IEEE Open Journal of
Intelligent Transportation
Systems

Received 7 November 2021; revised 21 March 2022; accepted 20 April 2022. Date of publication 28 April 2022; date of current version 10 May 2022.

Digital Object Identifier 10.1109/OJITS.2022.3171072

ITANS: Incremental Task and Network Scheduling for Time-Sensitive Networks

ANNA ARESTOVA, WOJCIECH BARON, KAI-STEFFEN J. HIELSCHER, AND REINHARD GERMAN

Department of Computer Networks and Communication Systems, University of Erlangen–Nürnberg, 91058 Erlangen, Germany

CORRESPONDING AUTHOR: A. ARESTOVA (e-mail: anna.arestova@fau.de)

This research has been developed as part of the MBPLE4Mobility project and is funded by the Federal Ministry of Economic Affairs and Climate Action.

ABSTRACT Recent trends such as automated driving in the automotive field and digitization in factory automation confront designers of real-time systems with new challenges. These challenges have arisen due to the increasing amount of data and an intensified interconnection of functions. For distributed safety-critical systems, this progression has the impact that the complexity of scheduling tasks with precedence constraints organized in so-called cause-effect chains increases the more data has to be exchanged between tasks and the more functions are involved. Especially when data has to be transmitted over an Ethernet-based communication network, the coordination between the tasks running on different end-devices and the network flows has to be ensured to meet strict end-to-end deadlines. In this work, we present an incremental heuristic approach that computes schedules for distributed and data-dependent cause-effect chains consisting of multi-rate tasks and network flows in time-sensitive networks. On the one hand, we provide a common task model for tasks and network flows. On the other hand, we introduce the concept of earliest and latest start times to speed up the solution discovery process and to discard infeasible solutions at an early stage. Our algorithm is able to solve large problems for synthetic network topologies with randomized data dependencies in a few seconds on average under strict end-to-end deadlines. We have achieved a high success rate for multi-rate cause-effect chains and an even better result for homogeneous or harmonic chains. Our approach also showed low jitter for homogeonous cause-effect chains.

INDEX TERMS Cause-effect-chains, real time, task scheduling, time-sensitive networks.

I. INTRODUCTION

UPCOMING innovations entail a steadily increasing amount of new functionalities, highly interconnected smart devices such as sensors and actuators, and an increasing volume of data transmitted over a communication network. The future project *Industry 4.0* is a promoter of this trend. It propels the digitization of industrial production and strives for greater connectivity between sensors, actuators, and machines [1], [2]. This leads to a growth of data in the system and creates new challenges to coordinate the amount of information and functionalities on processing and network systems and makes more difficult to comply to timing requirements. In the automotive context, the emergence of automated driving and advanced driver assistance systems make interconnectivity and consequently end-to-end constraints even more important. Both functions rely on tightly coupled processing chains consisting of sensors like radar, LiDAR, and cameras [3], that perceive the world and further processing tasks that generate proper outputs for actuators. Also, the railway domain drives research to raise the grade of automation of the Automatic Train Operation (ATO) system introducing new applications and dependencies to achieve a semi-automatic or even an unattended train operation [4].

Especially if tasks from safety-related domains like the aforementioned are related to each other by data dependency in so-called Cause-Effect Chains (CECs) the end-to-end timing behavior becomes more relevant and more difficult to handle. Relevant end-to-end metrics are the end-to-end system response time and the maximum data age [5]. The former expresses the reaction of the system to a change in value [6]. Data age, in contrast, describes the maximum time that input data influences the output of a CEC [7]. The Sensor-to-Actuator delay constraint is a known representative of such end-to-end chains [7].

The review of this article was arranged by Associate Editor Erik Jenelius.

Meeting the end-to-end requirements goes beyond the scheduling and analysis of a single device to the examination of a whole distributed system. This also includes the communication system in addition. Recent developments entail new communication technologies on top of Ethernet like the Time-Sensitive Networking (TSN) [8] that seem to be promising candidates to implement the future communication systems for different safety-critical domains and to establish a common and standardized development ground for future innovations. Even though TSN provides beneficial mechanisms to realize reliable real-time communication it still relies on proper configuration and scheduling. Network flow scheduling in time-sensitive networks and task scheduling is a widely explored area. The generation of combined schedules for chained tasks and network flows in distributed systems is a multi-objective problem that has been little addressed. It requires an adaptation to the applied technologies and a careful analysis. Some works look at the problem from the other side: they provide formal methods to analyze the maximum end-to-end delays of already scheduled CECs [6], [9]. Other singular papers exist that address the combined task and network scheduling with the help of exact methods such as Integer Linear Programming (ILP) [10] and satisfiability approaches [11], which are able to provide optimal solutions but can lead to long runtimes of several hours depending on the complexity of the problem. Often, an optimal solution is not necessary. Neither do these works consider the characteristics of TSN mechanisms. Since TSN has not been regarded in combination with CECs yet and due to the lack of heuristic solutions in the field of joint task and network scheduling, we want to introduce a heuristic approach *ITANS* for incrementally scheduling multi-rate CECs in time-sensitive networks using the Time-Aware Shaper (TAS). The emphasis is first put on the consideration of the end-to-end deadlines for the system response time. In this work, we will address the most common difficulties that arise for scheduling distributed and chained tasks and network flows. Additionally, will introduce a common task model for task and network flows for distributed real-time systems. Furthermore, we will apply the concept of the *earliest start time* (*EST*) and *latest start time* (*LST*) that enables finding feasible as well as invalid solutions faster. Finally, we will demonstrate by means of selected use cases from the automotive domain that the ITANS heuristic allows to schedule more than a thousand tasks and network flows organized in CECs of different lengths and with different system configurations within a few seconds on average. We achieve a high success rate for multi-rate CECs and provide even better results for homogeneous and harmonic CECs. Moreover, we are able to provide a low jitter for homogeneous CECs.

The paper is organized as follows. In Section II, we introduce the applied TSN mechanisms. We present the system model in Section III and formulate the necessary scheduling constraints in Section IV. The heuristic approach is presented in Section V. We evaluate and reflect on the performance of ITANS in Section VI. In Section VII we review the related work. Finally, we discuss and conclude our approach and give a preview of future activities in Section VIII.

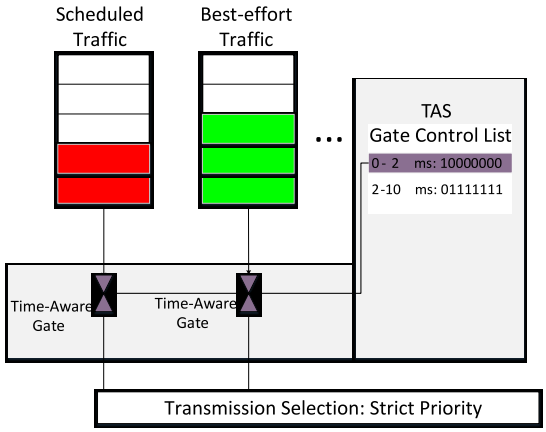


FIGURE 1. TAS components, cf. [19].

II. FUNDAMENTALS

A. TIME-SENSITIVE NETWORKING

Since the presented algorithm exploits characteristic properties of TSN, we want to introduce TSN in more detail to the reader. TSN provides a set of standardized mechanisms based on IEEE 802.3 Ethernet that make the Ethernet technology real-time capable [12], [13] and enable the coexistence of network traffic with mixed criticality, e.g., jitter- and latency-sensitive real-time traffic and best-effort traffic without jitter and latency requirements, solely resource guarantees [14]. Moreover, TSN in combination with Ethernet supports link speeds greater or equal to 1 gigabit per second and thus provides high capacities for communication links as opposed to the PROFINET technology. TSN enriches the Ethernet with the following aspects:

- Synchronization of time
- Reliability
- Resource management
- Latency guarantees

Latency guarantees promote deterministic behavior in the network and, depending on the applied mechanism, also promote low jitter to time-sensitive network traffic. Mechanisms such as frame preemption (IEEE 802.1 Qbu [15]), cyclic queuing and forwarding (IEEE 802.1 Qch [16]), asynchronous traffic shaping (IEEE 802.1 Qcr [17]), and TAS (IEEE 802.1 Qbv [18]) can facilitate these goals. In this work, we focus on the latter.

TAS introduces a Time Division Multiple Access (TDMA)-based principle that enables the assignment of transmission slots to different traffic classes. Therefore, each egress queue of a physical network port that supports 802.1 Qbv is preceded by a time-aware gate. The time-aware gates can open (state 1) or close (state 0) according to a cyclic schedule specified in the gate control list (GCL) (Fig. 1). Solely packets located in queues with open gates are considered for transmission by the transmission selection. If

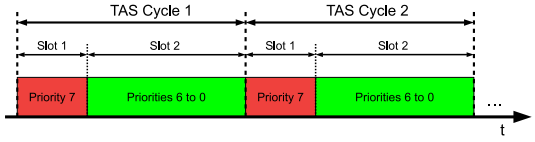


FIGURE 2. Serialization on the wire, cf. [19].

several queues are open, the transmission selection operates according to the established algorithm. In most cases, strict priority is applied.

An exemplary configuration of the TAS components is illustrated in Fig. 1. The critical traffic is transmitted in the relative interval [0, 2] ms within a cycle of 10 ms. The binary representation 10000000 shows the states of the time-aware gates. Each position represents one of 8 time-aware gates. In this example and also in our approach, the transmission selection policy is set to strict priority and one egress queue is dedicated for time- and safety-critical traffic. This queue assignment is also proposed in [20], [21]. The resulting serialization on the wire is depicted in Fig. 2. The example from Fig. 1 shows a possible TAS configuration for one egress port. To be able to guarantee latency and jitter bounds for one or several flows, a feasible configuration of the gates associated with time-critical traffic has to be found on each egress port that is traversed by critical flows.

Additionally, 802.1 Qbv introduces guard bands. They prevent the transmission of packets a certain interval before the closing time of their gate and before the opening time of other gates. This mechanism prevents non-critical traffic from delaying critical traffic in their own transmission window. Without the guard bands, a non-critical network packet could initiate a send process just an instant before the closing time of its gate. Since the transmission process cannot be interrupted at any point in time, the non-critical packet might finish or might be interrupted in the transmission slot of the critical traffic. Such a safety interval may have the size of the transmission duration of a Maximum Transmission Unit (MTU). Further guard band mechanisms are described in [18]. TAS requires a common notion of time in all involved network devices and end-device. This can be ensured by the generalized Precision Time Protocol (PTP) specified in [22].

B. TSN ON THE DATA LINK LAYER

TSN mechanisms are located on the data link layer in the Open Systems Interconnection (OSI) model. As TSN is part of the IEEE 802.1 Working Group, network flows that want to make use of TSN services carry a Virtual Local Area Network (VLAN) tag. This tag contains a priority code point (PCP) field that is used for traffic classification. The PCP value that has the integer range of 0-7 assigns a priority to the flow. Each priority value can be mapped to one egress queue if a physical port has eight egress queues, which usually applies. Additionally, TSN introduces the terms *talker* that refers to the source application and *listener* that identifies the destination application of a TSN flow. TSN flows can be propagated in a unicast or multicast fashion.

C. TSN IN END-DEVICES

TSN mechanisms not only apply to network devices but can also be configured in end-devices. Linux-based operating systems adopted special techniques to enable time-triggered injection of network traffic. The traffic control module *tc* of Linux allows to apply Quality of Service (QoS) mechanisms such as shaping, scheduling, policing, and packet dropping to network traffic [23]. The submodule *qdisc* (queuing discipline) allows the configuration of traffic control disciplines for each network interface. Queuing disciplines can control the behavior of the whole network interface card (NIC) or certain network classes. The queuing discipline of interest to achieve real-time behavior is *Earliest TxTime First* (ETF) [24]. ETF allows users to determine on application level the time when a network packet of a certain traffic class should be dequeued from the traffic control buffer into the hardware queue of a NIC. Furthermore, it gives users the ability to specify the time when the packet should leave the NIC if the function *offload* is specified and supported [24]. Special network drivers are necessary to realize the ETF functionality. Additionally, the system clock and the hardware clock of the NIC have to be synchronized in order to achieve the desired behavior.

The mapping of socket priorities to traffic classes and mapping of traffic classes to hardware queues can be done either by the *Multiqueue Priority* (MQPRIO) queuing discipline [25] or the *Time Aware Priority Shaper* (TAPRIO) queuing discipline [26]. Both disciplines are *classfull*, i.e., they are able to create classes to which other queuing disciplines can be applied. ETF, in contrast, works on a traffic class. TAPRIO additionally provides an implementation of the TAS operating principle as a software solution (at this point in time). When using TAPRIO, it is possible to configure time slices for network classes of an egress port. These time slices determine the interval when packets of a certain class are forwarded to the NIC.

Queuing disciplines can be combined by switching them hierarchically. When TAPRIO acts as the superior discipline and ETF is applied to one or multiple classes of TAPRIO, TAPRIO controls the transmission windows of different traffic classes and ETF specifies when a certain packet of a traffic class is forwarded to the NIC and put on the wire [27], [28]. The combination of MQPRIO and ETF allows the classification of outgoing network traffic and the timely injection of individual packets of a certain traffic class on the wire. MQPRIO does not subdivide the time space into transmission windows for dedicated traffic classes. We refer at least to the MQPRIO-ETF combination in our scheduling approach. However, we prefer and advise using the TAPRIO-ETF configuration to avoid the collision with packets from non-critical traffic classes that are generated in the same end-device. Our goal is to specify the transmission offset and deterministic travel time through the

communication network for each TSN flow with the ITANS algorithm.

III. SYSTEM MODEL

First, we introduce a system model for a distributed real-time system consisting of processing and network resources and tasks. We divide the system model into an architectural model and a task model. We adopt the concept of a common representation of processing and network tasks from [10].

A. ARCHITECTURAL MODEL

We denote the architectural view of the system as a graph $\mathcal{G}(\mathcal{V}, \mathcal{L})$ consisting of vertices $\mathcal{V}$ and links $\mathcal{L}$. The set $\mathcal{V}$ comprises end-devices, denoted as $v_i^e \in \mathcal{V}$, and network switches, represented by $v_i^s \in \mathcal{V}$. Each end-device node v_i^e owns one or several processing cores. The devices are physically interconnected by bi-directional full-duplex Ethernet links. For each physical link between two distinct nodes v_j and v_k $(j! = k)$ the set $\mathcal{L}$ contains two directed logical Ethernet links $l_{j,k}$ and $l_{k,j}$. The links uniquely identify the physical source and destination ports. Each logical link $l_{j,k}$ is characterized by the tuple $\langle r_{j,k}, s_{j,k}\rangle$, with $r_{j,k}$ describing the link's data rate in bit per second and $s_{j,k}$ denoting the length of the link in meter.

B. TASK MODEL

The task set $\mathcal{T}$ is divided into the representation of processing tasks $\tau_i^p \in \mathcal{T}$, and network tasks $\tau_i^n \in \mathcal{T}$. Each processing task τ_i^p is defined by the tuple $\langle p_i, \phi_i, d_i, e_i^j, \omega_i^j\rangle$, with p_i denoting the task period, ϕ_i the release offset referred to the relative beginning $t' = 0$ of p_i, and d_i the relative deadline. The worst-case execution time (WCET) is depicted by e_i^j and the worst-case response time (WCRT) is denoted by ω_i^j. Index j refers to the end-device v_j^e and expresses that WCET and WCRT are hardware-dependent. The WCRT comprises the difference between the release time ϕ_i and the completion time of the task that is $\phi_i + e_i^j$ plus the delay $D_{preempt}^i$ due to preemption by other tasks and the scheduling overhead D_{sched}^j on v_j^e. The preemption delay depends on the number of tasks on the same core that have a higher priority and whose execution overlap with the task of interest. The scheduling overhead arises due to context switches and depends on the hardware and the operating system (OS).

Whenever tasks are connected by a precedence order due to data dependencies, they form a CEC [29]. It means that tasks supply data to their immediate successor tasks. Additionally, we introduce the term *job* that describes a task in execution. $\tau_{i,k}$ is the k-th execution or job of task τ_i. Thus, ϕ_i can be adjusted to $\phi_{i,k}$ of the k-th instance, which corresponds to $\phi_{i,k} = \phi_k + (k-1) \times p_i$ since the first release $(k = 1)$ of task τ_i. Also ω_i^j may be different in each period. Thus, $\omega_{i,k}^j$ defines the WCRT of the k-th instance. Tasks can be data-dependent on each other. We refer to the dependencies between jobs as job-level dependencies [29]. $\{\tau_{1,1}^p, \tau_{2,3}^n, \tau_{3,2}^p\}$ and $\{\tau_{1,2}^p, \tau_{2,5}^n, \tau_{3,2}^p\}$ illustrate end-to-end job-level paths in Fig. 3. Since we allow different periods within a CEC the temporal distances between the same jobs on different end-to-end job paths may vary. This variation is among others caused by task preemption ($\tau_{1,1}^p$ and $\tau_{3,2}^p$), and the delay of task execution ($\tau_{1,4}^p$ and $\tau_{3,2}^p$). Another major reason for the jitter of end-to-end latency is the transition of periods. p_3 is four times greater than p_2 and two times greater than p_1 in the example of Fig. 3. Thus, the job-level path consisting of the jobs $\{\tau_{1,1}^p, \tau_{2,3}^n, \tau_{3,2}^p\}$ in Fig. 3 is longer than the path $\{\tau_{1,2}^p, \tau_{2,5}^n, \tau_{3,2}^p\}$. Whenever a task with predecessors has a higher period than the first task in a CEC, the jitter regarding the system response time on different job-level paths is inevitable. Also in case of homogeneous periods within a CEC, the path length can differ due to preemption or priority scheduling.

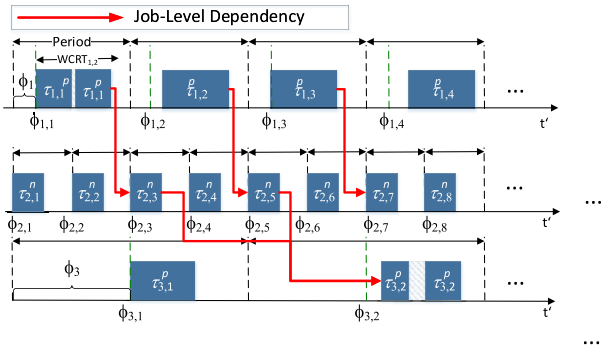


FIGURE 3. Job-level dependencies for $\tau_1 \prec \tau_2 \prec \tau_3$.

One CEC $cec_i \in \zeta$ is captured by the tuple $\langle T_i, e2e_i, hp_i\rangle$, with T_i comprising all processing and network tasks that are related by data dependencies, $e2e_i$ denoting the maximum allowed end-to-end duration on all job-level dependency paths, and hp_i the hyperperiod of cec_i. The end-to-end time refers in our case primarily to the maximum response time of a system. The hyperperiod, in general, is the least common multiple of the periods of tasks on the same core after which a task schedule is repeated [30]. The hyperperiod of a CEC is the least common multiple of all hyperperiods of the cores on which its processing tasks are running and the hyperperiod of the communication network if the CEC has network tasks. A network task is created when a processing task of a CEC is located on a different end-device than its preceding task. As illustrated in Fig. 4, task τ_1^p has two successor tasks τ_2^p and τ_3^p. While τ_1^p and τ_2^p share the same end-device v_1^e, τ_3^p has been allocated to end-device v_2^e. Thus, the network task τ_4^n has to be inserted and becomes the immediate successor of τ_1^p and the new predecessor of τ_3^p. The identification of the predecessors (previous tasks) of an allocated task is determined by the function $prev : \mathcal{T} \longrightarrow \mathcal{T}$. The determination of successors (next tasks) of a task is defined by the mapping $next : \mathcal{T} \longrightarrow \mathcal{T}$.

The network task set comprises TSN flows that arise due to allocation of CEC tasks to different nodes. We assume that each TSN flow consists of one Ethernet frame to simplify the

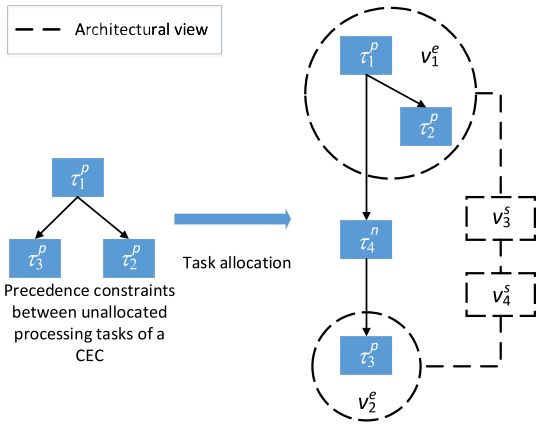


FIGURE 4. CEC allocation to end-devices.

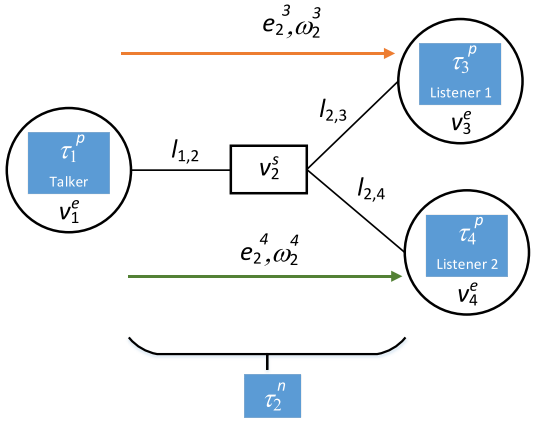


FIGURE 5. Example of network task characteristics.

characterization. To unify the representation of processing and network tasks, each flow or network task τ_i^n is characterized by the tuple $\langle p_i, \phi_i, d_i, e_i^j, \omega_i^j, s_i, pt_j \rangle$, with p_i denoting the period, ϕ_i the relative transmission start offset referred to the beginning of p_i, and d_i the maximum allowed end-to-end network latency. In contrast to processing tasks, e_i^j determines the worst-case transmission time (WCTT) through the communication network from the source node to the sink v_j^e. We pursue a no-wait principle introduced in [31] and applied to TSN flows in [20] and discussed in our former work [32] where flows get exclusive access to all communication links of their path as soon as they are injected onto the wire by the talker node. It means that they will not experience any interference with other flows and they do not have to wait in egress queues of the traversed ports. Thus, preemption of network tasks is not allowed and consequently the WCRT ω_i^j of τ_i^n equals the WCTT e_i^j. We maintain the index j of the node for the WCTT and WCRT to clarify that if the flow has multiple listeners (consuming processing tasks) e_i^j and ω_i^j may differ for each listener if the listeners are located on different nodes. The size of the flow in Byte is defined by s_i, and pt_j specifies the flow path. pt_j consists of an ordered sequence of Ethernet links included in $\mathcal{L}$ starting from the talker node and reaching to the listener node v_j^e. It is included in the list $\mathcal{P}_i$ that contains the paths to all listeners. Furthermore, we introduce a mapping of the flow to the talker node and the listener nodes. The assignment of a talker node to the flow is captured by the function $talkV : \mathcal{T} \longrightarrow \mathcal{V}$. The mapping of listener nodes to a flow is defined by $listsV : \mathcal{T} \longrightarrow \mathcal{V}$. The talker application of τ_i^n can be retrieved by $prev(\tau_i^n)$ and the listeners by $next(\tau_i^n)$ accordingly.

Fig. 5 shows some characteristics of the network task τ_2^n. The talker and at the same time the predecessor task of τ_2^n is τ_1^p. The listeners are represented by the processing tasks τ_3^p and τ_4^p. The listener nodes v_3^e and v_4^e can be retrieved with the function $listsV(\tau_2^n)$. Each listener has its individual WCTT and WCRT. The path pt_3 from v_1^e to v_3^e is described by the ordered sequence $\{l_{1,2}, l_{2,3}\}$ and the path pt_4 from v_1^e to v_4^e is described with $\{l_{1,2}, l_{2,4}\}$.

IV. FORMAL CONSTRAINTS

We declare a resulting ITANS schedule as feasible if the following set of constraints is fulfilled. The constraints describe the required timing behavior of the set of tasks to be scheduled. We divide them into CEC, task constraints, as well as *LST* and *EST* specification and constraints.

A. CEC CONSTRAINTS

We distinguish between processing and network tasks. Each CEC has one producer task and a variable number of intermediate and last tasks. The producer task and all last tasks of a task chain are always processing tasks. Intermediate tasks can be processing or network tasks. Each network task has exactly one predecessor task and a variable number of successor tasks. The predecessor and the successors of a network task are processing tasks. A processing task can follow and precede a variable number of processing tasks (same end-device) as well as network tasks. Each task is member of one or multiple CECs. The processing tasks of a CEC can have different periodic activation patterns, homogeneous (same periods), harmonic (multiple of each other), or non-harmonic. Conversely, the periods of network tasks are harmonic to each other. All CECs are loop-free, e.g., from sensor to actuator, or the other way around.

For each CEC cec_i we have to determine the jobs of the tasks $\in T_i$ that form a job-level dependency and minimize the end-to-end latency as illustrated in Fig. 3 and Fig. 6. To determine the worst-case end-to-end latency we have to form feasible job-level paths. Each path must provide a forward reachability in the time domain [6]. Backwards jumps are not allowed. To determine the response time, we search for the shortest feasible path from the first job to the last on a dependency path. This semantic is described as the *first-to-first* path delay in [6]. For example, $\tau_{1,1}^p$ from Fig. 6 cannot provide data to $\tau_{2,3}^p$. In the case of system response time, there are hp_i/p_a paths, where hp_i is the hyperperiod of cec_i and p_a describes the period of the producer task $\tau_a \in T_i$.

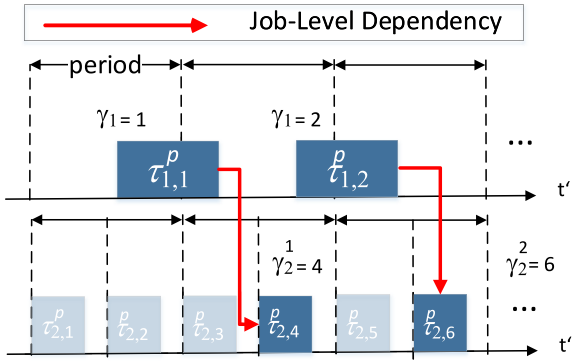


FIGURE 6. Job-level dependency between τ_1 and τ_2.

In Eq. (1) we match the jobs of the first task and its direct successors that build a job-level dependency for each CEC. At first, we first introduce the variable γ_a that depicts the γ_a-th instance of the first task $\tau^p_a \in T_i$. We formulate the precedence constraint for the immediate successors $\tau_k \in next(\tau^p_a)$. Therefore, we determine execution number γ^a_k that specifies the successor job τ_{k,γ^a_k} of job τ^p_{a,γ_a}:

$$\forall cec_i \in \zeta, \tau^p_a = first\{T_i\}, \forall \tau_k \in next(\tau^p_a), \gamma_a \in \left\{1, \ldots, \frac{hp_i}{p_a}\right\}, \gamma^a_k \in \mathbb{N}, \forall \gamma_a \; \exists \gamma^a_k : \arg\min_{\gamma^a_k} \left\{\phi_{k,\gamma^a_k} : \phi_{k,\gamma^a_k} \geq \phi_{a,\gamma_a} + \omega^j_{a,\gamma_a} + D_{comm}\right\}, \quad (1)$$

where τ_{k,γ^a_k} is the earliest job that follows after τ^p_{a,γ_a} has finished and after the additional delay D_{comm}. Therefore, γ^a_k specifies the selected task instances in a job-level dependency. Not all jobs are considered. In Fig. 3 the relevant jobs of τ^n_2 are $\tau^n_{2,3}$ and $\tau^n_{2,5}$. Selecting the jobs $\tau^n_{2,4}$ and $\tau^n_{2,6}$ would lead to the same end-to-end duration. The function $first\{T_i\}$ returns the first task of the set of tasks T_i of cec_i. The index j in the WCRT ω^j_{a,γ_a} indicates that τ^p_a is running on device v^e_j. We maintain the index j to have a common expression for network and processing tasks. Network tasks depend on the index as it indicates the destination node of one of its listeners and the WCRT might differ for each listener. The delay D_{comm} describes the delay caused by the inter-process communication if τ^p_a and τ_k are both processing tasks. Otherwise, D_{comm} represents the delay caused by the communication stack when packing or unpacking a network packet and the delay caused by the NIC before sending or receiving the packet. If the predecessor is a processing task and the successor a network task then D_{comm} depends on the device that the processing task is running on. Vice versa D_{comm} depends on the listener's node of the network task. If $p_k > p_a$ then jobs of p_k exist that are not part of any data path (compare τ^n_2 and τ^p_1 in Fig. 3).

Eq. (2) describes how a job-level dependency is extended for all direct and indirect successor tasks of τ^p_a. Here, τ_k is an intermediate task and τ_m its immediate successor. The variable γ^a_k depicts the relevant task executions that have been identified to be on a job-level path starting with τ^p_{a,γ_a}. γ^a_m identifies the instance of τ_m. To satisfy the precedence constraint, we have to identify the relevant job τ_{m,γ^a_m} that starts after the runtime of τ_{k,γ^a_k} and the additional delay D_{comm}.

$$\forall cec_i \in \zeta, \tau_k \in T_i, \forall \tau_m \in next(\tau_k) \neq \emptyset, \gamma^a_m \in \mathbb{N}, \gamma_a \in \left\{1, \ldots, \frac{hp_i}{p_a}\right\}, \forall \gamma^a_k \; \exists \gamma^a_m : \arg\min_{\gamma^a_m} \left\{\phi_{m,\gamma^a_m} : \phi_{m,\gamma^a_m} \geq \phi_{k,\gamma^a_k} + \omega^j_{k,\gamma^a_k} + D_{comm}\right\}. \quad (2)$$

Each CEC has an end-to-end deadline constraint on the job-level dependency. This contraint applies to all identified job-level paths. The difference between the release time ϕ_{a,γ_a} of the first job on a job-level path and the end time of the last job τ^p_{k,γ_k} in the same job-level dependency has to be less than or equal to the specified end-to-end latency $e2e_i$:

$$\forall cec_i \in \zeta, \tau^p_a = first\{T_i\}, \forall \gamma_a \in \left\{1, \ldots, \frac{hp_i}{p_a}\right\} : \max_{\tau^p_k \in last(T_i)} \left(\phi_{k,\gamma^a_k} + \omega^j_{k,\gamma^a_k}\right) - \phi_{a,\gamma_a} \leq e2e_i. \quad (3)$$

The function $last(T_i)$ returns the tasks without successors within the task set T_i of cec_i. The variable γ^a_k identifies the job that is reachable from τ^p_{a,γ_a}

B. TASK CONSTRAINTS

Each processing and network task has to be executed once in its period. Processing tasks are allowed to start in one period and to finish in the consecutive one, as shown in Fig. 6. However, jobs of a task are not allowed to overlap [33]. If the task τ_k has a relative offset $\phi_k > 0$, we adjust d_k to $d_k = p_k + \phi_k$. We define the maximum response of all task executions as $\omega^j_{k,max}$. Network tasks have the same WCTT for all execution. Thus $\omega^j_{k,max} = \omega^j_k$. Also, network tasks are restricted by the common network cycle *TASCycle* that is discussed in Section V-C. Thus, d_k of a network task τ^n_k is specified as $d_k = TASCycle$. The period constraint is described with:

$$\forall \tau_k \in \mathcal{T} : \phi_k + \omega^j_{k,max} \leq d_k. \quad (4)$$

Eq. (4) applies to all task executions. Processing tasks on the same processing core can preempt each other. Network tasks are planned collision-free according to the no-wait principle. Thus, the dynamic queuing delay $D^{l_{x,y}}_q$ on the link $l_{x,y}$ assumes zero. This measure contributes to less jitter and more determinism, as the remaining delays in the network are static. But it also requires a careful planning approach. As already described in our previous work [34], the static network delays cover:

- the transmission delay $D^{l_{x,y}}_{tans,k} = (s_k \times 8bit/Byte)/r_{x,y}$ of network task τ^n_k on link $l_{x,y}$
- the propagation delay $D^{l_{x,y}}_{prop}$ in dependence of the cable length $s_{x,y}$ and material composition,
- and the processing delay D^x_{proc} on a network node v^s_x.

D^x_{proc} depends on the switching factory. The WCRT ω^j_k of a network task τ^n_k from the talker to a listener is formed as follows:

$$\forall v^e_j \in listsV\left(\tau^n_k\right), pt_j \in \mathcal{P}_k :$$
$$sum_{links} = \sum_{l_{x,y} \in pt_j \setminus \{first(pt_j)\}} D^x_{proc} + D^{l_{x,y}}_{trans,k} + D^{l_{x,y}}_{prop}$$
$$\omega^j_k = D^{first(pt_j)}_{trans,k} + D^{first(pt_j)}_{prop} + sum_{links}, \tag{5}$$

with v^e_j specifying a listener's node and pt_j denoting the flow path of the network task τ^n_k to the listener's node v^e_j. The function $first(pt_j)$ returns the first Ethernet link of the set pt_j consisting of ordered links. Eq. (5) indirectly includes the constraint, that the transmission on an intermediate or last link $l_{x,y}$ cannot start until all bits of the flow have been pushed on the previous link, the flow has been propagated over the previous link, and before it has been processed on the source node v^s_x of link $l_{x,y}$. It is also evident from the formulas that we focus on store-and-forward switches.

C. LATEST START TIME

Since we provide an incremental scheduling approach, we determine for each task with predecessors the *EST* and *LST*. These parameters define the scope and the mobility for the task scheduling. Furthermore, the algorithm can use these parameters to detect invalid plans at an early stage. The calculation of the *EST* is elaborated in Section IV-D. LST_k of task τ_k tells which maximum offset the task can have within a job-level path to still comply to the end-to-end times. The LST_k of τ_k can be calculated solely with the knowledge of the maximum end-to-end delay of the CEC and the precedence relationships. The relative value is identical for all task execution. All *LSTs* are determined starting with the last task in a CEC, and before the actual scheduling process. The *LST* of the last task is the maximum allowed end-to-end time of a CEC minus the WCET and the overhead delay caused by the hardware or the OS. The *LST* of an intermediate task is derived from the *LST* of the successor with the smallest *LST* value:

$$\forall \tau_k \in \mathcal{T}, prev(\tau_k) \neq \emptyset, next(\tau_k) \neq \emptyset :$$
$$LST_k = \min_{\tau_m \in next(\tau_k)} LST_m - \left(e^j_k + D_k\right), \tag{6}$$

where τ_m is a successor task of τ_k. D_k summarizes the occurring overhead delays arisen due to inter-process communication, the communication stack, and scheduling. Since the calculation takes place before the actual scheduling process, we do not yet know the WCRT of the tasks. After determining the job-level dependencies, each relevant job of τ_k is assigned an aligned LST_k to keep the LST_k value consistent on each job-level path. Therefore, the LST_{k,γ^a_k} value of a job τ_{k,γ^a_k} is calculated by shifting LST_k by the actual start time of the first job τ^p_{a,γ_a} on the same job-level path. As illustrated in Fig. 7, the LST_2 of τ^p_2 is determined by $e2e - (WCET_2 + D_2) + \phi_{1,1}$. $\phi_{1,1}$ is the release time and the

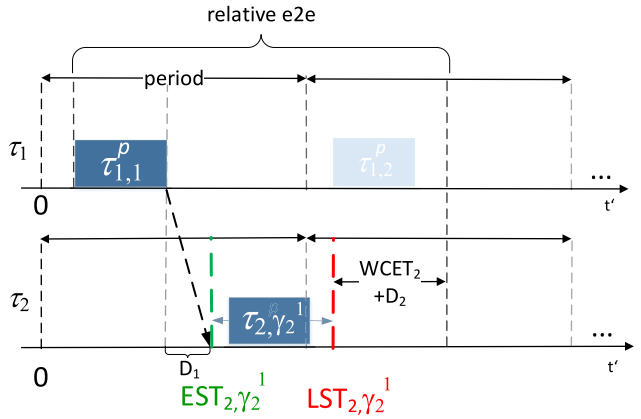


FIGURE 7. Determination of EST_2 and LST_2 for $\tau_1 \prec \tau_2$.

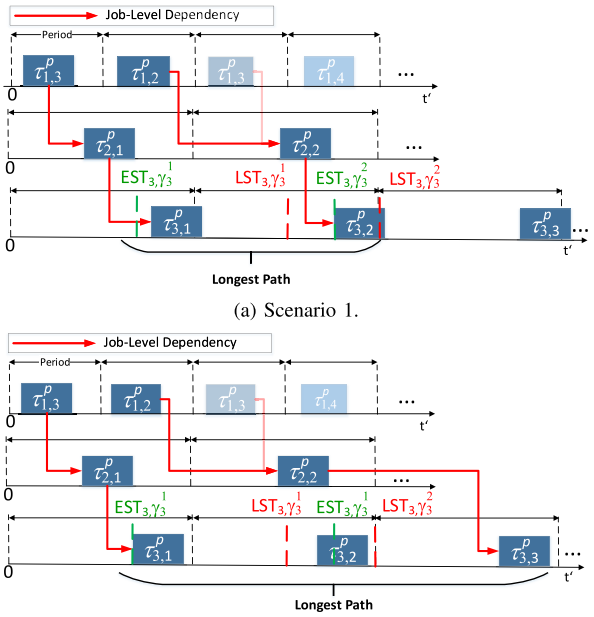
actual start time in this example. The job τ^p_{2,γ_2^1} can start its execution in the scope of EST_{2,γ_2^1} and LST_{2,γ_2^1} in order to not miss the end-to-end deadline.

D. EARLIEST START TIME

The specification of the *EST* is a major part of ITANS. We use the *EST* value to find the best task offset for all job-level paths that helps to fulfil the end-to-end deadlines. The task scheduling process will try to plan a task τ_k with the initial offset $EST_k \% p_k$. EST_k is calculated after all predecessors of τ_k have been scheduled. On a certain job-level path, the best *EST* value is the finish time of the direct predecessor job plus an overhead delay. The relative *EST* value can vary in each job-level dependency in contrast to the *LST*.

To determine the best EST_k value, we observe the duration of all job-level paths including all scheduled predecessors of τ_k and excluding τ_k and its successors. After, we calculate the EST_{k,γ^a_k} value on each path. Then, we identify the longest path and try to select the proper value for EST_k. The goal is to avoid an unnecessary extension of the longest path. The best EST_k value in order to not worsen the longest path is the finish time of the predecessor job on the longest path plus the overhead delay by hardware and the OS. The example in Fig. 8 identifies the path $\{\tau^p_{1,2}, \tau^p_{2,2}\}$ as the longest path in scenario 1. If we set EST_3 to EST_{3,γ_3^2} (the earliest start time on the second path) and manage to schedule the τ_3 at $EST_3 \% p_3$ then all deadlines are met. Respectively, $\tau^p_{3,1}$ starts before LST_{3,γ_3^1} value (latest start time on the first path) and $\tau^p_{3,2}$ does not violate LST_{3,γ_3^2}. Both absolute values refer to $t' = 0$ that is the absolute release time of the first task. If we choose $EST_3 = EST_{3,\gamma_3^1}$ then we extend the second path such that the end-to-end deadline is missed, more specifically LST_{3,γ_3^2} is exceeded.

However, the optimum value for the longest path might make another path long enough to miss LST_k. This can be the case if the period of τ_k is significantly bigger than the period of its predecessor. The worst-case on any path is that a job start time results to be just an instant before the calculated *EST* value on the path. This would enlarge the path by about



(a) Scenario 1.

(b) Scenario 2.

FIGURE 8. Determination of EST_3.

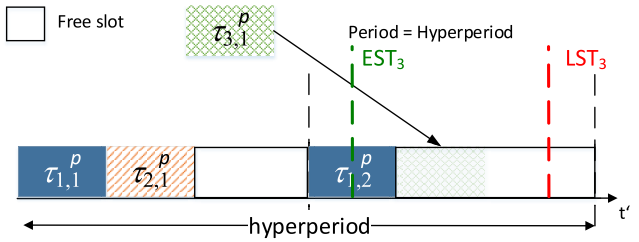


FIGURE 9. Distribution of producer tasks.

$p_k + e_k^j$. To avoid this situation, we initially set EST_k to the optimal value for the longest path and shift it incrementally until we find a feasible value that does not violate LST_k on any path. We shift by a maximum of one period of the predecessor task on the longest path. All further shift operations would repeat the behavior. In the case of several predecessors, we still determine the longest path and try to find the EST_k value that satisfies all paths. The mobility of a task is the difference between the ascertained relative EST_k and the earliest relative LST_k of all paths.

The initial EST_a for the producer tasks τ_a are zero. The LST_a values are set to p_k to provide mobility to the producer tasks. The definition of *EST* and *LST* values does not guarantee a feasible schedule. It is a support to narrow the solution space for a certain task schedule order and to eliminate invalid solutions faster.

V. SCHEDULING ALGORITHM

The proposed scheduling algorithm includes both the planning of processing tasks and the scheduling of network tasks on their assigned resources. While processing tasks allow preemption, network tasks have to run without interference. The goal of the algorithm is to schedule the tasks of all CECs so that the aforementioned constraints are met. We do not provide solutions for the route planning of flows, nor do we optimize the allocation of tasks to resources. However, we presume a proper allocation of tasks to resources and fixed flow paths.

A. ALLOCATION OF TIME RESOURCES

We manage all processing and network resources in time slots. The time slots can be occupied or free. Each time slot is described by the size in microseconds. Each occupied slot is associated with a task. Occupied slots register the start and end times of the occupancy by a task. Computational cores are the resources of the processing tasks and communication links are referred to as the resources of network tasks. In case of the communication links, the relevant resources for network tasks are egress ports.

A single processing task can occupy one slot or several split slots per period if preemption is allowed. A network task can only have one slot per period and the slots have to be equidistant within the hyperperiod of the network that is the least common multiple of all network task periods. This measure serves to comply to the no-wait principle for network flows and to avoid dynamic queuing delays.

The following procedures are explained in advance to better understand the overall algorithm from Section V-E.

B. MANAGEMENT OF PROCESSING TASK RESOURCES

The *planProcessingTask* method applied in Section V-E schedules one task τ_k^p on its computing resource c_k. The function looks for fitting free slots for task τ_k^p in the interval of one period p_k and within the hyperperiod hp_{c_k} of all task periods that are executed on the core c_k. Hence, τ_k^p has to be scheduled hp_{c_k}/p_k times per hyperperiod. The initial slot search offset for τ_k^p is $EST_k\%p_k$. From there on, further slots are searched at intervals of the task period. The search offset increases gradually if no feasible slots can be found. It does not exceed $min\{EST_k\%p_k + p_k, EST_k\%p_k + task\ mobility\}$. Here, the constraint Eq. (4) is regarded. Already scheduled tasks will not be shifted.

Whenever a task finds fitting slots within the hyperperiod, the affected free slots are reduced by the size of the task (WCET) and D_{sched}^j of the system. This leads to new occupied slots that are associated with the planned task. On success, the *slotSearchProc* function sets the earliest release time ϕ_k, all actual start times of the jobs, and all WCRTs $\omega_{k,\gamma_k^a}^j$ (on device v_j^e). The slots are organized in a ring buffer. I.e., the slot search is allowed to exceed the absolute hyperperiod value but is restricted to $EST_k\%p_k + hp_{c_k}$ initially. Each planned task is assigned a priority that is the consecutive number in planning order, which is managed for each core independently. τ_k^p from Fig. 9 gets the priority 3 for instance. The priority is higher the smaller the value is. If slots that cover the task's size and overhead are not available, the algorithm looks for a composition of free slots

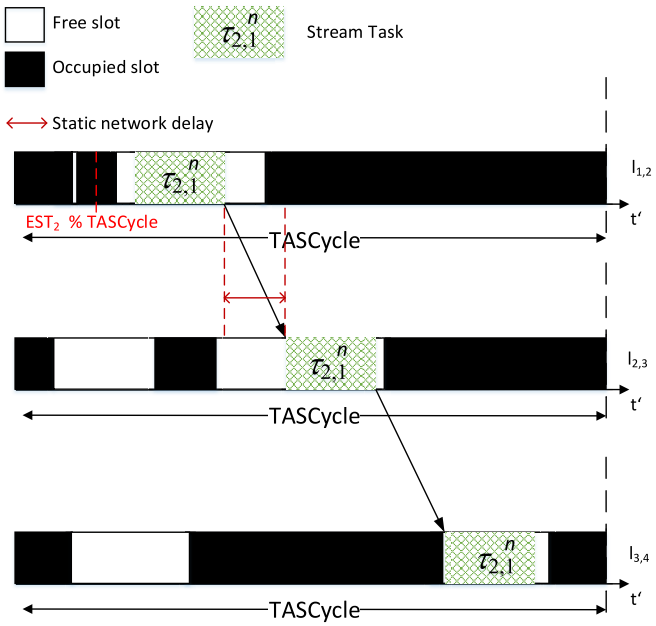


FIGURE 10. Example of slot search for a network task.

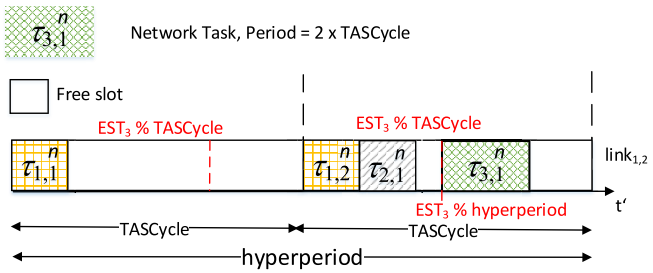


FIGURE 11. Example of flow planning.

comprising the duration of the task and each considering the overhead delays. Composite slots indicate task preemption. Whether the preemption occurs depends on the actual execution time that is less than or equal to the WCET.

C. MANAGEMENT OF NETWORK TASK RESOURCES

We introduce a management unit for network task resources that is the smallest organizing unit. Each management unit organizes its free and occupied slots. On an egress port, one management unit has the size of the smallest occurring task period *TASCycle* in the network. The flow planning is more complex as one flow has to be planned across several links. Thus, on each link free slots have to be detected that occupy the size of the flow. The size comprises the transmission duration of the flow on the link plus the interframe gap in our case. Physically, we look for resources on the source port of the link. The slots on subsequent links must be properly aligned. Fig. 10 shows a scheduling example of network task τ_2^n on the path $\{l_{1,2}, l_{2,3}, l_{3,4}\}$. If there is a fitting slot on $l_{1,2}$, we will search on $l_{2,3}$ for slots that start exactly at the same time or before the flow arrives at the egress queue of the source port of $l_{2,3}$. The arrival time on $l_{2,3}$ includes the static network delay that comprises the transmission and propagation on $l_{1,2}$ and the processing on device v_2^s (source node of $l_{2,3}$). If the slot on $l_{1,2}$ is bigger than the necessary size, $\tau_{2,1}^n$ can be shifted within the slot to fit in the next one. The same rule applies to the link $l_{3,4}$. The occupied slots are representatives of GCL entries. After finding proper slots, the GCL entries on the corresponding devices will be configured such that the gates for the critical egress queues will be opened when the slots start (see Section V-D). If the flow arrived before the start of a slot, it would have to wait. This situation would result in the need to buffer the flow and in a queuing delay. Consequently, this would violate the no-wait principle. Extending ITANS to consider queuing delays is a subject of future work.

For the whole network, we determine a common scheduling cycle *TASCycle* that corresponds to the smallest period of all occurring critical flows/network tasks in the network. The periods of critical flows must be a multiple of *TASCycle*. We refer to the harmonic periods which are also a subject in PROFINET [35] or TTEthernet [36]. Non-harmonic periods contribute to more complexity and make it more difficult to find a valid schedule. We handle the free and occupied slots over the hyperperiod of the network. Let us assume that the network cycle is 1 ms and the hyperperiod is 4 ms. A network task with a period of 1 ms has to find equidistant free slots over the hyperperiod, thus in all 4 management units and on all links. A network task with a period of 4 ms has to be scheduled in one of four management units on each link. The restriction defined in Eq. (4) must be taken into account.

Let us consider the following example in Fig. 11: $p_3 = 2\,\text{ms}$, $TASCycle = 1\,\text{ms}$, and $EST_3 = 3.5ms$. The network hyperperiod corresponds to p_3 in the example. To minimize the *e2e* time of the corresponding CEC, we normalize the EST_3 value to specify in which management unit we have to look for slots first. The first step translates the EST_3 value to the hyperperiod: $EST_3\%hyperperiod = 1.5ms$. The second step determines the index of the management unit (that begins with 1): $\lfloor (1.5ms)/TASCycle \rfloor + 1 = 2$. Hence, we search for free slots in the second management unit on the whole path starting on link $l_{1,2}$. The initial offset is $EST_3\%TASCycle = 0.5ms$. If no slots can be found in the specified management unit, we continue the slot search in the next management unit. The search offset will then be reset to zero. We resume the search until we find suitable slots across the hyperperiod or fail to find a feasible solution. In the course of the scheduling process, the number of occupied slots rises and the number of free slots decreases. Still, solutions can be easily found since the network delay of critical flows is small due to high link speeds in time-sensitive networks. The described procedure is applied in the method *planNetworkTask* in Algorithm 2 (Section V-E).

Fig. 12 shows the management of resources and the ratio of the periods applied for the scheduling of processing and

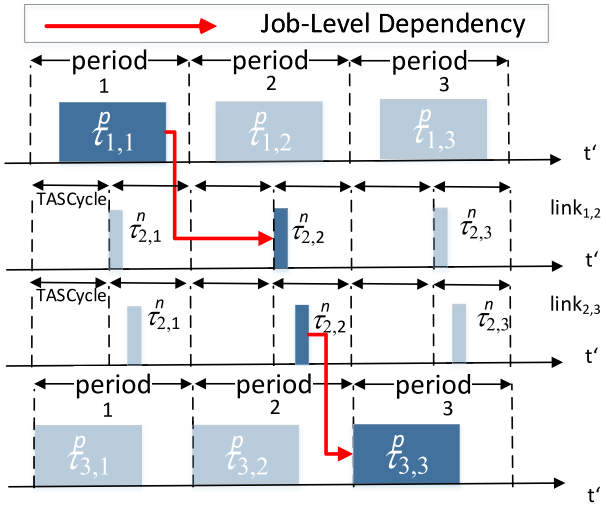


FIGURE 12. Example of a CEC schedule.

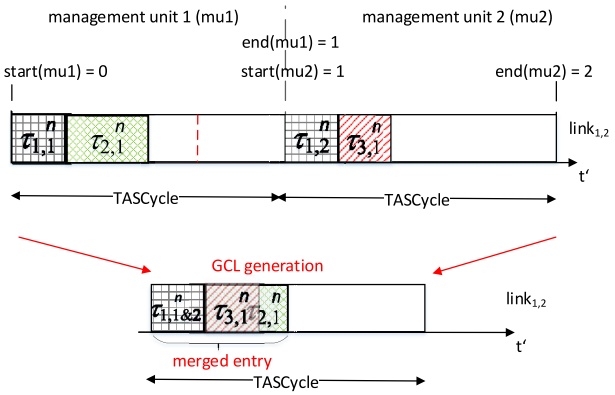


FIGURE 13. Example of a GCL merge.

network tasks. The assumed network hyperperiod and task period is *period*. *TASCycle* is *period*/2. The flow τ_2^n did not find fitting slots in the first management unit. Thus, the algorithm resumed the slot search in the second management unit and found suitable slots on the links $l_{1,2}$ and $l_{2,3}$.

D. GENERATION OF GATE CONTROL LISTS

The GCLs are generated after the scheduling algorithm has found a feasible solution. The TAS cycle in the GCLs corresponds to *TASCycle*. After one *TASCycle*, the GCL schedule is repeated in each TSN-capable egress port or queuing discipline. To construct the GCL, we collect all occupied slots associated with network tasks from all management units of the port and translate the start and end times of the slots to the range of *TASCycle* by applying the modulo operator. After sorting the translated times in ascending order, we create the GCL entries. First, we assume that each flow gets its own slot. However, we merge two slots if:

- two slots directly follow each other ($\tau_{1,1}^n$ and $\tau_{2,1}^n$ as well as $\tau_{1,2}^n$ and $\tau_{3,1}^n$ in Fig. 13),
- two slots overlap ($\tau_{2,1}^n$ and $\tau_{3,1}^n$ as well as $\tau_{1,1}^n$ and $\tau_{1,2}^n$ in Fig. 13),
- two adjacent slots are less apart than the transmission time of the smallest Ethernet frame on wire plus the interframe gap
- not enough entries are available

Planning with the smallest period reduces the number of entries in the GCL of a TSN-capable egress port. The maximum number of GCL entries is limited. Flows with a greater period than *TASCycle* can occupy one slot alternatively if they use different *TASCycles* or management units. However, if the alternation is not optimal, the whole slot or a part of the slot will be reserved in every *TASCycle*, even though it is not used. On the other side, if we constructed the GCLs using the hyperperiod and if the difference between the biggest and the smallest flow period in the system were large, the number of necessary entries in the GCL could exceed the number of maximum allowed entries in the hardware. Furthermore, after each critical slot, we have to create at least one GCL entry for less critical slots in addition. Moreover, the configuration would become less manageable due to many entries. Heterogeneous periods increase the hyperperiod and the complexity in network task planning even more. ITANS may not be able to choose the smallest period to be the *TASCycle* any more. Consequently, we only allow harmonic periods for network tasks since they have a greatest common divisor that is easy to determine and allow to plan within the smallest flow period to initially reduce GCL entries.

E. OVERALL SCHEDULING PROCESS

The proposed scheduling heuristic represents an incremental process that is dependent on the input sequence of CECs. The program flow is illustrated in Fig. 14. First, we calculate the *LST* values for all tasks. Then we sort the CECs by their mobility. The mobility of a chain cec_i is $e2e_i$ minus the highest sum of sequential execution times of all tasks $\in T_i$. Finally, we plan all tasks of the CECs on the corresponding resources in *planAllTasks* (see Section V-E1). If the method does not find a feasible solution we retry the scheduling by applying a repair and reordering function Section V-F. If the algorithm does not succeed in planning all CECs after fifty iterations, the overall scheduling process fails. Otherwise, if all CECs are planned, we return the GCLs of TAS-capable egress ports or queuing disciplines (when TAPRIO is used) and the start offsets of the processing and network tasks. Those can be used to configure the schedules in the end-device or switch hardware.

1) PLAN ALL TASKS

Algorithm 1 describes how the tasks of all CECs are scheduled. The first step of the method is the generation of a list of tasks that are plannable (line 1). The set contains tasks, whose *EST* and *LST* values have already been defined. This primarily accounts for all producer tasks in the first iteration. In other iterations the initial task set may contain intermediate or last tasks. The successor tasks will be appended to the task set during the schedule run. In the

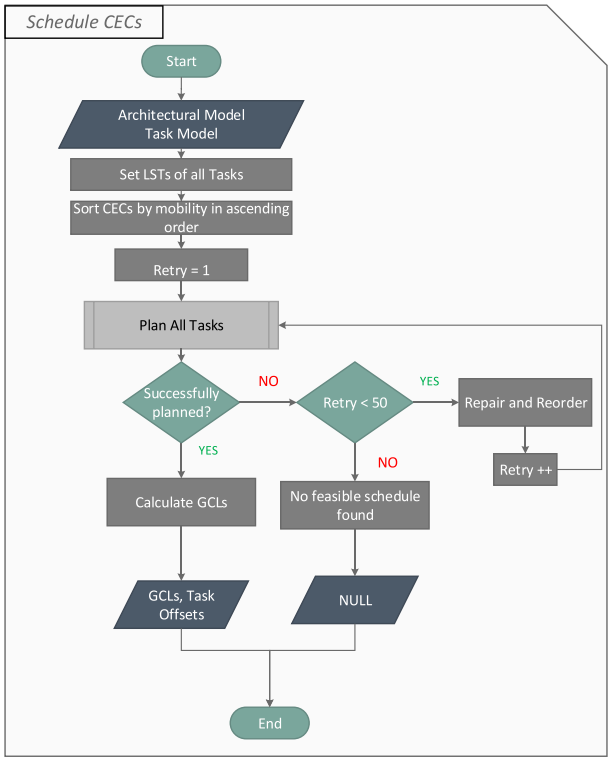


FIGURE 14. Overall program flow chart.

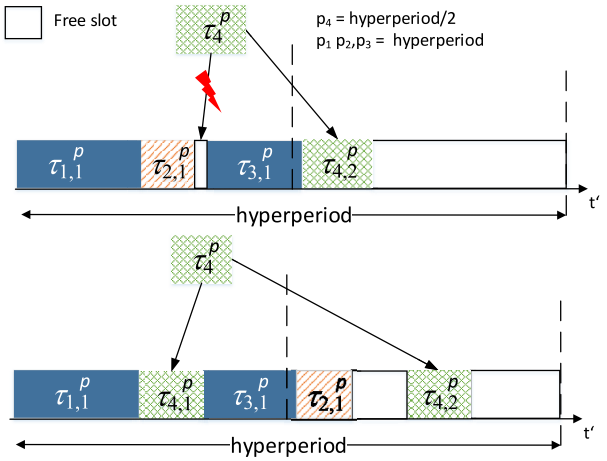


FIGURE 15. Scheduling tasks with small periods on a core.

Algorithm 1: planAllTasks($\mathcal{G}$, ζ)

```
1  taskSet ← createTaskSet(G, ζ);
2  sort(taskSet);
3  while taskSet is not empty do
4  |   τ_k^p ← getNextTask(taskSet);
5  |   success ← planProcessingTask(τ_k^p);
6  |   if success == false then
7  |   |   success ← reschedule(τ_k^p);
8  |   |   if success == false then
9  |   |   |   return -1;
10 end
11 return 0;
```

Algorithm 2: activateSuccessorTasks($\mathcal{G}$, τ_k^p)

```
1  foreach successor in next(τ_k^p) do
2  |   success ← setEST(successor);
3  |   if success == false then
4  |   |   return -1;
5  |   if successor is NetworkFlow then
6  |   |   success ← planNetworkTask(successor);
7  |   |   if success then
8  |   |   |   foreach s_successor in next(successor) do
9  |   |   |   |   success ← setEST(s_successor);
10 |   |   |   |   if success == false then
11 |   |   |   |   |   return -1;
12 |   |   |   |   else
13 |   |   |   |   |   appendToTaskSet(s_successor);
14 |   |   |   |   end
15 |   |   |   end
16 |   |   else
17 |   |   |   return -1;
18 |   |   end
19 |   else
20 |   |   appendToTaskSet(successor);
21 |   end
22 end
23 return 0;
```

second step, we sort the task set by task mobility and by the period. This approach leads to urgent tasks and tasks with a small period to be planned first. The smaller the priority the more often the task has to be planned throughout the hyperperiod of its assigned core. If tasks with high periods are planned first they could occupy so much space that a task with a smaller period will either experience much jitter or will not be able to find fitting slots. Fig. 15 shows that the task τ_4^p with the lowest period is scheduled after task τ_2^p. It does not find possible slots in the first scenario. When planning τ_4^p before τ_2^p we might get a feasible plan, as shown in the second scenario. After getting the first task from the task set (line 4) we try to plan it (line 5). The *planProcessingTask* function searches for suitable time slots throughout the hyperperiod for a given input task (see also Section V-B). If the process succeeds. i.e., if the tasks find enough slots and the *LST* value is respected, the successor tasks will be activated in the procedure *activateSuccessorTasks* (see Algorithm 2). If the task cannot be planned, the activation of successors fails, or if the *LST* is exceeded, as illustrated in Fig. 16, the task releases its slots and an infeasible planning process is reported by *planProcessingTask*. In this case a reschedule function as explained in Section V-F is applied Section V-F (line 7). If the reschedule trial does not find a feasible solution either the

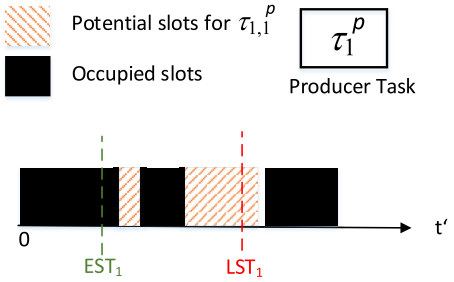


FIGURE 16. Job $\tau_{1,1}^{p}$ exceeds the *LST* of an activated task.

schedule iteration will be terminated and the *planAllTasks* function reports an impossible planning process.

Algorithm 2 describes how the successors of a planned task τ_k^p are configured and activated in the method *activateSuccessorTasks*. First, we determine the *EST* value for each successor within the job-level dependency (line 2). If the obtained *EST* value exceeds the *LST*, then a failure is reported. If the successor task of τ_k^p is a network flow (line 6), then we plan the flow in the method *planNetworkTask* (see Section V-C). The function *planNetworkTask* sets the transmission offset and the WCTT. After, we try to set the *EST* bounds of each successor of the flow and append them to the task set (line 9-14) if all their predecessor have already been scheduled.

If the successor of τ_k^p is not a flow, we will append it to the task set if possible (line 20).

F. REPAIR FUNCTIONS

Whenever a processing task or a network task could not be planned or the *EST* value exceeded the *LST* value, we have taken a closer look at the cause. A frequent source of failure was that a task with a higher period or WCET/WCTT was scheduled before a task with a lower value on the same core or egress port. Another common cause was that the planning according to the order of tasks within a CEC has not worked out. Thus, we identified different rescheduling strategies. In summary the countermeasures are:

- Exchanging the order of tasks on the core of the failed processing task
- Exchanging the order of flows on the busiest egress port of the failed network task
- Exchanging the order of tasks within the CECs of the failed tasks
- Permutation of the start order of the CECs
- Assigning another offset to the producer tasks of the CECs

Exchanging the order of tasks on a core or egress port is applied in Algorithm 1, line 7. It is done by gradually swapping the position of already scheduled tasks with the failed task, starting with the last planned task to avoid big changes in the existing plan. After a change in the order, ITANS checks if the modified tasks still meet their deadline. Only in this case, the iteration can be resumed. Otherwise, the iteration fails. The other countermeasures are applied after a whole iteration fails, see Fig. 14. Whenever a whole iteration is not successful the algorithm resets all scheduled tasks and resource occupancy. Then it identifies the conflicting tasks of the failed task. If the conflicting tasks are part of the same CEC we try to change the order of the tasks within the chain and to schedule all predecessor tasks of the failed task and the failed task itself. If this step works out, the succeeding tasks are placed in the initial task set, see Algorithm 1 line 1. The remaining order stays unchained. If the conflicting tasks are part of other CECs ITANS puts the CECs of the failed task before the conflicting CECs and begins a new iteration. In this case, the failed task is not scheduled yet before the next iteration, but the producer tasks of its CECs are scheduled before the producer tasks of the conflicting CECs. If the same task fails several times ITANS either randomly assigns start offsets to the producer tasks of the CECs or randomly permutates the order of the CECs. The experience showed that rescheduling within the core or egress port of the failed tasks and rescheduling within the CECs have solved the majority of schedule problems.

VI. CASE STUDY

In this section, we evaluate the proposed ITANS heuristic by alternating different parameters. We look at the success rate, runtime, and jitter development. Additionally, we compare ITANS with other approaches.

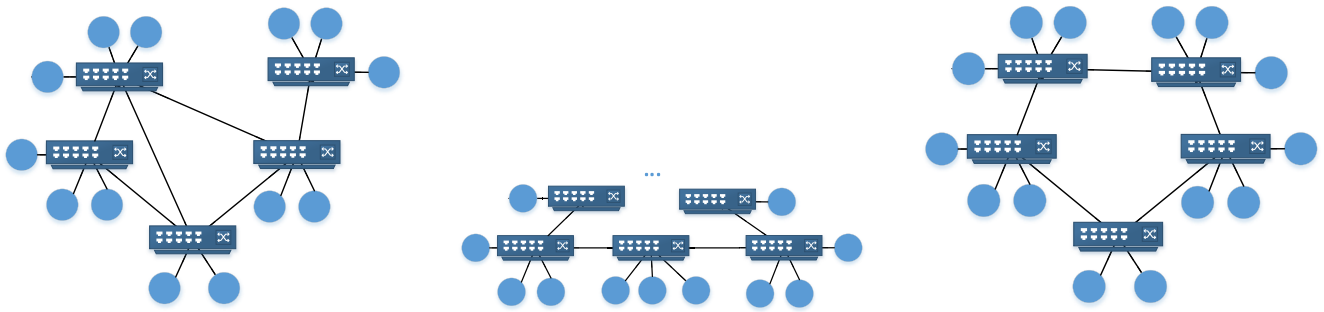
A. EXPERIMENTAL SETUP

The main emphasis of this case study is to evaluate the usability and performance of our approach. CEC with strict end-to-end delay constraints are especially common in the context of automated driving and assisted driving in the automotive context and in the railway domain for ATO. In these domains, the majority of data dependency paths turns out to be Sensor-to-Actuator paths [7]. This kind of CEC involves many complex steps as stated and detailed in [37]:

- sensing and fusing sensor data
- localization
- interpretation and generation of a world model
- trajectory calculation
- reactive control
- diagnosis and fault management
- trajectory execution
- platform stabilization

Based on this use case and with the information taken from real world automotive benchmarks [33] we have carried out different case studies with synthetic tasks and network topologies varying the following parameters:

- Core utilization $\in \{0.5, 0.6, 0.65\}$
- Number of CECs $\in \{10, 30, 50, 100, 120, 150\}$
- Number of processing tasks within a CEC $\in \{2, .., 10\}$
- Number of network tasks within a CEC $\in \{1, .., 5\}$
- Harmonic and non-harmonic processing task periods
- Harmonic network task periods



(a) Mesh topology. (b) Hierarchical topology. (c) Ring topology.

FIGURE 17. Considered network topologies.

TABLE 1. Description of experiments.

Experiment	Description	Chosen periods
MeshNH	Mesh topology	Non-harmonic processing periods $\in$ {10,20,25,50,100} ms Harmonic network periods $\in$ {5,10,20} ms
RingNH	Ring topology	Non-harmonic processing periods $\in$ {10,20,25,50,100} ms Harmonic network periods $\in$ {5,10,20} ms
HierNH	Hierarchical topology	Non-harmonic processing periods $\in$ {10,20,25,50,100} ms Harmonic network periods $\in$ {5,10,20} ms
MeshH	Mesh topology	Harmonic processing periods $\in$ {2,4,8,16,32} ms Harmonic network periods $\in$ {2,4,8,16} ms
RingH	Ring topology	Harmonic processing periods $\in$ {2,4,8,16,32} ms Harmonic network periods $\in$ {2,4,8,16} ms
HierH	Hierarchical topology	Harmonic processing periods $\in$ {2,4,8,16,32} ms Harmonic network periods $\in$ {2,4,8,16} ms

- Number of switches in the network $\in$ {5, 10, 20, 30}
- Link speed of egress ports $\in$ {100*Mbps*, 1*Gbps*}
- Network topologies $\in$ {Mesh, Ring, Hierarchical}

The number of end-devices depends on the utilization. The number of processing cores was randomly chosen between one and two cores per end-device, where predominantly one core was selected to force the distribution of tasks across the system. Each task was assigned to one processing core in a manner to comply to the average regarded utilization and to create an overall rate of 25-30% of network tasks proportionally to the overall number of all tasks. The assumed WCET of a processing task was less than or equal 1%-10% of the task period. The heuristic was implemented and tested on a Core i7-8550U CPU, 1.88 GHz processor with 24GB RAM. 1 gigabit per (Gbps) second was selected preferentially over 100 megabit per (Mbps). The network flows periods were selected to be the greatest common divisors of its predecessor and successors or the smallest available network period to not introduce additional overhead.

TABLE 2. Average number of tasks over all experiments.

# CECs	# proc. tasks	# net. tasks	Total number of tasks
5	27	18	45
10	60	32	92
30	120	50	170
50	280	108	388
100	595	228	823
120	655	198	853
150	825	189	1014

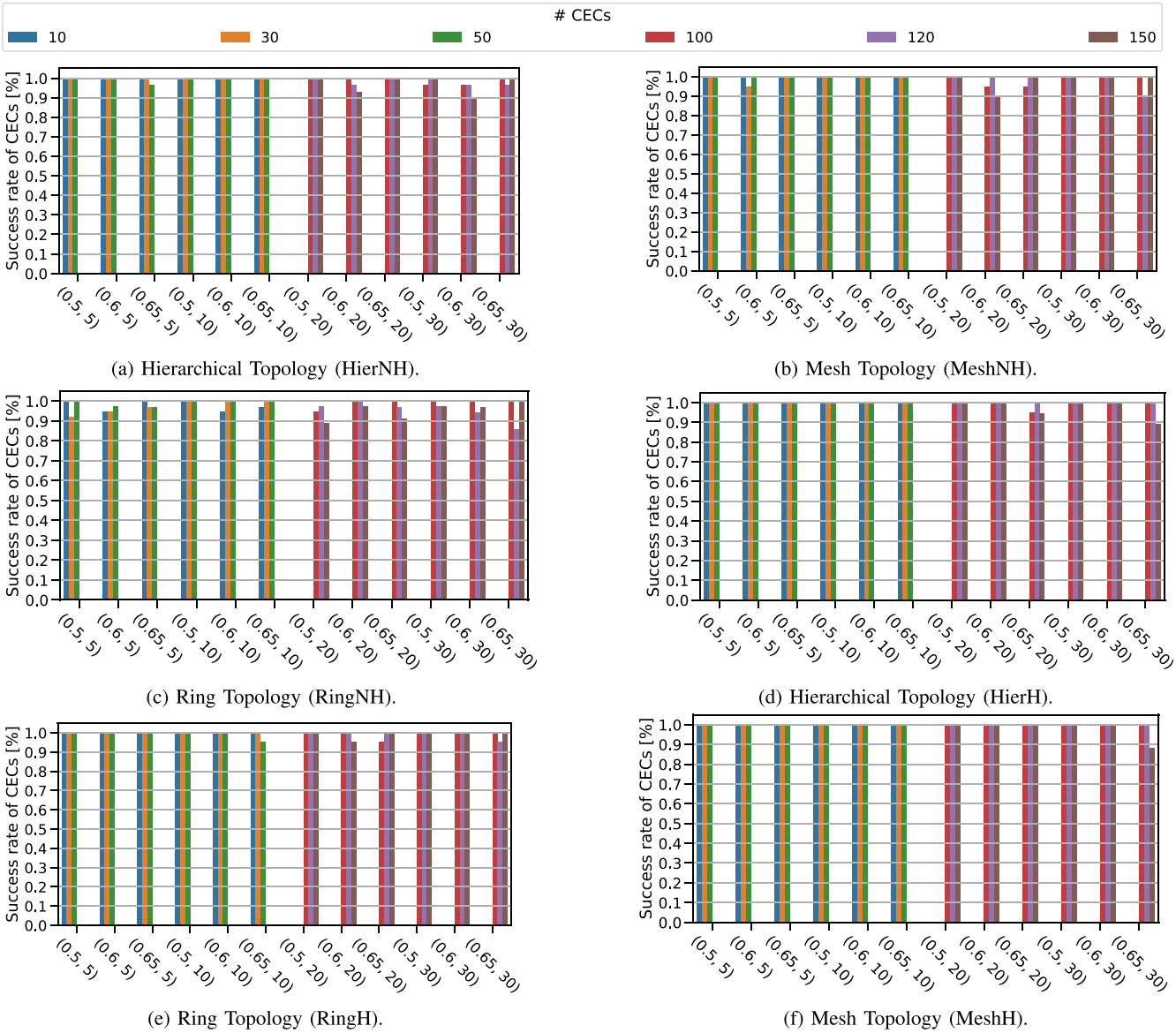
As described in Table 1, we carry out 4 major experiments varying the parameters mentioned previously. Each experiment was divided into a small- to middle-sized (S) and a large system (L):

- S: {5,10} switches, {10,30,50} CECs
- L: {20,30} switches, {100,120,150} CECs

We execute the most experiments with non-harmonic periods. Nevertheless, we make a comparison with a task set with harmonic periods as well. As stated in [33], 70 % of engine control applications use homogeneous and harmonic task periods. The mesh network (Fig. 17(a)) is a random interconnected network topology, but not necessarily fully-meshed. The ring topology (Fig. 17(c)) defines a ring of switches and interconnects one or several end-devices. The hierarchical network (Fig. 17(b)) interconnects different subtopologies. For example: a line topology with a link to upper floors, as can be found in industrial automation, or an extended star topology. The path length for the flows was between one hop and (number of switches)/2 for the ring and hierarchical topology. In contrast, the number of hops of the network tasks in the mesh topology was only a few hops. The end-to-end deadlines were calculated as proposed in [33].

B. RESULTS

In the first step, we have determined the success rate for a different number of CECs in dependence of the processor utilization, number of switches, and the topology. The success rate is defined as $success\ rate = \frac{\#successful\ runs}{\#successful\ runs\ +\ \#failed\ runs}$.



(a) Hierarchical Topology (HierNH).

(b) Mesh Topology (MeshNH).

(c) Ring Topology (RingNH).

(d) Hierarchical Topology (HierH).

(e) Ring Topology (RingH).

(f) Mesh Topology (MeshH).

FIGURE 18. Success rate for a different number of CECs in dependence of the tuple (utilization, number of switches).

The failure rate is $1 - \textit{success rate}$ accordingly. Fig. 18 illustrates the success rate in different network topologies for non-harmonic CECs and harmonic CECs. In summary, we can observe that ITANS provides a high success rate for randomized synthetic topologies and tasks and in particular for a utilization rate of 50 %. If we take a closer look, we can state that ITANS performs better for harmonic CECs. We can also see that the failure rate is particularly higher for a higher number of CECs and a higher utilization. A higher number of CECs raises the complexity. The number of possible scheduling orders increases exponentially with the number of tasks. Furthermore, we can see that the ring topology shows a higher failure rate than other topologies. The network schedule can have a higher impact in a ring topology. Certain ports may have a higher traffic load than others. Here, the optimization of network task schedule can be more important than processing task schedule. Especially, if the ports have a lower link speed. In this case, the WCTT might even be higher than the WCET of the tasks. On long flow paths and for maximum sized Ethernet frames, the WCTT can have a size of over 1 ms. This is a multiple of common processing task WCETs [33]. This has to be taken into consideration in the calculation of end-to-end deadlines.

TABLE 3. Minimum | average | maximum runtime in dependence of CECs and switches for (S) system.

# CECs	5 switches			10 switches		
10	0.04s	0.24s	8.53s	0.04a	0.29a	6.43s
30	0.23s	1.00s	18.63s	0.25s	0.48s	8.9s
50	0.25s	0.69s	21.23s	0.29s	0.47s	0.76s

We illustrate the runtime of the ITANS heuristic in Table 4. Therefore, we have recorded the minimum, average, and maximum execution time of successful runs with different

TABLE 4. Minimum | average | maximum runtime in dependence of CECs and switches for (L) system.

# CECs	20 switches			30 switches		
100	0.51s	1.05s	19.25s	0.57s	0.89s	1.69s
120	0.50s	2.03s	113.8s	0.69s	1.58s	24.85s
150	0.67s	2.85s	35.46s	0.78s	4.11s	105.40s

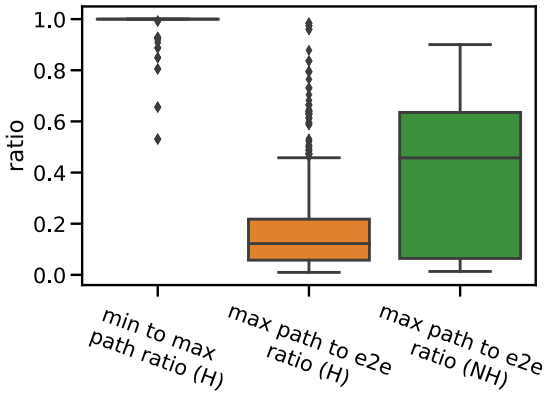


FIGURE 19. Path ratios.

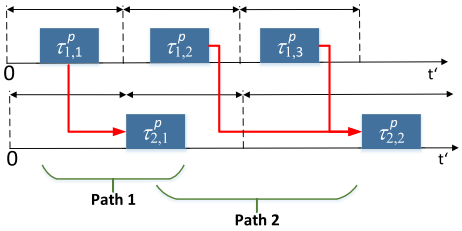


FIGURE 20. Inevitable difference of job-level path lengths.

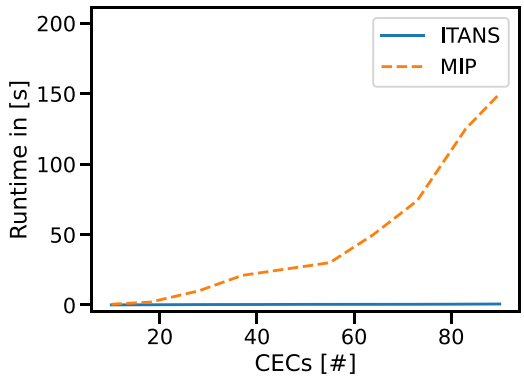


FIGURE 21. Runtime of ITANS compared to MIP.

utilizations varying the number of CECs and the number of switches in the system. The most influencing factor is the number of CECs. The average runtime increases with the number of CECs. We are able to find feasible solutions for more than 1000 network and processing tasks in under 5 s on average. In the case of maximum values, we can detect outliers like 120 CECs and 20 switches or 150 CECs and 30 switches. Those outliers especially arise in combination with a higher utilization and a higher number of CECs. We can also see that the executions in topologies with 5 switches entail a higher maximum execution time. This is especially caused when an egress ports or a sequence of egress ports transmits many network tasks. A feasible order of network tasks has to be found. We could observe up to 40 flows per port (with 1 Gbps) in certain scenarios. In case of 100 Mbps egress ports, the complexity of finding a working sequence of flows increases since network flows occupy ten times more resources than in 1 Gbps egress ports.

Jitter is an important factor, particularly in motion control. We have investigated the jitter of end-to-end delays of the CECs for homogeneous CECs during our experiments. Therefore, we have compared the smallest and the biggest path length as $pathRatio = \frac{smallest\ path\ duration}{longest\ path\ duration}$. The higher the *pathRatio* value, the lower the jitter. Fig. 19 illustrates the path ratio for homogeneous CECs. We can provide very low jitter for the majority of the experiments. Regarding the jitter for harmonic or heterogeneous CECs, we can state that the path lengths will always vary if any task has a higher period than the first task in the chain, see Fig. 20. This jitter is inevitable. In our approach, the jitter is predictable in the most cases, due to static priority-based scheduling and no-wait task scheduling.

Furthermore, we have analyzed the ratio of the longest path to the end-to-end deadline of a CEC as $maxPathRatio = \frac{longest\ path\ duration}{end-to-end\ deadline}$. Fig. 19 shows the *maxPathRatio* for homogeneous (H) and non-homogeneous (NH) periods. On average, we are well below the end-to-end deadline. This ratio also indicates that the average system response time is low compared to the maximum end-to-end deadline. Non-harmonic and harmonic CECs logically show a higher ratio than harmonic due to period transitions.

C. COMPARISON TO OTHER APPROACHES

A Mixed Integer Programming (MIP) approach is presented in [10] that describes a task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems. It is able to calculate optimized solutions regarding multiple objects link maximal response time of all applications. On the one hand, we have reproduced the case study to analyze the maximum response time. In a tree network topology with homogeneous 30 CECs and maximum 5 tasks per chain, the authors achieve a maximal end-to-end latency of all applications of 1740.72 µs. Our algorithm is able to provide a maximum response time of 2950.00 µs.

This result of ITANS satisfies all deadlines and is obtained in 0.41 s in comparison to several seconds of the MIP approach. Although the runtime of the case study is not explicitly mentioned, we can deduce the runtime from the scalability analysis of the work. Furthermore, we are able achieve low jitter. Comparing the runtime development depending on number of CECs and topology size, we can illustrate that we outrun the MIP approach, as presented in Fig. 21. The MIP approach develops an exponential course. ITANS, in contrast, performs under one second.

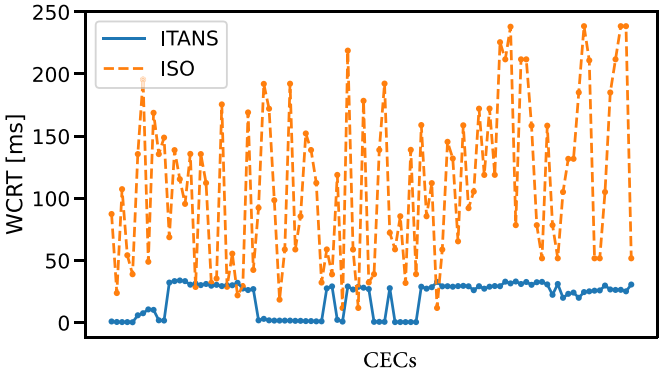


FIGURE 22. Comparison of end-to-end response times of ITANS and an isochronous scheduling.

Furthermore, we want to compare ITANS to isochronous scheduling that is often associated for TAS [38] and application scheduling in time-sensitive applications like motion control, as also stated in IEC/IEEE 60802 TSN profile for Industrial Automation (TSN-IA) [39]. A common application data cycle is defined in which a fixed share of the cycle is dedicated for the execution of the processing tasks and the other for the transfer time of critical network traffic. For example: the application data cycle is defined as the greatest common divisor of all processing and network tasks. The first 60% are dedicated for task execution and the other 40% for network. The applications are not executed during the network share and other and also vice versa. Thus, the processing and network tasks are coordinated. We have compared this common scheduling paradigm to the ITANS heuristic for harmonic CECs that are conventional in this domain. Therefore, we have defined 100 CECs with different periods {2,4,8,16,32} and have scheduled them using ITANS and an optimized isochronous scheduling with an application data cycle of 2 ms. The result is, that the isochronous scheduling provides predictable and fixed time slots for processing and network tasks, but it introduces a large delay to the end-to-end semantic as shown in Fig. 22. It is however easier to implement. Nevertheless, ITANS is also able to provide lower jitter for homogeneous CECs and is suitable for this kind of application.

D. SCHEDULE PLAUSIBILITY

To check the plausibility of the resulting schedules, we have simulated the runs and compared them to the resulting schedules. Therefore, we have performed the following steps:

- Examination of the schedules of each task chain and its deadline
- Evaluation of each flow schedule from the talker node to each listener
- Investigation of the complete schedule on each traversed network link
- Validation of the schedules on each processing core

Additionally, we were able to import our synthetic topologies and network task descriptions into the simulation environment in OMNeT++ and were able to verify our flow schedules with the help of the INET and NeSTiNg [40] library. The plausibility checks have shown that the simulated schedules have matched with the calculated schedules in all runs.

VII. RELATED WORK

Scheduling of CECs combines different research areas. We want to address the related work that deals with the scheduling of TSN flows applying the TAS mechanism, joint task and network planning, and CECs. In the field of flow planning in time-sensitive networks, different scheduling approaches have arisen. Craciunas *et al.* [41] formulate scheduling constraints for 802.1Qbv-compliant networks. They solve the problem of scheduling of up to 1000 TSN flows with predetermined paths by using Satisfiability Modulo Theories (SMT) and Optimization Modulo Theories (OMT). The approaches presented in [21], [32] apply genetic algorithms to schedule time-triggered traffic in time-sensitive networks. Dürr and Nayak [20] present a mapping of the Job Shop Scheduling Problem (JSP) to the planning of TSN traffic. They create flow schedules by integrating the no-wait method and the *Tabu Search* principle. In [42], a window-based approach that determines the GCL entries is presented. The authors embed a worst-case delay analysis to guarantee upper delay bounds instead of using strict temporal isolation. These works primarily focus on the network scheduling. As we combine the task and network schedule, we employ an incremental first-fit strategy to satisfy both, the processing and the network tasks. The number of regarded network flows is about 200. In comparison to the discussed works, we show a lower runtime for the same amount of network flows plus additional processing tasks but do not further optimize the flow schedules.

Regarding combined task and network scheduling strategies, Zhang *et al.* [10] propose a MIP-based approach for non-preemptive periodic applications and network tasks. They define the objectives to optimize the end-to-end task chain latencies and response times of applications. They are able to find optimal solutions for small until middle-sized industrial use cases with up to 90 task chains consisting of 2 processing and one network task in less than 150 s. As already discussed, we are able to outrun the execution times of the MIP approach still providing low response times and jitter. The SMT-based approach presented in [11] solves the combined task and network scheduling problem for middle- to large-sized industrial use cases. The runtime of the SMT-based variant varies from a few milliseconds to several hours. The authors focus on the earliest deadline first (EDF) scheduling approach for preemptive processing tasks. They formulate network, task, and memory constraints. The approach was evaluated in terms of runtime and scalability that increases exponentially with the system size. ITANS can keep up with the regarded small, middle, and large system

sizes and scales well. Besides, [10], [11] focus on time-triggered Ethernet networks using protocols like TTEthernet. We put the emphasis to TSN-based communication networks integrating the 802.1Qbv mechanisms.

The CEC topic is also well researched. Kramer *et al.* [33] provide automotive benchmarks taking typical CEC characteristics into consideration. They present among others common task periods and execution times of tasks in the automotive field. Becker *et al.* [9] introduce an end-to-end response time analysis for CECs in automotive embedded systems. They focus on the computation of the maximum data age of CECs. The ITANS heuristic works the other way around. We try to schedule the tasks of the CECs so that we comply to given end-to-end times. Schlatow *et al.* [7] propose a Mixed Integer Linear Program (MILP)-based optimization for the data age of CECs comprising preemptive processing tasks. They determine the phase offset, priority, and processor mapping of processing tasks minimizing the data age. Network tasks are not considered in this work. The authors show that their approach is applicable to real work automotive use cases such as Advanced Driver Assistance Systems (ADAS) and engine control.

Even though ITANS does not focus on optimal solutions, it is able solve complex and large scheduling problems in a short time. Additionally, ITANS is tailored to consider the characteristics of complex CEC and TAS. Consequently, ITANS addresses real-world and future-related scenarios. Nevertheless, it must be mentioned that heuristic approaches do not always find a solution. But in this case, heuristics can be backed up with exact algorithms or meta-heuristics.

VIII. DISCUSSION AND CONCLUSION

We have shown that it is possible to calculate feasible schedules for time-triggered processing and network tasks with different requirements. The methods were chosen to satisfy both. Other approaches are also plausible. The mentioned isochronous scheduling approach fits under this category. Moreover, it is, e.g., conceivable that first all processing tasks of the CECs are planned and then a feasible schedule for the network has to be found. However, in this case the release times of succeeding processing tasks on different end-devices have to be estimated as the intermediate network task will be scheduled later. Moreover, this approach still requires a worst-case estimation of network delays, e.g., through analytical methods like *network calculus*, to determine the earliest start time of the succeeding processing tasks on different end-devices. If the network delay is underestimated data will be delivered to a later execution of the succeeding processing task and data age will rise. It is especially difficult to meet the end-to-end deadlines of the CECs when regarding both separately. An incremental and combined approach allows to better synchronize the processing and network tasks and to find less pessimistic bounds for network latency and jitter when applying the TAS mechanism.

The ITANS approach involves the use of static priorities for processing tasks. Dynamic priorities, as, e.g., used for scheduling policies like EDF, might reorder already scheduled tasks in our approach. This can lead to the tasks no longer being coordinated with other tasks in the task chain and to missing the end-to-end deadline. Moreover, the proposed priority-based scheduling promotes a similar execution of the tasks throughout the hyperperiod.This in turn helps to reduce the jitter on different job-level paths.

EDF and RM scheduling can achieve 100 % core utilization when using harmonic tasks [43]. Yet, it is difficult to achieve this utilization for CECs, especially when release offsets and tight end-to-end deadlines are involved. Therefore, we account for some slack times.

We allow multi-rate CECs. In practice, 70 % of engine control applications as can be found in the automotive, industrial automation, and railway domain use homogeneous and harmonic periods [33]. Heterogeneous and harmonic periods among CECs can lead to unnecessary task execution [44] due to over- or undersampling. Since ITANS overall performs best for homogeneous CECs, our algorithm proves to be well-suited and applicable.

Nonetheless, we have to annotate that we neglect the consideration of memory resources in our approach. This concerns memory that is necessary for the execution of processing tasks and the buffers for network traffic. Among others, there are buffer restrictions in the NICs of end-devices and in the traffic control module, if it is available in the OS. The difference between the best-case execution time (BCET) and WCET of processing tasks leads to network packets arriving at traffic control modules or NICs earlier than the computed and configured forwarding time. Thus, early packets have to be buffered for some time. Also when network packets arrive at network switches and before they are processed and forwarded to the proper egress ports, they have to be buffered. Depending on the number of ingress ports this can be more or less critical. The memory consumption due to queuing in egress ports is in our case not of interest as we avoid queuing delays.

We do not elaborate on the problem of shared or exclusive resources for processing tasks that is referenced extensively in [30]. We leave the analysis to system or application designers. Nevertheless, we can integrate those times in the WCRTs. Also, the clock drift between clocks of different devices was not considered. Synchronized clocks are necessary for the TAS mechanisms and the CEC task sequence. We have assumed ideal clocks. The PTP protocol and adaptations of it are common time synchronization protocols for Ethernet-based systems that promise a maximum deviation in the sub-microsecond range between clocks. This deviation can be considered in the proposed formulas.

In summary, we presented a heuristic approach for the joint task and network scheduling covering common problems in a variety of complex application domains. Furthermore, we have shown that our algorithm is capable to find feasible solutions for middle- to large-sized problems with a high success rate and low end-to-end jitter in just a few seconds. The definition of earliest and latest start times

allows a more efficient exploration of the solution space and provides a quick assessment of the feasibility of the certain schedule. In the future, we plan to extend ITANS to consider the maximum data age. Moreover, we intend to elaborate the algorithm by including meta-heuristics like genetic algorithms to determine a more optimal input order of CECs. Finally, we plan to involve further scheduling mechanisms besides TAS and embed network flow analysis in ITANS.

REFERENCES

[1] C. Mannweiler *et al.*, "Reliable and deterministic mobile communications for industry 4.0: Key challenges and solutions for the integration of the 3GPP 5G system with IEEE," in *Proc. Mobile Commun. Technol. Appl. ITG-Symp.*, 2019, pp. 1–6.

[2] L. Silva, P. Pedreiras, P. Fonseca, and L. Almeida, "On the adequacy of SDN and TSN for industry 4.0," in *Proc. IEEE 22nd Int. Symp. Real-Time Distrib. Comput. (ISORC)*, 2019, pp. 43–51.

[3] Z. Wang, Y. Wu, and Q. Niu, "Multi-sensor fusion in automated driving: A survey," *IEEE Access*, vol. 8, pp. 2847–2868, 2020.

[4] R. Lagay and G. M. Adell, "The autonomous train: A game changer for the railways industry," in *Proc. 16th Int. Conf. Intell. Transp. Syst. Telecommun. (ITST)*, 2018, pp. 1–5.

[5] M. Dürr, G. Von Der Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5S, p. 58, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3358181

[6] N. Feiertag, K. Richter, J. E. Nordlander, and J. Å. Jönsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," presented at the Workshop RTSS, Barcelona, Spain, 2008.

[7] J. Schlatow, M. Mostl, S. Tobuschat, T. Ishigooka, and R. Ernst, "Data-age analysis and optimisation for cause-effect chains in automotive control systems," in *Proc. IEEE 13th Int. Symp. Ind. Embedded Syst. (SIES)*, 2018, pp. 1–9.

[8] "Time-Sensitive Networking (TSN) Task Group." [Online]. Available: https://1.ieee802.org/tsn/ (Accessed: Mar. 16, 2022).

[9] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *J. Syst. Archit.*, vol. 80, pp. 104–113, Oct. 2017.

[10] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task- and network-level schedule co-synthesis of ethernet-based time-triggered systems," in *Proc. Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2014, pp. 119–124.

[11] S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Syst.*, vol. 52, pp. 161–200, Mar. 2016.

[12] N. Finn, "Introduction to time-sensitive networking," *IEEE Commun. Stand. Mag.*, vol. 2, no. 2, pp. 22–28, Jun. 2018, doi: 10.1109/MCOMSTD.2018.1700076.

[13] J. L. Messenger, "Time-sensitive networking: An introduction," *IEEE Commun. Stand. Mag.*, vol. 2, no. 2, pp. 29–33, Jun. 2018.

[14] A. Ademaj *et al.*, "Time sensitive networks for flexible manufacturing Testbed—Description of converged traffic types," IIC Consortium, Boston, MA, USA, Rep. IIC:WHT:IS3:V1.0:PB:20180418, 2018. Accessed: Mar. 19, 2022. [Online]. Available: https://www.iiconsortium.org/pdf/IIC_TSN_Testbed_Traffic_Whitepaper_20180418.pdf

[15] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 26: Frame Preemption*, IEEE Standard 802.1Qbu-2016, Aug. 2016.

[16] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 29: Cyclic Queuing and Forwarding*, IEEE Standard 802.1Qch-2017, 2017.

[17] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 34: Asynchronous Traffic Shaping*, IEEE Standard 802.1Qcr-2020, 2020.

[18] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic*, IEEE Standard 802.1Qbv-2015, Mar. 2016.

[19] A. Arestova, M. Martin, K.-S. J. Hielscher, and R. German, "A service-oriented real-time communication scheme for AUTOSAR adaptive using OPC UA and time-sensitive networking," *Sensors*, vol. 21, no. 7, p. 2337, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/7/2337

[20] F. Dürr and N. G. Nayak, "No-wait packet scheduling for ieee time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 203–212.

[21] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2018, pp. 337–344.

[22] *IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications*, IEEE Standard 802.1AS-2020, 2020.

[23] "TC(8)—Linux Manual Page." [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html (Accessed: Sep. 16, 2021).

[24] "TC-ETF(8)—Linux Manual Page." [Online]. Available: https://man7.org/linux/man-pages/man8/tc-etf.8.html (Accessed: Aug. 16, 2021).

[25] "TC-MQPRIO(8)—Linux Manual Page." [Online]. Available: https://man7.org/linux/man-pages/man8/tc-mqprio.8.html (Accessed: Aug. 16, 2021).

[26] "TC-TAPRIO(8)—Linux Manual Page." [Online]. Available: https://man7.org/linux/man-pages/man8/tc-taprio.8.html (Accessed: Aug. 16, 2021).

[27] "Scheduled Packet Transmission: ETF." [Online]. Available: https://lwn.net/Articles/758592/ (Accessed: Sep. 16, 2021).

[28] C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. M. Vilches, "Real-time linux communications: An evaluation of the Linux communication stack for real-time robotic applications," 2018, *arXiv:1808.10821*.

[29] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *Proc. IEEE 22nd Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, 2016, pp. 159–169.

[30] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications* (Real-Time Systems Series 24), 3rd ed. Cham, Switzerland: Springer, 2011.

[31] A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," *Eur. J. Oper. Res.*, vol. 143, no. 3, pp. 498–517, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221701003381

[32] A. Arestova, K.-S. J. Hielscher, and R. German, "Design of a hybrid genetic algorithm for time-sensitive networking," in *Measurement, Modelling and Evaluation of Computing Systems*. Cham, Switzerland: Springer Int., Mar. 2020, pp. 99–117.

[33] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *Proc. 6th Int. Workshop Anal. Tools Methodol. Embedded Real-Time Syst.*, 2015. [Online]. Available: http://waters2015.inria.fr/

[34] A. Arestova, K.-S. J. Hielscher, and R. German, "Simulative evaluation of the TSN mechanisms time-aware shaper and frame preemption and their suitability for industrial use cases," in *Proc. IFIP Netw. Conf. (IFIP Netw.)*, 2021, pp. 1–6.

[35] *Industrial Communication Networks—Profiles—Part 2: Additional Fieldbus Profiles for Realtime Networks Based on ISO/IEC/IEEE 88023*, IEC Standard 61784-2:2019, Jul. 2020.

[36] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered Ethernet (TTE) design," in *Proc. 8th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput. (ISORC)*, May 2005, pp. 22–33.

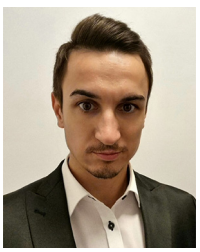
[37] S. Behere and M. Törngren, "A functional reference architecture for autonomous driving," *Inf. Softw. Technol.*, vol. 73, pp. 136–150, May 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584915002177

[38] M. Kim, D. Hyeon, and J. Paek, "eTAS: Enhanced time-aware shaper for supporting non-isochronous emergency traffic in time-sensitive networks," *IEEE Internet Things J.*, early access, Nov. 13, 2021, doi: 10.1109/JIOT.2021.3124508.

[39] "IEC/IEEE 60802 TSN Profile for Industrial Automation (D1.2)." [Online]. Available: https://www.ieee802.org/1/files/private/60802-drafts/d1/60802-d1-2.pdf (Accessed: Mar. 20, 2022).

[40] J. Falk *et al.*, "NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++," in *Proc. Int. Conf. Netw. Syst. (NetSys)*, 2019, pp. 1–8.
[41] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 183–192.
[42] N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Window-based schedule synthesis for industrial IEEE 802.1Qbv TSN networks," in *Proc. 16th IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, 2020, pp. 1–4.
[43] M. Mohaqeqi, M. Nasri, Y. Xu, A. Cervin, and K. Årzén, "Optimal harmonic period assignment: Complexity results and approximation algorithms," *Real-Time Syst.*, vol. 54, pp. 830–860, Apr. 2018.
[44] E. Farcas, C. Farcas, W. Pree, and J. Templ, "Transparent distribution of real-time components based on logical execution time," *ACM SIGPLAN Not.*, vol. 40, no. 7, pp. 31–39, Jul. 2005.

ANNA ARESTOVA received the bachelor's and M.Sc. degrees in 2018 in information and communication technology from the University of Erlangen–Nürnberg, Germany, where she is currently pursuing the Ph.D. degree with the Department of Computer Science. Her research interests are time-sensitive networking, modeling and simulation of communication networks, as well as real-time scheduling.



WOJCIECH BARON received the B.Sc. and M.Sc. degrees in information and communication technology from the University of Erlangen–Nürnberg, Germany, in 2015 and 2018, respectively, where he is currently pursuing the Ph.D. degree with the Department of Computer Science. He collaborates with AUDI AG in diverse projects that evaluate synchronization mechanisms and real-time capabilities in distributed simulation systems in the context of automated driving.

KAI-STEFFEN J. HIELSCHER was born in Münchberg, Germany, in 1972. He received the Ph.D. degree in computer science from the University of Erlangen–Nürnberg in 2008, where he is currently working as a Postdoctoral Researcher with the Department of Computer Science (Computer Networks and Communication Systems). His focus of research includes measurement, modeling and simulation of distributed systems as well as deterministic and stochastic Network Calculus.

REINHARD GERMAN received the master's degree in computer science and the Ph.D. degree from the Computer Science Department, Technical University of Berlin, Germany, in 1991 and 1994, respectively. He is a Full Professor with the Computer Networks Lab, Department of Computer Science, University Erlangen–Nürnberg, Germany. He is also an Adjunct Professor with the Faculty of Information Technology, Monash University, Melbourne, Australia. His research interests include performance and dependability analysis of interconnected systems based on numerical analysis, network calculus, discrete-event simulation, measurements, and testing. Vehicular communications and smart energy constitute major application domains.