and flow identifier as noted in Section III-B. The next phase (PHASE II) identifies GCL for mandatory flows. We follow the deadline-monotonic order and determine the corresponding gate-open and gate-close times for each queue. After generating GCLs for all mandatory packets, the last phase (PHASE III) checks for slacks during the entire analysis window. Optional packets can be scheduled only if they can finish transmission within the slack interval without interfering with mandatory packets. If a GCL interval in Q_L precedes a mandatory packet, we add a guard band to ensure that sufficient buffer remains before the schedule of mandatory packets. A detailed description of the Lazy Search algorithm is presented next. Appendix B derives search complexity.

D. Lazy Search Heuristic

Our intention here is to maximize the scheduling of optional packets without missing any mandatory packet deadlines. Algorithm 1 formally introduces the Lazy Search heuristic. PHASE I uses pre-defined rules to calculate flow ordering for each queue (Lines 5–7). PHASE II finds the candidate packets at each epoch t within the analysis window A, with the intention that all mandatory packet deadlines are met, although this is not guaranteed (Lines 8–28). Finally, PHASE III searches for available intervals (slacks) in between the scheduled mandatory packets (Line 30). Since all optional packets share the same queue, we schedule them within these intervals based on their arrival times and absolute deadlines (Lines 31–48). If all mandatory packets are completed before their deadline, the flow set is deemed *schedulable* (Lines 50–53).

Algorithm 1 takes the set of all packets, $\mathcal{P}_M \cup \mathcal{P}_O$, as input, and generates the schedule (GCL), which contains gate opening and closing times $\mathcal{G}_{Q_k}$ for each queue Q_k. We now describe the three phases of Algorithm 1 in detail.

1) PHASE I—*Flow Ordering:* TSN switches maintain a FIFO queue for each traffic class [8]. The packets are enqueued in the order of their arrival. Optional packets are placed in Q_L according to their arrival time. When multiple optional packets (from different traffic classes) arrive at the same time, we first check for their designer-given weights λ_k to prioritize them. As depicted in Fig. 4, for equal weights, we use Rule 1–Rule 3 for tie-breaking (see Section III-B).

2) PHASE II—*GCL Construction for Mandatory Packets:* All mandatory packets from $\mathcal{Q}_M$ must retain their deadlines. We follow the deadline-monotonic priority order for GCL construction. The set of packets from each queue head is defined as:

$$\mathcal{P}_{head} = \{p_{ij} \mid p_{ij} \text{ is the head packet of } Q_k, \forall Q_k\}. \quad (4)$$

Given the set of packets from each queue head, we find the

Algorithm 1 GCL Construction using Lazy Search

Input: The set of all packets, $\mathcal{P}_M \cup \mathcal{P}_O$
Output: GCL $\mathcal{G}_{Q_k}, \forall Q_k$ and a list of admissible optional packets
1: Get mandatory and optional packet lists $\mathcal{P}_M$ and $\mathcal{P}_O$
2: **for** each queue Q_k **do**
3: $\mathcal{G}_{Q_k} \leftarrow \emptyset$, $n_{Q_k} \leftarrow 0$ /* *Initialize GCL and interval counter* */
4: **end for**
5: **PHASE I: Determine Flow Ordering**
6: For $\forall p_{ij} \in \mathcal{P}_M$: sort packets by using Eq. (3)
7: For $\forall p_{ij} \in \mathcal{P}_O$: sort packets based on arrival, tie-break using λ_i and Rule 1–Rule 3
8: **PHASE II: GCL Construction for Mandatory Packets**
9: Set current time, $t \leftarrow 0$ and busy window, $B \leftarrow \emptyset$
10: $\sigma \leftarrow \emptyset$ /* *Initialize scheduled packet set* */
11: /* *Select candidate packet for each decision epoch t* */
12: **while** $t < A$, $p_{ij} \in \mathcal{P}_M$ and $\mathcal{P}_M \neq \emptyset$ **do**
13: $\mathcal{P}_{\text{head}} \leftarrow \{p_{ij} \mid p_{ij}$ is the head packet of $Q_k, \forall Q_k \in \mathcal{Q}_M\}$
14: $\mathcal{E}_p(t) \leftarrow \{p_{ij} \in \mathcal{P}_{\text{head}} \mid a_{p_{ij}} \leq t\}$
15: **if** $\mathcal{E}_p(t) = \emptyset$ **then** $t \leftarrow \min\{a_{p_{ij}} \mid a_{p_{ij}} > t\}$ **continue**
16: **end if**
17: /* *Tie-breaking using Rule 1–Rule 3* */
18: $p_{ij}^{tx}(t) \leftarrow \text{argmin}_{p_{ij} \in \mathcal{E}_p(t)}\{d_{p_{ij}}, -e_{p_{ij}}, i\}$
19: /* *Update GCL interval counters* */
20: $Q_k \leftarrow Q_i$, $l \leftarrow n_{Q_k} + 1$, $n_{Q_k} \leftarrow n_{Q_k} + 1$
21: $s_{p_{ij}} \leftarrow t$ /* *Update packet start time* */
22: $go^l_{Q_k} \leftarrow t$ /* *Set gate open time* */
23: $gc^l_{Q_k} \leftarrow t + e_{p_{ij}}$ /* *Set gate close time* */
24: $t \leftarrow gc^l_{Q_k} + c_{ipg}$ /* *Update next transmission time* */
25: /* *Update GCL, busy window, and scheduled packet set* */
26: $\mathcal{G}_{Q_k} \leftarrow \mathcal{G}_{Q_k} \cup \{[go^l_{Q_k}, gc^l_{Q_k})\}$
27: $B \leftarrow B \cup \{[go^l_{Q_k}, gc^l_{Q_k})\}$, $\sigma \leftarrow \sigma \cup \{p_{ij}^{tx}(t)\}$
28: **end while**
29: **PHASE III: GCL Construction for Optional Packets**
30: $\mathcal{I} \leftarrow [0, A] \setminus B$ /* *Calculate slacks between mandatory packets* */
31: **for** each packet $p_{ij} \in \mathcal{P}_O$ **do**
32: **for** each interval $[\hat{s}, \hat{e}] \in \mathcal{I}$ **do**
33: /* *Check for mandatory packets after the interval* */
34: *successor* $\leftarrow (\hat{e} < A) \wedge (\exists p_{uv} \in \mathcal{P}_M : s_{p_{uv}} = \hat{e})$
35: /* *Determine if a guard band is needed* */
36: **if** *successor = True* **then** $gb_{duration} \leftarrow gb$
37: **else** $gb_{duration} \leftarrow 0$
38: **end if**
39: $go^l_{Q_L} \leftarrow \max(a_{p_{ij}}, s)$, $s_{p_{ij}} \leftarrow go^l_{Q_L}$
40: $gc^l_{Q_L} \leftarrow go^l_{Q_L} + e_{p_{ij}}$
41: /* *Schedule only if fits and meets deadline; otherwise drop* */
42: **if** $s_{p_{ij}} + e_{p_{ij}} + gb_{duration} \leq e$ **and** $gc^l_{Q_L} \leq d_{p_{ij}}$ **then**
43: /* *Update GCL, busy window, scheduled packet set* */
44: $\mathcal{G}_{Q_L} \leftarrow \mathcal{G}_{Q_L} \cup \{[go^l_{Q_L}, gc^l_{Q_L})\}$; $l, n_{Q_L} \leftarrow n_{Q_L} + 1$
45: $B \leftarrow B \cup \{[go^l_{Q_L}, gc^l_{Q_L})\}$; $\sigma \leftarrow \sigma \cup \{p_{ij}\}$ **break**
46: **end if**
47: **end for**
48: **end for**
49: /* *Check whether all mandatory packets meet their deadlines* */
50: **if** $\exists p_{ij} \in \mathcal{P}_M : s_{p_{ij}} + e_{p_{ij}} > d_{p_{ij}}$ **then**
51: **return** UNSCHEDULABLE
52: **end if**
53: **return** $\mathcal{G}_{Q_k}, \forall Q_k$ and $\{\sigma \cap \mathcal{P}_O\}$ /* *GCL and selected packets* */

candidate packet to be scheduled at each decision epoch t as:

$$\mathcal{E}_p(t) = \{p_{ij} \in \mathcal{P}_{head} \mid a_{p_{ij}} \leq t\}. \quad (5)$$

The set $\mathcal{E}_p(t)$ contains the packets from the queue head that are available to be scheduled at time t. If no packet is available at time t (i.e., $\mathcal{E}_p = \emptyset$), then the next transmission time t' is the earliest arrival time among the head packet set and it is defined as: $t' = \min\{a_{p_{ij}} \mid a_{p_{ij}} > t',\ \forall p_{ij} \in \mathcal{P}_{head}\}$.

When $\mathcal{E}_p(t)$ contains multiple packets, we must select a candidate packet for transmission, as only one packet can be transmitted through an egress port of a TSN switch at a given time. Hence, the candidate packet selected for transmission at time t is obtained by the following condition:

$$p_{ij}^{tx}(t) = \underset{p_{ij} \in \mathcal{E}_p(t)}{\operatorname{argmin}} \{d_{p_{ij}}, -e_{p_{ij}}, i\}, \quad (6)$$

i.e., we follow Rule 1–Rule 3 to determine which packet is selected for transmission.

Once we know the candidate packet $p_{ij}^{tx}(t)$, we open the gate for transmitting it. Let $p_{ij}^{tx}(t) \in Q_k$ and time t belongs to the l^{th} gate interval for Q_k. The gate-open and gate-close times are then set to $go_{Q_k}^l = t$ and $gc_{Q_k}^l = t + e_{p_{ij}}$, respectively. We also maintain a counter, n_{Q_k}, to keep track of the number of open intervals in each queue. The interval $[go_{Q_k}^l, gc_{Q_k}^l)$ represents the reserved GCL for packet $p_{ij}^{tx}(t)$.

According to the TSN standard, after a packet departs, the egress port should idle for an inter-packet gap duration (denoted as c_{ipg}) before the subsequent transmission can start. Hence, the next earliest time at which the egress link becomes available for transmission is: $t' = gc_{Q_k}^l + c_{ipg}$. The above process is repeated until all mandatory packets have been assigned a GCL. Once the packet is transmitted, it will be added to the scheduled packet set σ.

3) PHASE III—*Scheduling Optional Packets:* Recall that we aim to maximize the service rate for optional packets. We do so by identifying slack intervals in which optional packets can be accommodated. After all mandatory packet gate intervals are generated from PHASE II, we calculate the remaining available slots $\mathcal{I}$, where $\mathcal{I} = [0, A] \backslash B$. The variable B denotes *busy window*, i.e., the intervals of time when the egress port is unavailable due to the transmission of a mandatory packet. For each available interval $[\hat{s}, \hat{e}] \in \mathcal{I}$, where $[\hat{s}, \hat{e}]$ denotes the start and end time of an interval, respectively, we first check if the interval is in between any open GCL interval reserved for mandatory packets. If a mandatory packet is scheduled after the interval $[\hat{s}, \hat{e}]$, we place a guard band to ensure the optional packet does not collide with the transmission of the following mandatory packet. Figure 3 shows the use of a guard band. For instance, when an optional packet from Q_3 is scheduled between the intervals of two mandatory packets, a guard band (of MTU size) is inserted. If no mandatory packet is scheduled after the optional packet, no guard band is required. To schedule an optional packet, the execution time of the packet, together with the guard band, must be less than or equal to the length of the available interval; otherwise, the packet will be dropped.

After obtaining the GCL for all mandatory and optional packets, we verify whether the gate-close times of all mandatory packets occur before their deadlines. If this condition holds, the flow set is considered *schedulable* under its weakly-hard requirements.

E. Limitations of Lazy Search

The Lazy Search technique introduced above serves as an initial, heuristic-based approach for synthesizing GCLs. This method provides a practical starting point by incrementally constructing feasible schedules for mandatory and optional packets without the computational burden of exhaustively exploring the entire scheduling space. While such heuristics are valuable for providing insights into GCL construction, they inherently lack mathematical guarantees of optimality. For instance, because GCLs are created using a first-fit strategy, there is no guarantee that the timing of all mandatory packets will be satisfied during construction. Hence, for a given flow set, there may exist a feasible GCL that can be derived by another algorithm $\hat{\mathcal{A}}$—if such an algorithm is known—that Lazy Search may fail to find.

IV. GCL CONSTRUCTION USING OPTIMIZATION

To overcome the limitations of heuristic search, we aim to construct an optimization-based formulation using Integer Linear Programming (ILP). The ILP approach systematically explores the solution space to identify GCL intervals that satisfy all mandatory deadlines while maximizing the service rate of optional packets. The resultant GCLs are "optimal" for a *given* mandatory and optional set of packets $\mathcal{P}_M$ and $\mathcal{P}_O$. The *objective* of our optimization formulation is to maximize the number of optional packet transmissions. We also have the following *constraints*: *(a) temporal constraints* (packets must be scheduled after arrival and meet their deadlines), *(b) order constraints* (packets belonging to the same traffic class are considered for scheduling as they arrive), *(c) non-preemption constraints* (all packets must wait until a currently serving packet departs from the egress), and *(d) non-overlapping constraints* (as the switch can dispatch one packet at a time, only a single queue gate is open at a given time).

Although ILP problems are generally considered NP-hard, the GCL construction problem developed here can be solved using existing techniques such as branch-and-bound [14] by off-the-shelf optimization solvers (e.g., Gurobi [15], GLPK [16], etc.). We now introduce the mathematical formulations of the objective function and the constraints.

A. Optimization Formulation: Objective

Let $\delta_{p_{ij}}$ be a binary decision variable, where $\delta_{p_{ij}} = 1$ implies the packet p_{ij} is scheduled and $\delta_{p_{ij}} = 0$ indicates the packet is dropped. For a feasible system, $\delta_{p_{ij}} = 1$ for $\forall p_{ij} \in \mathcal{P}_M$. The objective of our ILP formulation is to maximize the weighted number of optional packet transmissions, defined as:

$$\text{Maximize} \sum_{\forall p_{ij} \in \mathcal{P}_O} \lambda_i \delta_{p_{ij}}. \quad (7)$$

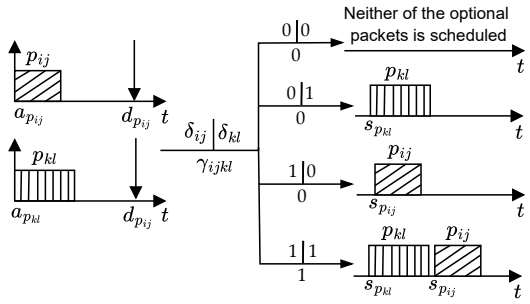


Fig. 5. Deterministic flow ordering for optional packets using decision variables $\delta_{p_{ij}}$, $\delta_{p_{kl}}$, and γ_{ijkl} when they arrive at the same time.

A higher weight λ_i indicates that the optional packets of F_i are more preferable for (best-effort) service guarantees; hence, they should be prioritized over other flows' optional packets.

B. Optimization Formulation: Constraints

Our optimization formulation includes four constraints, as we present below.

1) Temporal Constraints: A packet $p_{ij} \in \{\mathcal{P}_M \cup \mathcal{P}_O\}$ can only be scheduled after its arrival, i.e.,

$$s_{p_{ij}} \geq a_{p_{ij}} \delta_{p_{ij}}, \tag{8}$$

where $s_{p_{ij}}$ is the start time of p_{ij}. We note that $s_{p_{ij}}$ is an unknown parameter (to be derived from the optimizer). Further, the packets must depart from the egress port before their deadlines. We present this constraint as follows:

$$s_{p_{ij}} + e_{p_{ij}} \leq d_{p_{ij}} + M(1 - \delta_{p_{ij}}), \tag{9}$$

where M is a sufficiently large constant. We use the variable M to relax the deadline constraint for the ILP optimizer when it cannot schedule an optional packet (e.g., $\delta_{p_{ij}} = 0$). Note that, by definition, for any mandatory packet $p_{ij} \in \mathcal{P}_M$, the decision variable $\delta_{p_{ij}} = 0$ implies an infeasible configuration.

2) Flow Ordering in Same Queue: Each traffic class maintains a FIFO queue. Note that for packets from same flow, $a_{p_{ij}} > a_{p_{il}}$ when $j > l$. The optimizer selects a pair of packets and tries to determine their starting time when the gate can be opened. For two packets p_{ij} and p_{kl} with $a_{p_{ij}} < a_{p_{kl}}$, the FIFO constraint is represented as follows:

$$s_{p_{ij}} + e_{p_{ij}} + c_{ipg} \leq s_{p_{kl}}, \tag{10}$$

where if $j = l$ then $i \neq k$ (i.e., packets from different flows) and $i = k$ implies $j \neq l$ (i.e., different packets from the same flow). The constraint in Eq. (10) ensures that due to the non-preemptive nature of transmission, the packet that arrives late (p_{kl}) can only begin after the completion of the preceding packet, p_{ij}, plus the additional inter-packet gap waiting time.

When multiple mandatory packets arrive simultaneously, we use Rule 1–Rule 3, as defined in Eq. (3), for tie-breaking. The analysis of optional packets for the same arrivals is complex because they can be dropped to accommodate the timing constraints of mandatory packets. For a pair of optional packets, there are four possibilities: both can be dropped or scheduled, or only one can be scheduled. Hence, we introduce

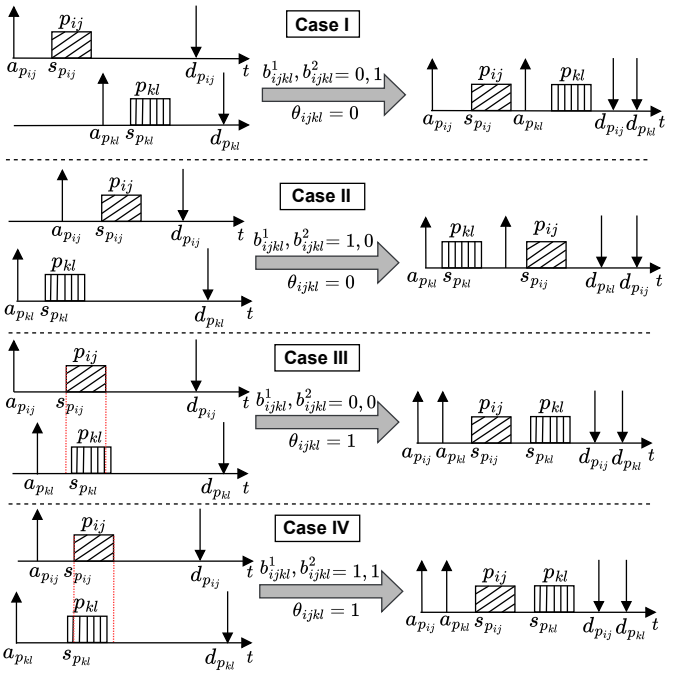


Fig. 6. Determining the packet start times from multiple queues using auxiliary variables b^1_{ijkl}, b^2_{ijkl}, and θ_{ijkl}.

an auxiliary binary variable $\gamma_{ijkl} \in \{0, 1\}$ that equals 1 if and only if both packets p_{ij} and p_{kl} are scheduled (i.e., $\delta_{p_{ij}} = \delta_{p_{kl}} = 1$), as depicted in Fig. 5. To relate these three decision variables $\delta_{p_{ij}}, \delta_{p_{kl}}, \gamma_{ijkl} \in \{0, 1\}$, we include the following conditions: *(a)* $\gamma_{ijkl} \leq \delta_{p_{ij}}$, *(b)* $\gamma_{ijkl} \leq \delta_{p_{kl}}$, and *(c)* $\gamma_{ijkl} \geq \delta_{p_{ij}} + \delta_{p_{kl}} - 1$. Therefore, if p_{ij} is selected before p_{kl} (denoted by $p_{ij} \prec p_{kl}$), the flow ordering constraint for optional packets is given by:

$$s_{p_{ij}} + e_{p_{ij}} + c_{ipg} \leq s_{p_{kl}} + M(1 - \gamma_{ijkl}), \quad \forall_{p_{ij}, p_{kl}} : p_{ij} \prec p_{kl}. \tag{11}$$

In the above equation, the large constant M is used to relax the constraints for dropped packet.

3) Flow Ordering Across Queues: Recall that, packet transmissions are non-preemptive. Consider two mandatory packets p_{ij} and p_{kl} from different flow class (queue), i.e., $cls_i \neq cls_k$. Assume $d_{p_{ij}} < d_{p_{kl}}$. Then, the following scenarios exist: *(a)* p_{ij} departs from egress before the p_{kl} starts (and vice-versa), and *(b)* their execution overlaps (e.g., p_{ij} starts while p_{kl} is ready (and vice-versa)). We represent these four cases with two binary variables b^1_{ijkl} and b^2_{ijkl}, as shown in Fig. 6. For instance, when $(b^1_{ijkl}, b^2_{ijkl}) = (0, 1)$, p_{ij} completes before the start of p_{kl}. Therefore, when the packet arrivals do not overlap (i.e., either b^1_{ijkl} or b^2_{ijkl} is 1), their start times are given by:

$$s_{p_{ij}} + e_{p_{ij}} + c_{ipg} \leq s_{p_{kl}} + M b^1_{ijkl} \text{ and} \tag{12}$$

$$s_{p_{kl}} + e_{p_{kl}} + c_{ipg} \leq s_{p_{ij}} + M b^2_{ijkl}. \tag{13}$$

The case $(b^1_{ijkl}, b^2_{ijkl}) = (0, 0)$ or $(b^1_{ijkl}, b^2_{ijkl}) = (1, 1)$ implies the packet execution overlaps. However, because transmission is non-preemptive, once p_{ij} starts, p_{kl} can

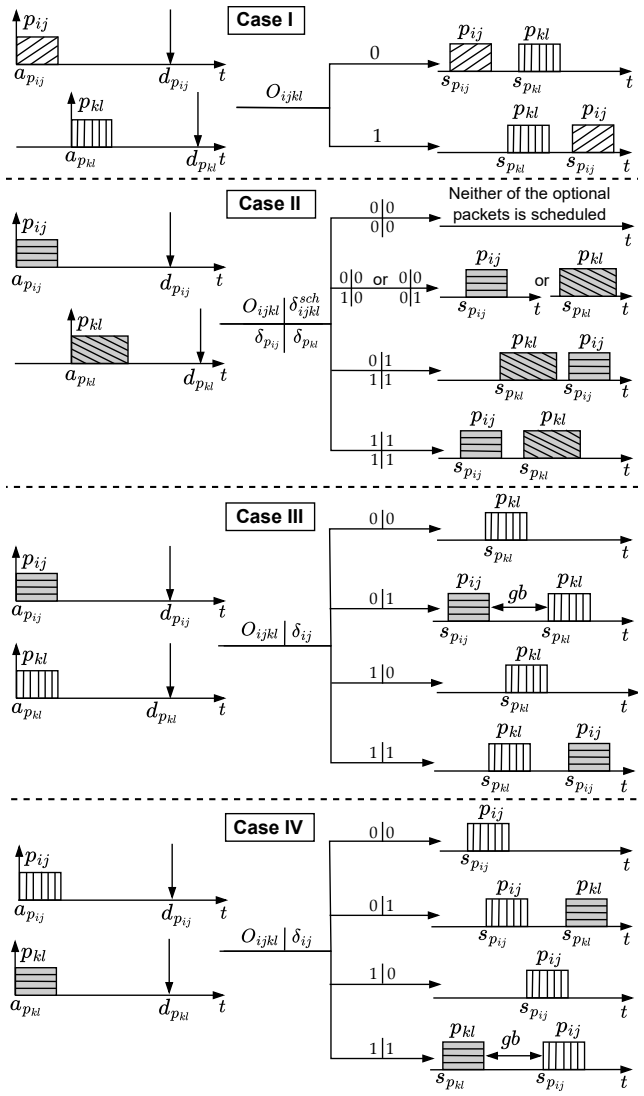


Fig. 7. Ordering of mandatory and optional packets. An optional packet can precede the mandatory only if we can place a buffer (guard band).

only be scheduled after p_{ij} departs. Hence, we use an auxiliary binary variable θ_{ijkl} to tackle this with the following conditions: *(a)* $b^1_{ijkl} + b^2_{ijkl} \geq 1 - \theta_{ijkl}$, *(b)* $b^1_{ijkl} + b^2_{ijkl} \leq 1 + \theta_{ijkl}$, *(c)* $b^1_{ijkl} - b^2_{ijkl} \leq 1 - \theta_{ijkl}$, and *(d)* $b^2_{ijkl} - b^1_{ijkl} \leq 1 - \theta_{ijkl}$. The variable $\theta_{ijkl} = 1$ implies there are overlaps in packet transmission, which should not be the case. Therefore, the mandatory packet start times from multiple queues are:

$$s_{p_{ij}} + e_{p_{ij}} + c_{ipg} \leq s_{p_{kl}} + M(1 - \theta_{ijkl}). \tag{14}$$

4) Ensuring Non-Overlapping Flow Executions: At any given time, only one packet can be transmitted through the TSN egress port [2]. To account for this hardware constraint, we define a binary variable $O_{ijkl} \in \{0, 1\}$ that controls the relative ordering between a pair of packets p_{ij} and p_{kl}. There are four cases: both packets are mandatory, one is mandatory and the other is optional, or both are optional. Each of these cases needs to be considered individually, as described below.

Case I: Both p_{ij} and p_{kl} are Mandatory. Let $O_{ijkl} = 0$ implies packet p_{ij} precedes packet p_{kl} (and vice versa for $O_{ijkl} = 1$), as depicted in Fig. 7 (see Case I). Hence, their start times are constrained by:

$$s_{p_{ij}} + e_{p_{ij}} + c_{ipg} \leq s_{p_{kl}} + MO_{ijkl} \text{ and} \tag{15}$$

$$s_{p_{kl}} + e_{p_{kl}} + c_{ipg} \leq s_{p_{ij}} + M(1 - O_{ijkl}). \tag{16}$$

Hence, Eq. (16) is the complementary case of Eq. (15) which considers the case when p_{kl} is selected to be transmitted first instead of p_{ij}.

Case II: Both p_{ij} and p_{kl} are Optional. When both packets are optional, there are several possible outcomes (see Fig. 7, Case II): *(a)* both can be scheduled, *(b)* either of them is scheduled, or *(c)* neither of them are scheduled. Hence we use a binary variable, δ^{sch}_{ijkl} to determine if both are scheduled or not. Specifically, the relation with δ^{sch}_{ijkl} and the other decision variables (i.e., $\delta_{p_{ij}}$ and $\delta_{p_{kl}}$) is given by: $\delta^{sch}_{ijkl} \leq \delta_{p_{ij}}$, $\delta^{sch}_{ijkl} \leq \delta_{p_{kl}}$, and $\delta^{sch}_{ijkl} \geq \delta_{p_{ij}} + \delta_{p_{kl}} - 1$. The above conditions imply that $\delta^{sch}_{ijkl} = 1$, if and only if both packets are scheduled. Then, the conditions for start times are given by:

$$s_{p_{ij}} + e_{p_{ij}} + c_{ipg} \leq s_{p_{kl}} + M(1 - O_{ijkl}) + M(1 - \delta^{sch}_{ijkl}) \text{ and} \tag{17}$$

$$s_{p_{kl}} + e_{p_{kl}} + c_{ipg} \leq s_{p_{ij}} + MO_{ijkl} + M(1 - \delta^{sch}_{ijkl}). \tag{18}$$

Case III: p_{ij} is Optional and p_{kl} is Mandatory. In this case, without careful selection of start times, p_{ij} can interfere with the mandatory packet p_{kl}. Hence, we need a guard band, as shown in Fig. 7 (see Case III). The constraints are given by:

$$s_{p_{ij}} + e_{p_{ij}} + gb \leq s_{p_{kl}} + MO_{ijkl} + M(1 - \delta_{p_{ij}}) \text{ and} \tag{19}$$

$$s_{p_{kl}} + e_{p_{kl}} + c_{ipg} \leq s_{p_{ij}} + M(1 - O_{ijkl}) + M(1 - \delta_{p_{ij}}). \tag{20}$$

When $O_{ijkl} = 0$, the optional packet p_{ij} is scheduled before the mandatory packet p_{kl} (i.e., $\delta_{p_{ij}} = 1$). This case, the constraints in Eq. (19) is used. We place a guard band (gb) after the transmission of the optional packet to prevent any interference with mandatory packet. The condition in Eq. (20) tackles the opposite ordering where mandatory packet p_{kl} is scheduled before the optional packet p_{ij}. For this case, we do not need a guard band.

Case IV: p_{ij} is Mandatory and p_{kl} is Optional. This is similar to Case III except the packet types are reversed (see Fig. 7, Case IV). The constraints are same as before:

$$s_{p_{ij}} + e_{p_{ij}} + c_{ipg} \leq s_{p_{kl}} + MO_{ijkl} + M(1 - \delta_{p_{kl}}) \text{ and} \tag{21}$$

$$s_{p_{kl}} + e_{p_{kl}} + gb \leq s_{p_{ij}} + M(1 - O_{ijkl}) + M(1 - \delta_{p_{kl}}). \tag{22}$$

C. GCL Construction

Once we obtain all start times $s_{p_{ij}}$ of p_{ij} belong to Q_k from the ILP optimizer, we can construct the GCL for each l^{th} segment as follows: $go^l_{Q_k} = s_{p_{ij}}$, $gc^l_{Q_k} = s_{p_{ij}} + e_{p_{ij}}$. The final schedule (GCL) is then given by: $\mathcal{G}_{Q_k} = \cup_{\forall l}\{[go^l_{Q_k}, gc^l_{Q_k})\}$.

V. Evaluation

We evaluate weakly-hard TSN flow scheduling on two fronts: *(a)* performance comparisons using synthetically generated flow sets—to analyze the design space (Section V-B) and *(b)* experiments with a real TSN switch—to test the behavior of the schedulers on hardware (Section V-C). Our implementation is **publicly available** [6].

A. Evaluation Metrics

We use the following metrics for our evaluation.

1) Schedulability Ratio (SR): This is a standard metric used for testing the performance of real-time schedulers and defined as follows: $SR = \frac{\mathcal{F}_{sched}}{\mathcal{F}_{total}}$, where $\mathcal{F}_{sched}$ is the number of schedulable flow set and $\mathcal{F}_{total}$ denotes the total flow sets. Recall that for a flow set to be schedulable, *all* mandatory packets must meet deadlines.

2) Optional Packet Admissibility Ratio (OPAR): The next metric calculates the fraction of admissible optional packets and is defined as follows: $OPAR = \frac{1}{\hat{N}} \sum_{i=1}^{\hat{N}} \frac{O_i^{sched}}{O_i}$, where $\hat{N}$ denotes the total number of scheduled flow sets, O_i^{sched} is the number of admissible optional packets in flow set i, and O_i represents the total number of optional packets in flow set i.

3) Computation Time: How much time it takes to get GCLs (or infeasibility results) from the optimization engine.

4) Normalized Response Time (NRT): This metric shows the "spread" of the flow, which is defined as: $NRT_i = \frac{R_i}{D_i}$, where R_i is the response time (difference between flow arrival and completion) and D_i is the deadline. $NRT_i > 1$ implies that the flow misses its deadline.

B. Experiments with Synthetically Generated Flows

1) Parameters and Experiment Setup: For each experiment, we randomly generated N periodic flows with a target system utilization, $U = \sum_{i=1}^{N} \frac{e_i}{T_i}$. Individual flow utilizations were generated using the UUnifast algorithm [17]. The flow periods were selected from $\{50, 100, 200, 400\}$. The deadlines were equal to the periods. The execution time is calculated as follows: $e_i = T_i U_i$, where, e_i was selected from $[600, 12000]$. We used these numbers to match the processing times of standard Ethernet frames (64-1522 bytes) on a 1 Gbps TSN egress port [18]. We used Gurobi [15] to solve the ILP.

2) Experiment #1—Comparing Lazy Search with ILP: In our first set of experiments (see Fig. 8), we compare the performance of Lazy Search and the ILP solution. We varied the number of flows, $N \in \{16, 32, 48\}$. For each N, the switch utilization U was varied from $\{0.4, 0.5, \ldots, 1.2\}$. For each pair of (N, U), we generated 100 independent flow sets. We assume the weakly-hard requirements are $(w, h) = (1, 2)$, i.e., one deadline miss is acceptable after two successful mandatory packet transmissions. The analysis window for this setup was 1200 (derived from Eq. (2)) and all flows have equal weights.

The schedulability and optional packet admissibility (SR and $OPAR$) for Lazy Search and optimization solution is presented in Fig. 8. Total utilization refers to the utilization of all mandatory and optional packets, not only the admitted optional packets. As expected, schedulability decreases with

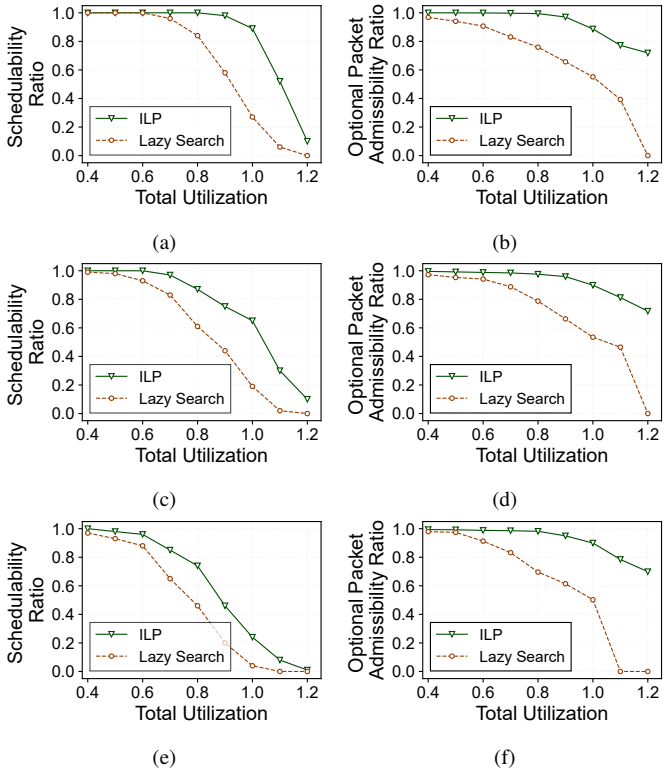


Fig. 8. Schedulability (left) and optional packet admissibility (right) under varying network utilization for $N = 16$ (top), $N = 32$ (middle), and $N = 48$ (bottom). Lazy Search—despite its efficient first-fit lookup—fails to discover all feasible GCL configurations. For optional packet admissibility, Lazy Search performs the worst, particularly in systems with a large number of flows.

higher utilization due to higher loads. In addition, the optimal solution outperforms the Lazy Search. This is because Lazy search locally explores gate intervals without considering the entire flow configuration. As the results show, the optimal solution finds more admissible packets than the heuristic search (Lazy). The performance gains are more pronounced for overloaded scenarios (i.e., when $U > 1$).

3) Experiment #2—Stress Testing the Optimizer: While the optimal solution shows performance gains, it comes with a cost: computational overheads and intractability. In the next set of experiments, we study the extent to which an optimal solution is computationally feasible to obtain. To test this, we vary the optimization problem space (e.g., number of constraints). Note from our formulation that the number of constraints depends on the number of packets. For this experiment, we used $N = 48$, $U = 0.8$, and $(w, h) = (1, 2)$, and varied the number of packets (constraints). Table I shows the relation between the number of packets and the number of constraints obtained for our synthetic flow sets. We repeated each packet-count configuration with 100 flow sets and even for a fixed number of packets, the number of constraints varies as they depend on packet types (e.g., the composition of mandatory and optional packets). We set a cutoff time of 1 hour to check whether we get *any* decision (schedulable or unschedulable) from the optimizer within this time window. The experiments were conducted on

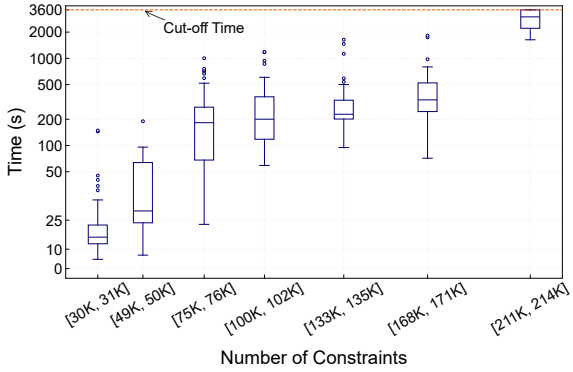


Fig. 9. Computation times to obtain optimal GCL configurations for various problem sizes. The experiments were run on an Intel Core i9 machine with 32 GB RAM.

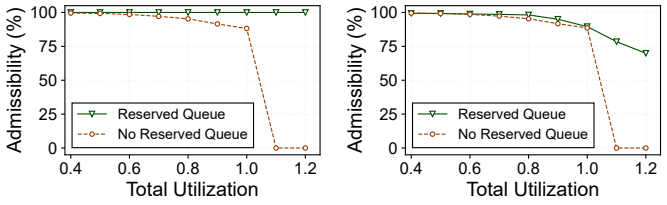
TABLE I
RELATION BETWEEN PACKET AND CONSTRAINT NUMBERS

Number of Packets	Number of Constraints
201	[30,470, 31,638]
252	[49,503, 50,605]
306	[75,022, 76,620]
351	[100,320, 102,105]
402	[133,520, 135,684]
450	[168,619, 171,102]
501	[211,465, 214,744]

a general-purpose computer (Intel Core i9 CPU and 32 GB RAM).

As expected, run times increase with a large number of constraints (see Fig. 9). Specifically, the average runtime increases from 20–30 seconds for 30K constraints to 45+ minutes for 210K constraints. For a smaller problem size (e.g., 200–250 packets, up to 50K constraints), the median runtime to get an optimal GCL is less than 30 s. We observed that even for a moderately large setup (~100K constraints), the typical runtime is 3–6 minutes on a mid-range computer. Hence, an optimal GCL can be achieved without a significant computational burden. For completeness, we also measured the runtime of the Lazy Search algorithm on the same hardware, which ranges from 0.21 s to 0.50 s for 201–500 packets.

4) Experiment #3—Impact of Weight on Optional Packet Admissibility: Recall that we use weights to assign relative priority for optional packets. Network designers can assign these weights based on application requirements to prioritize the service rate for optional packets from targeted flow classes. We now analyze how the weights influence GCL construction and the selection of optional packets. For this experiment, we set $U = 1$ and $N = 48$. Half of the flows (24) have $(w_i, h_i) = (1, 2)$ and the rest have $(w_i, h_i) = (1, 1)$. We experimented with three cases: *(a)* all flows have equal weight [Configuration 1], *(b)* flows with $(w_i, h_i) = (1, 2)$ have higher weight than flows with $(w_i, h_i) = (1, 1)$ [Configuration 2], and *(c)* flows with $(w_i, h_i) = (1, 1)$ have higher weight than flows with $(w_i, h_i) = (1, 2)$ [Configuration 3]. For each configuration, we repeated the experiments with 100 flow sets and report the average. Table II shows our findings. Regardless of



(a) Mandatory Packets (b) Optional Packets

Fig. 10. RQ vs. NRQ: NRQ minimizes response times without imposing hard timing requirements, which leads to reduced admissibility.

TABLE II
EFFECT OF WEIGHT ON OPTIONAL PACKET ADMISSION*

Configuration	Weighted OPAR (%)	Non-Weighted OPAR (%)	Total OPAR (%)
Conf. 1	–	–	82.71
Conf. 2	88.33	77.72	81.88
Conf. 3	89.00	62.10	78.91

*All weights are equal for Conf. 1; hence, the weighted and non-weighted OPAR columns are left blank.

weights, the percentage of admissible optional packets remains fairly unchanged. When weights are imposed, higher-weight flows—irrespective of their weakly-hard requirements—are favored during selection. Hence, by tuning the weights, network designers can adjust the service rate for certain flow classes during overload conditions.

5) Experiment #4—Evaluating Dedicated Queue Reservation for Optional Flows: One of our evaluation goals is to explore the design-space and assess *(a)* whether it is better to allocate a dedicated queue for optional packets or *(b)* keep them alongside their original queue. Hence, we develop another scheme that schedules mandatory and optional packets in the same queue based on their traffic class. To distinguish these two techniques, we term our original formulation in Section IV as *Reserved Queue (RQ)* and the latter one as *No Reserved Queue (NRQ)*. For NRQ, we attempt to reduce the response times of flows, with the aim of meeting the deadlines for all (or most) packets. One benefit of the NRQ scheme is that, for a switch with L queues, it can support L traffic classes, whereas the RQ scheme can support $L - 1$ classes, as one queue is reserved for serving optional packets.

For the NRQ scheme to work, we formulate an ILP optimization problem as follows: minimize $\sum_{p_{ij}} R_{p_{ij}}$, subject to Eq. (8), and Eqs. (10)–(22), where $R_{p_{ij}}$ is the response time of p_{ij}. Our rationale here is that the optimizer should be able to pick the "best" response times (and the corresponding GCLs) for the packets. If the response time is less than the deadline for all mandatory packets of all flows, the flow set is schedulable. From the GCL obtained from the solver, we then check for the response times of the flows and calculate how many mandatory and optional packets are *admissible* (i.e., $R_{p_{ij}} < d_{p_{ij}}$). Note that by formulation, the RQ scheme has 100% mandatory packet admissibility for a feasible case (e.g., when the optimizer returns a solution). The y-axis of Fig. 10 shows the percentage of admissible mandatory (left)

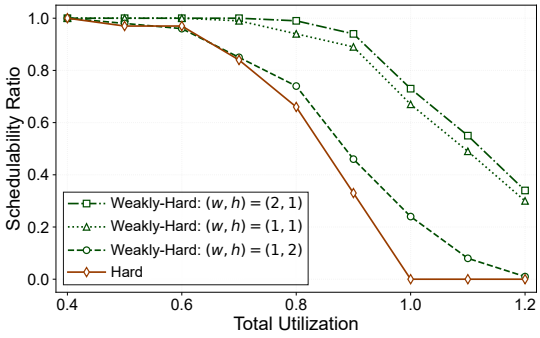


Fig. 11. Schedulability ratio under weakly-hard (dotted green) and hard (solid red) requirements. Stringent requirements reduce schedulability.

and optional (right) packets. In this experiment, we considered $N = 48$ and $(w, h) = (1, 2)$, and there were 100 flow sets for each utilization. As the figures show, NRQ performs poorly in high utilization and overloaded scenarios. This is because the NRQ scheme operates in a best-effort manner, attempting to minimize response times with the intention of meeting deadlines, even though it does not enforce any hard constraints.

6) Experiment #5—Studying Weakly-Hard Requirements: We also compare weakly-hard constraints with various (w, h) parameters to a configuration with "hard" requirements, namely $(w, h) = (0, 1)$. Conceptually, the hard requirements are similar to the NRQ setup with an added deadline constraint (i.e., Eq. (9)). Figure 11 presents the schedulability ratio (SR) as a function of total utilization for 48 flows with equal weights. As expected, allowing "tolerable" deadline misses improves flow admissibility when compared to hard real-time constraints (the red line in the figure). More stringent weakly-hard requirements—for instance, allowing one deadline miss over three consecutive transmissions, i.e., the $(w, h) = (1, 2)$ configuration—reduce SR.

C. Evaluation on a TSN Switch

We further experimented with a TSN switch to evaluate the performance of weakly-hard real-time flows on real hardware. For our evaluation, we use a TSN Real-Time HAT from InnoRoute [19], as shown in Fig. 12.

In this experiment, we considered 16 flows. The periods were set to $\{1, 2, 4, 16\}$ ms, with a total utilization of 1. Each flow has $(w_i, h_i) = (1, 2)$ weakly-hard requirements. From this configuration, we generated 216 packets, of which 144 are mandatory and 72 are optional. We tested the performance of both RQ and NRQ schemes. Based on the solutions returned by the optimizer, we configured the gate open and close times accordingly. The traffic flows were generated and dumped into a `.pcap` file. We then relayed the traffic trace using the Linux `tcpreplay` utility and pushed it to the switch using `tcpdump`. From the switch trace (ingress and egress timestamps), we calculated the packet response times. Specifically, we measured their normalized response time (NRT). Recall from the definition (Section V-A), $NRT > 1$ indicates a missed deadline.

Fig. 12. TSN switch used in our experiments.

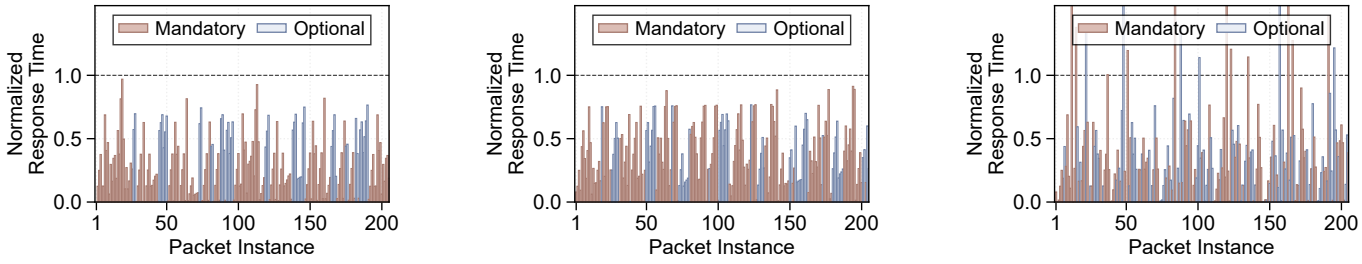
Figure 13 shows NRT of mandatory and optional packets for RQ and NRQ schemes. For RQ, when GCLs are obtained by Lazy Search (Fig. 13a), 36.11% optional packets cannot be accommodated. In contrast, when GCLs are being obtained by ILP (Fig. 13b), only 9.7% of the optional packets were discarded by the optimization decision. For both cases, all packets have $NRT < 1$, which ensures that no mandatory packet missed its deadline—consistent with the theoretical expectation for a feasible flow set. For the NRQ scheme (Fig. 13c), several mandatory (7.64%) and optional (8.33%) packets missed deadlines (e.g., $NRT > 1$) for the same flow set. We also observed a similar trend in our synthetic evaluations. These results stem from the design of the NRQ optimization: it cannot enforce hard requirements, and therefore the resulting schedules operate in a best-effort manner, which may lead to deadline misses for some instances.

VI. Discussion

We construct the mandatory packet set $\mathcal{P}_{M_i}$ using the (w_i, h_i) parameters derived from the (m_i, k_i) constraint. Although the (w, h) abstraction simplifies the schedulability analysis, a known and unavoidable consequence of transforming an (m, k) constraint into a (w, h) constraint is increased pessimism [7], [20]. Specifically, the (w, h) constraint is provably stricter than the (m, k) constraint: any task (flow in our context) that satisfies a (w, h) constraint also satisfies the corresponding (m, k) constraint, but not vice versa (see Lemma 3 in prior work [20]). For any given mandatory and optional packet sets, the GCL produced by our formulation preserves the original (m_i, k_i) semantics. For instance, once the ILP returns a feasible solution, Eq. (9) enforces deadlines for *all* mandatory packets by construction (and hence, ensures weakly-hard guarantees).

As a design-time tool, we do not make specific assumptions about how system-level enforcement mechanisms are realized at runtime. For example, designers may leverage IEEE 802.1Qci (PSFP) [8], which provides ingress admission control before packets enter the queues. Since the optional packets that must be dropped are known a priori through our design-time synthesis, we can configure PSFP stream gates to enforce these dropping decisions at the ingress and prevent non-admissible (optional) packets from entering the queues. Another option could be to use ingress filtering via programmable NIC [21] to extract scheduling attributes (e.g., the ordering criteria in Rules 1–3) and release packets in the desired order. Finally, it is also possible to suppress packet generation at the source; however, this approach is less practical, as switches typically do not have control over hosts or the applications that generate the flows.

In this work, we determine the schedule for a single switch. As flows are routed through the network, it is often necessary to know the end-to-end response times and



(a) RQ (Lazy Search) (b) RQ (ILP) (c) NRQ

Fig. 13. Normalized response times from the switch trace. RQ schemes (both Lazy Search and ILP) were able to meet all deadlines while NRQ missed some.

to determine a schedule for each switch along the flow path that preserves the weakly-hard semantics. Determining a network-wide global GCL is challenging, as it requires accounting for the dependency of flow arrivals on other switches. As future work, the proposed framework can be extended to support network-wide GCL construction.

VII. Related Work

Substantial research has been conducted on scheduling, optimization, and formal analysis for TSN. Researchers explore the problem of scheduling TSN traffic, primarily for hard real-time flows [22], [23]. TSNSCHED [24] is an SMT-based tool for schedule construction. Oliver et al. [25] encode scheduling constraints using first-order array theory to synthesize optimized GCLs. These efforts output deterministic schedules, but all assume a strictly hard real-time model in which every packet of every flow must meet its deadline.

Recent efforts have explored QoS and mixed-traffic scheduling in TSN. Houtan et al. [26] introduce models that improve best-effort service while ensuring feasibility for real-time traffic, showing that best-effort latency can be reduced without violating strict guarantees. A network calculus-based analysis is used to jointly analyze time-triggered, credit-based, and best-effort traffic, deriving delay and buffer bounds under different scheduling and preemption policies [27]. A recent study [28] evaluates several state-of-the-art TAS-based scheduling methods using both simulations and TSN testbeds. Although these works incorporate heterogeneous traffic classes, they do not incorporate weakly-hard constraints.

While real-time community has extensively studied weakly-hard models [5], [29], [7], they are primarily in the CPU scheduling context (see the review [30]). These works establish rigorous foundations for predictable deadline misses, but they target CPU/task-level timing behavior rather than network-level scheduling or TSN-specific constraints. Due to different CPU and TSN scheduling semantics, adapting existing task scheduling weakly-hard models for TSN is non-trivial. To the best of our knowledge, no prior work integrates weakly-hard requirements into TSN or synthesizes GCLs that explicitly differentiate mandatory and optional packets under bounded-miss constraints.

VIII. Conclusion

We introduce a design-time scheduling framework that integrates weakly-hard real-time semantics with TSN to enhance flow admissibility under overloaded conditions. Our results demonstrate that incorporating weakly-hard semantics into TSN is feasible and does not compromise the timing guarantees required by critical flows. With the optimal GCL construction developed in this work, real-time network engineers have a more informed basis for improving service rates in their applications without affecting the timing guarantees of critical traffic.

Appendix

A. Derivation of Analysis Window

Flow F_i releases exactly one packet every T_i time units. Let the k^{th} packet of F_i is released at time $r_i(k) = (k-1)T_i$, $k \in \mathbb{N}^+$. As noted before, each flow F_i follows a fixed mandatory/optional pattern with length $x_i = w_i + h_i$. For a flow F_i, a time shift by A advances the packet index of F_i by exactly $\frac{A}{T_i}$. Let $\ell_i := \frac{A}{T_i}$ and since $\frac{x_i T_i}{A} \in \mathbb{N}^+$, it follows that $\frac{x_i}{\ell_i} \in \mathbb{N}^+$. Therefore, for any packet index $k \in \mathbb{N}^+$, we have $(k-1+\ell_i) \bmod x_i = (k-1) \bmod x_i$. By the definition of $\mathcal{P}_{M_i}$ and $\mathcal{P}_{O_i}$, packet k is mandatory when $((k-1) \bmod x_i)+1 \le h_i$ and since $\frac{x_i}{\ell_i} \in \mathbb{N}^+$, the same condition holds for packet $k+\ell_i$. Hence, packet k and packet $k+\ell_i$ have the same type (e.g., mandatory/optional). Moreover, their release instances satisfy $r_i(k)+A = (k-1)T_i+A = (k-1+\ell_i)T_i = r_i(k+\ell_i)$. Thus, both release times and packet types (mandatory/optional) repeat exactly after time A for every flow.

B. Lazy Search Complexity

Sorting mandatory and optional packets in Phase I to derive their orders has total complexity $O(|\mathcal{P}_M| \log |\mathcal{P}_M| + |\mathcal{P}_O| \log |\mathcal{P}_O|)$. In Phase II, getting packets from the queue head and tie-breaking complexity is given by: $O(|\mathcal{P}_M||Q_M|)$. Since we have at most $(|\mathcal{P}_M| - 1)$ available intervals during Phase III, the complexity of determining opportunistic GCL intervals is $O(|\mathcal{P}_O||\mathcal{P}_M| \cdot 1) = O(|\mathcal{P}_O||\mathcal{P}_M|)$. Therefore, the total complexity of the Lazy Search algorithm:

$$
\begin{aligned}
&O(|\mathcal{P}_M| \log |\mathcal{P}_M| + |\mathcal{P}_O| \log |\mathcal{P}_O|) \\
&+ O(|\mathcal{P}_M| \cdot |Q_M|) + O(|\mathcal{P}_O| \cdot |\mathcal{P}_M|) \\
&= O(|\mathcal{P}_M| \log |\mathcal{P}_M| + |\mathcal{P}_O| \log |\mathcal{P}_O| + |\mathcal{P}_M||Q_M| + |\mathcal{P}_O||\mathcal{P}_M|).
\end{aligned}
$$

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers and the shepherd for their valuable comments and recommendations. This research was supported in part by the U.S. National Science Foundation Award 2345653. Any findings, opinions, recommendations, or conclusions expressed in the paper are those of the authors and do not necessarily reflect the view of the sponsor.

REFERENCES

[1] IEEE 802.1 Working Group, "Time-sensitive networking (TSN) task group." https://1.ieee802.org/tsn/. Accessed: 06-13-2025.

[2] "IEEE standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016.

[3] D. Lüdtke, T. Firchau, C. G. Cortes, A. Lund, A. M. Nepal, M. M. Elbarrawy, Z. H. Hammadeh, J.-G. Meß, P. Kenny, F. Brömer, *et al.*, "ScOSA on the way to orbit: Reconfigurable high-performance computing for spacecraft," in *2023 IEEE Space Computing Conference (SCC)*, pp. 34–44, IEEE, 2023.

[4] A. Lund, Z. A. Haj Hammadeh, P. Kenny, V. Vishav, A. Kovalov, H. Watolla, A. Gerndt, and D. Lüdtke, "ScOSA system software: The reliable and scalable middleware for a heterogeneous and distributed on-board computer architecture," *CEAS Space Journal*, vol. 14, no. 1, pp. 161–171, 2022.

[5] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE transactions on Computers*, vol. 50, no. 4, pp. 308–321, 2001.

[6] "Weakly-hard real-time flow scheduling in time-sensitive networks: Code repository." https://github.com/CPS2RL/TSN_Overload, 2026. Accessed: 03-03-2026.

[7] V. G. Moyano, Z. A. H. Hammadeh, S. Saidi, and D. Lüdtke, "Efficient scheduling of weakly-hard real-time tasks with sufficient schedulability condition," in *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*, pp. 541–550, 2025.

[8] "IEEE standard for local and metropolitan area network–bridges and bridged networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pp. 1–1993, 2018.

[9] D. Thiele, R. Ernst, and J. Diemer, "Formal worst-case timing analysis of Ethernet TSN's time-aware and peristaltic shapers," in *2015 IEEE Vehicular Networking Conference (VNC)*, pp. 251–258, IEEE, 2015.

[10] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA)*, vol. 1, pp. 337–344, IEEE, 2018.

[11] M. L. Raagaard and P. Pop, "Optimization algorithms for the scheduling of IEEE 802.1 time-sensitive networking (TSN)," *Tech. Univ. Denmark, Lyngby, Denmark, Tech. Rep*, vol. 1, 2017.

[12] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Stability-aware integrated routing and scheduling for control applications in ethernet networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 682–687, IEEE, 2018.

[13] "IEEE standard for Ethernet," *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, pp. 1–5600, 2018.

[14] B. Taylor, "Integer programming: The branch and bound method," *Introduction to Management Science*, p. 464, 2009.

[15] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2023. Version 10.0.

[16] A. Makhorin, "GLPK (GNU linear programming kit)," *http://www. gnu. org/s/glpk/glpk. html*, 2008.

[17] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-time systems*, vol. 30, no. 1, pp. 129–154, 2005.

[18] T. Zhang, G. Wang, C. Xue, J. Wang, M. Nixon, and S. Han, "Time-sensitive networking (TSN) for industrial automation: Current advances and future directions," *ACM Computing Surveys*, vol. 57, no. 2, pp. 1–38, 2024.

[19] "Real-time HAT™: Time-sensitive networking hardware for Raspberry Pi." https://innoroute.com/realtimehat/, 2021. Accessed: 04-10-2026.

[20] V. G. Moyano, Z. A. Haj Hammadeh, S. Saidi, and D. Lüdtke, "Global scheduling of weakly-hard real-time tasks using job-level priority classes," *ACM Transactions on Embedded Computing Systems*, 2024.

[21] A. Kaufmann, S. Peter, N. K. Sharma, T. Anderson, and A. Krishnamurthy, "High performance packet processing with FlexNIC," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 67–81, 2016.

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

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

[24] A. C. T. dos Santos, B. Schneider, and V. Nigam, "TSNSCHED: Automated schedule generation for time sensitive networking," in *2019 Formal Methods in Computer Aided Design (FMCAD)*, pp. 69–77, IEEE, 2019.

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

[26] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, "Synthesising schedules to improve QoS of best-effort traffic in TSN networks," in *Proceedings of the 29th International Conference on Real-Time Networks and Systems*, pp. 68–77, 2021.

[27] H. Daigmorte, M. Boyer, and L. Zhao, "Modelling in network calculus a TSN architecture mixing time-triggered, credit based shaper and best-effort queues," 2018.

[28] C. Xue, T. Zhang, Y. Zhou, M. Nixon, A. Loveless, and S. Han, "Real-time scheduling for 802.1Qbv time-sensitive networking (TSN): A systematic review and experimental study," in *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 108–121, 2024.

[29] Y. Sun and M. D. Natale, "Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–19, 2017.

[30] G. Buttazzo and S. Babamir, "Handling overload conditions in real-time systems," in *Real-Time Systems, Architecture, Scheduling, and Application*, vol. 7, InTech, 2012.