Embedded Reconfiguration of TSN: Dual Reconfiguration with Dropping and Reclaiming

Álex Gracia[1]
Department of Computer Sciences and Systems Engineering (DIIS), Aragon Institute of Engineering Research (I3A), University of Zaragoza, HiPEAC European NoE, Spain

Alitzel G. Torres-Macías
Department of Computer Sciences and Systems Engineering (DIIS), Aragon Institute of Engineering Research (I3A), University of Zaragoza, HiPEAC European NoE, Spain
CINVESTAV Unidad Guadalajara, Mexico

Juan Segarra
Department of Computer Sciences and Systems Engineering (DIIS), Aragon Institute of Engineering Research (I3A), University of Zaragoza, HiPEAC European NoE, Spain

José Luis Briz
Department of Computer Sciences and Systems Engineering (DIIS), Aragon Institute of Engineering Research (I3A), University of Zaragoza, HiPEAC European NoE, Spain

Antonio Ramírez-Treviño
CINVESTAV Unidad Guadalajara, Mexico

Héctor Blanco-Alcaine
Intel Deutschland GmbH, Neubiberg, Germany

Abstract

This paper presents our solution to the Industrial Challenge on embedded reconfiguration of Time-Sensitive Networking (TSN), held at the 37th ECRTS. The challenge requires restoring, at runtime and without precomputed solutions, the schedules of streams affected by a link failure in a realistic avionics network. Our solution applies a DROP policy at all bridges to purge the affected streams, updates Gate Control Lists (GCLs) accordingly, reclaims the transmission windows held by the dropped streams on operational links, and applies incremental scheduling to restore as many streams as possible in priority order. Transmission windows of the rescheduled streams are fit into the available gaps without modifying the schedule of unaffected streams. After the incremental scheduling stage, a last-resort mechanism purges all lower-priority streams than any yet-unscheduled critical stream, reclaims their resources, and retries; sacrificed streams are subsequently tested for rescheduling. Evaluated on the benchmark of the challenge – 5 bridges, 15 end-stations, and 241 streams across eight criticality classes – the system achieves perfect recovery in all eight single-link failure scenarios, with total reconfiguration times below 4.4 s. In a harder scenario, reducing the topology to a linear chain through four simultaneous link failures, all safety-critical streams are still recovered. Activating the last-resort mechanism recovers six additional lower-priority streams, with 96.9% of sacrificed streams subsequently re-rescheduled. This paper extends our original challenge submission with a formal algorithm description, a comprehensive experimental evaluation using Gurobi, and the priority-driven last-resort mechanism.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Dependable and fault-tolerant systems and networks; Networks → Network protocols; Networks → Network performance evaluation

Keywords and phrases 802.1Qbv, TSN, TAS, GCL, Scheduling, Real-Time

Digital Object Identifier 10.4230/LITES.11.1.2

[1] Corresponding author

Related Version *Previous Version*: https://github.com/ecrtsorg/ecrts-IC-solutions/blob/main/ECRTS25-IC_Gracia-et-al.pdf

Funding This work was supported by MCIN/AEI/10.13039/501100011033 (grant PID2022-136454NB-C22), and by Dept. of Science, University and Knowledge Society, Government of Aragón (grant to reference research group T58_23R), Spain.
Alitzel G. Torres-Macías: SECIHTI, grant 1141966, Mexico.

Received 2026-05-01 **Accepted** 2026-06-08 **Published** 2026-08-20

1 Introduction

Critical domains such as automotive, aerospace, and industrial automation require networked communication with strict real-time guarantees: bounded end-to-end latency, minimal jitter, and high availability. Time-Sensitive Networking (TSN) addresses these requirements through a suite of IEEE standards that bring deterministic behavior to standard Ethernet infrastructure [1]. Among them, IEEE 802.1Qbv introduces the Time-Aware Shaper (TAS), which controls frame transmission through time-aware gates governed by Gate Control Lists (GCLs), enabling predictable communication over shared links. In operational systems, network links may permanently fail. Any such failure renders the existing schedule infeasible for the streams traversing the failing link, requiring reconfiguration to restore real-time guarantees, if possible.

We propose a reconfiguration system for TSN networks upon link failures, built on three mechanisms: *dropping and purge*, *gap reclaiming*, and *incremental scheduling*. On failure detection, an explicit DROP policy applied at all bridges drops the affected streams and purges their GCL windows. On operational links, the transmission windows formerly held by the dropped streams are reclaimed as available gaps. An optimal incremental scheduler then exploits these gaps – combined with pre-existing idle intervals – to restore as many streams as possible, prioritizing the most critical ones. When a stream remains unschedulable after reclaiming, the system activates a last-resort stage that sacrifices lower-priority streams to recover their resources.

The incremental scheduler at the core of our approach, proposed in [8], schedules a new stream into an existing feasible schedule by allocating its transmission windows within the available gaps, without modifying the existing traffic. It uses a Mixed Integer Linear Programming (MILP) formulation to obtain optimal schedules and assigns queues to frames, ensuring frame isolation whenever possible. We deploy this methodology in a fault-recovery context, where the notion of available gaps is extended to include the reclaimed windows of dropped streams. All streams are treated as time-sensitive regardless of their nominal scheduling class, extending strict temporal guarantees to best-effort traffic.

This paper extends our original submission to the Industrial Challenge held at the 37$^{\text{th}}$ ECRTS [6]. The main contributions added in this extended version are: (i) a formal description of the reconfiguration algorithm, including the dropping and purge and gap reclaiming mechanisms; (ii) a thorough experimental evaluation using Gurobi as the underlying MILP solver; (iii) the priority-driven last-resort mechanism with bulk dropping and re-insertion; and (iv) a systematic analysis of recovery costs across single-link and multi-link failure scenarios.

The specific contributions of this paper are as follows:

- **Formal dropping and purge mechanism.** Upon link failure detection, a DROP policy applied at all bridges removes the transmission windows of the affected streams, guaranteeing that no stale frames remain in the network after a bounded transition period.
- **Gap reclaiming.** The transmission windows held by dropped streams on operational links are recovered as available gaps, extending the resource pool beyond the pre-existing idle intervals.

- **Incremental scheduling in a fault-recovery context.** The incremental scheduling proposal in [8] is applied sequentially over the affected streams in priority order to ensure the most critical traffic is restored first.
- **Priority-driven last-resort mechanism.** When a stream remains unschedulable after reclaiming, the system drops all lower-priority streams, reclaims their resources, and retries. We experimentally characterize the cost in sacrificed streams per recovered critical stream.
- **Experimental evaluation on the ECRTS Industrial Challenge benchmark.** We evaluate the system on the open avionics case study of the Industrial Challenge held at the 37$^{\text{th}}$ ECRTS using Gurobi as the underlying solver, measuring recovery times across single-link and multi-link failure scenarios.

Sec. 2 introduces the benchmark case study, and Sec. 3 discusses the related work. Sec. 4 describes the proposed reconfiguration algorithm, whose experimental evaluation is presented in Sec. 5. Sec. 6 concludes the paper.

2 Challenge Summary

This section describes the benchmark case study [1]: a realistic avionics scenario for TSN reconfiguration upon link failure, comprising a concrete network topology, a heterogeneous stream set, and a reconfiguration objective.

2.1 Network Topology

The network comprises 5 bridges (SW1–SW5) and 15 end-stations (ES1–ES15) interconnected in a five-pip dice topology (Fig. 1). Each peripheral bridge connects to three or four end-stations and three other bridges. The central bridge SW1 connects to two end-stations and the four surrounding bridges, making it the most connected node and the dominant bottleneck. Links operate at 1 Gbps.

The topology offers path diversity for most source-destination pairs: most streams can be rerouted through two or more alternative paths upon link failure. The network is dimensioned to reaccommodate all affected streams upon a single link failure, but not under simultaneous failure of multiple links.

2.2 Streams to Schedule

The benchmark defines 241 time-sensitive streams, each specified by a source end-station, a period, a minimum and maximum frame size, a traffic class, a utility value, and a pre-assigned path[2]. All streams are unique (non-replicated).

Streams are distributed across eight traffic classes (TC0–TC7), with TC7 carrying the highest-priority safety-critical traffic and TC0 the lowest. Three scheduling classes are defined: Time-Aware Shaper (TAS) streams in TC7, governed by GCLs; Credit-Based Shaper (CBS) streams in TC6–TC2, governed by *idleSlope* parameters; and Best-Effort (BE) streams in TC1–TC0, with no timing guarantees. Tab. 1 shows the distribution of streams across traffic classes together with their scheduling class, timing requirements, and frame isolation constraints.

Stream periods take values in $\{200, 320, 400, 800, 1600, 3200, 6400\}$ µs. The dominant period is 400 µs, shared by 146 out of 241 streams (60.6%). The hyperperiod (network cycle) is $T = 6400$ µs. The number of frames per hyperperiod ranges from 1 (period 6.4 ms) to 32 (period 200 µs).

[2] Three of the 241 streams have an invalid pre-assigned path; we assume they use the shortest path.

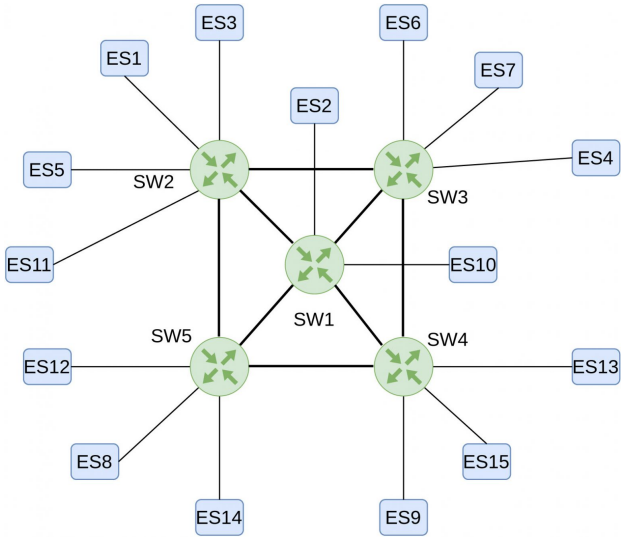

Figure 1 TSN network topology: 5 bridges (SW1–SW5) and 15 end-stations (ES1–ES15) [1].

Table 1 Stream distribution by traffic class. P denotes the stream period; deadline and jitter are expressed as multiples of P.

TC	Sched.	Streams	Deadline	Jitter	Isolation
TC7	TAS	32	$0.5 \cdot P$	$0.2 \cdot P$	Required
TC6	CBS	39	P	—	No
TC5	CBS	45	P	—	No
TC4	CBS	29	$2 \cdot P$	—	No
TC3	CBS	20	$2 \cdot P$	—	No
TC2	CBS	19	$2 \cdot P$	—	No
TC1	BE	40	—	—	No
TC0	BE	17	—	—	No
Total		**241**			

This heterogeneity has a direct impact on scheduling complexity: streams with many frames per hyperperiod require more binary variables in the MILP model and are substantially harder to schedule, as confirmed by our experimental results (Sec. 5).

2.3 Initial Schedule

The challenge specification [1] does not prescribe a method for computing the initial feasible schedule S that serves as the baseline prior to any failure. We compute S using the incremental MILP scheduler in [8], applying it sequentially to all 241 streams in decreasing order of priority: TC7 streams are scheduled first, followed by TC6, TC5, and so on down to TC0. Although TC1 and TC0 are Best-Effort classes with no explicit timing constraints in the challenge specification, we conservatively assign them a deadline equal to their period during scheduling to bound resource contention.

For parameters not specified in the challenge, we adopt the following values. The maximum time error between any two nodes is set to 500 ns. The propagation delay of all links is set to 0 ns. The processing time at each bridge is set to 1000 ns. Each stream is scheduled using its maximum frame size to ensure feasibility under worst-case conditions.

2.4 Objective

When a link fails, all streams whose pre-assigned path traverses the failed link become infeasible. The objective is to restore as many of these streams as possible, prioritizing the most critical ones, as rapidly as possible. Reconfiguration must be performed online, from scratch at failure time; no precomputed solutions are permitted [1]. *Frame Replication for Reliability* (FRER) [7] is also excluded, so each stream has a unique schedule on its pre-assigned path.

3 Related Work: Solutions to the ECRTS Industrial Challenge

The challenge [1] attracted several solutions addressing the same reconfiguration problem on the same benchmark, making them the most directly relevant comparison for this work.

Boyer *et al.* [2] propose WPEx-TAS, a novel scheduling class designed for resilient and incremental TAS reconfiguration. The key idea is to compute an initial schedule with deliberate slack, achieved through window enlargement and protective gating, that allows new streams to be inserted into existing TAS windows without modifying the GCL. Reconfiguration is therefore extremely fast: no GCL update is required upon failure, only updated path information. This approach requires, however, an initial schedule designed offline with WPEx constraints in mind, imposing structural requirements that limit baseline network utilization. Our approach is transparent to the initial scheduling method and applies to any feasible pre-existing schedule.

Bujosa *et al.* [3] propose a hybrid approach combining precomputed backup paths with HERMES [4] for Scheduled Traffic (ST) rescheduling and worst-case response time analysis for Audio-Video Bridging traffic. Upon link failure, affected streams are immediately assigned their precomputed backup paths, and HERMES recomputes the ST schedule. This achieves fast recovery for ST streams (around 2.5 ms) but requires offline precomputation of backup paths, violating the no-precomputation constraint of the challenge [1]. HERMES also provides no optimality guarantees, and circular dependencies in the path recomputation mechanism left two of the eight failure scenarios unschedulable. Our approach imposes no precomputation requirement, guarantees optimality per stream via MILP, and achieves perfect recovery in all eight single-link failure scenarios.

The key differentiator of our proposal is the reclaiming mechanism: by recovering the transmission windows formerly held by dropped streams on operational links, we extend the resource pool available to the incremental scheduler beyond the pre-existing idle intervals. This mechanism is absent in WPEx-TAS, which relies on predesigned slack, and in the backup-path approach, which relies on precomputed alternative paths. Combined with the priority-driven last-resort mechanism, our approach delivers provably optimal per-stream recovery without any offline preparation specific to the failure scenario. Our proposal also treats all streams as time-sensitive, including those classified as CBS or BE in the challenge.

4 Proposal

We model the TSN network as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each vertex represents a TSN node (bridge or end-station) and each directed edge $(a, b) \in \mathcal{E}$ represents a unidirectional physical link from node a to node b, with a bijection between links and egress ports. A path p in $\mathcal{G}$ is an ordered sequence of links $(a, b) \in \mathcal{E}$, where the destination node of each link is the source node of the next. We assume the network operates under a feasible TSN schedule S prior to the failure, computed by any scheduling method. S is a set of 8-tuples $\langle s, f, (a, b), q, \text{queued}, \text{open}, \text{complete}, \text{close}\rangle$, assigning each frame f of each stream s to a queue q on the egress port to b in bridge a. It specifies the arrival (queued) and transmission completion times of f, along with the gate opening and closing times for its transmission through link (a, b). We follow the notation in [8].

Each stream s carries a priority determined by its traffic class: TC7 streams have the highest priority and TC0 the lowest. We define Φ as the set of all streams, ordered by decreasing priority and, within the same priority, by index in the benchmark specification; recovery follows this order. The utility field specified in the challenge is not used, as the traffic class already provides a natural and operationally meaningful priority ordering. The benchmark also specifies a fixed path p_s for each stream $s \in \Phi$.

Fault detection is outside the scope of this work; the algorithm is triggered upon notification of a failing bidirectional link $(a, b), (b, a) \in \mathcal{E}$. Henceforth, *failing link* refers to a failing bidirectional link. Upon failure, the topology is updated to $\mathcal{G}' = (\mathcal{V}, \mathcal{E} \setminus \{(a, b), (b, a)\})$, and all streams whose path p_s contains any direction of the failing link become *affected streams*, collected in the ordered set $\mathcal{D} = \Phi^{(a,b)} \cup \Phi^{(b,a)}$, where $\Phi^{(a,b)} \subset \Phi$ is the set of streams traversing (a, b) and $\Phi^{(b,a)}$ is defined analogously. $\mathcal{D}$ follows the same ordering as Φ. Unaffected streams retain their schedules without modification.

Our solution (Alg. 1) purges the network of frames belonging to the affected streams, reclaims their former transmission windows as gaps, and attempts to reschedule them within the resulting free intervals. The first stage (ll. 1–3) purges the system of streams affected by the failure. It applies a DROP policy to all streams in $\mathcal{D}$, updates all GCLs accordingly, removes the failing links from the topology $\mathcal{G}$, and removes the purged streams from the schedule data $\mathcal{S}'$. An incremental scheduling stage (ll. 4–18) and a last-resort stage (ll. 19–35) follow.

Incremental scheduling attempts to reschedule each stream in $\mathcal{D}$, one at a time, in decreasing order of priority. For each stream s, the algorithm enumerates the available paths in $\mathcal{G}'$, sorted by hop count, and attempts to schedule s on each path in turn using an incremental scheduler[3] [8]. If a feasible schedule is found, the corresponding transmission windows are synthesized, $\mathcal{S}'$ is updated, and s is deployed and removed from $\mathcal{D}$. Each successful schedule consumes transmission gaps that may reduce the options available for subsequent streams. Scheduling in priority order therefore ensures the most critical traffic is tested for recovery first. The schedule obtained for each stream is optimal for that stream and path in the current network state, as guaranteed by the MILP formulation. Incremental scheduling comprises both temporal scheduling (setting optimal transmission times) and queue and gate assignment at bridges. For this challenge, queue assignment follows a direct mapping from traffic class to queue index: a stream of class TCk is assigned to queue k at every bridge it traverses. This policy does not guarantee frame isolation in general, since streams of the same class sharing an egress port are assigned to the same queue. However, for TC7 streams, which require frame isolation per the benchmark specification, we explicitly verify that no two TC7 frames overlap in any egress queue along the path. Although this simple mapping suffices for all scenarios in this challenge, more sophisticated assignment methods could be used, e.g., [8].

After the incremental scheduling stage, the last-resort mechanism attempts to schedule any streams remaining in $\mathcal{D}$ through a more aggressive strategy: purging low-priority scheduled streams to recover their resources. First, line 20 collects in $\mathcal{C}$ all scheduled streams with lower priority than any yet-unscheduled stream in $\mathcal{D}$. These streams are purged and added to $\mathcal{D}$ (ll. 21–22), and the incremental scheduling loop is repeated (ll. 24–35). This guarantees that no low-priority stream can directly prevent the scheduling of a higher-priority one, while retaining as many streams as possible in the running schedule.

[3] Publicly available at `https://gitlab.com/uz-gaz/incremental-tsn-scheduler`

Algorithm 1 Reconfiguration algorithm.

Require: Set of streams Φ
Require: Feasible schedule S prior to failure
Require: Boolean *lastResort* // Enable last-resort mechanism
Ensure: New schedule $\mathcal{S}'$ and set of unscheduled affected streams $\mathcal{D}$

$\mathcal{G}' \leftarrow \mathcal{G} \setminus \{(a,b),(b,a)\}$
$\mathcal{D} \leftarrow \Phi^{(a,b)} \cup \Phi^{(b,a)}$
$\mathcal{S}' \leftarrow purge(\mathcal{D}, S)$ // Apply DROP policy, remove transmission windows, disable talkers
$\mathcal{D}_0 \leftarrow \mathcal{D}$
for all $s \in \mathcal{D}_0$ **do**
  $\mathcal{P} \leftarrow$ paths p for s in $\mathcal{G}'$, sorted by hop count
  for all $p \in \mathcal{P}$ **do**
    Generate MILP model for s on p using gaps in $\mathcal{S}'$ [8]
    $\mathcal{S}_s \leftarrow$ Solve model
    if schedulable **then**
      $\mathcal{S}' \leftarrow \mathcal{S}' \cup \mathcal{S}_s$
      $deploy(\mathcal{S}_s)$ // Stop DROP policy, set GCLs, adjust and enable talker
      $\mathcal{D} \leftarrow \mathcal{D} \setminus \{s\}$
      break
  if $s \in \mathcal{D}$ **then**
    print s unschedulable // No feasible schedule without last-resort mechanism
    if *lastResort* **then**
      break // Continue with last-resort mechanism
if *lastResort* **and** $\mathcal{D} \neq \emptyset$ **then**
  $\mathcal{C} \leftarrow$ streams in $\mathcal{S}'$ with lower priority than highest-priority stream in $\mathcal{D}$
  $\mathcal{S}' \leftarrow purge(\mathcal{C}, \mathcal{S}')$
  $\mathcal{D} \leftarrow sort_{\downarrow}(\mathcal{D} \cup \mathcal{C})$ // Merge and re-sort by decreasing priority
  $\mathcal{D}_0 \leftarrow \mathcal{D}$
  for all $s \in \mathcal{D}_0$ **do**
    $\mathcal{P} \leftarrow$ paths p for s in $\mathcal{G}'$, sorted by hop count
    for all $p \in \mathcal{P}$ **do**
      Generate MILP model for s on p using gaps in $\mathcal{S}'$ [8]
      $\mathcal{S}_s \leftarrow$ Solve model
      if schedulable **then**
        $\mathcal{S}' \leftarrow \mathcal{S}' \cup \mathcal{S}_s$
        $deploy(\mathcal{S}_s)$ // Stop DROP policy, set GCLs, adjust and enable talker
        $\mathcal{D} \leftarrow \mathcal{D} \setminus \{s\}$
        break
    if $s \in \mathcal{D}$ **then**
      print s permanently unschedulable // No feasible schedule in current topology

Table 2 Results for single-link failure scenarios. $\Delta\Gamma$ is the additional gap capacity recovered by reclaiming, computed over inter-bridge links only.

Faulty link	$\lvert\mathcal{D}\rvert$	Succ.	Unsucc.	$\Delta\Gamma$ (ns)	$\Delta\Gamma$ (%)	Resched.
SW1-SW2	48	48	0	4 800 840	+8.3 %	1.23 s
SW1-SW3	51	51	0	4 047 248	+7.0 %	1.27 s
SW1-SW4	36	36	0	3 458 224	+6.0 %	2.11 s
SW1-SW5	43	43	0	5 091 976	+8.9 %	1.23 s
SW2-SW3	39	39	0	1 364 656	+2.4 %	2.23 s
SW2-SW5	43	43	0	4 142 912	+7.1 %	3.20 s
SW3-SW4	35	35	0	3 282 400	+5.7 %	2.85 s
SW4-SW5	35	35	0	4 780 896	+8.3 %	2.06 s
Average	**41.3**	**41.3**	**0**	**3 871 144**	**+6.7 %**	**2.02 s**

5 Experimental Results

All experiments run on an Intel Xeon Gold 5120 CPU (28 cores, 56 threads, 2017) with Gurobi Optimizer 12.0.2 as the MILP solver. This platform is representative of the compute capacity available on modern aircraft, as required by the challenge: computation must use resources embedded in the vehicle, with no access to external infrastructure [1]. Multi-core processors are an established and regulated component in safety-critical avionics, governed by EASA/FAA guidance AMC 20-193 [5]. Certified ARINC 653 operating systems such as Wind River VxWorks 653 Multi-Core Edition run on embedded multicore platforms, including the Intel Xeon D family, 11$^{\text{th}}$-generation Intel Core, and quad-core Arm Cortex-A53 (Xilinx Zynq UltraScale+ MPSoC) processors, integrated into avionics-grade single-board computers and deployed in production aircraft such as the Boeing 787 and the Airbus A400M [9, 10]. The embedded Intel Xeon D family offers up to 20 cores per socket, comparable in core count to our dual-socket experimental machine, but at a lower base clock frequency (1.8–2.4 GHz vs. 2.2 GHz). As MILP solve time scales primarily with single-core performance, reconfiguration times on such a platform would be at most approximately twice those reported here, remaining well within the reconfiguration budget.

The baseline schedule S is computed offline prior to any failure using [8]. We explore two scenarios: single-link failures and a multi-link failure, preserving end-to-end reachability between all end-station pairs.

5.1 Single-Link Failures

Tab. 2 reports results for all eight single-link failure scenarios. For each scenario, we report the number of affected streams $|\mathcal{D}|$, the number successfully rescheduled, and the number unsuccessfully rescheduled. We also report the gap gain ($\Delta\Gamma$): the additional transmission capacity recovered by reclaiming, expressed in nanoseconds and as a percentage of the total available capacity on operational links after the failure. Only bridge-to-bridge links are considered, as end-station links are not subject to failure in the challenge.

The rescheduling time ranging from 1.23 s to 3.20 s depending on the failing link is driven primarily by the number of frames per hyperperiod each affected stream requires (up to 32 frames per stream). All eight single-link failure scenarios achieve perfect recovery: every affected stream is successfully rescheduled with zero drops and zero disruption to unaffected traffic. Reclaiming the

Table 3 Results for the linear topology under four simultaneous link failures that preserve end-station connectivity, without activating the last-resort mechanism (*lastResort* = FALSE in Alg. 1). $\Delta\Gamma$ is the total gap capacity recovered through reclaiming across all inter-bridge links over the hyperperiod.

$\lvert\mathcal{D}\rvert$	**Succ.**	**Unsucc.**	$\Delta\Gamma$ (ns)	$\Delta\Gamma$% .	**Resched.**	**TC7 recovered**
139	104	35	8 785 312	+26.1 %	5.04 s	**100 %**

transmission windows of affected streams contributes between +2.4 % and +8.9 % of additional aggregated gap capacity on inter-bridge links. The total reconfiguration time remains below 4.3 s in all single-link scenarios.

As described in [1], critical avionics hosts are redundantly connected to both a TSN network and a duplicated AFDX network, so critical applications continue communicating over AFDX without interruption even when reconfiguration time exceeds the deadlines of the affected streams. Reconfiguration runs in the background to isolate the faulty components and restore TSN service, while critical traffic remains protected by the redundant network. The recovery budget is therefore bounded by the time the system tolerates operating without a redundant TSN path, on the order of the mean time between independent link failures. Even accounting for the lower clock frequency of embedded avionics processors (Sec. 5), reconfiguration completes well within this budget.

Rescheduling time varies notably across links: the SW2-SW5 failure (3.20 s) takes more than twice as long as SW1-SW2 (1.23 s). despite involving a similar number of streams. This is consistent with the behavior observed in [8]: streams with many frames per hyperperiod are significantly harder to schedule, and the sets of streams affected by each link failure differ in their period distribution.

5.2 Multi-Link Failure: Linear Topology

To stress-test the system under severe degradation, we evaluate a scenario in which four links fail simultaneously: SW1-SW3, SW1-SW4, SW1-SW5, and SW2-SW5. This combination reduces the network to a single linear chain SW1–SW2–SW3–SW4–SW5, eliminating all alternative paths but maintaining reachability, thus concentrating all traffic through a single sequence of bottleneck links.

Tab. 3 summarizes the results when reconfiguration relies solely on dropping, reclaiming, and the incremental MILP scheduler (*lastResort* = FALSE in Alg. 1). Of the 139 affected streams, 104 are successfully rescheduled (74.8 %), including all TC7–TC5 streams. The remaining 35 streams (TC4–TC0) cannot be accommodated in the degraded topology: all paths between their source and destination nodes cross at least one failed link, and the surviving chain links lack sufficient residual capacity.

Reclaiming contributes +26.1 % of additional aggregated gap capacity across all inter-bridge links over the hyperperiod, substantially larger than in single-link failure cases. This is expected: four failing links release a proportionally larger pool of reclaimed windows, and in a heavily degraded topology where available paths are scarce, this additional capacity is essential to achieve a higher recovery rate.

The rescheduling time (5.04 s) reflects the larger number of affected streams (139 vs. 41.3 on average in single-link failures). Frame isolation for TC7 streams was verified after queue assignment; no violation was detected.

Table 4 Streams successfully scheduled per traffic class: incremental-only (no last-resort) vs. incremental plus last-resort. Δ shows the net change per class.

Traffic class	Total	Incr. only	Incr.+LR	Δ
TC7	32	32	32	0
TC6	39	39	39	0
TC5	45	45	45	0
TC4	29	26	28	+2
TC3	20	19	20	+1
TC2	19	16	19	+3
TC1	40	21	22	+1
TC0	17	8	7	−1
Total	**241**	**206**	**212**	**+6**

The previous results confirm that the incremental scheduler alone recovers all TC7–TC5 streams, leaving 35 streams unscheduled. To evaluate the last-resort mechanism, we run Alg. 1 with *lastResort* = TRUE: when an affected stream cannot be scheduled, all lower-priority streams are dropped, their transmission windows reclaimed as gaps, and scheduling is retried with these additional resources. Tab. 4 breaks down scheduled streams per traffic class for both configurations.

The last-resort mechanism triggers when a TC4 stream fails to schedule, purging the 32 TC3–TC0 streams operational at that point (Alg. 1, ll. 19–22). Upon re-insertion, the incremental scheduler recovers 31 of them; the single collateral loss is one TC1 stream for which no feasible schedule exists in the remaining gaps. Gains concentrate in TC2–TC4: three additional TC2 streams, two TC4, and one TC3 are recovered. TC1 shows a net gain of one stream: although the purged TC1 stream is lost, several TC1 streams that were unschedulable in the incremental-only run become schedulable once the reclaimed gaps are released, offsetting that loss. The sole net loss occurs at TC0: one TC0 stream affected by the link failure, successfully rescheduled in the incremental-only baseline, cannot be accommodated once the freed gaps are consumed by higher-priority affected streams. The net result is six additional recovered streams.

This outcome is by design: the mechanism explicitly prioritizes higher-criticality traffic, accepting marginal collateral at the lowest priority class in exchange for substantial gains at TC2–TC4. The results also expose a limitation of the current approach: two TC4 streams compete for the same bottleneck resources, and because one is processed first and consumes the freed gaps, the other remains permanently unschedulable even after the full sacrifice. Joint scheduling of competing streams at the same priority level, and heuristics to estimate whether a sacrifice yields sufficient capacity before committing to the purge, are directions for future work.

6 Conclusions

This paper presents a reconfiguration system for TSN networks upon link failures, built on three mechanisms: dropping and purge, gap reclaiming, and incremental scheduling. When a link failure is detected, affected streams are dropped via an explicit DROP policy at all bridges, and their transmission windows are purged. The freed windows on operational links are reclaimed as additional scheduling capacity, extending the resource pool beyond pre-existing idle intervals. An optimal incremental MILP scheduler then restores as many streams as possible in priority order. When a stream cannot be accommodated even after reclaiming, a last-resort mechanism purges all lower-priority streams simultaneously, reclaims their resources, and retries. Evaluated

on the open benchmark of the Industrial Challenge on embedded reconfiguration of Time-Sensitive Networking (TSN), held at the 37$^{\text{th}}$ ECRTS, our proposal achieves perfect recovery in all eight single-link failure scenarios, with total reconfiguration times below 4.3 s. In a multi-link failure scenario, reducing the topology to a linear chain, the incremental scheduler recovers all TC7–TC5 streams without activating the last-resort. With the last-resort mechanism, six additional streams are recovered at the cost of 32 transient sacrifices, of which 31 are subsequently re-rescheduled.

References

1 Marc Boyer and Rafik Henia. Industrial challenge: Embedded reconfiguration of TSN. working paper or preprint, July 2024. URL: `https://hal.science/hal-04630862`.

2 Marc Boyer, Rafik Henia, and Cédric Pralet. ECRTS industrial challenge: Designing a resilient time-aware shaper configuration for TSN. `https://github.com/ecrtsorg/ecrts-IC-solutions/tree/main`, 2025. Solution to the 37th ECRTS Industrial Challenge.

3 Daniel Bujosa, Mohammad Ashjaei, and Saad Mubeen. Online reconfiguration for TSN in avionics using backup paths and integrated mapping, scheduling, and analysis tools. `https://github.com/ecrtsorg/ecrts-IC-solutions/tree/main`, 2025. Solution to the 37th ECRTS Industrial Challenge.

4 Daniel Bujosa, Mohammad Ashjaei, Alessandro V. Papadopoulos, Thomas Nolte, and Julián Proenza. HERMES: Heuristic multi-queue scheduler for TSN time-triggered traffic with zero reception jitter capabilities. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems (RTNS 2022)*, pages 70–80. ACM, 2022. `doi:10.1145/3534879.3534906`.

5 European Union Aviation Safety Agency. AMC 20-193: Use of multi-core processors. EASA Acceptable Means of Compliance, 2022.

6 Álex Gracia, Alitzel G. Torres-Macías, Juan Segarra, José L. Briz, Antonio Ramírez-Treviño, and Héctor Blanco-Alcaine. Embedded reconfiguration of TSN: Dual reconfiguration with dropping and reclaiming. `https://github.com/ecrtsorg/ecrts-IC-solutions/tree/main`, 2025. Solution to the 37th ECRTS Industrial Challenge.

7 IEEE. IEEE standard for local and metropolitan area networks – frame replication and elimination for reliability, 2017. `doi:10.1109/IEEESTD.2017.8091139`.

8 A.G. Torres-Macías, Á. Gracia, J. Segarra, J.L. Briz, A. Ramírez-Treviño, and H. Blanco-Alcaine. Optimal Fast IEEE802.1Qbv Incremental Scheduling. *Ad Hoc Networks*, 191:104280, 2026. `doi:10.1016/j.adhoc.2026.104280`.

9 Wind River Systems. Building real-time avionics systems optimized for Intel multi-core processors. Wind River Systems – technical resource, 2024.

10 Wind River Systems. VxWorks 653 multi-core edition: IMA platform for safety-critical applications. `https://www.windriver.com/resource/vxworks-653-product-overview`, 2024.