Fast Time-Aware Shaper Scheduling for In-Vehicle Networks via Deep Reinforcement Learning

Mohammadparsa Karimi, Majid Nabi, Andrew Nelson, Kees Goossens, *Member, IEEE*, and Twan Basten, *Senior Member, IEEE*

***Abstract*—Modern vehicles increasingly rely on distributed computing platforms that exchange large volumes of sensor and control data with strict timing requirements. Ensuring that this traffic meets its deadlines over Ethernet-based in-vehicle networks requires Time-Sensitive Networking (TSN) and, in particular, effective configuration of the Time-Aware Shaper (TAS). However, generating and updating TAS schedules that remain valid as traffic patterns evolve is an NP-hard problem that traditional optimization or heuristic methods address only partially.**

This paper introduces a Deep Reinforcement Learning (DRL) scheduler that learns to configure TAS schedules directly from network state while preserving standard compliance through analytical validation. The proposed DRL scheduler encodes the scenario (network topology and workload) of the in-vehicle network using a Graph Neural Network (GNN) and learns scheduling policies that balance deadline satisfaction, latency, and resource utilization. Evaluation on a comprehensive benchmark shows that the proposed approach consistently outperforms state-of-the-art heuristics and a topology-specific DRL baseline, achieving higher success rate and lower delay while maintaining efficient bandwidth use. Once trained, it can adapt to new traffic scenarios within milliseconds, demonstrating the potential of the DRL-based scheduler as a foundation for adaptive and reliable communication in next-generation software-defined vehicles.

***Index Terms*—Adaptive scheduling, Automotive Ethernet, Deep Reinforcement Learning, Graph Neural Networks, In-Vehicle Networks, Proximal Policy Optimization, Time-Aware Shaper, Time-Sensitive Networking**

I. Introduction

Software is rapidly becoming the defining feature of modern automobiles. As perception, decision-making, and control migrate to networked computing platforms, vehicles must process sensor and control data with tight timing requirements to deliver safety, energy-aware operation, and a refined user experience [1], [2]. In this setting, the in-vehicle network (IVN) is the vehicle's nervous system; its design and configuration directly determine whether critical signals arrive on time and whether resources are used efficiently [3].

Time-Sensitive Networking (TSN) [4] has emerged as the foundation for IVNs because it combines support for streaming data, deterministic delivery, bounded latency, and high reliability with Ethernet bandwidth and support for mixed traffic classes [5]. Within TSN, the Time-Aware Shaper (TAS), defined by the IEEE 802.1Qbv standard [6], uses a Gate Control List (GCL) per switch port to open and close per-queue transmission windows in a periodic fashion, aligned to global time. By isolating time-triggered traffic from interference and by structuring access to the medium in globally aligned cycles, TAS enables safety-critical flows to meet deadlines while keeping the network well utilized. These properties are essential in vehicles that operate under changing conditions and tight resource budgets.

The challenge is that computing valid and high-quality TAS schedules is NP-hard [7]. Yet, they must be recomputed as traffic evolves, because in-vehicle workloads can change dynamically during operation [2]. Sudden bursts in perception streams combined with shifting control loops and evolving topology updates occur in ways that can quickly invalidate static plans [8]. Valid schedules must be quantized to the time base, insert guard bands, enforce time-triggered windows (referred to as 'segments'), and coordinate many ports and queues over each cycle. The combinatorial search space grows with the topology size, the number of streams, and the number of traffic classes. This creates a persistent tension between real-time adaptability and strict timing guarantees [9].

In software-defined vehicles, network scenarios evolve as software updates introduce new sensing and control functions, functionalities change dynamically at runtime, and after-market modifications are applied [10]. Following the system-scenario-based design paradigm [11], we interpret network scenarios in the TSN context as being induced by the combination of network topology and workload. These structural factors jointly determine the set of runtime operating conditions and their associated performance trade-offs. The network topology represents the physical devices and communication links, while the workload represents the collection of traffic streams generated over the topology. In practical IVNs, workload characteristics may vary frequently at runtime due to dynamic application behavior, whereas topology changes are comparatively rare and typically occur only due to hardware reconfiguration, failures, or after-market modifications.

Our previous work [23] introduced a DRL scheduler that selects gate control parameters from observed network state, trained with Proximal Policy Optimization (PPO) and a reward shaped by deadline satisfaction, latency, and link utilization computed via analytic delay evaluation. That study improved success rate and latency over an Earliest Deadline First (EDF) baseline across multiple scenario classes and demonstrated online adaptiveness. At the same time, opportunities remained to expose network scenarios more directly to the neural

This work has received funding from the European Chips Joint Undertaking under Framework Partnership Agreement No. 101139789 (HAL4SDV).

The authors are with the Electronic Systems (ES) group, Department of Electrical Engineering, Eindhoven University of Technology (TU/e), 5612 AZ Eindhoven, The Netherlands. (e-mail: {m.karimi, m.nabi, a.t.nelson, k.g.w.goossens, a.a.basten}@tue.nl).

network and to reduce action complexity.

This paper[1] advances this work into a framework that is scenario-aware, robust against scenario changes and modest topology changes, better scalable, more reliable, and easier to deploy. The contributions compared to [23] are as follows.

- **Scenario-aware state encoding:** We design a graph neural network (GNN) [24] that aggregates node and link-level features into a global embedding representing both network topology and traffic workload. This encoding exposes structural properties of the IVN to the policy and enables generalization across varying scenarios, including substantial workload variations and moderate topology changes. Compared to our prior work, the GNN-based representation improves resulting schedules (in terms of success rate and delay) while reducing the sensitivity to topology modifications.
- **Topology-independent action design with constraint-preserving GCL synthesis:** We reformulate TAS control as a structured, phase-based action space that is invariant to network size and port-level dimensionality. This abstraction preserves sufficient expressiveness for feasible schedule generation while eliminating the topology-dependent action dimensionality present in prior PPO-based TAS schedulers. This reduces decision complexity and enables scalable policy learning across different network scenarios. To bridge the gap between this high-level action representation and IEEE 802.1Qbv-compliant per-port GCLs, we introduce a deterministic, standards-aware compiler that maps abstract phase allocations into valid GCL configurations while enforcing feasibility constraints by construction. Together, this design decouples learning from low-level schedule synthesis and guarantees standard compliance without constraining the policy optimization process.
- **Staged scenario evaluation with topology variation:** We conduct a staged evaluation across progressively harder network scenarios by increasing traffic load, tightening timing constraints, and introducing moderate topology modifications. This structured progression enables quantitative assessment of generalization under both dynamic workload conditions and small topology changes.

Fig. 1 illustrates the high-level idea of this work. Part (a) shows the training loop, where scenarios from the scenario pool are processed by the environment, which contains the analytic evaluation framework. Given the current scenario, the environment produces the corresponding network state representation, which is encoded by the GNN into graph embeddings and provided to the DRL scheduler. The agent outputs a scheduling action, which is compiled into per-port GCLs and passed back to the evaluation framework inside the environment. This framework evaluates the resulting schedule and computes the corresponding reward values, success rate, delay, and utilization.

Part (b) shows the runtime phase, where the trained agent receives a new scenario, the GNN encodes its graph structure, and the agent produces the corresponding action. The resulting schedule is verified for validity before deployment.

[1]The source code of the work described in this paper is avaialble open source via the TU/e ES GitHub repository (https://github.com/TUE-EE-ES).

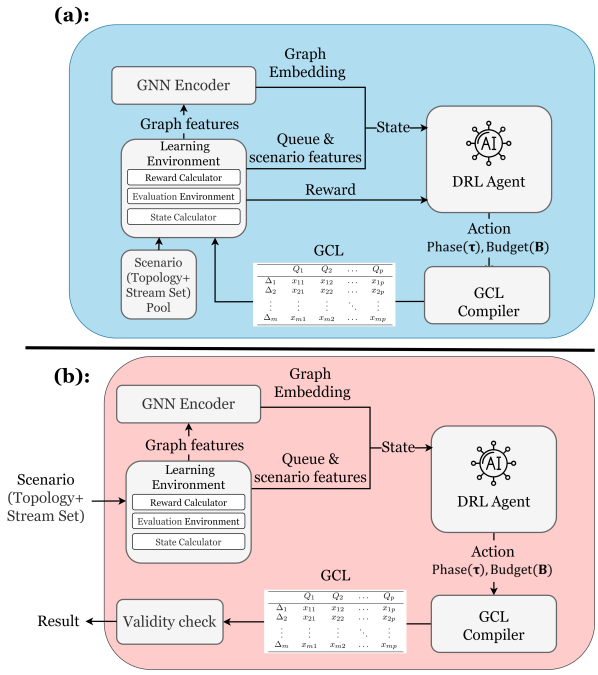


Fig. 1: High-level overview of the proposed framework. (a) Training loop. (b) Runtime deployment.

If the analytic validation detects deadline misses, a higher-level supervisory controller can take corrective actions such as reverting to the last verified configuration, temporarily blocking non-critical traffic, or activating a predefined fallback schedule to ensure continued safe and deterministic operation. The design and implementation of this higher-level controller are beyond the scope of this work.

Compared to our earlier work, the extended framework achieves higher scheduling success rates and more efficient training through faster convergence and reduced action complexity. Its scenario-aware GNN encoder and compact template action also provide strong adaptability to unseen network scenarios at runtime.

The remainder of the paper is organized as follows. Section II reviews prior work on TAS scheduling and positions our contribution. Section III formalizes the scheduling problem and the network and traffic models. Section IV presents the DRL scheduler, including the scenario-aware encoder, compact template action, and deterministic compiler. Section V describes the experimental setup and reports results. Section VI concludes the paper and outlines directions for future work.

II. Related Work

Prior work on TAS scheduling pursued exact optimization, heuristic construction, and learning-based control, each approach offering different trade-offs between validity, solution quality, and adaptability. A recent benchmark [12] compares some influential offline algorithms and documents systematic trade-offs among runtime, success rate, latency, and GCL length. We draw on these findings to frame our design and select strong baselines.

Heuristics are attractive for fast construction and support for frequent recomputation [13]. Tabu search [14] and related

list-based refinements minimize flow time under a no-wait model and often find first solutions rapidly, but can miss valid cases in tight settings and may require longer runs for improvement. Move to Front tabling (M2F) [15] discretizes time and produces very short GCLs with back to back transmissions, but introduces queueing delays that raise latency in larger instances. A Greedy Randomized Adaptive Search Procedure (GRASP) [16] builds solutions with a randomized greedy constructor and improves them by local search, which increases schedulability at the cost of higher latency due to queuing and sensitivity to parameter tuning.

The exact methods provide correctness by construction and can certify infeasibility. The no-wait integer linear program (NoWait-ILP) [17] achieves minimal latencies, but its runtime grows quickly with problem size. An incremental satisfiability modulo theories (SMT) [18] formulation supports multiple traffic classes; a decomposition variant that groups streams into subsets improves scalability. It is the only exact method in the benchmark [12] that consistently handles very large instances, although it may trade some schedulability for speed. Overall, exact models deliver strong guarantees and low latency when they finish, yet their scalability limits restrict online use.

Learning-based techniques have recently been explored for different optimization tasks in TSN. For instance, [19] investigates DRL for wireless TSN, where the objective is multi-link selection and deterministic time-slot allocation in a WiFi-based architecture. Although this work incorporates a wireless time-aware shaper, its optimization focus is on assigning flows to predefined wireless time slots and selecting transmission links, rather than directly generating GCLs for Ethernet TAS scheduling. Similarly, [20] applies reinforcement learning to the joint routing and scheduling problem with reliability guarantees. The proposed approach primarily optimizes redundant path selection and performs time-slot allocation along selected routes, with the main objective of improving load balancing and reliability.

Regarding DRL-based approaches that directly target TAS scheduling, recent works address different formulations of the problem. In particular, [21] focuses on constrained modifications of existing schedules using a restricted action space, limiting scalability and structural generalization. DeepScheduler [22] performs fine-grained time-slot assignment on individual links through sequential flow ordering and greedy slot allocation, operating at a different abstraction level from IEEE 802.1Qbv TAS by allocating discrete transmission slots rather than synthesizing standard-compliant GCL windows.

Our prior work [23] directly configures TAS parameters using DRL, demonstrating improved scheduling quality compared to EDF. However, it relies on topology-specific action spaces, which fundamentally prevents generalization even under minor topology variations. For the experimental comparisons in this paper, we retrained this prior model for each evaluated topology instance to enable direct and fair performance assessment.

III. Scheduling Problem

This section formally defines the TAS scheduling problem for IVNs. We first outline the network and traffic models, then describe the timing and validity constraints, and finally present the optimization objective and associated challenges.

A. Network Model

We model the IVN as a time-sensitive Ethernet-based communication fabric that interconnects a set of Electronic Control Units (ECUs), computers, sensors, actuators, and switches. The network can be represented as a directed graph:

$$G = (V, E), \tag{1}$$

where V (vertices) denotes the set of network devices and E (edges) denotes the set of directed communication links [25]. Each node $v \in V$ can represent either:

- **An end system or ECU**, such as a sensor, camera, controller, or central computing unit that transmits or receives frames.
- **A switch or gateway**, responsible for forwarding frames between ports according to their assigned scheduling configuration.

Each directed edge $e \in E$ is defined as

$$e = (v_e^{\text{src}}, v_e^{\text{dst}}, b_e), \tag{2}$$

where $v_e^{\text{src}} \in V$ and $v_e^{\text{dst}} \in V$ denote the source and destination nodes of the link, respectively, and b_e denotes the available bandwidth.

B. Traffic Model

Let $S = \{s_1, s_2, \ldots, s_n\}$ denote the set of all Time Triggered (TT) streams that coexist in the network at a given time [26]. Each stream $s \in S$ represents a periodic data flow generated by a talker and received by a listener[2]. Each TT stream s is formally defined as:

$$s = (v_{\text{talker},s}, v_{\text{listener},s}, T_s, n_{f,s}, \mathbf{t}_{\text{release},s}, f_s, d_s), \tag{3}$$

where:

- $v_{\text{talker},s} \in V$ is the *talker*, i.e., the source node that generates frames of stream s.
- $v_{\text{listener},s} \in V$ is the *listener*, i.e., the destination node that receives the frames of stream s.
- T_s is the *period* of the stream, representing the time after which its complete frame-release pattern repeats.
- $n_{f,s}$ is the number of frames transmitted in each period.
- $\mathbf{t}_{\text{release},s} = \left[t_{\text{release},s}(1), \ldots, t_{\text{release},s}(n_{f,s})\right]$ is the vector of release times within one period, where $t_{\text{release},s}(i)$ denotes the release time of the i-th frame relative to the beginning of the period $(1 \leq i \leq n_{f,s})$.
- f_s is the *frame size* in bytes, typically ranging from a few tens of bytes to the Ethernet maximum of 1518 bytes.
- d_s is the *end-to-end deadline* of the stream, representing the maximum allowable delay between the release of a frame at the talker and its reception by the listener.

[2]Although the TSN standard supports multiple listeners per talker, it is common practice for streaming communication to support only unicast streams with a single listener. Multicast transmission can be represented by decomposing each multicast stream into multiple unicast streams.

Each stream therefore periodically generates a sequence of time-stamped frames that must traverse the network along a fixed route from talker to listener, as specified in the scenario definition, subject to TAS scheduling. When multiple streams share network resources, their timing requirements collectively determine the scheduling complexity.

C. TAS Model

TAS is a traffic shaping mechanism designed to provide deterministic transmission in TSN. TAS operates by allocating precisely defined time windows during which specific traffic classes or queues are permitted to transmit. By aligning these transmission opportunities across all nodes in the network, TAS guarantees that critical traffic (e.g., control or sensor data) can traverse the network with bounded latency and no interference from lower-priority or best-effort traffic.

At each egress port, TAS divides the transmission buffers into multiple priority queues, denoted as $\{Q_1, Q_2, \ldots, Q_p\}$, where Q_1 typically represents the highest-priority class and Q_p the lowest. There is a one-to-one mapping between traffic classes and queues. A transmission gate is associated with each queue and can be either *open* (allowing transmission) or *closed* (blocking transmission). The state of these gates over time is defined by a GCL.

A GCL specifies a cyclic schedule that determines when each queue is allowed to transmit. It is organized as a repeating cycle of duration T_{cycle}, partitioned into non-overlapping time segments $\Delta_1, \Delta_2, \ldots, \Delta_m$, where m denotes the number of time segments within one TAS cycle. Each segment Δ_i corresponds to a continuous time window in the cycle during which a subset of queues, possibly none of them, is allowed to transmit.

The GCL configuration for a given port can therefore be represented as a binary matrix $X = [x_{ij}]_{m\times p}$, where:

$$x_{ij} = \begin{cases} 1, & \text{if the gate of } Q_j \text{ is open during segment } \Delta_i, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Hence, the complete GCL for a port can be expressed as a table structure, as shown in Table I, where each row corresponds to a time segment Δ_i and each column represents a queue Q_j.

TABLE I: Structure of a GCL

Time Segment	Q_1	Q_2	$\cdots$	Q_p
Δ_1	x_{11}	x_{12}	$\cdots$	x_{1p}
Δ_2	x_{21}	x_{22}	$\cdots$	x_{2p}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
Δ_m	x_{m1}	x_{m2}	$\cdots$	x_{mp}

When a queue's gate is open, frames stored in that queue can be transmitted according to their internal First-In, First-Out (FIFO) order. If multiple queues are open simultaneously, the hardware transmission selection typically follows priority-based arbitration, ensuring that higher-priority queues are served first. Conversely, when the gate of a queue is closed, no frame from that queue may be transmitted, even if the link is idle [27].

The precise alignment of GCLs across all switches and ECUs is crucial for achieving deterministic end-to-end performance. By coordinating the open/closed intervals along each frame's route, TAS ensures that frames can traverse multiple hops without unnecessary waiting time, minimizing jitter and guaranteeing that deadlines are met.

Because different streams in the network may have distinct transmission periods T_s, the TAS cycle time T_{cycle} is typically chosen as the least common multiple of all stream periods. This ensures that the entire network schedule repeats deterministically after every T_{cycle}.

D. Timing and Validity Constraints

In a TAS-based network, each frame must not only meet its end-to-end deadline but also comply with the transmission windows defined by the GCLs along its path. All network devices are time-synchronized (using IEEE 802.1AS [28]) to ensure consistent operation of these time-triggered schedules.

Let $t_{\text{start},s}(k)$ denote the start time of the k-th period of stream s, and $t_{\text{delivery},s}(i,k)$ the time at which the i-th frame in that period reaches its destination. The end-to-end delay of each transmitted frame is defined as:

$$D_s(i,k) = t_{\text{delivery},s}(i,k) - \big(t_{\text{start},s}(k) + t_{\text{release},s}(i)\big). \tag{5}$$

Each frame must therefore satisfy $D_s(i,k) \le d_s$. Because multiple streams share network resources such as links and queues, these timing relations become interdependent across hops. The overall schedule must ensure that all streams meet their deadlines under the combined effect of GCL timing, link bandwidths, and propagation delays. This interdependence forms a coupled temporal constraint system, where the validity of one stream depends on the alignment of others sharing the same network resources.

E. Optimization Objective

The goal is to compute, for every egress port, a GCL that satisfies the end-to-end timing constraint and optimizes overall performance. The decision variables are the per-segment gate states $x_{ij} \in \{0,1\}$, number of segments m, and the segment durations that form the TAS cycle. The optimization objectives are the following.

Maximizing deadline satisfaction: For a given scenario, the primary objective is to maximize the number of frames that meet their end-to-end deadlines. Let $\delta_{s,i} = 1$ if frame i of stream s meets its deadline and $\delta_{s,i} = 0$ otherwise. The success rate is then defined as

$$SR = \frac{\sum_{s\in S}\sum_{i=1}^{n_{f,s}} \delta_{s,i}}{\sum_{s\in S} n_{f,s}} \times 100\%, \tag{6}$$

and the objective is to maximize SR.

Delay minimization: Among all valid schedules, another objective is to minimize the aggregate end-to-end delay across all frames. Since the schedule repeats in cycles, we consider only the first cycle to compute the average delay.

$$D = \frac{1}{\sum_{s\in S} n_{f,s}} \sum_{s\in S}\sum_{i=1}^{n_{f,s}} D_s(i,1). \tag{7}$$

Maximizing resource efficiency: We measure how effectively links are used during the time they are *allocated* for transmission by the GCLs. For each link $e \in E$, let OccupiedTime_e be the total time the link actually transmits within a cycle, and AllocatedTime_e the total time at least one permitting gate is open on that link. The overall utilization is

$$OU = \frac{\sum_{e \in E} \text{OccupiedTime}_e}{\sum_{e \in E} \text{AllocatedTime}_e} \times 100\%, \tag{8}$$

with complementary average idle time equal to 100% - OU.

This definition ensures that a high OU reflects how well those allocated windows are actually used for transmissions, and indirectly how much time is allocated by the GCLs [23].

F. Adaptive Scheduling

The literature recognizes several definitions and categorizations of scheduling algorithms, such as *offline* versus *online*, *static* versus *dynamic*, and *adaptive* versus *non-adaptive* scheduling. For example, in [29], offline scheduling is defined as the case where all tasks and their parameters are fully known in advance and the complete schedule is precomputed before runtime, while online scheduling refers to scenarios where the scheduler must make decisions at runtime with incomplete or evolving information. In this work, we adopt the definition of *adaptive scheduling* from [30]. A scheduler is considered adaptive if it can dynamically generate or update schedules during runtime in response to changes in traffic, topology, or new application requests, and can do so within an acceptable computation time.

In summary, the TAS scheduling problem for IVNs is to determine, for every egress port in the network, a valid GCL that ensures all time-triggered streams meet their end-to-end deadlines while minimizing overall latency and maximizing link utilization. For *adaptive scheduling*, a key challenge is to derive schedules within very short time frames.

IV. Deep Reinforcement Learning Scheduler

This section introduces the overall architecture of the proposed DRL-based scheduler and explains how its main components interact. The proposed scheduler uses a feedback (closed) loop: the policy generates a high-level schedule template, that is compiled into per-port GCLs that satisfy IEEE 802.1Qbv timing rules, and scored on deadline success, latency, and utilization. This score is used to update the policy. The scheduler structure is modular, separating learning, compilation, and evaluation, as illustrated in Fig. 1.

A. Evaluation Environment

The evaluation environment is part of the learning environment that consists of three components: the evaluation environment, a reward calculator, and a state calculator. The evaluation environment builds on a unified network performance model that provides a fair and consistent evaluation for all schedulers. Fig. 2 illustrates how the network performance model, which serves as the evaluation environment, interfaces with the scheduler plug-ins used in this study. All schedulers

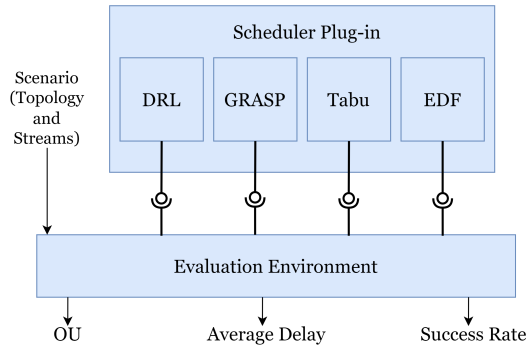


Fig. 2: Integration of scheduler plug-ins with the evaluation environment.

are implemented as plug-ins that generate their own GCLs. These GCLs, together with the network topology and traffic scenario, are provided as inputs to the model, which executes a cycle-accurate TAS simulation. The model then evaluates the schedules and produces key performance metrics, including the success rate, average delay, and overall utilization. This evaluation module is powered by the **INSIM** application [31], which provides the underlying network analysis engine and ensures that all schedulers are tested under identical assumptions and timing semantics.

Inputs: The environment receives three inputs: (i) the network topology $G = (V, E)$ and link parameters defined in (2), (ii) the set of periodic streams $S = \{s_1, \ldots, s_n\}$ as defined in (3), and (iii) the corresponding per-port GCLs generated by the scheduler under test.

Functionality: Each stream s has a fixed path from its talker to its listener. The environment simulates frame transmissions along these paths. For each link $e = (u, v, b_e)$, the transmission time of one frame is $\tau_{s,e} = \frac{8 f_s}{b_e}$, where f_s is in bytes and b_e in bits per second.

The environment simulates frame transmission during one cycle. It tracks the delivery time $t_{\text{delivery},s}(i)$ of each frame i, whether the frame meets its deadline, and how long each link is active or idle under the generated GCLs.

Outputs and Metrics: The model reports the three metrics SR, D, and OU defined in Sec. III-E.

B. DRL scheduler

Several DRL architectures could, in principle, be applied to the TAS scheduling problem, such as value-based methods [32], policy-gradient approaches [33], actor–critic variants [34], deterministic continuous-control algorithms [35], and entropy-regularized off-policy techniques [36]. We choose PPO [37] because it balances stability and simplicity for our "compile–simulate–score" loop. PPO is on-policy (no replay buffer), uses clipped updates that are easy to tune, and naturally emits our mixed action outputs (positive segment durations and traffic class budgets, see below, via simple heads). Its built-in entropy term encourages exploration in the combinatorial schedule space, and it integrates cleanly with our GNN encoder and batched scenario evaluation. In practice, it yields reliable improvements in deadline success and latency while keeping utilization high.

PPO is an actor–critic method. The *actor* is the policy $\pi_\theta(act \mid a)$, parameterized with parameters θ, that maps (network and traffic) *states* a to a probability distribution over *actions* act. The system state is the GNN-encoded representation of the network scenario at hand. The precise definitions of system states and scheduling actions are elaborated below, in Section IV-D. The *critic* is a scalar baseline $J_\phi(a)$ with parameters ϕ; it predicts the expected return from state a and reduces the variance of the policy gradient. Let a and a' denote the current and next network state, respectively. Similarly, θ and θ' denote the policy parameters before and after the update, and ϕ and ϕ' denote the critic parameters before and after the update. Training proceeds in short cycles:

1) **Collect rollouts.** For each state a in the rollout episode, sample an action act from the policy distribution $\pi_\theta(\cdot|a)$, compile it to GCLs, simulate the resulting schedule, and obtain the next state a' together with the corresponding reward $r(a')$ (detailed in Sec. IV-E).
2) **Estimate advantages.** For each (a, act), compute $\mathrm{Adv}(a, act)$, as specified below, which measures how much the selected action performed better or worse than the critic's baseline prediction $J_\phi(a)$.
3) **Constrain the update.** PPO uses a clipped probability ratio between the updated policy $\pi_{\theta'}$ and the old policy π_θ to prevent large policy shifts in a single update of the policy parameters θ', yielding stable learning with minimal tuning.
4) **Compute importance ratio.** To control the magnitude of policy updates, PPO introduces an importance ratio $\rho(a, act; \theta, \theta')$ that measures how the probability of selecting action act changes between the new policy θ' and the policy used to collect the rollout data.
5) **Update the critic.** Fit $J_{\phi'}$ by updating parameters ϕ' to the observed empirical returns so that subsequent advantage estimates become well centered and less noisy.

The advantage and importance ratio are defined as:

$$\mathrm{Adv}(a, act) = r(a') - J_\phi(a), \tag{9}$$

$$\rho(a, act; \theta, \theta') = \frac{\pi_{\theta'}(act \mid a)}{\pi_\theta(act \mid a)}, \tag{10}$$

where $r(a')$ is the reward for the next state a', $J_\phi(a)$ is the critic's estimated value for state a, $\pi_{\theta'}(act \mid a)$ is the updated policy, and $\pi_\theta(act \mid a)$ is the previous policy before the update.

With a minibatch $\mathcal{M}$ of rollout samples, the PPO objective that we *maximize* is

$$\begin{aligned}\mathcal{L}_{\text{PPO}}(\theta', \phi') = \mathbb{E}_{(a,act)\in\mathcal{M}}\Big[&\min\Big(\rho(a, act; \theta, \theta')\,\mathrm{Adv}(a, act), \\ &\mathrm{clip}\big(\rho(a, act; \theta, \theta'), 1-\epsilon, 1+\epsilon\big)\,\mathrm{Adv}(a, act)\Big)\Big] \\ &- \beta\,\mathbb{E}_{a\in\mathcal{M}}\big[\gamma(\pi_{\theta'}(\cdot|a))\big] - \lambda\,\mathbb{E}_{a\in\mathcal{M}}\big[(J_{\phi'}(a) - r(a'))^2\big].\end{aligned} \tag{11}$$

Here, ϵ is the clipping parameter, $\gamma(\cdot)$ denotes the policy entropy (scaled by β), and λ weights the critic loss. Each minibatch $\mathcal{M}$ contains a collection of rollout samples gathered from multiple scenarios in parallel, representing diverse network states and traffic configurations. This batching ensures that every PPO update reflects an average behavior across different conditions, improving learning stability.

C. GNN Encoder

The policy employs a scenario-aware encoder that transforms the IVN graph into compact, learnable representations. The encoder is implemented as a Graph Neural Network (GNN) that aggregates structural and traffic information across the graph and streams defined in (1)–(3). This embedding provides each network node with a summary of its position, role, and communication load in the IVN. In our implementation, the GNN encoder consists of three message-passing layers with hidden embedding size of 128 per node. Each layer applies a ReLU activation and layer normalization, followed by residual aggregation to stabilize training. We observed that using fewer than three layers limited the DRL scheduler's ability to capture multi-hop dependencies, while deeper configurations caused over-smoothing and slower convergence of the PPO training without noticeable performance gain. The chosen depth provides a good balance between representation capacity and computational efficiency for IVN topologies.

Input Features: Each node $v \in V$ (either a switch or an end system) is initialized with a feature vector $\mathbf{fv}_v \in \mathbb{R}^{d_v}$ containing: (i) one-hot indicator of its type or role (switch, controller, sensor, and actuator), (ii) normalized in-degree and out-degree, and (iii) an estimated load derived from the number of streams traversing that node. Each directed link $(u, v) \in E$ carries an edge feature vector $\boldsymbol{\eta}_{(u,v)} \in \mathbb{R}^{d_e}$ that includes normalized link capacity, relative utilization, normalized node depths, and indicators of the transmitter and receiver types. Here, d_v and d_e denote the dimensions of the node and edge feature vectors, respectively. These input features enable the GNN to encode both topological and traffic-related properties of the network.

Message Passing and Aggregation: The encoder consists of $(L = 3)$ message-passing layers that iteratively refine node embeddings by combining local features with information from their neighbors. Let $\mathbf{h}_v^{(\ell)}$ denote the embedding of node v at layer ℓ, initialized as $\mathbf{h}_v^{(0)} = \mathbf{fv}_v$. At each layer, node states are updated according to:

$$\mathbf{h}_v^{(\ell+1)} = \sigma\left(W_s^{(\ell)}\mathbf{h}_v^{(\ell)} + \sum_{(u,v)\in E} \mathrm{MLP}^{(\ell)}\big([\mathbf{h}_u^{(\ell)} \,\|\, \boldsymbol{\eta}_{(u,v)}]\big)\right), \tag{12}$$

where $W_s^{(\ell)}$ is a learnable $\mathbb{R}^{d_v} \times \mathbb{R}^{d_v}$ self-projection matrix that preserves and transforms each node's own embedding across layers, $\mathrm{MLP}^{(\ell)} : \mathbb{R}^{d_v+d_e} \to \mathbb{R}^{d_v}$ is a small multi-layer perceptron network that produces edge-conditioned vectors from neighboring nodes, $[\cdot\|\cdot]$ denotes concatenation, and $\sigma(\cdot)$ is the ReLU activation function. Layer normalization is applied after each update to improve stability. After L layers, each node embedding $\mathbf{h}_v^{(L)}$ summarizes both its local characteristics and multi-hop connectivity context.

D. State, Action Representation, and GCL Compiler

The DRL scheduler operates on graph-structured states and outputs a compact scheduling template that is later com-

piled into per-port GCLs. This separation allows the learning module to explore high-level allocation strategies while a deterministic compiler generates IEEE 802.1Qbv compliant GCLs.

State Representation: The policy needs a state description that captures both the global topology and the local transmission context of each queue. To achieve this, the GNN encoder provides a scenario-aware embedding for every node, which is extended with queue- and scenario-specific features.

For each node v, the encoder outputs an embedding $\mathbf{h}_v^{(L)}$ that summarizes its structural and traffic context. For every queue $Q_{v,i_p,c}$ (on node v, port index i_p, and traffic class c), a feature vector is constructed as:

$$\mathbf{z}_{v,i_p,c} = [\,\mathbf{h}_v^{(L)} \,\|\, \mathbf{Q}_{v,i_p,c} \,\|\, \mathbf{y}_g\,], \tag{13}$$

where $\mathbf{Q}_{v,i_p,c}$ encodes local queue attributes (priority, normalized load, and the total number of active queues of the queue's port) and $\mathbf{y}_g$ is a global context vector summarizing scenario statistics, namely the number of streams, number of ports, number of queues (p), average stream period, normalized T_{cycle}, and minimum deadline.

The overall system state a, which serves as the input to the PPO policy described in Section IV-B, is the set of all $\mathbf{z}_{v,i_p,c}$ across queues and ports, ensuring that each scheduling decision reflects both its local conditions and the global network status.

Action Representation: The action defines a compact template that globally synchronizes transmission opportunities in a TAS cycle. Let K denote the number of transmission phases per TAS cycle. This value is the same for all ports and all scenarios. The DRL scheduler does not decide K; it only decides how the cycle is partitioned into phases and how these phases are shared among traffic classes.

The agent outputs two parameter sets. The first is a vector $\boldsymbol{\tau} = [\tau_1, \ldots, \tau_K]$ of non-negative values, where each entry represents the relative length of one phase. The compiler scales these values so that the resulting K phase durations add up exactly to the TAS cycle length T_{cycle}. This defines a global partitioning of the cycle into K consecutive phases with well-defined start and end times, shared by all ports.

The second output is a $K \times p$ class-budget matrix $\mathbf{B} = [b_{i,c}]$, with $b_{i,c} \in [0, 1]$ indicating the fraction of phase i reserved for traffic class c (Recall from Sec. III-C that there are p traffic classes.). If the fractions for a given phase do not sum to one, gates are closed for the remaining part of the phase.

The choice of K determines the temporal resolution of the schedule template and is an implementation decision. We use $K = 5$ in our experiments as it offers a good trade-off between schedule flexibility and training efficiency. Larger values can be used if finer-grained control is desired.

An important observation is that the resulting action space, consisting of $\boldsymbol{\tau}$ and $\mathbf{B}$ pairs, is independent of the number of ports and queues, meaning it has much lower dimensionality than an action space consisting of per-port GCLs. The TSN standard defines up to eight traffic classes. This implies an action dimensionality of $(K + 1) \times p$, which in our implementation is 6×8. Available TAS-capable switches support different maximum numbers of time segments per cycle in their GCLs. Reported implementations support a wide range of maximum GCL sizes, typically between 8 and 1024 entries, depending on the hardware platform and configuration [40]. This corresponds to per-port GCL dimensions between 8×8 and 1024×8 (time slots by traffic classes). Directly generating GCLs as outputs, one for every port in the network, results in an action space that is an order of magnitude larger than in our approach.

GCL Compiler: A deterministic TAS compiler converts the template $(\boldsymbol{\tau}, \mathbf{B})$ into concrete per-port GCLs. It first computes the phase durations and boundaries from $\boldsymbol{\tau}$. Then, for each phase and traffic class, it allocates a GCL time segment Δ whose length equals the class fraction $b_{i,c}$ multiplied by the phase duration. For a given port, only the queues corresponding to an active traffic class receive such a segment; other queues remain closed. In this way, the compiler always produces IEEE 802.1Qbv-compliant GCLs, while the DRL scheduler operates only on high-level phase and budget parameters.

As an example, assume $T_{\text{cycle}} = 100$ μs, $K = 5$ phases, and $p = 8$ traffic classes. Suppose the agent outputs $\boldsymbol{\tau} = [1, 1, 1, 1, 1]$ and

$$\mathbf{B} = \begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The entries of $\boldsymbol{\tau}$ are relative lengths. The compiler first computes the weight $W = \sum_{k=1}^{5} \tau_k = 5$ and gives phase i a duration $\frac{\tau_i}{W} T_{\text{cycle}}$. Since all τ_i are 1, each phase has duration $100/5 = 20$ μs, so the phases together cover $[0, 100$ μs$)$.

From $\mathbf{B}$, the compiler derives the class windows in each phase. For example, in phase 3 (time interval $[40, 60)$ μs) class 1 receives 0.5×20 μs $= 10$ μs and class 2 also receives 10 μs. In phase 4, all entries in the corresponding row of $\mathbf{B}$ are zero, so all gates are closed in that phase. In phase 5, class 1 and class 3 each receive 0.5×20 μs $= 10$ μs.

The port in the example implements only traffic classes 1, 2, and 3. The compiler constructs the following GCL for that port, using seven consecutive segments that cover the full cycle, one segment in phase 1, one in phase 2, two in phase 3, none in phase 4, and two in phase 5:

Segment	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
$\Delta_1 = [0, 20)$	1	0	0	0	0	0	0	0
$\Delta_2 = [20, 40)$	0	1	0	0	0	0	0	0
$\Delta_3 = [40, 50)$	1	0	0	0	0	0	0	0
$\Delta_4 = [50, 60)$	0	1	0	0	0	0	0	0
$\Delta_5 = [60, 80)$	0	0	0	0	0	0	0	0
$\Delta_6 = [80, 90)$	1	0	0	0	0	0	0	0
$\Delta_7 = [90, 100)$	0	0	1	0	0	0	0	0

During Δ_5, all queues are closed. For inactive queues, all entries in the corresponding column are set to zero.

E. Reward Function

The reward function guides the agent toward schedules that jointly improve deadline satisfaction, latency, and link utilization. For a given state a, being the GNN-encoding of a specific scenario, the agent produces a scheduling action

that is compiled into per-port GCLs, resulting in a new network configuration corresponding to the resulting state a'. Specifically, each action is compiled into per-port GCLs, transforming the system state from a to a'. The corresponding configuration is evaluated by the environment to obtain the success rate, delay, and overall utilization, denoted respectively as SR, D, and OU, as defined in Section III-E. The scalar reward provided to the agent after each evaluation, the $r(a')$ value in the advantage computation of Eq. (9) and the PPO objective of Eq. (11), is a weighted combination of these three objectives, $\alpha_1 SR - \alpha_2 D + \alpha_3 OU$, where $\alpha_1, \alpha_2, \alpha_3 > 0$ are scalar coefficients controlling the relative influence of each term. The first term rewards schedulers that deliver a higher fraction of frames within their deadlines, the second penalizes longer delays, and the third promotes efficient utilization of open transmission windows. All metrics are normalized to the range $[0, 1]$ to ensure numerical stability and consistent scaling across scenarios. The influence of reward weight selection is analyzed in Section V-E.

F. Complexity

The computational complexity of the proposed DRL scheduler can be analyzed separately for the inference and training phases.

During inference, the GNN encoder performs $L = 3$ message-passing layers over the network graph $G = (V, E)$, processing each node and edge at every layer. Since L, d_v, and d_e are fixed hyperparameters, the inference cost scales linearly with the number of nodes $|V|$ and edges $|E|$. The policy head produces an action of fixed dimensionality $(K + 1) \cdot p = 48$, independent of topology size, and the deterministic GCL compiler maps this template to per-port GCLs in time proportional to the number of ports. Overall, the time and memory complexity of a single scheduling decision are $\mathcal{O}(|V|+|E|)$ and $\mathcal{O}(|V|\cdot d_v+|E|\cdot d_e)$, respectively, making inference fast and well suited for runtime configuration.

After a a schedule is generated, it will be validated by the analytical evaluation module, which tracks frame transmissions across all streams $s \in S$ and links in E. This step scales with the number of streams and network size in both time and memory.

During training, each episode combines a GNN forward pass with analytical validation over a minibatch ($\mathcal{M}$) of scenarios, so memory scales as $\mathcal{O}(|\mathcal{M}| \cdot (|V| \cdot d_v + |E| \cdot d_e))$. This cost is incurred entirely offline; once trained, the agent generates scheduling decisions in a single forward pass.

V. Experiments and Results

This section evaluates the performance, scalability, and adaptability of the proposed DRL scheduler. We assess its ability to generate valid high-quality TAS schedules under diverse traffic conditions and compare it with established heuristic and DRL-based baselines.

First, we analyze the runtimes of the baseline heuristic and DRL-based solvers and the proposed DRL method to assess their degree of adaptivity and to identify suitable hyperparameter settings for two of the reference algorithms that keep their runtimes similar to the DRL runtime. Then, we compare all schedulers in terms of schedule quality (success rate, average end-to-end delay, and overall utilization) across increasingly challenging test scenarios. Finally, we conduct two additional studies: (i) an analysis of the effect of reward weight selection on performance trade-offs, and (ii) a component ablation study to quantify the individual effect of the GNN encoder and the compact action design.

A. Experimental Setup

To train and evaluate the proposed scheduler, we generated a comprehensive set of 360 synthetic TAS scenarios. Out of the 360 generated scenarios, 300 were used for training the DRL scheduler and 60 were reserved for testing and comparison. The test set was selected such that it spans the full range of complexity levels. Each scenario represents a complete in-vehicle network instance with defined topology, traffic flows, and timing requirements.

The scenarios were produced automatically using a dedicated generator that incrementally increases complexity while ensuring that all periods and deadlines remain physically feasible with respect to the network topology, link capacities, and frame sizes. For every traffic flow, the generator calculates the minimum possible transmission time along its path and ensures that both the stream period and deadline are large enough to make real-time transmission achievable.

Scenario difficulty grows gradually through several mechanisms. At each step, additional traffic flows are introduced between sensors, zone controllers, and the central computer, frame sizes are increased, deadlines are tightened, and release times are varied to create overlapping transmissions and congestion on shared links. Early scenarios therefore represent light and easily schedulable workloads, whereas later ones contain dense and highly interdependent communication patterns that more strongly challenge the schedulers. Table II summarizes the key characteristics of the test scenarios across different difficulty stages. In TSN applications, a stream is typically classified as time-critical if its deadline-to-period ratio is less than 0.5 [39]. We follow the same definition in this table.

All algorithms were evaluated using the analytical simulation framework described in Section IV-A. Experiments were implemented in the **INSIM** engine [31] and executed on a machine with an Intel Core i7-13700H CPU, 16 GB RAM, and an NVIDIA RTX A1000 6GB GPU running on windows. The proposed DRL scheduler required approximately 1500 training episodes to achieve stable convergence, corresponding to about 2–3 hours of training time on the specified hardware. Fig. 3 illustrates the episode reward over time, where results are averaged over five independent runs with different random seeds and smoothed using a moving average with a window size of 50 episodes.

The PPO loss hyperparameters are set as follows: the clipping parameter is $\epsilon = 0.2$, the critic loss weight is $\lambda = 0.5$, and the entropy regularization coefficient β is linearly annealed from 0.01 to 0.001 during training to encourage exploration in early stages and stabilize convergence.

TABLE II: Detailed specifications of test scenarios across staged complexity levels

Scenario ID	Topology Description	Number of Streams	Time-Critical (%)
1–12	6 Zones, 6 Zone Switches, 6 Zone Controllers, 1 Central Switch, 1 Central Computer, 18 Sensors/Actuators	12–59	0–10
13–24	6 Zones, 6 Zone Switches, 6 Zone Controllers, 1 Central Switch, 1 Central Computer, 20 Sensors/Actuators	64–117	10–20
25–36	6 Zones, 6 Zone Switches, 6 Zone Controllers, 1 Central Switch, 1 Central Computer, 22 Sensors/Actuators	121–177	20–25
37–48	6 Zones, 6 Zone Switches, 6 Zone Controllers, 1 Central Switch, 1 Central Computer, 24 Sensors/Actuators	183–255	25–28
49–60	6 Zones, 6 Zone Switches, 6 Zone Controllers, 1 Central Switch, 1 Central Computer, 26 Sensors/Actuators	262–351	28–31

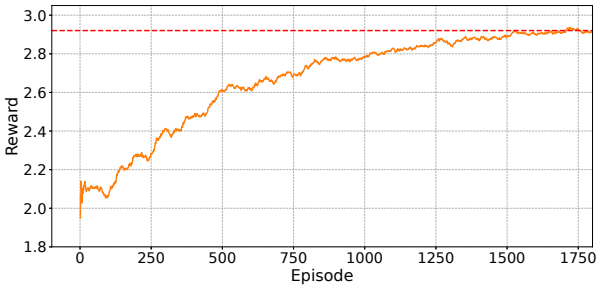


Fig. 3: Learning curve of the proposed DRL scheduler (50-episode moving average and averaged over five random seeds).

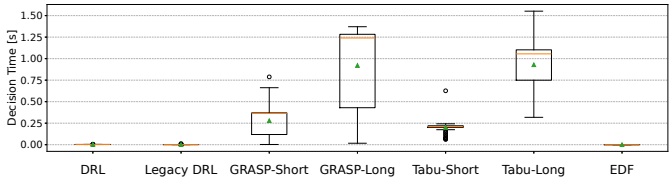


Fig. 4: Distribution of scheduling decision times across all 60 test scenarios for each algorithm.

B. Baseline Algorithms

Based on the adaptiveness definition provided in Section III, this study focuses on comparing algorithms that can generate schedules under changing network conditions. Exact optimization approaches such as ILP or SMT solvers, while capable of finding optimal schedules for static configurations, are excluded from our comparison due to their high computational cost and limited runtime adaptability. Instead, we evaluate methods that can construct or update valid GCLs within practical time bounds.

Among heuristic schedulers, two algorithms have demonstrated strong performance in recent benchmarking studies of TAS scheduling [12]: the Greedy Randomized Adaptive Search Procedure and Tabu Search. In addition to these, we also include the EDF scheduler [38] as a lightweight reference algorithm as it provides very fast scheduling decisions, making it relevant when evaluating adaptiveness where runtime efficiency and rapid reconfiguration are key. Beyond heuristic baselines, we also evaluate against our prior DRL-based TAS scheduler [23].

1) EDF: EDF prioritizes frames based on their deadlines and transmits those with the earliest deadlines first. It offers extremely low computational cost and fast decision making, which makes it suitable as a baseline for adaptivity evaluation.

2) GRASP: The GRASP scheduler constructs initial valid schedules through a randomized greedy procedure and then refines them by local search. Its stochastic element helps escape local minima and explore diverse valid solutions, while its adaptive construction ensures moderate runtime scalability.

3) Tabu Search: The Tabu scheduler iteratively explores neighboring schedules while maintaining a short-term memory of visited configurations to prevent cycling. It efficiently improves schedule quality through guided local exploration and has been shown to outperform many earlier heuristic approaches on large-scale TAS instances.

4) Legecy DRL-based TAS Scheduler: The DRL scheduler proposed in [23] (referred to in this work as Legacy-DRL) directly generates TAS parameters using a flat state representation and a port-based action space. Its action dimensionality is explicitly tied to the number of ports in the network, which fundamentally restricts structural generalization and prevents operation under topology changes. For this study, we trained five independent instances of this agent, one per evaluated topology, to enable direct performance comparison with the proposed scenario-aware framework.

Together, these four baselines represent different levels of adaptiveness and computational complexity, providing a sound basis for assessment of the proposed DRL-based scheduler in terms of runtime efficiency and scheduling quality.

C. Adaptivity and Inference Time Analysis

According to the definition provided in Section III, all five evaluated algorithms are adaptive, as they can generate new TAS schedules when network or traffic conditions change. However, their degree of adaptivity also depends on how quickly they can recompute a valid schedule. To analyze this property, we measured the inference time of each scheduler across all 60 test scenarios. The distributions of these results are presented in Fig. 4 as box plots. The central orange line in each plot indicates the median value, the box height represents the interquartile range (IQR) between the 25th and 75th percentiles, and the green triangle marks the mean. Whiskers show the spread of data within 1.5 times the IQR, and individual circles represent outliers beyond this range.

As shown, the DRL, Legacy-DRL and EDF schedulers exhibit consistently low inference times close to zero, as they compute schedules in a single forward pass without iterative optimization. This property makes them highly adaptive and suitable for runtime reconfiguration. In contrast, the GRASP

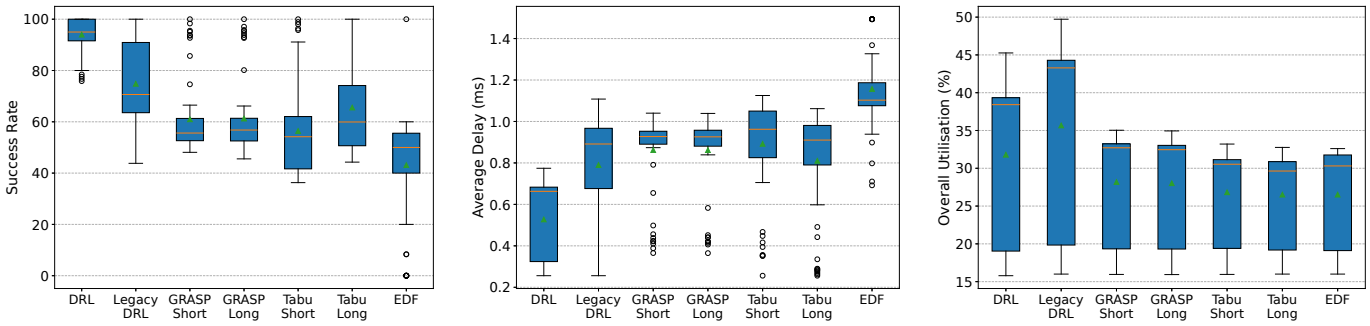


Fig. 5: Distribution of performance metrics across all 60 test scenarios for DRL, GRASP, Tabu, and EDF schedulers. (a) Success rate, (b) average end-to-end delay, and (c) overall utilization. For GRASP and Tabu, results are averaged over 30 independent runs with different random seeds.

and Tabu schedulers rely on iterative search procedures whose runtime depends on the number of iterations and the size of the local or neighborhood search. To illustrate this dependency, two configurations of each algorithm were analyzed. For GRASP, the first configuration, referred to as GRASP-Short, uses 30 global iterations and 15 local refinements per iteration, while the second configuration, GRASP-Long, increases these numbers to 90 and 45, respectively. Similarly, Tabu-Short uses 10 iterations with a neighborhood size of 3, whereas Tabu-Long extends these to 50 iterations and a neighborhood size of 30. The results show that increasing these parameters significantly raises the computation time and its variability across scenarios. The longer variants are less adaptive due to their higher decision latency. Nevertheless, we compare our DRL scheduler with both the short and long instances of GRASP and Tabu, because runtimes up to several seconds may be acceptable, depending on the use case.

D. Results and Discussion

The performance comparison of the schedulers across the 60 test scenarios is shown in Fig. 5. Subfigures (a)–(c) present the distribution of success rate, average end-to-end delay, and overall utilization, respectively, following the box-plot conventions introduced in Section V-C. For GRASP and Tabu, the reported results are obtained by averaging over 30 independent runs with different random seeds to reduce stochastic variation and provide a fair statistical basis.

The results indicate that the proposed DRL scheduler achieves consistently higher success rates and lower average delay while maintaining reasonable link utilization across all scenarios. GRASP and Tabu show a larger dispersion in performance due to their randomized local search mechanisms and sensitivity to initialization. Tabu provides better results with more iterations and larger neighborhoods. GRASP-Long does not improve over GRASP-Short. The EDF baseline shows the lowest success rate and the highest delay, reflecting its limits in handling contention and network dependencies.

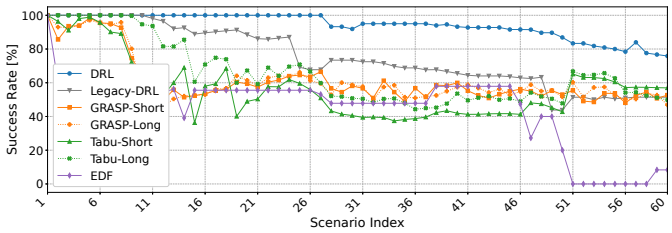
Compared to Legacy-DRL, the proposed scheduler generally attains higher success rates in complex scenarios, whereas Legacy-DRL occasionally achieves higher overall utilization. This behavior can be attributed to the different action space designs. While Legacy-DRL directly manipulates per-port GCL parameters and can tightly pack transmission windows to maximize utilization, the proposed template-based phase abstraction prioritizes global coordination and deadline satisfaction. As a result, improved schedulability may in some cases come at the cost of reduced utilization efficiency. A more detailed analysis of this trade-off is provided in Section V-E.

Fig. 6 provides a per-scenario view of the performance trends across the 60 test scenarios. Subfigures (a)–(c) illustrate the evolution of success rate, average delay, and overall utilization, respectively, as scenario complexity increases.

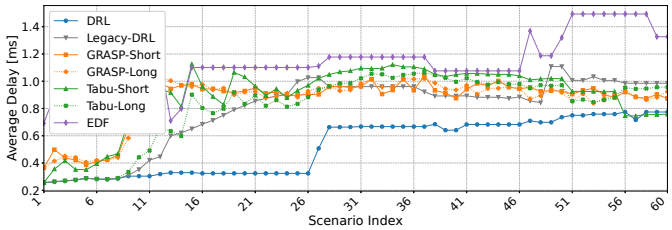
The results show that the DRL scheduler maintains a nearly constant and high success rate throughout all scenarios, indicating that the learned policy adapts reliably even as network load grows. In contrast, the heuristic baselines exhibit larger fluctuations due to the randomized search and initialization sensitivity. As the number of streams and their path overlaps increase, both GRASP and Tabu begin to lose validity for some flows, resulting in visible drops in success rate and a corresponding decline in utilization efficiency. The EDF scheduler shows the lowest success rate in dense scenarios. Legacy-DRL demonstrates competitive success rate in simpler scenarios but exhibits reduced performance in complex scenarios. The more abstract phase-based action design of the DRL scheduler proposed in this paper makes it easier to find feasible schedules for a wider variety of encountered scenarios.

In terms of delay, DRL consistently achieves the lowest average delay, followed by GRASP and Tabu, and finally EDF. The smooth profile of the DRL curve suggests that the policy produces well-aligned transmission windows across all nodes, minimizing waiting times at intermediate switches. The heuristic approaches exhibit more pronounced variability.

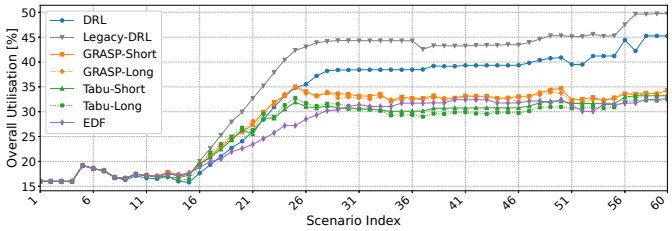
Overall utilization shows a generally increasing trend as the traffic load grows. In easier scenarios, all algorithms achieve similar utilization, and in some cases the heuristics reach slightly higher values than the DRL scheduler. As scenarios grow more demanding, DRL maintains a stable and competitive level of utilization with higher success rates and lower delays. The DRL scheduler effectively balances resource use and timing performance across different network



(a)

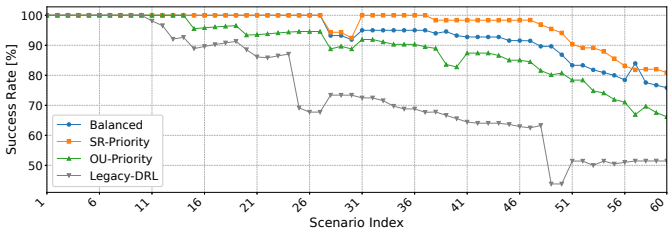


(b)

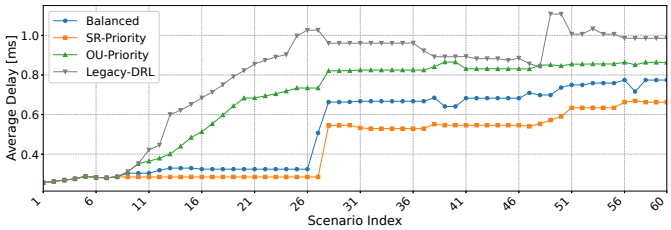


(c)

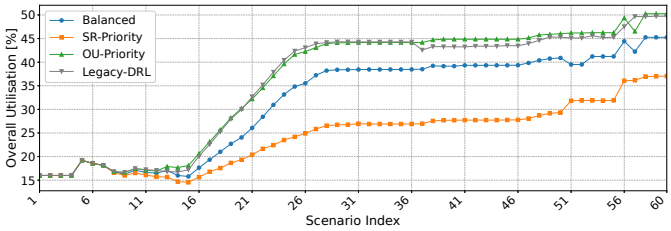
Fig. 6: Per-scenario comparison of DRL, GRASP, Tabu, and EDF schedulers across all 60 test scenarios. (a) Success rate, (b) average end-to-end delay, and (c) overall utilization. For GRASP and Tabu, results are averaged over 30 independent runs with different random seeds.



(a)



(b)



(c)

Fig. 7: Per-scenario comparison of the three reward-weight configurations of the proposed DRL scheduler across all test scenarios. (a) Success rate, (b) average end-to-end delay, and (c) overall utilization.

conditions. Legacy-DRL achieves higher overall utilization due to its direct per-port action design.

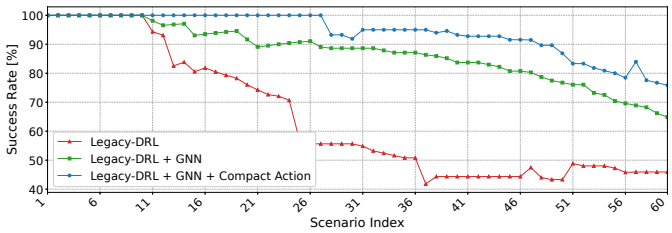
Regarding the output validity, all algorithms are evaluated through the analytical model described in Section IV-A. This evaluation layer checks every generated schedule against the timing, queueing, and transmission constraints of the TAS standard.

In a practical deployment, an invalid schedule should trigger a higher-level supervisory mechanism within the vehicle's network management system. This supervisory layer should enforce corrective actions, such as temporarily blocking non-critical traffic, adjusting control loop rates, or switching to a fallback schedule to maintain safety and determinism.

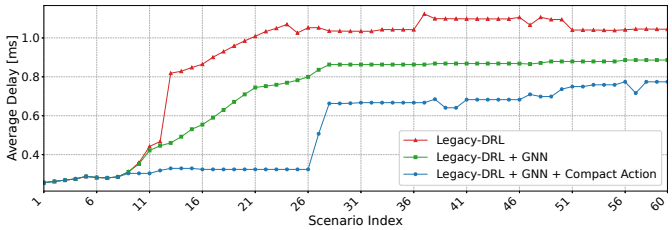
E. Effect of Reward Weights

To evaluate the influence of reward weighting on scheduling behavior, we trained three independent instances of the proposed DRL scheduler using different reward coefficient configurations. The first configuration, referred to as *Balanced*, employs weights $(\alpha_1, \alpha_2, \alpha_3) = (3.0, 0.15, 3.0)$ for success rate, average delay, and overall utilization, respectively. This configuration is used throughout Section V-D and Section V-F. The second configuration, denoted as *SR-Priority*, emphasizes deadline satisfaction and low delay with weights $(5.0, 1.0, 0.5)$. In this setting, both timing-related weights are increased relative to utilization. The third configuration, denoted as *OU-Priority*, prioritizes resource efficiency using weights $(1.0, 0.5, 5.0)$. In this case, the weights of both success rate and delay are reduced, reflecting a shift away from strict timing optimality toward higher bandwidth utilization.

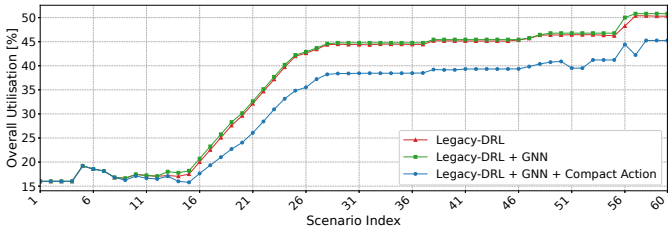
Fig. 7 illustrates the results for the three trained agents, with respect to success rates, average end-to-end delay, and overall utilization across scenarios. For clearer comparison of trade-offs, we additionally include the results of the Legacy-DRL baseline in this figure. The SR-Priority DRL scheduler achieves the highest success rate in most scenarios. However, this improvement comes at the cost of reduced overall utilization, indicating increased idle time within allocated transmission windows. In contrast, the OU-Priority DRL scheduler attains the highest utilization but exhibits the lowest success rate, showing that aggressively maximizing bandwidth efficiency can adversely impact schedulability. The Balanced configuration lies between these extremes, providing a favorable compromise between success rate and resource utilization. Notably, as observed in the figure, the OU-Priority configuration achieves overall utilization values close to those of Legacy-DRL, while outperforming it in terms of success rate across all scenarios. Moreover, the proposed framework preserves this performance without topology-specific retrain-



(a)



(b)



(c)

Fig. 8: Per-scenario comparison of Legacy-DRL, Legacy-DRL+GNN, and the proposed DRL framework across all test scenarios. (a) Success rate, (b) average end-to-end delay, and (c) overall utilization.

ing.

F. Component Ablation Study

To analyze the effect of each architectural component in the proposed framework, we conducted an ablation study by training three different DRL scheduler variants.

The first variant corresponds to the Legacy-DRL baseline described in Section V-B. This scheduler employs a flat state representation without scenario-aware encoding and utilizes a direct per-port action space for TAS configuration, making it impossible to handle topology changes. As discussed earlier, separate instances of this scheduler were trained for each of the five evaluated network topologies.

The second variant, denoted as Legacy-DRL+GNN, augments the Legacy-DRL architecture with the proposed GNN-based scenario-aware state encoder while retaining the original per-port action representation. This configuration isolates the impact of the GNN encoder on scheduling performance.

The third variant corresponds to the complete proposed framework, referred to as Legacy-DRL+GNN+Compact Action. This scheduler integrates all components introduced in this paper, including the scenario-aware GNN encoder, the compact phase-based action representation, and the deterministic IEEE 802.1Qbv-compliant GCL compiler, enabling topology-independent operation.

Fig. 8 compares the performance of the three scheduler variants across all test scenarios. Incorporating the GNN encoder significantly improves success rate and reduces average end-to-end delay, while having only a minor impact on overall utilization. Introducing the compact action abstraction and GCL compiler further enhances schedulability and latency performance by removing topology sensitivity and promoting global coordination across ports. However, this improvement comes at the cost of a reduction in OU.

Overall, the ablation results reveal a trade-off between maximizing utilization and achieving robust deadline satisfaction across varying scenarios. While scenario-aware encoding and compact action design substantially improve scheduling reliability and delay performance, tighter per-port control can in some cases yield higher utilization.

VI. Conclusion

This paper presented a scenario-aware DRL scheduling agent for TAS in IVNs. The approach combines a GNN encoder with a compact policy representation trained using the PPO algorithm. It learns to allocate transmission opportunities in a manner that satisfies strict timing constraints while balancing high link utilization and low latency.

Experimental evaluation demonstrates that the proposed DRL scheduler outperforms representative heuristic baselines, namely GRASP and Tabu Search, as well as the Legacy-DRL baseline, in terms of success rate and average delay. Across a wide range of scenarios with increasing complexity, the learned policy maintains a high success rate and stable performance, showing a strong generalization to unseen network scenarios. Its inference time is negligible compared to iterative search methods, making it suitable for adaptive or online reconfiguration in software-defined vehicles.

The analytical evaluation framework ensures that all algorithms were tested under identical timing semantics and provides guaranteed validation of the generated schedules. This integration of learning-based decision making with formal analytical verification represents an important step toward safe and adaptive real-time communication systems.

Although the proposed framework ensures that all generated schedules are analytically validated for standard compliance, a higher-level supervisory mechanism is still required in practical deployments to handle situations where the DRL scheduler fails to produce a feasible configuration within strict timing constraints. Moreover, while the structured compact action space removes the fixed-topology limitation of prior DRL approaches and improves schedulability under complex scenarios, it reduces direct fine-grained per-port control, which can lead to a modest decrease in overall link utilization compared to topology-specific DRL schedulers.

Future work may extend this framework to support multi-agent scheduling across distributed switches, integrate traffic prediction for proactive adaptation, and explore hardware-in-the-loop validation in automotive prototypes.

Acknowledgment

This work received funding from the European Chips Joint Undertaking under Framework Partnership Agreement No

101139789 (HAL4SDV).

REFERENCES

[1] Z. Liu, W. Zhang, and F. Zhao, "Impact, Challenges and Prospect of Software-Defined Vehicles," *Automotive Innovation*, vol. 5, no. 2, pp. 180–194, 2022.

[2] Y. Peng, B. Shi, T. Jiang, X. Tu, D. Xu, and K. Hua, "A Survey on In-Vehicle Time-Sensitive Networking," *IEEE IoT Journal*, vol. 10, no. 16, pp. 14375–14396, 2023.

[3] Y. Xu, J. Shang, and H. Tang, "Recent Trends of In-Vehicle Time Sensitive Networking Technologies, Applications and Challenges," *China Communications*, vol. 20, no. 11, pp. 30–55, 2023.

[4] J. Farkas, L. Lo Bello, and C. Gunther, "Time-Sensitive Networking Standards," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 20–21, 2018.

[5] W. Kong, M. Nabi, and K. Goossens, "Run-time Per-Class Routing of AVB Flows in In-Vehicle TSN via Composable Delay Analysis," in *VTC2022-Spring*, 2022, pp. 1–7.

[6] "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.

[7] Y. Zhang, J. Wu, M. Liu, and A. Tan, "TSN-Based Routing and Scheduling Scheme for Industrial Internet of Things in Underground Mining," *Eng. Appl. Artif. Intell.*, vol. 115, p. 105314, 2022.

[8] L. Lo Bello, G. Patti, and G. Vasta, "Assessments of Real-Time Communications over TSN Automotive Networks," *Electronics*, vol. 10, no. 5, p. 556, 2021.

[9] T. Stüber *et al.*, "A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (TSN)," *IEEE Access*, vol. 11, pp. 61192–61233, 2023.

[10] B. Li *et al.*, "Over-the-Air Upgrading for Enhancing Security of Intelligent Connected Vehicles: A Survey," *Artificial Intelligence Review*, vol. 57, no. 11, p. 314, 2024.

[11] S. V. Gheorghita *et al.*, "System-scenario-based design of dynamic embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 1, Article 3, pp. 1–45, Jan. 2009.

[12] T. Stüber *et al.*, "Performance Comparison of Offline Scheduling Algorithms for the Time-Aware Shaper (TAS)," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 7, pp. 9736–9748, 2024.

[13] D. Bujosa *et al.*, "HERMES: Heuristic Multi-Queue Scheduler for TSN Time-Triggered Traffic with Zero Reception Jitter Capabilities," in *RTNS*, 2022.

[14] F. Dürr and N. G. Nayak, "No-Wait Packet Scheduling for IEEE Time-Sensitive Networks (TSN)," in *RTNS*, 2016.

[15] X. Jin *et al.*, "Real-Time Scheduling of Massive Data in Time Sensitive Networks With a Limited Number of Schedule Entries," *IEEE Access*, vol. 8, pp. 6751–6767, 2020.

[16] V. Gavriluţ *et al.*, "AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN," *IEEE Access*, vol. 6, pp. 75229–75243, 2018.

[17] E. Schweissguth et al., "ILP-Based Routing and Scheduling of Multicast Realtime Traffic in Time-Sensitive Networks," in *RTCSA*, 2020, pp. 1–11.

[18] S. S. Craciunas et al., "Scheduling Real-Time Communication in IEEE 802.1Qbv Time-Sensitive Networks," in *RTNS*, 2016.

[19] X. Wang *et al.*, "Towards wireless time-sensitive networking: Multi-link deterministic scheduling via deep reinforcement learning," *Computer Networks*, vol. 261, p. 111119, 2025.

[20] H. Cheng *et al.*, "Reliable routing and scheduling in time sensitive networks based on reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 12, no. 4, pp. 2415-2427, 2025.

[21] A. Roberty et al., "Configuring the IEEE 802.1Q Time-Aware Shaper with Deep Reinforcement Learning," in *NOMS*, 2024, pp. 1–7.

[22] X. He *et al.*, "DeepScheduler: Enabling Flow-Aware Scheduling in Time-Sensitive Networking," in *IEEE INFOCOM*, 2023.

[23] M. Karimi *et al.*, "Deep-Reinforcement-Learning-based Scheduler for Time-Aware Shaper in In-Vehicle Networks," in *2025 VTC-Spring*, IEEE, 2025.

[24] Z. Tian *et al.*, "GTSNet: A Generalized Traffic Scheduler for Time-Sensitive Networking Based on Graph Neural Network," *IEEE Transactions on Industrial Informatics*, vol. 21, no. 1, pp. 208-217, 2024.

[25] P. Pop, K. Alexandris, and T. Wang, "Configuration of multi-shaper Time-Sensitive Networking for Industrial Applications," *IET Networks*, vol. 13, no. 5–6, pp. 434–454, 2024.

[26] G. P. Sharma et al., "End-to-End No-wait Scheduling for Time-Triggered Streams in Mixed Wired-Wireless Networks," *Journal of Network and Systems Management*, vol. 32, no. 3, p. 65, 2024.

[27] Q. Li *et al.*, "A Simple and Efficient Time-Sensitive Networking Traffic Scheduling Method for Industrial Scenarios," *Electronics*, vol. 9, no. 12, p. 2131, 2020.

[28] G. Garner, "Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications," *IEEE Draft Standard 802.1AS*, 2016.

[29] P. Li, *et al.*, "Learning for Sustainable Online Scheduling with Competitive Fairness Guarantees," in *Proc. 16th ACM Int. Conf. Future and Sustainable Energy Systems*, 2025.

[30] D. Mourtzis, "Advances in Adaptive Scheduling in Industry 4.0," *Frontiers in Manufacturing Technology*, vol. 2, p. 937889, 2022.

[31] M. Karimi *et al.*, "INSIM: A Modular Simulation Platform for TSN-based In-Vehicle Networks," in *2025 VTC-Fall*, IEEE, 2025.

[32] J. Clifton and E. Laber, "Q-Learning: Theory and Applications," *Annual Review of Statistics and Its Application*, vol. 7, no. 1, pp. 279–301, 2020.

[33] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[34] X. Xin *et al.*, "Supervised Advantage Actor-Critic for Recommender Systems," in *WSDM*, 2022.

[35] D. Silver *et al.*, "Deterministic Policy Gradient Algorithms," in *ICML*, PMLR, 2014.

[36] T. Haarnoja *et al.*, "Soft Actor-Critic Algorithms and Applications," *arXiv preprint* arXiv:1812.05905, 2018.

[37] J. Schulman *et al.*, "Proximal Policy Optimization Algorithms," *arXiv preprint* arXiv:1707.06347, 2017.

[38] R. Dobrin, N. Desai, and S. Punnekkat, 'On Fault-Tolerant Scheduling of Time Sensitive Networks," in *CERTS*, 2019.

[39] A. Ademaj *et al.*, "Industrial Automation Traffic Types and Their Mapping to QoS/TSN Mechanisms," in *TSN Mechanisms*, vol. 3, 2019.

[40] C. Xue *et al.*, "Real-time scheduling for IEEE 802.1Qbv time-sensitive networking (TSN): A systematic review and experimental study," *arXiv preprint arXiv:2305.16772*, 2023.

Mohammadparsa Karimi (Member, IEEE) received the B.Sc. degree from Shahid Beheshti University in 2019 and the M.Sc. degree from Iran University of Science and Technology in 2021. He is currently a Ph.D. candidate with the Department of Electrical Engineering at Eindhoven University of Technology. His research interests include Time-Sensitive Networking, AI for network management and configuration, and real-time communication systems.

Majid Nabi (Member, IEEE) received the B.Sc. degree in computer engineering from Isfahan University of Technology, Isfahan, Iran, in 2001, the M.Sc. degree in computer engineering from Tehran University, Tehran, Iran, in 2007, and the Ph.D. degree in electrical and computer engineering from Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands, in 2013. He is currently an Assistant Professor at the Department of Electrical Engineering, TU/e. His research interests include efficient and reliable networked embedded systems, low-power wireless sensor networks, the Internet of Things, and efficient machine-learning techniques for network management.

Andrew Nelson (Member, IEEE) received his M.Sc. degree in Embedded Systems at Eindhoven University of Technology, Netherlands in 2009. After this, he moved to Delft University of Technology, Netherlands and received his Ph.D. there in 2014. He is currently employed as an assistant professor at Eindhoven University of Technology. His research interests include realtime(including mixed time-criticality) low-power multi-core embedded-systems and the timing analyses thereof.

Kees Goossens has a PhD from the University of Edinburgh in 1993 on hardware verification using embeddings of formal semantics of hardware description languages in proof systems. He worked for Philips/NXP from 1995 to 2010 on real-time networks on chip for consumer electronics. He was part-time full professor at Delft university from 2007 to 2010, and since then has been full professor at the Eindhoven University of Technology, researching composable, predictable, low-power embedded systems, supporting multiple models of computation since then. He published 4 books, 200+ papers, 17 patents, and the DRAM-power open-source software.

Twan Basten (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in computing science from Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands. He is currently a Professor with the Department of Electrical Engineering, TU/e. His current research interests include the design of embedded and cyber-physical systems, performance engineering, dependable computing, and computational models.