Hierarchical Scheduling of Cooperative TSN for Mixed Critical Wireless Systems

Jannusch Bigge* and Christoph Sommer*
*TU Dresden, Faculty of Computer Science, Germany
https://www.cms-labs.org/people/{ bigge , sommer }

***Abstract*—Inside highly mobile vehicles, In-Vehicle Networks (IVNs) are designed to meet the requirements of reliable and deterministic communication. When two vehicles communicate, however, decentralized best-effort wireless connections threaten to re-introduce the problems that IVNs were designed to solve. Moving to centralized coordination cannot meet the requirements of Vehicle to Everything (V2X) systems for a wealth of reasons from capacity bounds to privacy and safety. In this paper, we thus propose hierarchical cooperative Time Sensitive Networking (TSN) as an approach for cooperating (rather than centrally managed) TSN networks via best-effort wireless links. Our approach is both suited to mixed IVN and V2X systems of highly mobile nodes like platooning and generalizable to similar systems requiring streams across cooperating TSN networks. We also present an approach for the simulative performance evaluation of such systems and describe an Open Source reference implementation using well-established simulation tools. We conclude with the results of a proof of concept evaluation to demonstrate the feasibility and dynamic adaptivity of our approach.**

I. Introduction

Inside road vehicles, In-Vehicle Networks (IVNs) have always had to meet the requirements of reliable and deterministic communication. More recently, though, the gradual introduction of modern Advanced Driver-Assistance Systems (ADAS) has led to ever higher general loads in the network and ever stricter requirements on the bus systems. To meet these requirements, Time Sensitive Networking (TSN) over Ethernet (already a well established technique in the industry for factory automation) is being adapted for IVNs in the context of IEEE 802.1DG [1]. Even though there are still many open questions, such as those surrounding dynamic scheduling, such solutions hold great promise for the future of IVN.

Between two vehicles, at the same time, wireless connections that forgo infrastructure rely on best-effort wireless connections, which threatens to re-introduce the problems surrounding reliability and determinism that TSN just tried to solve inside the vehicle. To meet reliability challenges in wireless networks, 5G-TSN [2] recently started to work towards real-time wireless communication channels based on TSN. One key aspect of such 5G systems, however, is the requirement of a centralized base station which manages the connections. In the context of highly mobile ad hoc systems like Vehicle to Everything (V2X), such centralized coordination introduces problems like capacity bounds, overhead, and a dependency on infrastructure.

Even assuming that these are solved problems, centralized coordination would also, by definition, need to govern the whole network. This would require universal cooperation between all manufacturers and the acceptance of an external application for scheduling the internal IVN. Moreover, aside from such technical challenges, a centralized approach would also raise concerns regarding security, privacy, and safety. First promising approaches to circumvent these limitations were already presented in the context of decentralized management of networks, but are mainly specific to single (mainly industrial) use cases and cannot be easily transferred to the context of highly mobile V2X systems.

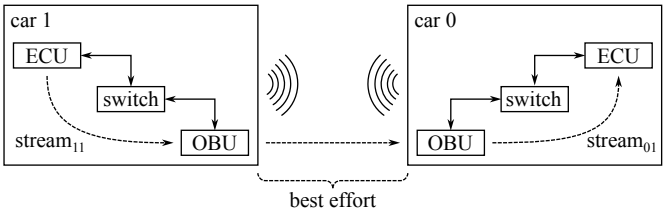


Figure 1. Logical stream across two cars connecting corresponding Electronic Control Units (ECUs), which is divided into two separate streams bounded by the limits of each individual In-Vehicle Network (IVN) and crossing a best-effort wireless channel in between the cars.

To fill this gap, we propose *hierarchical cooperative TSN* as an approach for cooperating (rather than centrally managed) TSN networks via best-effort wireless links, creating the system architecture illustrated in Figure 1. Our approach is both suited to mixed IVN and V2X systems of highly mobile nodes like platooning [3] and generalizable to similar systems requiring streams[1] across cooperating TSN networks. We also present an approach for the simulative performance evaluation of such systems, describe an Open Source reference implementation using well-established simulation tools, and present the results of a proof of concept evaluation to demonstrate the feasibility and dynamic adaptivity of our approach.

II. Related Work

To enable deterministic real-time communication over Ethernet, the TSN set of standards was developed by the IEEE 802.1 TSN Task Group [4]. Different standards from this set can be used and combined to realize different use cases. The standards are grouped into four categories: time

[1]As the approach we describe spans the full protocol stack, for clarity, we forgo using different terminology for each protocol/service data unit of each protocol layer; instead, we consistently use the term *stream* for TSN-governed data in transit, *packet* for a unit of data, and the terms *talker* and *listener* for the sender and receiver of such data, respectively.

20th Wireless On-demand Network systems and Services Conference (WONS 2025)

synchronization, high availability / ultra reliability, bounded low latency, and dedicated resources plus API. By enabling time synchronization, one can combine features from bounded low latency for the scheduling together with dedicated resource allocation for these streams to achieve zero congestion loss.

A rich set of related work exists for TSN in the context of industrial applications like factory automation, its original focus. Here, network configuration and management is usually done by a Central Network Configuration (CNC) in combination with a Centralized User Configuration (CUC) [5]. Luo et al. [6] define three different levels of management: fully centralized, fully decentralized, and a centralized network with distributed user configuration. With the rise of Industrial Internet of Things (IIOT) and smart factories, the requirements for the network started to change. To address these new problems of complex, highly dynamic typologies in combination with spatio-temporal burst traffic they propose a hierarchical network topology in which multiple Computing Network Collaboration Domains (CNCDs) collaborate. We build on this approach and expand the idea of multiple TSN domains to the context of vehicular networking, but now with completely independent TSN networks. The approach of Luo et al. [6], in turn, is based on offloading the computational load with fog/edge computing nodes, originally designed for smart villages [7].

In vehicular networking, similar problems exist. Karle et al. [8] developed a complex simulation of IVNs to investigate networking in the context of autonomous driving. Andronovici et al. [9] developed a TSN testbed and compared the measurements with an OMNeT++ and INET based simulation for typical streams and configurations of an IVN. As the load by Electronic Control Units (ECUs) is constantly increasing and modern ADAS are introducing new requirements for the network, as Zou et al. [10] point out, the authors particularly stress the high amount of spatio-temporal data and the loss in usefulness if delivered incompletely, which is often a problem with time triggered streams. To solve this, they proposed a new scheduling approach with a focus on in-car networking. New standards like IEEE 802.1DG [1] (currently at draft version 4) focus directly on TSN for vehicular communication to address some of these issues. Different approaches are pursued in parallel, such as one based on the Time Aware Shaper (TAS) and Asynchronous Traffic Shaping (ATS), proposed by Zhou et al. [11]. For their investigations they simulated the IVN with OMNeT++ and CORE4INET.

The necessity of combining the IVN with V2X was shown by Buse et al. [12], who described a way to couple hardware-in-the-loop simulation with Vehicular Ad-Hoc Network (VANET) simulation. With the increasing requirements brought about by advanced sensors and actuators and the connections between them, the demand for deterministic connections is not only increasing for wired connections but for wireless connections as well, which is also addressed by the IEEE and the 3GPP [13]. These hybrid TSN networks try to include the wireless connection and manage it with different approaches depending on the technology stack. One fundamental problem common to all hybrid approaches according to Seijo et al. [14] is time synchronization and resource allocation. Atiq et al. [13] compared multiple integration approaches and especially compared the differences between WLAN and 5G regarding wireless TSN capabilities. Different approaches to how TSN could be integrated in WLAN were investigated by Seijo et al. [14], both in theory and in a testbed. In contrast, Durisi et al. [15] investigated the performance of TSN capable 5G systems. These approaches have in common that they focus on the design of existing TSN network architectures [5], which treat the whole network as one big network, managed and configured in a centralized approach with one CUC and CNC, which incurs all the drawbacks mentioned above.

The simulation of wireless TSN was shown by Kinabo et al. [16]. A simulation for TSN capable 5G systems was proposed by Debnath et al. [2]. Common to all simulation approaches is that they are very specific to a particular use case. A first simulation approach in which the simulation of the TSN based IVN was combined with the simulation of the wireless connection was shown by Turcanu and Sommer [17], a simulation approach on which we build in this paper.

III. COOPERATIVE TIME SENSITIVE NETWORKING (TSN)

In an automotive context, we face similar problems as in smart factories with IIOT: With features like modern ADAS, not only the network load in general increases but also the requirements for this kind of traffic are increasing. A new dimension to the problem is that of constant changes in the network topology as vehicles not just join or leave the network but permanently change distances to each other. Another critical problem is the discoverability and the management of the network. Managing a whole network as well as dividing it into multiple CNCDs requires knowledge about the network in general and all its devices. Moreover, it is highly unlikely that two vehicles from different vendors would allow the adaption of the IVN schedule by the other one if the internal network structure would be propagated to other vehicles or infrastructure.

In this paper, we address these problems by proposing a hierarchical cooperative TSN approach. Instead of looking at the entire network, which is very difficult given the high dynamics, we treat the individual subnetworks as independent networks. This is in stark contrast to existing approaches, in which the network is managed as a whole, similar to the approach of CNCDs. Still, despite our goal of keeping the networks independent, we want to achieve better performance than simple best-effort connections in-between the independent TSN networks. To achieve this, we want each subnetwork to be aware of wireless connections from others and to react to incoming messages.

Figure 2 illustrates this approach for two separate TSN networks (TSN D1 and TSN D2) connected with a best-effort wireless connection which is not included in any of the TSN networks. Every subnetwork should be as big as possible to increase the amount of deterministic connections in the whole network. At the same time, it should be only so big that it is manageable by a single configurator. This is in contrast to the

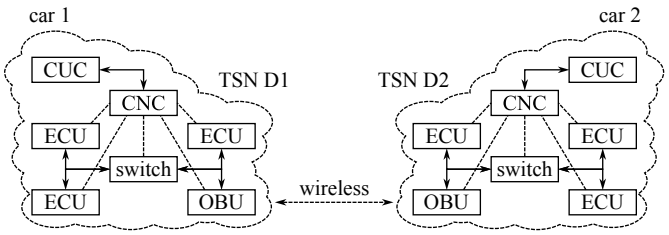


Figure 2. Logical representation of hierarchical cooperative TSN domains which are connected via an unmanaged wireless channel. Two completely independent TSN domains are created, each with its own CNC and CUC.

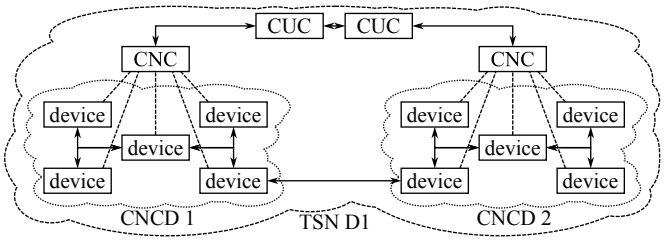


Figure 3. Traditional approach to decentralized TSN configuration: unified configuration across two collaborating domains, which requires a global CNC and CUC as proposed by, e.g., Luo et al. [6].

traditional approach illustrated in Figure 3, where the whole network is managed as a single domain.

Our concept realizes the following goals:

a) Each car is only responsible for its own IVN schedule. No other car can dictate its schedule, neither can it dictate that of other networks.
b) Well-scheduled streams across multiple IVNs (connected via V2X communication) are the result of cooperation between all participants of the stream.
c) Whenever a new packet arrives, the receiver should remember this packet for a certain time t_{remember}. If a second packet arrives that can be sorted into the same stream, the IVN schedule should be adapted to fit the new stream as well as possible without breaking any constrains of the IVN.
d) At the same time, every participant of the V2X network should, whenever it notices a new periodic message, attempt to soft reserve a corresponding slot in its own schedule. This means that, in the IVN schedule, this time slot should not be used to send data.

Together, this approach leads to a network where, after an initial transient period, every participant is aware of other senders and can reduce the latency for their streams.

To improve the placement of the streams in the schedule even further, each stream should be adapted during its existence. This is because of inherent variations in the arrival time of incoming messages even if they belong to the same stream. This can have multiple reasons like a non-deterministic sender-IVN, a high utilization of the V2X link which delays channel access (like CSMA/CA channel access) or requires retries, or simply variations of the distance between sender and receiver. We address this by ensuring that each slot of a stream is big enough to fit any one of a number of past messages, plus a safety buffer commensurate with the expected jitter. Equation (3) gives the concrete implementation chosen for the proof of concept evaluation in this paper.

Respecting these rules, a communication path from one ECU in a car c_1 to another ECU in a different car, c_0, could look as shown in Figure 1. In a traditional TSN approach, one stream would be established from the talker (ECU in c_1) to the listener (ECU in c_0) including the wireless path. With hierarchical cooperative TSN, this stream is divided into two streams. The first stream $stream_{11}$ only includes the part from the ECU in c_1 up to the On Board Unit (OBU) in c_1. A second stream $stream_{01}$ is configured only in c_0 and goes in the opposite direction, starting from its OBU and sending in its ECU. Note that, following our approach, $stream_{01}$ will be created with the second message that arrives in the specified time interval t_{remember}, at the OBU of c_0. The initial cycle time t_{cycle} can be calculated based on the arrival time of the first message t_{a1} and the arrival time of the second message t_{a2}.

As there is a high probability that t_{cycle} is not perfect due to the discussed sources of jitter, one should adapt the parameters when more messages arrived. One way would be the recalculation of t_{cycle} which, depending on the algorithm, can lead to a very precise value – though only in the complete absence of any noise. To avoid this problem and at the same time increase the slot-hit-rate, we propose an approach in which the slot length of the $stream_{01}$ will be adapted instead. Consistent with our goals, the wireless transmission between c_0 and c_1 is not directly managed by any TSN network, but only indirectly influenced by $stream_{11}$.

IV. Simulating cooperative Time Sensitive Networking (TSN)

At a high level, simulating cooperative TSNs encompasses two main challenges, discussed in the following.

A. Internal and External Network

A crucial aspect is the network topology extraction. Traditional network configuration is performed on the complete network, resulting in one big network graph including all network nodes being extracted. For the approach of hierarchical cooperative TSN this could be a problem as we want to separate every IVN into its subnetwork. Having every IVN represented as its individual network topology assures that no vehicle is ever attempting to configure the network of another vehicle. Furthermore, we need a representation of the connections between the individual vehicles without the IVN, as we do not want to configure the subnetworks. Depending on the configuration and the module which tries to calculate and apply a configuration, the correct network topology needs to be provided.

In more detail, simulating multiple IVNs and the connection between the vehicles requires special care regarding the visibility of the individual network devices. By making sure

that the visibility is limited to the network the device belongs to, we assure, at the same time, that we cannot configure other networks. This is especially important for the scheduler, which should not be able to access an IVN which is not managed by it. Still, it is important that the OBUs and the actual sender in the IVN can address the receiver OBU directly instead of simply broadcasting everything. To make this possible, the forwarding tables in the IVNs need to be updated if a new vehicle is created or deleted. The address of the external interface of every OBU is added to the forwarding table, which allows every network device to address the destination vehicle directly without creating unnecessary multicast or broadcast traffic in the IVN. Moreover, the receiving OBU needs a mapping for every incoming traffic to the corresponding internal network device, as well as the priority this traffic should have. This allows every subnetwork to independently choose the priority this traffic should have in the IVN and also eliminates the need to publish the internal endpoint to other senders.

B. Dynamic Time Sensitive Networking (TSN) scheduling

A key design choice for scheduling was to adapt an existing, well-established TSN scheduler to our needs to build on validated code. As we want to leave the responsibility for each IVN schedule within in the vehicle, every vehicle has its own scheduler. This approach was chosen to make sure that the scheduler never includes connections (like the wireless ones or from other cars) it is not allowed to have. Still, the scheduler needs to distinguish between all different senders. Every external sender that should be included in the IVN is represented as an application running on the Network Interface Card (NIC) which received the message first. This application contains all information necessary for the scheduler to include the external partner into the schedule of the IVN. As a consequence, the scheduler does not know that this virtual application is only a representation of a real external communication partner and treats it as every other application in the IVN. This also allows us to modify the parameters in a way that we can easily adapt to changes from outside. Typically, a wireless transmission in the context of vehicles is influenced by the distance of the sender and receiver. If this distance varies in a given range, and respectively also the transmission time, we can adapt our slot length parameters for the scheduler to increase the slot hit rate for this application.

V. Simulation study

To show the effects of hierarchical scheduling of cooperative TSN, we simulate a platoon of five vehicles.

For this, we use OMNeT++ as a discrete event simulator in combination with the model libraries and tools listed in Table I: SUMO [18] runs microscopic simulation models of road traffic. Veins serves to mirror its simulated vehicles as agents of an OMNeT++ based network simulation and integrates wireless communication models suitable for vehicular networks. Plexe [3] extends Veins and SUMO with models of common Cooperative Adaptive Cruise Control (CACC) approaches, which enables simulations of vehicle platooning.

Table I
USED SOFTWARE VERSIONS.

software	version	modified
OMNeT++	6.0.3	no
INET	4.5.2	yes[2]
Veins	5.3	no
Plexe	3.1	no
SUMO	1.20	no
TSNSCHED	1.1	yes[2]

INET simulates both the Ethernet based IVN and the V2X communication. Some additional modules were created by us for the connection of the libraries and additional functionality, which was necessary as a result of the simultaneous simulation of IVN and V2X. The full source code is available online.[2]

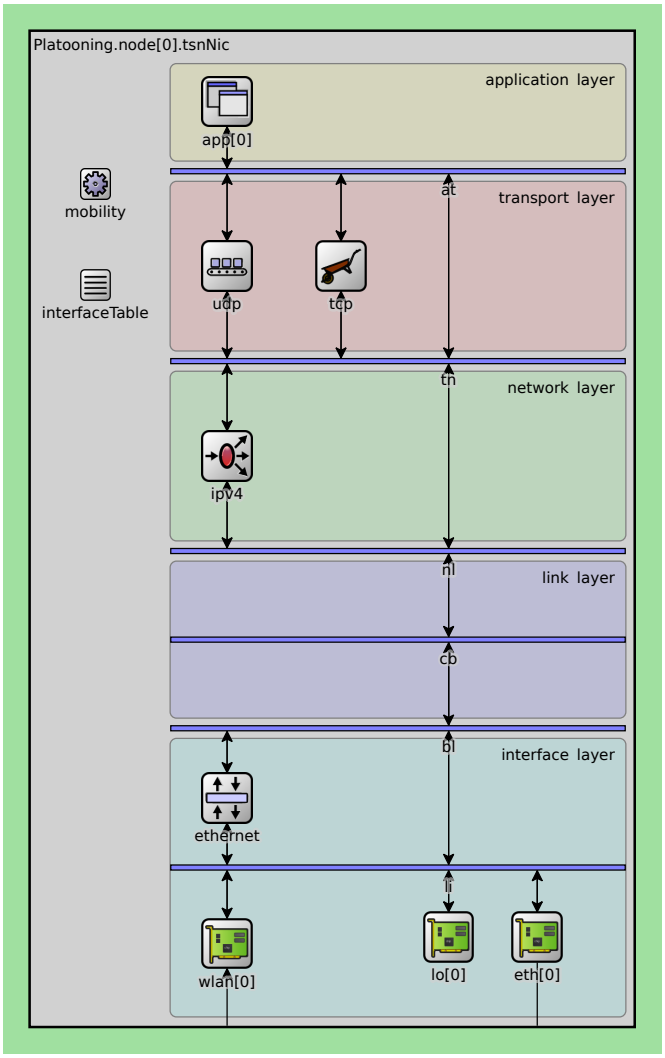
Connecting multiple simulation libraries brings unique challenges. Taking the position of a module into account for model calculations such as path loss or propagation time is a common problem and has been solved already in multiple frameworks. Instead of having multiple independent *mobility* modules (i.e., those tracking module position), they were replaced by *attached mobility* modules, each from its original framework but listening to mobility changes from another. Similar to the representation of the mobility also the representation of a packet is solved by multiple frameworks. With the introduction of driver modules, this can be solved. They extract key information, create a new packet for the new framework, and finally encapsulate the original packet in the newly created. Incorrect packet lengths would be caused by treating the old packet as payload or data, and so it needs to be avoided. In the connection layer at the receiving module, the driver performs the same operation in the other direction.

Figures 4a and 4b illustrate the resulting topology of agents in the combined simulation: Figure 4a shows a single protocol stack of a host in INET with submodules closely mirroring OSI layers one through seven. Figure 4b demonstrates how multiple networked devices, each containing such submodules, form the network topology of a single vehicle module in our combined simulation. Of note is that the internal components of the Plexe stack are not encapsulated in a network module and instead placed at the root level of the vehicle module (red dashed line). This is for compatibility reasons, as some modules reference other modules by relative paths and otherwise Plexe would need to be modified. More generally, extensions to existing module libraries are restricted to shims that are designed to not impact model validity. The only modifications of existing model code were necessary to allow INET to nest network devices and to visualize the state of internal components.

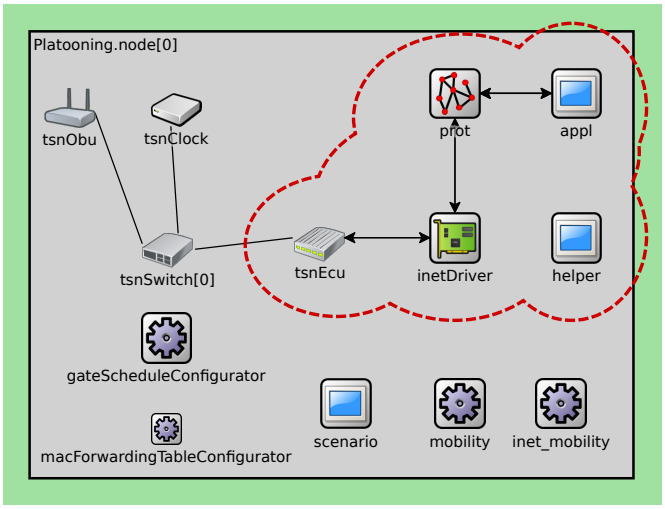
Thus assembled, each vehicle in the combined simulation contains four networking devices (connected by 1 m long full-duplex Ethernet cables operating at 100 Mbit/s):

- an ECU which consumes and produces the data,
- a TSN clock,

[2]https://www.cms-labs.org/research/software/cooperativetsn/



(a) mobile agent in plain INET



(b) mobile agent in our combined simulation

Figure 4. Representations of mobile agents in plain INET (one agent is one host with one protocol stack) and in our combined simulation (one agent is one vehicle, which contains multiple networked devices, each of which then contains one protocol stack). Red cloud: subcomponents making up one logical device of the Plexe stack.

- an OBU as a gateway between the internal Ethernet-based TSN network and the external network which connects the individual cars via V2X communication technologies, and
- a TSN switch which connects the other three.

Messages are generated by an unmodified Plexe platooning application to realize a standard platooning use case. This also serves to highlight the fact that the proposed approach is not limited to a specific application or simulation module library. The only difference to a standard Plexe simulation is that the messages are not sent out via a wireless interface of the platooning ECU but instead via one of its wired interfaces, to be sent via the IVN to the OBU of the car.

Platooning messages are thus 246 Byte long and generated at each car at fixed intervals of 100 ms, starting with a random offset (uniformly distributed between 0.01 ms and 100 ms) after the initialization of the vehicle. Note that the fixed message generation interval of 100 ms is merely chosen for ease of simplicity; the employed scheduler TSNSCHED can just as well calculate and create a hypercycle.

As we do not create any other noise traffic, the path from the ECU to the OBU is completely free and the message can pass through without any additional queue time added by TSN. As we use store-and-forward switches, though, we do add some delay to the transmission, namely the time that is necessary for the packet to reach the switch completely. Taking this additional delay into account is important because the scheduler needs to know this delay for the placement of the slots. If we would only consider the arrival time of the octets needed to evaluate the switching, like it is done in cut-through switching, the slot would be already open even though the switch is not yet in a state where it can start the transmission; thus, the slot would end too early. The end-to-end delay in the sender Δ_{stream0} is always the sum of the propagation time t_{prop} and the transmission time t_{trans} of the packet.

$$\Delta_{\text{stream0}} = t_{\text{prop}} + t_{\text{trans}} = 2 \times (5\,\text{ns} + 19.68\,\mu\text{s}) = 39.37\,\mu\text{s} \quad (1)$$

At the receiver, if we assume that this message arrives at time t_{a} at the car and (depending on the start time of the slot t_{slot} and the arrival time at the switch t_{arr} of the whole packet) has to wait for a time $\Delta_{\text{slot}} = t_{\text{slot}} - t_{\text{a}}$ until the first slot in the schedule is free, and if we assume that it uses the same IVN topology as the sender, Δ_{stream1} should be the same as $\Delta_{\text{stream0}} + \Delta_{\text{slot}}$. The only delay not accounted for, then, is the delay added by the wireless connection Δ_{w} (to be discussed later in this section). Taking all delays into account, the total end-to-end delay is then:

$$\Delta_{\text{total}} = 2\Delta_{\text{stream}} + \Delta_{\text{w}} + \Delta_{\text{slot}} = 78.74\,\mu\text{s} + \Delta_{\text{w}} + \Delta_{\text{slot}} \quad (2)$$

Naturally, because this is a best-effort network, Δ_{w} cannot be computed. However, we can influence Δ_{w} indirectly by respecting the sending times of others as proposed in Section III to not increase it unnecessarily. Even better would be a prediction of Δ_{w} such that the next packet hits the slot as well. The used prediction is described later in this chapter. Easier to

Figure 5. Schedule for incoming traffic previous to the dynamic scheduling (0) and after the scheduling (1).

optimize is t_{slot} as we can directly move the placement of the slot by adapting our schedule dynamically during the runtime.

Based on these values, the schedule is dynamically created and adapted during the runtime of the simulation. To be able to do this, we use the scheduler TSNSCHED [19], which automatically calculates the network configuration based on given constrains for individual TSN streams. The schedule is not created immediately; instead every vehicle has to wait for a given time until the external streams are taken into account for the schedule creation. If a new message arrives for the first time, there is no dedicated slot in the schedule for it. Indeed, in the worst case, the scheduler cannot find a suitable schedule for the new stream. In both cases, however, the data must not get dropped; instead, it is important that we still have a slot in which arbitrary traffic can travel through the IVN even though we cannot guarantee any constraints anymore for this traffic. Due to this reason, the initial TSN configuration contains a slot which is big enough for all the packets arriving during one cycle. Only this slot can be used for incoming traffic at the start of the lifetime of every vehicle. Where this slot is placed in the cycle is not important; in our case, TSNSCHED places it at the beginning of each cycle. More precisely, the initial slot (shown in Figure 5 for *schedule 0*) is placed at the earliest possible position at which a packet could arrive at the switch: $t_{\text{slot}} = t_0 + t_{\text{prop}} + t_{\text{trans}}$.

For outgoing traffic, there are no constrains and the slot is open for the whole cycle. This leads to zero queuing times in the sender IVN. If we had used a simple schedule, e.g., the same schedule for every IVN for outgoing traffic as well, all OBUs would try to send at the same time due to the synchronized sending slots, which leads to collisions on the wireless channel.

The newly created streams need to be placed at such positions in the slot that Δ_{slot} will be zero to minimize Δ_{total}. To achieve this, not only the start time of the stream needs to be set correctly, but also the maximum allowed latency of the stream must be configured accordingly. This is necessary as TSNSCHED by default tries to optimize the schedule in a way that all slots are placed as close together as possible. This strategy increases the latency of streams as Δ_{slot} increases. The smallest possible allowed maximum latency is equal to the delay Δ_{stream1} in the receiver.

To adapt to slight changes in the arrival time, we need to update the schedule. For this, we consider the arrival times of the ten last messages received for a given stream as set $\mathcal{T}_{\text{a}}$. The difference between the earliest and the latest arrival time is taken and increased by 10 % to accommodate future semi-systematic changes of arrival times. This relative approach allows adapting to different scenarios. For example, in a scenario with a rapidly changing Δ_{w} due to a fast-changing distance, the additional slot length will be larger, which will increase the chances that the slot will fit the new message. However, we will have to block more time, which will not be used for other transmissions. In a scenario with low variation in Δ_{w}, the difference between the earliest and the latest arrival time will be shorter, resulting in a smaller additional time.

Taken together, the new slot length l_{slot} is calculated as:

$$l_{\text{slot}} = 1.1 \times (\max_{t_{\text{a}} \in \mathcal{T}_{\text{a}}} t_{\text{a}} - \min_{t_{\text{a}} \in \mathcal{T}_{\text{a}}} t_{\text{a}}) + t_{\text{trans}} \tag{3}$$

We decided to use nanoseconds as the resolution of the time stamps. With this resolution, we can save the individual time points with enough precision for a meaningful scheduling and at the same time not create unnecessarily high computational load. However, if we would not consider this loss in precision, the individual slots in the schedule would not fit. To solve this, we have to round down the arrival time stamps to make sure that every arrival time is within the slot. At the same time, we have to increase the total slot length by 1 ns because otherwise the packet cannot be transmitted completely in the slot, as its start time t_{slot} was previously moved to a not matching time point. In the worst case, this leads to a cycle up to 1 ns to big, which is equivalent to approx. one fifth of a minimal-length Ethernet frame for a bandwidth of 100 Mbit/s.

Using TSNSCHED for scheduling has some limitations. The slot length can only be set indirectly via the packet size in TSNSCHED. Increasing the packet size, however, does not only increase the slot size but at the same time also increases transmission time t_{trans}, which moves the schedule to a later time point and increases the size of the guard bands. As we still want to minimize the allowed latency to reduce Δ_{total} as much as possible, we have to recalculate the Δ_{stream1} as well. Due to the previously explained rounding errors and the wrong transmission time the scheduler assumes, the stream delay is artificially lengthened to ensure that the slot is big enough. This could lead to small queuing times, but as we will show in Section VI this was not the case, as the correct t_{arr} was late enough behind t_{slot} to completely avoid the potential offset.

We simulated two different scenarios to investigate the improvements of our proposed hierarchical cooperative TSN scheduling approach regarding the total end-to-end delay Δ_{total}.

A. Static platoon scenario

The first scenario uses the static example included with Plexe, which is not changing the distance between the vehicles during the simulation and ensures that no two cars try to send at the same time. As a result, Δ_{w} is constant and in combination with the deterministic behavior of the sender, we have a known

constant arrival time t_{a} in the receiver (relative to the start of a cycle) over the complete simulation time.

Due to these constant values, the scheduler does not need to update the schedule. This, together with the fact that we create no additional traffic – neither between nor within the vehicle – every slot-start can be placed exactly aligned with the stream-start and as a result Δ_{slot} is zero. This means that, after the first adaption of the schedule, the slot-hit-rate should be 100 % leading to zero jitter in the receiver network as well.

This scenario works as a baseline to show the improvements and at the same time how cooperative TSN behaves if all data is received with a constant arrival time t_{arr} relative to the starting time of the cycle.

B. Sinusoidal platoon scenario

The second scenario is the sinusoidal scenario, again included with Plexe. Here, the platoon leader speed is oscillating, leading to changing distances between the cars. The distance change results in different Δ_{w}, which, in turn, influences the arrival time at the receiver. Even though the distance change is only approx. 2 m, and thus the delay varies between 33 ns and 40 ns, this needs to be considered. If we ignored these changes, two things could happen. If a new packet arrives earlier, the arrival time t_{a} would not match with the slot start time t_{slot}, resulting in additional queuing time. If a new packet arrives later, the slot would not be big enough to transmit the packet. As we try to fit our slots exactly, we would expect zero jitter again, but the chances are given, that a packet have to wait for a whole cycle.

VI. Results

Our goal was to improve the end-to-end delay Δ_{total} for messages received from another vehicle without modifying the sender or managing the connection between the vehicles directly.

To achieve this, we dynamically reschedule the IVN of the reception if we identify a new stream. To reduce the load on the scheduler and sample more values to increase the chance of setting the slot length better regarding the slot hit rate without wasting too much time, we decided to update the schedule only after we received ten new values, which should lead to a sample time of approx. 1 s. Additionally, for ease of demonstration, we artificially delay the rescheduling by the index of the car plus five seconds.

Figure 6 shows the average end-to-end delay of all messages arriving within a cycle, independent of the sender, split by the receiver, for the static scenario. Each time point t_{reconf} at which for the first time a new configuration was applied (because we received enough messages from the same sender, as defined by $\mathcal{T}_{\mathrm{a}}$) and thus for the first time the network was rescheduled is marked with a dotted line. At the beginning of the simulation, the resulting average end-to-end delay is different for every car as every car sends at a different time. Due to the different sending times, the arrival time t_{a} in the car differs, but the slot start time t_{slot} is the same in every car. Thus, Δ_{slot} is varying, which influences Δ_{total} directly.

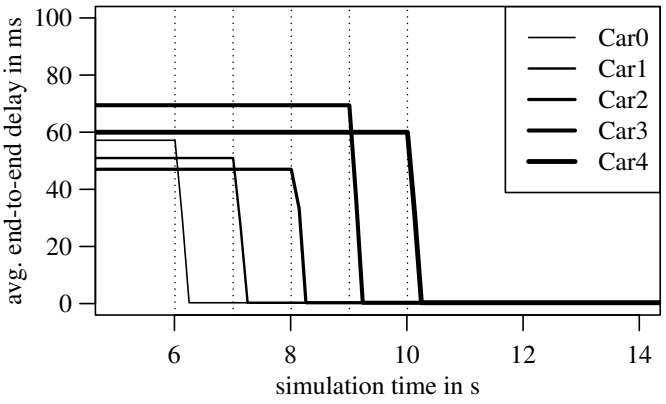


Figure 6. Total end-to-end delay Δ_{total} for all cars in the platoon, averaged over all incoming connections from all other cars.

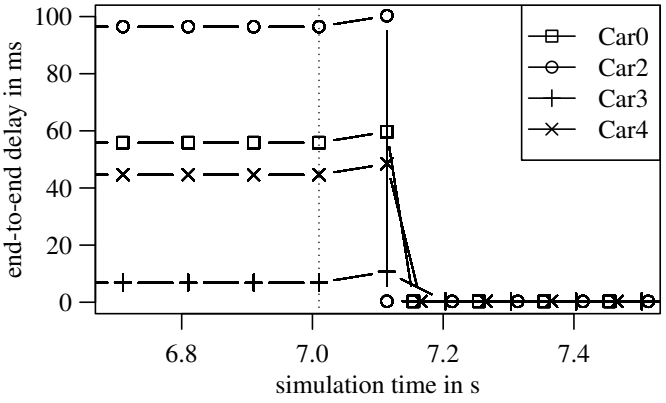


Figure 7. End-to-end delay Δ_{total} in c_1 with initial rescheduling at t_{reconf} (dotted line).

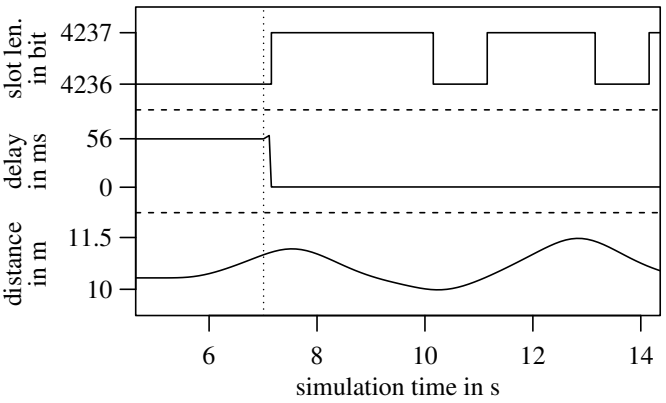


Figure 8. Comparison of the distance between c_0 and c_1 that influences Δ_{w} and thus increases the slot length l_{slot}. The delay Δ_{total} stays low after the initial rescheduling at t_{reconf} due to the big enough slot.

Figure 7 shows the end-to-end delay of individual messages for c_1 before and after the first run of the scheduler (marked with a dotted line), for the static scenario. For c_1, t_{reconf} is at simulation second 7.01 as at this time point the criteria for the creation of a new stream are fulfilled (and the artificial delay, which was introduced for visualization purposes, passed as well). Noticeable is the increase in the end-to-end delay of the first message from every sender directly after the rescheduling. This happens because the open slot was initially placed at the earliest possible position of the cycle, as shown in Figure 5 for *schedule 0*. After the rescheduling, this slot is still existent but moved to a later position in the cycle. The first open time t_{slot} is placed by the scheduler at the position so that Δ_{slot} is zero. As already mentioned in Section V, the scheduler always tries to place all slots as densely as possible. Thus, the slot is moved between the slots of the new streams. This results in additional new closing time at the start of the cycle and is increasing the queue time and Δ_{slot}. Every message that follows afterward hits its designated slot directly, resulting in no queue time and no added Δ_{slot} and zero jitter.

The only difference between the static and sinusoidal scenario regarding our investigations is the changing distance and, as a result, changing Δ_{w}. Comparing the wireless delay Δ_{w} with the total delay Δ_{total}, Δ_{w} is three magnitudes lower than Δ_{total}. To be able to see the influence of the dynamic rescheduling, Figure 8 shows not only the end-to-end delay, but also the distance between c_0 and c_1, and how the slot size changes with different distances for the sinusoidal scenario.

One drawback of the chosen scheduler can be seen in Figure 5: For *schedule 1* one can see that instead of multiple small slots, which match the streams exactly, one big slot is placed. This is a behavior by the scheduler, to avoid unnecessary state switching of the ports. In the internal representation of the scheduler, the slots are still exactly defined, so that another stream is not scheduled during this slot, as it would result in a collision. Due to this too big slot scheduled by TSNSCHED, even with a later arrival time, there is still enough time left in the slot so that the transmission can be transmitted. Only with the last stream in the cycle, this would be a problem.

VII. Conclusion and future work

In this paper, we proposed *hierarchical cooperative Time Sensitive Networking (TSN)* as an approach for cooperating (rather than centrally managed) TSN networks via best-effort wireless links. Our approach is both suited to mixed In-Vehicle Network (IVN) and Vehicle to Everything (V2X) systems of highly mobile nodes like platooning and generalizable to similar systems requiring streams across cooperating TSN networks.

We also presented an approach for the simulative performance evaluation of such systems and described an Open Source reference implementation using well-established simulation tools.

We concluded with the results of a proof of concept evaluation to demonstrate the feasibility and dynamic adaptivity of our approach.

In future work, we plan to investigate more advanced scheduling techniques to increase the utilization of the IVN, which will also be increased to real scale. For this a different scheduler will be necessary.

References

[1] "Time-Sensitive Networking Profile for Automotive In-Vehicle Ethernet Communications," IEEE, Draft Standard P802.1DG/D4.0, Jul. 2024.

[2] R. Debnath, M. S. Akinci, D. Ajith, and S. Steinhorst, "5GTQ: QoS-Aware 5G-TSN Simulation Framework," in *2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall)*, IEEE, Oct. 2023.

[3] M. Segata et al., "Multi-Technology Cooperative Driving: An Analysis Based on PLEXE," *IEEE Transactions on Mobile Computing*, vol. 22, no. 8, pp. 4792–4806, Aug. 2023.

[4] N. Finn, "Introduction to Time-Sensitive Networking," *IEEE Communications Standards Magazine*, vol. 6, no. 4, pp. 8–13, Dec. 2022.

[5] K. Zanbouri, M. Noor-A-Rahim, J. John, C. J. Sreenan, H. V. Poor, and D. Pesch, "A Comprehensive Survey of Wireless Time-Sensitive Networking (TSN): Architecture, Technologies, Applications, and Open Issues," arXiv, cs.RO 2312.01204, 2023.

[6] Z. Luo et al., "Hierarchical Computing Network Collaboration Architecture for Industrial Internet of Things," in *2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)*, vol. 6, IEEE, Jan. 2023, pp. 57–64.

[7] D. Puthal, S. Mohanty, S. Wilson, and U. Choppali, "Collaborative Edge Computing for Smart Villages," *IEEE Consumer Electronics Magazine*, vol. 10, no. 3, pp. 68–71, May 2021.

[8] P. Karle et al., "EDGAR: An Autonomous Driving Research Platform – From Feature Development to Real-World Application," arXiv, cs.RO 2309.15492, 2023.

[9] D. Andronovici, I. Turcanu, J. Bigge, and C. Sommer, "Cross-Validating Open Source In-Vehicle TSN Simulation Models With a COTS Hardware Testbed," in *2024 IEEE Vehicular Networking Conference (VNC)*, IEEE, May 2024, pp. 172–179.

[10] J. Zou, X. Dai, and J. A. McDermid, "reTSN: Resilient and Efficient Time-Sensitive Network for Automotive In-Vehicle Communication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 3, pp. 754–767, Mar. 2023.

[11] Z. Zhou, J. Lee, M. S. Berger, S. Park, and Y. Yan, "Simulating TSN traffic scheduling and shaping for future automotive Ethernet," *Journal of Communications and Networks*, vol. 23, no. 1, pp. 53–62, Feb. 2021.

[12] D. S. Buse, M. Schettler, N. Kothe, P. Reinold, C. Sommer, and F. Dressler, "Bridging Worlds: Integrating Hardware-in-the-Loop Testing with Large-Scale VANET Simulation," in *14th IEEE/IFIP Conference on Wireless On demand Network Systems and Services (WONS 2018)*, Isola 2000, France: IEEE, Feb. 2018, pp. 33–36.

[13] M. K. Atiq, R. Muzaffar, O. Seijo, I. Val, and H.-P. Bernhard, *When IEEE 802.11 and 5G Meet Time-Sensitive Networking*, 2022.

[14] O. Seijo, X. Iturbe, and I. Val, "Tackling the Challenges of the Integration of Wired and Wireless TSN With a Technology Proof-of-Concept," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 10, pp. 7361–7372, Oct. 2022.

[15] G. Durisi, T. Koch, and P. Popovski, "Toward Massive, Ultrareliable, and Low-Latency Wireless Communication With Short Packets," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1711–1726, Sep. 2016.

[16] A. B. D. Kinabo, J. B. Mwangama, and A. A. Lysko, "Towards Wi-Fi-based Time Sensitive Networking Using OMNeT++/NeSTiNg Simulation Models," in *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, IEEE, Dec. 2021.

[17] I. Turcanu and C. Sommer, "Poster: Potentials of Mixing TSN Wired Networks and Best-Effort Wireless Networks for V2X," in *2021 IEEE Vehicular Networking Conference (VNC)*, IEEE, Nov. 2021, pp. 135–136.

[18] P. A. Lopez et al., "Microscopic Traffic Simulation using SUMO," in *21st IEEE International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, Nov. 2018.

[19] A. C. T. dos Santos, B. Schneider, and V. Nigam, "TSNSCHED: Automated Schedule Generation for Time Sensitive Networking," in *2019 Formal Methods in Computer Aided Design (FMCAD)*, IEEE, Oct. 2019, pp. 69–77.