Deep-Reinforcement-Learning-based Scheduler for Time-Aware Shaper in In-Vehicle Networks

Mohammadparsa Karimi, Majid Nabi, Andrew Nelson, Kees Goossens, and Twan Basten
Electronic Systems (ES) group, Department of Electrical Engineering, Eindhoven University of Technology
The Netherlands
{m.karimi, m.nabi, a.t.nelson, k.g.w.goossens, and a.a.basten}@tue.nl

***Abstract*—As vehicles develop into software-defined platforms with powerful automated driving capabilities and driver support systems, their in-vehicle networks become significantly more complicated. A key technique for ensuring deterministic, low-latency connectivity for crucial data traffic in such settings is Time-Sensitive Networking (TSN), and specifically the Time-Aware Shaper (TAS). However, current TAS scheduling techniques have difficulty adjusting schedules to dynamically shifting traffic patterns and changing operating conditions. This paper presents an adaptive scheduler using Deep Reinforcement Learning (DRL), which aims to meet strict deadlines, reducing latency and providing near-ideal resource usage. Experimental results for different vehicle scenarios show that our DRL-based scheduler performs better in terms of success rate, low latency, and overall network performance than state-of-the-art heuristic algorithms such as earliest deadline first (EDF) scheduling.**

***Index Terms*—In-Vehicle Networking, TSN, TAS, Adaptive Scheduling, Deep Reinforcement Learning**

I. Introduction

The automotive industry is rapidly moving from traditional, hardware-centric vehicles to software-defined vehicles (SDV) [1]. These modern vehicles integrate advanced features, such as autonomous driving and Advanced Driver Assistance Systems (ADAS), to improve safety and driving comfort, and to move closer to full autonomy [2]. A key requirement for these emerging capabilities is a robust in-vehicle communication infrastructure. Next-generation vehicles must process and transmit large volumes of real-time data from sources such as high-resolution cameras, LIDAR, and radar, as well as from onboard computing units [3].

As vehicles become more self-reliant, the complexity of in-vehicle networks (IVNs) increases. Traditional networking technologies like CAN, FlexRay, and MOST are reaching their limits in terms of bandwidth, latency, and flexibility [4]. In contrast, Ethernet-based Time-Sensitive Networking (TSN) offers deterministic, low-latency, and highly reliable data exchange. These properties make TSN a foundational technology for upcoming automotive platforms, enabling reliable communication in challenging and varied conditions [5].

Among the TSN set of standards, the Time-Aware Shaper (TAS) is especially significant. TAS assigns precise time windows for data transmission, ensuring that critical traffic, such as obstacle detection or vehicle control commands, can be delivered on time without interference. This scheduling mechanism relies on Gate Control Lists (GCLs), which combine the timing of the opening and closing of network gates, preventing data bottlenecks, and ensuring that priority messages always meet their deadlines [6]. However, the challenge lies in determining effective and efficient TAS schedules, especially in the face of unpredictable and continuously changing traffic patterns found in real-world driving scenarios [7].

The complexity of modern autonomous vehicles only amplifies this difficulty. Driving conditions vary widely, from the stop-and-go nature of urban streets to the high-speed flow of highways. Each scenario imposes different requirements on network performance and data handling. Traditional TAS scheduling approaches, which often rely on static or offline-defined configurations, struggle to adapt when conditions shift mid-operation [8]. In addition, computing an optimal schedule involves exploring a large search space, making it difficult to achieve real-time online responsiveness while maintaining stringent timing guarantees [9].

To address some of these limitations, this paper proposes a Deep Reinforcement Learning (DRL)-based adaptive scheduler for TAS in IVNs. Using DRL, we can create a learning-based approach that refines its scheduling strategies as the conditions evolve. Our concrete **contributions** are:

- **A DRL-based Adaptive Scheduler:** We introduce a novel scheduler that applies DRL techniques to optimize TAS online during operation, accommodating changing in-vehicle conditions.
- **Performance Evaluation:** We present evaluations across various traffic scenarios, demonstrating that our DRL-based method outperforms other approaches in terms of adaptiveness, latency, and resource usage.
- **Web Application and Open-Source Release:** We implemented a web application integrating our proposed scheduler, allowing users to schedule desired flows and analyze results interactively. The source code is available as open-source via the TU/e ES GitHub repository (https://github.com/TUE-EE-ES).

The remainder of this paper is organized as follows. Section II reviews related work, focusing on existing TAS scheduling methods and their limitations. Section III defines the problem space and describes the challenges associated with adaptive TAS in IVNs. In Section IV, we detail our DRL-based scheduling algorithm, its architecture, and its implementation. Section V presents the experimental setup, results, and perfor-

mance analysis. Finally, Section VI concludes the paper and outlines potential directions for future research.

II. Related Work

Scheduling in TSN, especially for TAS, has received considerable attention. Many methods have been proposed to ensure reliable and low-latency communication in in-vehicle networks. These approaches fall into three main categories: heuristic, exact, and machine-learning-based. Each category offers unique benefits but also faces its own limitations [6].

A. Heuristic Approaches

Heuristic methods are commonly used because they are simple and can quickly generate workable schedules. As such, they are a practical starting point for large networks and can be re-run whenever conditions change, thanks to their low computational overhead. For example, Walrand [10] notes that heuristic-based traffic shaping can be executed at a low cost, providing relatively quick solutions.

Kim et al. [11] proposed a heuristic approach that first constructs a valid schedule and then refines it to minimize end-to-end latencies. EDF scheduling has also been adapted to TAS [12], [13]. For example, Dobrin et al. [13] introduced an EDF-based heuristic that adjusts deadlines to handle retransmissions before scheduling frames by their earliest deadlines, ensuring timely delivery under certain conditions.

As heuristics offer computational efficiency, they provide adaptiveness by recomputing schedules when conditions change. But, as the number of flows and the diversity of their requirements increase, heuristics may struggle to maintain near-optimal results. They rely on relatively simple decision criteria (e.g., sorting by deadlines or iteratively refining initial guesses), which can overlook complex interdependencies and lead to less effective utilization of available resources [6].

B. Exact Approaches

Exact methods use techniques like Integer Linear Programming (ILP) and Satisfiability Modulo Theories (SMT) to find optimal solutions with strict timing guarantees. For example, Stüber et al. [14] present an ILP solution to compute Efficient Robust Schedules (ERS) that provide deterministic performance under varying loads and (bounded) timing uncertainty. Other exact approaches are an ILP solution by Schweissguth et al. [15] and an SMT solution by Craciunas et al. [16]. These exact methods are slow and resource-intensive, making them unsuitable for online operation. They may also struggle to scale as in-vehicle networks become more complex and diverse. Because scheduling in TSN is an NP-hard problem [17], exact methods cannot easily adapt to complex scenarios.

C. Machine Learning-Based Approaches

Machine learning, and in particular DRL, has recently gained attention in TSN scheduling. Unlike traditional methods, DRL learns from experience and adapts as conditions change. Roberty et al. [18] show that DRL can produce better TAS schedules under dynamic conditions than older approaches. Islam and Muslim [19] use Graph Convolutional Networks (GCN) with DRL to handle more complex traffic demands in IVNs.

DRL-based techniques offer adaptability and can improve bandwidth use and reduce latency as traffic changes. They can overcome many of the limits of both heuristic and exact methods. Despite these advantages, existing DRL methods face challenges, including limited training on complex real-world scenarios, restricted parameter optimization, and a lack of complex scenario evaluations. Our approach addresses some of these gaps by extending optimization to include schedules, cycle lengths, and gate control sequences while evaluating performance across diverse traffic patterns and network configurations, especially for IVNs.

III. Problem Definition

This section presents the TAS scheduling problem, including how we represent the network components, the specification of time-triggered (TT) streams, and the formulation of scheduling constraints that must be met to ensure timely communication.

A. Network Model

We consider an IVN as a directed graph $G = (V, E)$, where each vertex $v \in V$ represents a network device (e.g., an ECU or a switch), and each directed edge $e \in E$ represents a communication link [20]. Each device v has one port for each of its outgoing links. Each port is associated with a GCL, which defines the scheduling of outgoing traffic for that port. The GCL is used to open and close transmission gates for queues associated with a port at precisely defined times, ensuring that time-sensitive streams are transmitted during their allocated time slots to achieve bounded latency. This is further explained below. Each link $e \in E$ is defined as

$$e = (\text{src}_e, \text{dest}_e, b_e), \tag{1}$$

where $\text{src}_e, \text{dest}_e \in V$ are the source and destination nodes of the link, and b_e is the transmission bandwidth of the link.

B. Time-Triggered (TT) Streams

We consider a set of TT streams at each point in time $S = \{s_1, s_2, \ldots, s_n\}$ [21]. Each TT stream s is specified as

$$s = (v_{\text{talker},s}, V_{\text{listener},s}, T_s, n_{f,s}, t_{\text{release},s}, f_s, d_s), \tag{2}$$

where $v_{\text{talker},s} \in V$ is the source (talker) node, $V_{\text{listener},s} \subseteq V$, is the set of one or more destination nodes for the stream (where $V_{\text{listener},s} \neq \emptyset$), T_s is the period of the stream, $n_{f,s}$ is the number of frames generated each period, $t_{\text{release},s}(i)$ is the release time of frame i in a period, relative to the start of the period, f_s is the frame size in bytes, and d_s is the stream's deadline, which is the maximum tolerated latency of the stream's frames. Multiple TT streams may share the network, all with their own timing constraints, resulting in a complex scheduling scenario. In the remainder, we assume that each stream is allocated to specific routes through the network to reach all its listeners, and to a shared queue inside each node in those routes.

C. GCL and Cycle Structure

As discussed, each device in the network has one or more ports, and each port is associated with a Gate Control List (GCL). This GCL controls opening and closing of transmission gates for a set of priority queues $\{Q_1, Q_2, \ldots, Q_p\}$ that feed frames into that port's outgoing link. When a queue's gate is open, frames from that queue may be transmitted onto the egress link; if multiple queues are open simultaneously, the highest priority traffic is sent first. When a gate is closed, the frames in that queue are not transmitted, even if they are ready.

A GCL defines a cyclic pattern of opening and closing times. To combine the timing of various TT streams with periods $T_{s_1}, T_{s_2}, \ldots, T_{s_n}$, the GCL cycle is set as follows:

$$T_{\text{cycle}} = \text{lcm}(T_{s_1}, T_{s_2}, \ldots, T_{s_n}). \tag{3}$$

Table I provides an example of a GCL configuration. The table defines a single cycle divided into m time segments $\Delta_1, \Delta_2, \ldots, \Delta_m$. Each Δ_i specifies the duration of a continuous interval of time in the cycle. The binary values x_{ij} specify, for each segment i and each queue Q_j, whether the gate of that queue is open (1) or closed (0).

TABLE I
EXAMPLE STRUCTURE OF A GCL

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

A GCL is properly defined if adding all the segment durations Δ_i yields the cycle time:

$$T_{\text{cycle}} = \Delta_1 + \Delta_2 + \cdots + \Delta_m. \tag{4}$$

As an example, consider a GCL cycle with $T_{\text{cycle}} = 5\,\text{ms}$ and two segments: $\Delta_1 = 2\,\text{ms}$ and $\Delta_2 = 3\,\text{ms}$. Suppose we have three queues Q_1, Q_2, Q_3. If during Δ_1, Q_1 and Q_2 are open ($x_{11} = x_{12} = 1$) and Q_3 is closed ($x_{13} = 0$), and during Δ_2 only Q_3 is open ($x_{21} = x_{22} = 0, x_{23} = 1$), then:

- for $t \in [0, 2)$ ms, Q_1 and Q_2 are open, and Q_3 is closed;
- for $t \in [2, 5)$ ms, Q_3 is open, while Q_1 and Q_2 are closed.

D. Scheduling Constraints

For each stream s, let $t_{\text{start},s}(k)$ be the time at which period k of the stream starts. Let $t_{\text{delivery},s}(i,k)$ be the time at which frame i of stream s generated in period k is successfully received by all its listeners. Recall that each stream s is associated with relative release times of its frames $t_{\text{release},s}$ and a relative deadline d_s measured from the release time of each frame (Eq. (2), [22]). Thus, every frame i of stream s must meet:

$$t_{\text{delivery},s}(i,k) \leq t_{\text{start},s}(k) + t_{\text{release},s}(i) + d_s. \tag{5}$$

E. Objective

The goal is to ***determine GCL parameters*** (time segments Δ_i, gate opening times x_{ij}, see Table I) for all GCLs in the network such that ***all frames meet their deadlines***, Eq. (5), ***while minimizing the total end-to-end latency of all frames in each period***. Considering all frames i of all streams s in all periods k of these streams in a hyperperiod of length T_{cycle}, the latter can be expressed as:

$$\min \sum_s \sum_k \sum_i \big(t_{\text{delivery},s}(i,k) - (t_{\text{start},s}(k) + t_{\text{release},s}(i))\big). \tag{6}$$

IV. DRL-BASED SCHEDULING APPROACH

This section presents our DRL-based adaptive scheduling approach for TAS in IVNs. Our method leverages an RL agent to configure GCLs. The agent tries to ensure that traffic meets its deadlines and refines its scheduling strategy as network conditions and traffic demands evolve.

A. Overall Framework

We model the scheduling problem as a Markov Decision Process, where the agent (the scheduler) observes the allocated set of streams to be scheduled, the reward for the latest tried schedule, and the network state resulting from that schedule. It then selects an action (scheduling decision, GCL parameters for all GCLs) that aims to minimize end-to-end latencies while respecting the timing constraints of the streams. The DRL agent uses a policy network —trained via the Proximal Policy Optimization (PPO) algorithm— to map system states to GCL parameter assignments. Over time, the agent refines this policy to improve adherence to deadlines and optimize performance metrics such as latency and bandwidth utilization.

Fig. 1 illustrates how the DRL-based scheduler interacts with the IVN in the training procedure. It shows the flow of information between the agent and the environment, showing how actions (GCL parameter assignments) affect network performance and how feedback (reward and next state) guides the agent's learning.

Once trained, the agent can be deployed online within a real or emulated IVN. Inference uses the learned policy to compute GCL parameters based on the current network scenario (set of streams, topology), without further tuning of the agent's neural-network weights. If the delay calculation for the resulting schedule (explained below) shows that one or more deadlines are missed, the agent reports a failure. In such a case, a higher-level controller needs to take action, e.g., reducing sampling rates of sensors, reconfiguring the flow allocation, or adapting application settings, to ensure that a feasible schedule is possible. This higher-level control is out of scope for this work.

B. State Representation

At each decision step during training, the DRL agent observes both the characteristics of all streams s and the network conditions. It combines these into a normalized vector

$$\mathbf{o}_s = [\bar{s}^*, \bar{\tau}_s^*, {b_{avg}}^*, {Q_{avg}}^*] \tag{7}$$

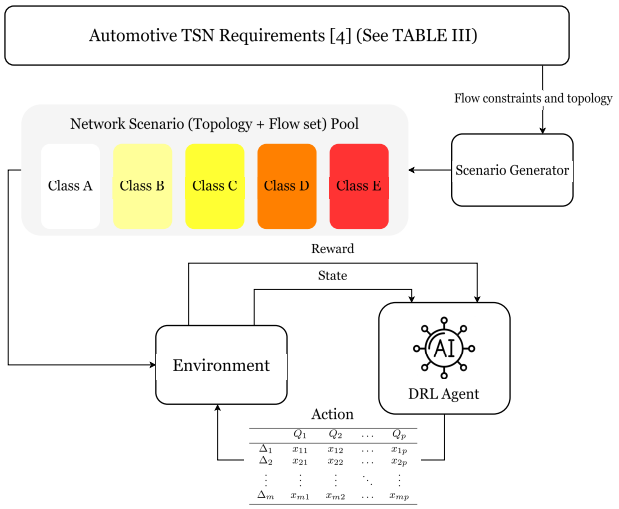

Fig. 1. A conceptual diagram showing the DRL agent training

where each parameter p is scaled to a dimensionless form p^*. Q_{avg} represents the average queue filling in the entire network and b_{avg} represents average available bandwidth on the links. The urgency factor τ_s for stream s measures how close that stream is to missing its deadline. Before normalization, τ_s is computed as

$$\tau_s = \frac{d_s - \frac{f_s}{b_e}}{d_s}. \tag{8}$$

When τ_s is near 1, the stream s still has ample slack. As τ_s approaches 0, that flow becomes increasingly urgent and must be prioritized.

C. Action Space

In our DRL-based scheduling approach, each action corresponds to configuring the GCLs for all device ports. Specifically, an action determines both the length of each time segment in the GCL cycle and which queues are open or closed during those segments. This way, the agent fully specifies the transmission schedule and gate states for every cycle. Formally, if a GCL has m time segments and p queues as in Table I, the action space must include the durations $\Delta_1, \Delta_2, \ldots, \Delta_m$ for the time slots and the $m \times p$ binary matrix defining which queues are open in each segment.

D. Delay Computation

We consider the delay per frame. Let the i-th frame of a TT stream s traverse a path of L nodes, $(v_0, v_1, \ldots, v_{L-1})$, where v_0 is the talker and v_{L-1} is a listener. For each hop $e_k \in E$ from v_k to v_{k+1} (for $k = 0, 1, \ldots, L-2$), the frame may incur a waiting time before it can be transmitted, followed by a transmission time on the outgoing link.

Formally, the total end-to-end delay $D_{s,i}$ for the i-th frame of stream s is:

$$D_{s,i} = \sum_{k=0}^{L-2} \big[W(v_k, q_s, t_{k,i}) + T_{\text{tx},k,s} \big], \tag{9}$$

where $W(v_k, q_s, t_{k,i})$ is the waiting time at node v_k for the queue q_s assigned to stream s, given that the frame arrives at time $t_{k,i}$ and $T_{\text{tx},k,s}$ is the transmission time for a frame of stream s over the link e_k given by $T_{\text{tx},k,s} = f_s / b_{e_k}$.

The waiting time $W(v_k, q_s, t_k)$ depends on the GCL configuration at node v_k and the status of the queues, including q_s, associated to the port to which also q_s is associated. The GCL specifies open intervals during which the queue q_s is allowed to transmit. If the frame arrives during an open interval in an empty queue, with all higher priority queues that are also open empty, and with sufficient time to transmit the frame at hand, the waiting time is 0. If it arrives in an open interval with sufficient time to transmit all earlier frames in its queue, all frames present and arriving in open higher priority queues, and the frame itself, then the waiting time equals the time to transmit all earlier frames in its queue and the higher priority frames. In other cases, the frame needs to wait until the next open interval, for which a similar calculation then needs to be performed.

E. Reward Function

Our objective is to find a scheduling policy that maximizes long-term performance. Although we compute a reward for each communicated frame, the agent's goal is to maximize the *total* reward over all frames. Let $\mathcal{F}$ be the set of frames considered during a scheduling horizon of one T_{cycle}. The total reward is defined as:

$$R_{\text{total}} = \sum_{i \in \mathcal{F}} R_i, \tag{10}$$

where per-frame reward R_i consists of three components that reflect minimizing delays (while meeting deadlines), maintaining a desired utilization, and avoiding late deliveries:

$$R_i = w_1 \cdot R_{\text{delay},i} + w_2 \cdot R_{\text{utilization},i} + w_3 \cdot R_{\text{penalty},i} \tag{11}$$

with w_1, w_2, and w_3 weights balancing the importance of each component.

Delay Component: This component rewards minimizing delays. For each frame $i \in \mathcal{F}$ belonging to stream s with absolute deadline d_s,

$$R_{\text{delay},i} = \begin{cases} \dfrac{d_s - D_{s,i}}{d_s}, & \text{if } D_{s,i} \le d_s \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

If the frame arrives on time (or early), $R_{\text{delay},i}$ is positive and can approach 1 as D_i becomes much smaller than d_s. If the frame is late, this term contributes no reward.

Utilization Component: We measure the overall utilization (OU) of the allocated time to ensure that the policy does not under-utilize or overload the network. Recall that E is the set of links. Let OccupiedTime_e be the total time a link $e \in E$ is transmitting data; let AllocatedTime_e be the total time a link $e \in E$ has at least one gate involved in transmitting data open. The overall utilization is:

$$\text{OU} = \left(\frac{\sum_{e \in E} \text{OccupiedTime}_e}{\sum_{e \in E} \text{AllocatedTime}_e} \right) \times 100. \tag{13}$$

From the given stream characteristics and network topology, we define acceptable utilization bounds $[U_{\min}, U_{\max}]$ and a

TargetUtilization (TU). If OU is within the acceptable range, the utilization component rewards how close OU is to TU:

$$R_{\text{utilization},i} = \begin{cases} 1 - \dfrac{|\,\text{OU} - \text{TU}\,|}{\text{TU}}, & \text{if } U_{\min} \leq \text{OU} \leq U_{\max} \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

This component is evaluated over a given time interval to prevent the scheduler from simply reducing latency by over-allocating gate opening times.

Penalty Component: If a frame i belonging to stream s is late, we apply a penalty proportional to its lateness:

$$R_{\text{penalty},i} = \begin{cases} -\frac{(D_{s,i} - d_s)}{d_s}, & \text{if } D_{s,i} > d_s \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

In summary, by defining the total reward as the sum of per-frame rewards, we provide immediate feedback to the agent for each scheduling decision while still focusing on improving long-term performance. Over time, the agent learns a scheduling policy that aims to meet deadlines, maintains efficient link utilization, and minimizes delivery time, in line with the objective set out in Sec. III-E.

F. Deep Reinforcement Learning Algorithm

We use the PPO [24] algorithm for training. PPO stabilizes the policy update process by restricting how much the policy can change at each iteration. The policy $\pi_\theta(action \mid space)$ is parameterized by θ (weights of a neural network). The agent iteratively interacts with the environment, collects trajectories, and updates θ to maximize the expected discounted return over a time horizon T_{cycle}:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T_{\text{Cycle}}} \gamma^t R_{\text{total},t}\right], \tag{16}$$

where $\gamma^t \in (0, 1)$ is the discount factor at time t. PPO uses clipped objective functions and advantage estimators to ensure stable and sample-efficient learning.

V. Experiments

In this section, we describe the experimental setup and evaluation results to validate our pre-trained DRL scheduler. The goals of our assessment are threefold: to demonstrate the adaptiveness of the scheduler to varying traffic conditions, to evaluate its latency improvements, and to show that it achieves lower idle times, which indicate more efficient use of resources. We first outline the automotive network architecture employed and describe the flow generation and scenario configurations. In all experiments, we assume that all devices are time-synchronized. Furthermore, we evaluated the meeting of the deadline using the delay calculation described in Sec. IV-D. To assess adaptiveness, we created several traffic classes, each containing diverse traffic flows, and simulated multiple network scenarios with varying levels of complexity and resource contention. Our DRL-based scheduler demonstrated the ability to rapidly adapt and generate efficient schedules for each scenario. Finally, we compare the performance of our approach to a known baseline to highlight its effectiveness in improving latency and resource efficiency.

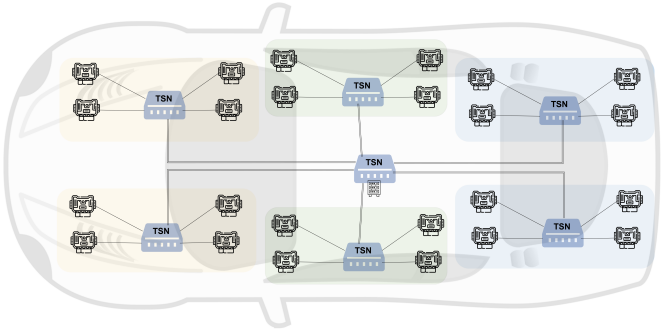

Fig. 2. A representation of the zonal architecture in this work

A. Network Architecture

We consider a fixed automotive network architecture based on a zonal design. In a zonal architecture, the vehicle's internal network is divided into different zones or regions, each containing a local switch or gateway [23]. Fig. 2 illustrates a conceptual layout of the zonal architecture for an in-vehicle network. Each zone aggregates traffic from local sensors and ECUs, forwarding time-sensitive streams to other zones and a central server via a high-bandwidth backbone.

B. Flow Generation and Scenarios

In our experiments, all links in the considered architecture were configured with a fixed bandwidth of 100 Mbps. For a comprehensive evaluation of our DRL-based scheduler, we defined five distinct classes of traffic scenarios. Each class progressively increases in complexity and difficulty, transitioning from conditions where flows encounter minimal contention and relaxed timing constraints to those where multiple flows compete aggressively for bandwidth and must meet stringent deadline requirements. Table II summarizes these classes, including their approximate bandwidth usage ranges, the ratio of flow deadlines to periods (deadline range), and the number of scenarios used for training and testing. In total, we generated 250 scenarios (50 per class) for training and an additional 50 scenarios (10 per class) for testing.

To maintain realism and consistency with automotive Time-Sensitive Networking (TSN) requirements, the generation process was guided by typical automotive constraints such as latency limits and frame sizes. Table III provides an overview of these typical TSN requirements [4].

C. Results and Comparison with EDF

To evaluate the effectiveness of our approach, we compared our DRL-based scheduling strategy against an EDF scheduler which is commonly referenced in TSN scheduling research and has been successfully adapted for TAS in prior work [13]. EDF is widely recognized as a strong heuristic approach due to its simplicity and ability to minimize delays for individual

TABLE II
PROPERTIES OF THE DEFINED SCENARIO CLASSES

Class	BW Range (%)	Deadline Range	Train/Test
A	30–35%	0.8–1.0	50/10
B	40–50%	0.7–0.9	50/10
C	65–75%	0.5–0.7	50/10
D	85–90%	0.3–0.5	50/10
E	90–100%	0.2–0.4	50/10

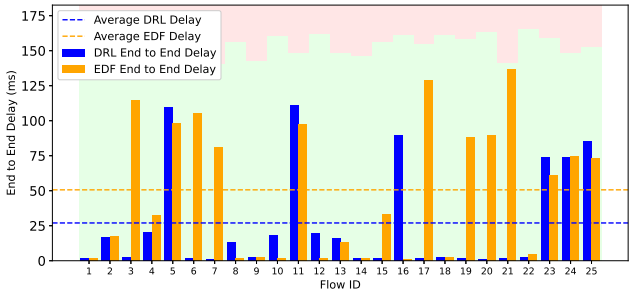

Fig. 3. Comparison of per-frame delay vs. deadline in a simple scenario

streams, making it a suitable benchmark for assessing new scheduling techniques. Since the EDF algorithm was not initially tested on our benchmark, we implemented and applied it to our scenarios.

Fig. 3 presents the per-frame delay distributions for both DRL and EDF in a simple scenario from class A, demonstrating how the DRL-based scheduler achieves lower average latency. The green area indicates the range where flow delays remain within their deadlines, signifying successful scheduling. In contrast, the red area represents the region where flow delays exceed their respective deadlines, indicating a failure to meet the timing constraints.

Fig. 4 compares the overall average delays across different classes under both scheduling policies with error bars representing the minimum and maximum observed delays. The results show that, as we progress from simpler scenarios to more complex ones and deadlines get tighter, the average latency decreases. However, DRL consistently achieves lower average latency compared to EDF.

We further evaluated the average overall success rates for each scenario class. A scenario is successful if all deadlines are met. Fig. 5 illustrates the average success rates in different scenario classes, and the error bars indicate min and max. For Class A scenarios, both schedulers successfully schedule all flows, achieving a success rate of 100%. However, starting from Class B scenarios, the EDF begins to fail in some cases, while the DRL-based scheduler maintains stable performance. For Classes C-E, the EDF struggles significantly, while the DRL-based scheduler, although not perfect, demonstrates notably higher success rates. These results highlight that EDF fails earlier and more consistently as scenario complexity increases, whereas DRL provides better robustness and adaptiveness across a wider range of scenarios.

In addition to evaluating latency, we also analyzed the idle time of links in various network scenarios. The DRL-

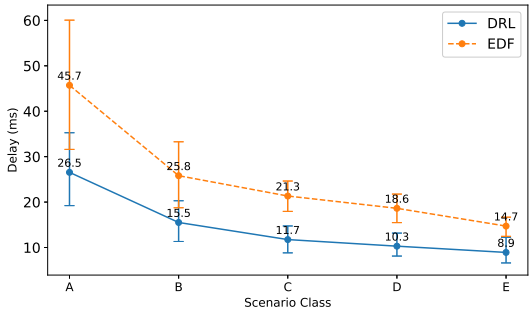

Fig. 4. Delay (average, min and max) comparison between DRL and EDF approaches

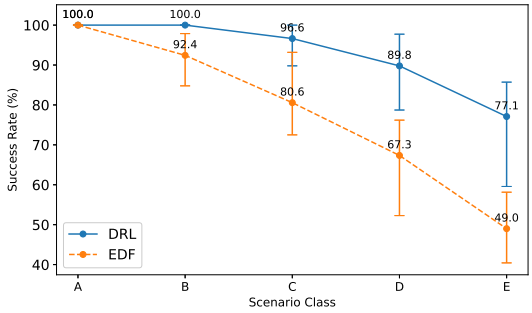

Fig. 5. Success rates (average, min and max) for meeting flow deadlines across different scenario classes

based scheduler demonstrated reduced idle times compared to the EDF scheduler because of the effective use of available resources. Fig. 6 illustrates the average idle time percentage across different scenario classes. We compute this percentage as IdleTime $= 100\% - \mathrm{OU}$, with OU as in Eq. (13), computed only for backbone links between zonal switches and the central server (Fig. 2).

Finally, we consider responsiveness to changes. Because the DRL scheduler is pre-trained, it can respond to new network configurations or adapt to changing traffic patterns by recomputing a schedule within a few seconds, often within one second. The exact response time depends on the computational hardware used. With modern setups, the inference time for generating new schedules is negligible, which makes the DRL approach adaptive for online deployment in IVNs.

VI. CONCLUSION

This paper presented a DRL-based adaptive scheduling method for the Time-Aware Shaper in IVNs, using RL based on proximity policy optimization. The proposed approach effectively addresses the limitations of heuristic scheduling methods, outperforming EDF scheduling, particularly in moderately complex and complex scenarios.

As other approaches, the DRL-based approach occasionally fails to find a feasible schedule as the complexity of the traffic scenario increases and bandwidth saturation approaches. This limitation is a potential challenge for practical implementation in highly complex real-world scenarios, where guaranteeing consistent performance under varying conditions is critical. To overcome this limitation, future research will focus on improving performance by combining offline and online training,

TABLE III
REPRESENTATIVE AUTOMOTIVE TSN REQUIREMENTS [4]

PCP	Traffic type	Attributes	Link Utilization	Loss Tolerance
0	Best Effort (Data Tx., Diag., Others)	Size: 64-1518 bytes; Timing constraint: 2000ms	$\geq 25\%$	Some
1	Video Stream 2	Size: 1518 bytes; Timing constraint: 50ms	1-20%	Some
2	Reserved for future use	N/A	-	N/A
3	Network Control/Management	Size: 64-500 bytes; Timing constraint: 100ms	1-5%	Few
4	Command & Control 2	Size: 64-1518 bytes; Timing constraint: 100ms	1-40%	Few
5	Video Stream 1	Size: 1518 bytes; Timing constraint: 2ms	1-5%	Few
6	Reserved for future use	N/A	-	N/A
7	Command & Control 1	Size: 64-512 bytes; Timing constraint: 1ms	1-5%	None

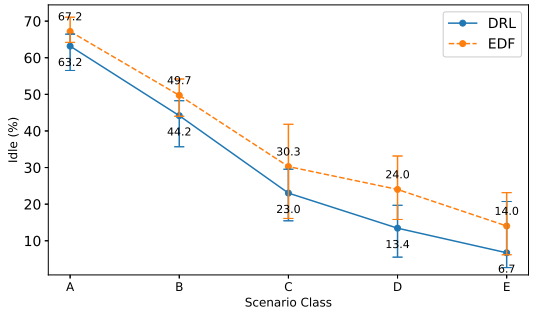

Fig. 6. Links idle time (average, min and max) percentage across different scenario classes

building a more extensive and varied flow dataset, and further refining the model to handle, e.g., network reconfigurations and extreme workload situations. As it can never be fully guaranteed to always find feasible schedules in heavy-load conditions, it is also interesting to integrate the DRL-based scheduler in an online application and resource orchestration framework that can adapt application settings or resource allocations in case of scheduling failures.

ACKNOWLEDGMENT

This work has 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] C. Srinivasan et al., "Advanced Driver Assistance System (ADAS) in Autonomous Vehicles: A Complete Analysis," in *Proc. 8th Int. Conf. Commun. Electron. Syst. (ICCES)*, 2023, pp. 1501–1505.

[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] Y. Peng et al., "A Survey on In-Vehicle Time-Sensitive Networking," *IEEE Internet Things J.*, vol. 10, no. 16, pp. 14375–14396, 2023.

[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 *Proc. IEEE 95th Veh. Technol. Conf. (VTC2022-Spring)*, 2022, pp. 1–7.

[6] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN)," *IEEE Access*, vol. 11, pp. 61192–61233, 2023.

[7] G. Papathanail et al., "Dynamic Schedule Computation for Time-Aware Shaper in Converged IoT-Cloud Environments," in *Proc. 27th Conf. Innov. Clouds, Internet Netw. (ICIN)*, 2024, pp. 1–8.

[8] A. A. Syed, S. Ayaz, T. Leinmüller, and M. Chandra, "Dynamic Scheduling and Routing for TSN based In-vehicle Networks," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, 2021, pp. 1–6.

[9] W. Kong, M. Nabi, and K. Goossens, "NPTSN: RL-Based Network Planning with Guaranteed Reliability for In-Vehicle TSSDN," in *Proc. 53rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, 2023, pp. 55–66.

[10] J. Walrand, "A Concise Tutorial on Traffic Shaping and Scheduling in Time-Sensitive Networks," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 3, pp. 1941–1953, 2023.

[11] H.-J. Kim et al., "Development of an Ethernet-Based Heuristic Time-Sensitive Networking Scheduling Algorithm for Real-Time In-Vehicle Data Transmission," *Electronics*, vol. 10, no. 2, p. 157, 2021.

[12] L. Leonardi, L. Lo Bello, and G. Patti, "Combining Earliest Deadline First Scheduling with Scheduled Traffic Support in Automotive TSN-Based Networks," *Appl. Syst. Innov.*, vol. 5, no. 6, p. 125, 2022.

[13] R. Dobrin, N. Desai, and S. Punnekkat, "On Fault-Tolerant Scheduling of Time Sensitive Networks," in *Proc. 4th Int. Workshop Security Dependability Crit. Embedded Real-Time Syst. (CERTS)*, 2019.

[14] T. Stüber, L. Osswald, and M. Menth, "Efficient Robust Schedules (ERS) for Time-Aware Shaping in Time-Sensitive Networking," *IEEE Open J. Commun. Soc.*, vol. 5, pp. 6655–6673, 2024.

[15] E. Schweissguth et al., "ILP-Based Routing and Scheduling of Multicast Realtime Traffic in Time-Sensitive Networks," in *Proc. IEEE 26th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, 2020, pp. 1–11.

[16] S. S. Craciunas et al., "Scheduling Real-Time Communication in IEEE 802.1Qbv Time-Sensitive Networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2016, pp. 183–192.

[17] 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.

[18] A. Roberty et al., "Configuring the IEEE 802.1Q Time-Aware Shaper with Deep Reinforcement Learning," in *Proc. IEEE Netw. Oper. Manag. Symp. (NOMS)*, 2024, pp. 1–7.

[19] S. T. Islam and A. B. Muslim, "AI-based Dynamic Schedule Calculation in Time Sensitive Networks using GCN-TD3," *arXiv preprint arXiv:2405.05019*, 2024.

[20] 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.

[21] 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.

[22] T. Stüber, M. Eppler, L. Osswald, and M. Menth, "Performance Comparison of Offline Scheduling Algorithms for the Time-Aware Shaper (TAS)," *IEEE Trans. on Ind. Inf.*, vol. 20, no. 7, pp. 9736–9748, 2024.

[23] A. Frigerio, B. Vermeulen, and K. G. W. Goossens, "Automotive Architecture Topologies: Analysis for Safety-Critical Autonomous Vehicle Applications," *IEEE Access*, vol. 9, pp. 62837–62846, 2021.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347*, 2017.