IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 August 2023, accepted 2 October 2023, date of publication 18 October 2023, date of current version 23 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3324957

RESEARCH ARTICLE

Optimization of Bandwidth Utilization and Gate Control List Configuration in 802.1Qbv Networks

ANNA ARESTOVA, KAI-STEFFEN J. HIELSCHER, AND REINHARD GERMAN
Computer Science 7, Computer Networks and Communication Systems, Friedrich-Alexander-Universität Erlangen-Nürnberg, 91058 Erlangen, Germany

Corresponding author: Anna Arestova (anna.arestova@fau.de)

This work was supported by the Federal Ministry of Economic Affairs and Climate Action through the MBPLE4Mobility Project.

ABSTRACT The IEEE 802.1 Time-Sensitive Networking (TSN) Task Group has been developing various standards since 2012 to provide deterministic and time-critical services through IEEE 802 networks. The Time-Aware Shaper (TAS) is a key component in TSN that was introduced to enable precise forwarding and shaping of time-sensitive network traffic. It allows configuring exclusive transmission slots for selected traffic classes. To achieve the best benefit from the TAS, numerous scheduling algorithms have been presented to find schedules for the transmission of time-critical network streams optimizing the stream end-to-end transmission times in TAS-based networks. However, most existing works do not address how the scheduling of critical network traffic affects the necessary number of critical transmission slots that is limited in hardware and the resulting bandwidth utilization for other traffic classes. In this paper, we have investigated how a selected set of heuristic and meta-heuristic algorithms for scheduling critical network streams in TAS-based networks affects the utilization of network resources, and how the configuration of TAS can be improved using a compression algorithm. Therefore, we make use of existing algorithms and propose alternatives that have an influence on input data ordering, scheduling procedures, and the TAS configuration. The evaluation showed that some of our proposed modifications can improve bandwidth utilization and reduce the number of transmission slots for critical traffic significantly. Further on, the compression algorithm can bring a remarkable improvement in isolated cases.

INDEX TERMS TSN, real-time, scheduling, ethernet networks.

I. INTRODUCTION

The introduction of Time-Sensitive Networking (TSN) [1] reveals the possibility of using Ethernet technology in real-time systems. It is particularly attractive because, on the one hand, the coexistence of critical and non-critical traffic in the same network is allowed and, on the other hand, vendor dependency can be avoided, but the interoperability between different vendors and devices can be established. Especially in relation to time-critical network traffic (TSN streams), it features various mechanisms to provide Quality of Service (QoS), in particular, latency guarantee and low jitter [2]. One of the most popular TSN shapers in research is the Time-Aware Shaper (TAS) introduced in [3]. It works on the principle of allocation of time slots to traffic classes.

The associate editor coordinating the review of this manuscript and approving it for publication was Frederico Guimarães.

However, this mechanism relies on a proper configuration that is not supported by the standard itself and is strongly dependent on the amount of critical network traffic and the network infrastructure. An inappropriate configuration can lead to even worse latency, jitter, and bandwidth utilization, or even to deadline misses that might have serious consequences foremost when safety-critical applications are involved. Innovations like autonomous driving come with a high number of safety-critical applications, numerous sensors and actuators, and consequently a higher load of safety-critical network traffic [4].

Several papers have addressed scheduling time-sensitive network traffic in conjunction with the configurations of time-critical slots in the TAS. However, the majority has focused on the adherence to the end-to-end delays of individual network streams and has not taken the difficulties and limitations of the TAS configuration into account. On the one hand, the number of configurable slots is limited by the hardware, which is often omitted and might invalidate the proposed algorithms. On the other hand, unusable gaps can occur between configured slots, resulting in wasted bandwidth. Since TSN promotes mixed-criticality, allowing critical and non-critical network traffic to coexist in the same network, the examination of wasted bandwidth and resource utilization of critical network traffic becomes relevant, particularly for less critical traffic. In transportation systems like cars or trains, we will find network traffic related to infotainment, telematics, and remote diagnostics among others. If real-time traffic is not planned effectively regarding bandwidth utilization, it can affect the performance of infotainment, telematics, and diagnostic systems by monopolizing and wasting essential resources. However, accounting for unusable and left-over bandwidth for non-critical traffic classes is frequently omitted.

In this paper, we introduce a set of heuristic algorithms and meta-heuristic algorithms that schedule periodic and critical TSN streams and take the limitations of the TAS configuration and bandwidth utilization into account to address the aforementioned limitations of existing algorithms. Therefore, we investigate the impact of our suggested algorithms in synthetic scenarios by altering input data attributes and sequences, scheduling optimization techniques, and specific TAS characteristics. We examine the algorithms for their execution time, schedulability, scalability, and bandwidth utilization and give a recommendation for the choice of the proper TAS algorithms based on the user's requirements. Moreover, we study the benefit of a schedule compression proposed in [5] promising to reduce the number of configured slots for critical traffic and bandwidth wastage.

Our main contribution is the algorithms' design that not only focuses on adherence to end-to-end latency but also considers input data attributes and resource characteristics and limitations from the beginning. The algorithms comprise a scheduling step and a TAS configuration step that is mostly neglected in existing works. Moreover, to the best of our knowledge, we present an exhaustive evaluation of the network resource allocation for 802.1Qbv networks that is unmatched in its scope.

The paper is structured as follows. Section II introduces the fundamentals of this paper. The related work is addressed in Section III. We present a system model in Section IV and the applied scheduling algorithms in Section V. In Section VI, we evaluate the algorithms and compare them. We draw a conclusion in Section VII.

II. FUNDAMENTALS

A. TIME-AWARE SHAPER

TSN offers a set of shapers and schedulers including the TAS to provide latency guarantees besides the mechanisms for synchronization of time, reliability, and resource management. The classification of the network traffic is a key feature for TAS. It divides the network traffic into several network classes based on the Priority Code Point (PCP) value in the Virtual Local Area Networks (VLAN) tag (in the Ethernet header) that has the integer range of 0-7, which results in 8 traffic classes. The term TSN stream defines a sequence of one or more coherent network packets starting at a source (*talker*) and being propagated to one or more destinations, the *listeners* [6]. The TAS operates according to the Time Division Multiple Access (TDMA) principle that divides the time into several slots that can be reserved for one or more traffic classes at the same time. The defined sequence of non-overlapping time slots is repeated after a configurable period. The schedule of the time slots is represented by the *Gate Control List (GCL)* [3]. To implement the time slotting principle, the standard 802.1Qbv has introduced the so-called *gates*, one for each egress queue of a physical port, see Fig. 1. Typically, 8 egress queues, one for each traffic class, are available in physical ports [7]. According to [8], one egress queue is enough for the highest priority traffic that includes real-time traffic. One gate lets the traffic pass whenever it is set active in a GCL entry. Otherwise, it is deactivated and prevents the frames in the queue from being transmitted and thus the queue is not considered by the transmission selection. In Fig. 1a, the queues are encoded in a bit array in each GCL entry. The left-most position is related to the highest priority and the right-most to the lowest. 0 means the gate is inactive and 1 indicates that it is active in the slot. The figure shows 8 egress queues with time-aware gates that are controlled by the gate driver according to a GCL. The queue with priority 7 represents the queue for scheduled/critical traffic. This queue is exclusively served in the intervals [T0;T1] and [T2;T3]. If several gates are active at the same time, the transmission selection operates according to a scheduling algorithm like strict-priority scheduling (SP) that first serves the queues with the highest priority. The frames in the queues are removed in a *first come, first serve* manner. To benefit from the TAS, a common notion of time is necessary. This can be provided in IEEE 802.1 AS [9] with a maximum deviation of 1 µs.

Moreover, the gating mechanism alone does not protect critical time slots from the intrusion of non-critical network traffic into their slots. If the transmission of a non-critical Ethernet frame has been initiated just an instant before the slot start time for critical traffic it cannot be preempted instantly. In the worst case, it means an intrusion of a Maximal Transmission Unit (MTU)-sized packet with 1542 Byte considering Ethernet data and physical layer overhead. To avoid intrusion situations, IEEE 802.1Qbv defines the so-called *guard bands* that are applied at the end of a time slot and are associated with gates. Packets that are ready for transmission in the interval of an active guard band are held back and will not intrude into time slots that are not assigned for their classes. This occurs if the guard band has a fixed size (MTU transmission size in the worst case) and does not feature a special logic to check if a frame can pass without causing intrusions [3]. Fig. 1b shows the serialization on the wire and demonstrates a use case where the guard band has a

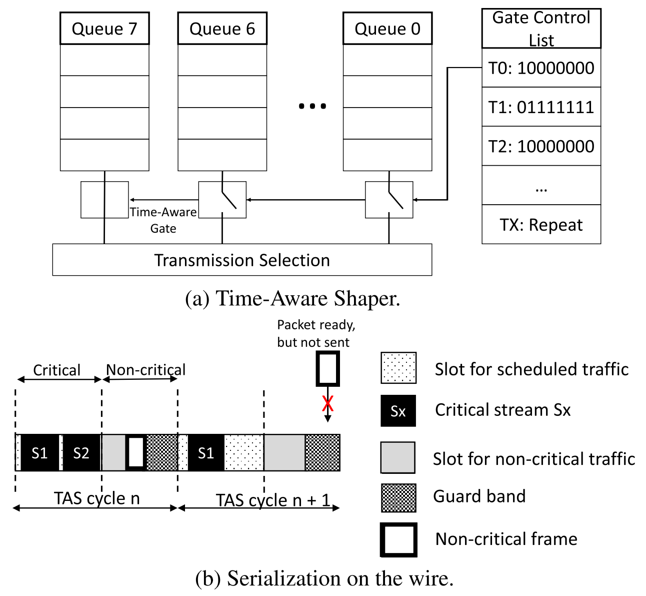


FIGURE 1. Example of a TAS configuration.

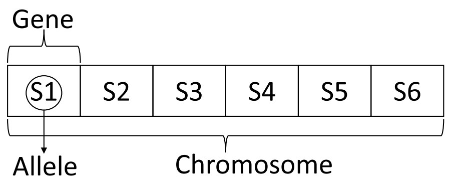


FIGURE 2. String encoding.

fixed size. One non-critical frame arrives in TAS cycle n an instant before the guard band starts and is transmitted. Another non-critical frame arrives in TAS cycle $n+1$. Even though it could fit into the non-critical slot, it is held back by the guard band and bandwidth is wasted. This illustration also makes clear that reserved slots do not have to be fully utilized. Unused slot times, especially in the critical slot, lead to bandwidth wastage. In Fig. 1b, such an unused slack arises between S_1 and S_2 in cycle n that could be assigned to lower priority traffic classes. While the TDMA principle provides benefits for critical traffic, it also shows disadvantages for other traffic classes.

However, finding a feasible gate configuration is a complex mission, since hundreds of time-critical network streams might have to be scheduled across small-sized or large-sized networks so far that all streams are able to meet their end-to-end transmission deadlines. The gate times of adjacent devices must be coordinated whenever two or more gates in different egress ports share at least one stream. The purpose is to avoid unnecessary delays. This requires careful scheduling approaches that we will address in this work.

We cannot make assumptions about wasted bandwidth in the non-critical slots since we do not have enough knowledge about it. But we can evaluate our scheduling algorithms by the number of necessary slots as well as resulting guard bands and determine the wasted bandwidth in critical slots. We will address the scheduling of TSN streams, the configuration of the GCLs in the network, as well as the bandwidth utilization in critical slots in more detail.

B. GENETIC ALGORITHMS

Since some of our considered algorithms use Genetic Algorithms (GA) [10], we will give an introduction in this section. A GA is inspired by the biological evolutionary process, which includes natural selection and genetics. It is a randomized search technique that consists of several steps, see [11]:

1) Initialization: An initial population of individuals is created. The population represents the first generation. Each individual represents a solution to a problem and consists of an encoded chromosome. The chromosome has a set of genes whose values often have a bit or string representation [12].
2) Fitness Evaluation: Each individual is assigned a fitness value through a fitness function that indicates the quality of the solution.
3) Selection: A percentage of individuals is selected for the next generation. Individuals with a better fitness value have a higher chance to survive.
4) Reproduction: In this step, the selected individuals are mated and mutated by applying crossover operations on two individuals or randomly mutating one or more individuals. This process generates new offspring.
5) Replacement: A fraction of the individuals is replaced by the generated offspring while maintaining the population size. Often, the best individuals are taken to the next generation as described in the elitist approach [13, p. 347].
6) Repeat or terminate: Either steps 1-5 are executed again or a termination condition is met, e.g. number of generations or a specified runtime is reached.

Scheduling a TSN stream can be mapped to a permutation problem [11]. Each chromosome encodes a sequence of all streams to be scheduled, and each gene, which can be perceived as a variable, contains the unique identification of a stream, e.g. in a string representation. The value of a gene is also referred to as *allele*. The chromosome describes the order in which the streams are scheduled from the left to the right. Fig. 2 gives an example of a stream encoding. During our evaluation phase, we discovered that the Position-based Crossover (PBX) and the *Swap* mutator are the most appropriate for our problem.

The PBX takes two individuals and determines a random number of positions that is smaller than the chromosome size. In Fig. 3, we choose indexes 1 and 4. Two new offspring are created. The chromosomes of the offspring inherit the genes that are determined by the selected positions, each offspring from one of the parents. *Offspring1* from Fig. 3 inherits *S2* at index 1 and *S5* at index 4 from *Individual1*. The remaining positions of the offspring are filled with unique elements from the other parent under the observance of the gene order of the parents. In our example, we would try to place the gene

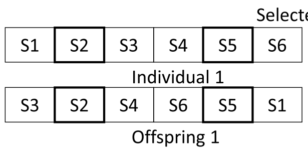


FIGURE 3. **PBX operator.**

with the value *S2* from *Individual2* at index 0 in *Offspring1*. But since *S2* is already present in *Offspring1*, we continue with *S3*. *S3* is not present in *Offspring1* and can be placed at index 0, etc. The Swap mutator takes one individual from the population and selects two random genes in the chromosome. Then, the gene positions are swapped. For more detailed information, we refer to our previous work [11].

III. RELATED WORK

In this work, we introduce a variety of algorithms for scheduling TSN streams and, in addition, provide a configuration method for the TAS while also performing an in-depth evaluation of network resource usage. The scheduling problem regarding the TAS has been addressed and reviewed multiple times. However, to the best of our knowledge, no other work presents a unified approach to scheduling and configuration while also optimization bandwidth utilization and wastage. We will first review related TSN scheduling approaches that mainly focus on adherence to end-to-end stream delays without considering the GCL configuration and resource utilization. One of the most reviewed works in this field is from Craciunas et al. [14]. The authors make use of Satisfiability Modulo Theories (SMT), which determines for given formulas (or in this case constraints) if they are satisfiable. This approach is able to schedule up to 1500 TSN streams in a range of 4 hours varying the size of flows and devices. They have mainly focused on the scheduling aspect and the resulting runtime performance. Besides, their algorithm allows the use of several egress queues. In terms of runtime, our performance surpasses that of similar network topologies. However, we cannot compare the success rate or the resultant end-to-end delays since that data has not been provided. The approach presented in [10] applies GAs to schedule time-triggered traffic in time-sensitive networks. The authors propose a GA for solving the routing and scheduling problem of TSN streams with inter-dependencies but do not go into detail about the GCL configuration. Given that we do not address the routing issue or the inter-dependencies of streams, a direct comparison with this work is not feasible. However, we share similarities in the design of the genetic algorithm for TAS-based scheduling, specifically in the chromosome encoding that associates one stream with one gene and the fitness function that outlines the resulting makespan of the schedule.

Also, a few works exist that draw attention to the joint scheduling of TSN streams and the configuration of the GCLs. Dürr and Nayak [5] have implemented a *Tabu Search* algorithm for scheduling TSN streams and integrated the no-wait principle. After the scheduling phase, they perform a

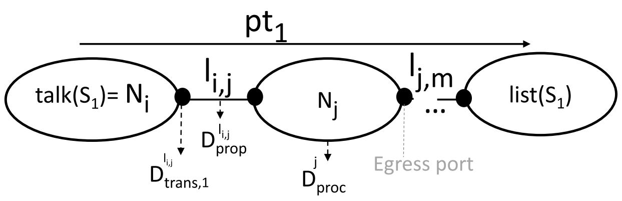


FIGURE 4. **Stream and delay characteristics for stream S_1.**

compression algorithm to reduce the number of GCL entries and guard bands. We adopt the no-wait approach and also perform a modified compression function afterward. For similar network configurations, our suggested algorithms demonstrate superior performance in terms of runtime compared to the cited work. Opposed to this work, we allow different harmonic or non-harmonic periods, while [5] considers the same period for all streams. Nonetheless, the authors have not presented any evaluations concerning schedulability or bandwidth utilization. Jing et al. [15] assume a fixed number of allowed GCL entries in the network using up to 4 queues to reduce guard bands. They solve the scheduling problem by applying SMT, Optimization Modulo Theories (OMT), and a custom heuristic approach. The SMT variant takes about 2 days to find feasible schedules for small use cases involving up to 100 flows. The OMT approach allows scheduling up to 10000 streams in two days. The heuristic approach outperforms other heuristic approaches, resulting in fewer GCL entries that have to be scheduled, but shows a slightly worse schedule ratio. Compared to this approach, we focus on one egress queue for scheduled traffic. Several of our proposed approaches can keep up or outperform the resulting number of GCL entries for up to 800 streams and 30 switches. However, we do not have enough knowledge about the used network resources in the paper. Further on, we show a lower runtime and carry out evaluations closer to industrial and automotive use cases. Moreover, we provide more diverse procedures to control the number of GCL entries and the bandwidth utilization. The work presented in [16] applies the SMT solver to schedule TSN streams. Similar to a set of our approaches, they set the GCL period to the greatest common divisor (GCD) of all stream periods. They evaluate their algorithm with small use cases and state that GCD-based approaches show a better runtime and allow reusing the configured slots. However the authors do not report statistics about GCL configuration or bandwidth utilization.

IV. SYSTEM MODEL

We describe the system model based on [17] and [18]. The network is denoted as a bidirectional graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ defines the set of network nodes and $\mathcal{L}$ the set of links between nodes. Each link $l_{i,j} \in \mathcal{L}$ defines a unidirectional logical link between the nodes $n_i \in \mathcal{N}$ and $n_j \in \mathcal{N}$, $(i! = j)$. The method $r(l_{i,j})$ returns the link rate of $l_{i,j}$. The streams to be scheduled are denoted by the set $\mathcal{S}$. Each stream $s_k \in \mathcal{S}$ is defined by the tuple $\langle p_k, \phi_k, d_k, l_k, pt_k \rangle$, with p_k denoting the stream period, ϕ_k the release offset at the talker

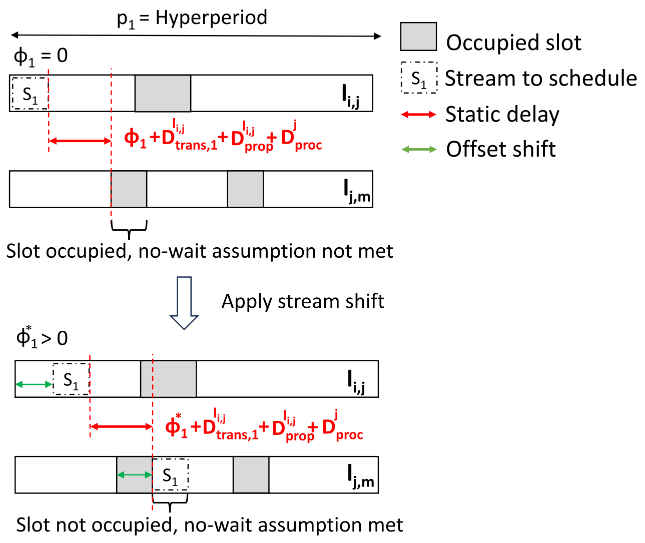

p1 = Hyperperiod
ϕ1 = 0
S1
li,j
ϕ1 + D trans,1 + D prop + D proc
lj,m
Slot occupied, no-wait assumption not met
Apply stream shift
ϕ1* > 0
S1
li,j
ϕ1* + D trans,1 + D prop + D proc
S1
lj,m
Slot not occupied, no-wait assumption met
Occupied slot
Stream to schedule
Static delay
Offset shift

FIGURE 5. No-wait scheduling of S_1 in reference to Fig. 4.

referred to the start of p_k, and d_k the relative deadline in reference to the start of p_k. We assume that the deadline equals the period. However, this assumption can be adapted. The resulting scheduling offset can be adjusted by the scheduling algorithms and is described as ϕ^*_k, $\phi^*_k \geq \phi_k \geq 0$. The size of s_k is defined as l_k including physical and data layer protocol overhead. The stream path is denoted by pt_k that consists of a sequence of links. The function $talk(s_k)$ returns the talker of the stream and $list(s_k)$ the listener. For the sake of simplicity, we assume that each stream has one listener and carries one Ethernet frame. Considerable stream delays are listed in the following, cf. [17]:

- transmission delay $D^{l_{i,j}}_{trans,k} = (l_k \times 8\ bit/Byte)/r(l_{i,j}))$ of stream s_k on link $l_{i,j}$ that describes the time it takes to put bits of s_k on the wire,
- propagation delay $D^{l_{i,j}}_{prop}$ that denotes the time it takes for a bit to travel from one end of the link to the other depending on the cable properties, and
- processing delay D^i_{proc} of a network node n_i depending on its switching logic.

Some stream delays and characteristics are illustrated in Fig. 4 for an example stream S_1. We focus on scheduling TSN streams in a no-wait manner that denotes that whenever a stream starts its transmission at the talker, it travels along its path without encountering other streams in egress port queues. Thus, streams have exclusive access to network resources and do not experience any queueing delays. The consideration of queuing delays defines its own field of research and can be performed e.g. by using analytical frameworks like *Network Calculus* [19], [20]. Fig. 5 depicts the no-wait scheduling for S_1. In the upper scenario, S_1 is allocated on the link $l_{i,j}$ with the offset $\phi_1 = 0$. The no-wait transmission start time on the next link $l_{j,m}$ would be at $\phi_1 + D^{l_{i,j}}_{trans,1} + D^{l_{i,j}}_{prop} + D^j_{proc}$. Since there is already another stream scheduled at this time step and S_1 would have to wait for its transmission, we would hurt the no-wait assumption. Thus, we have to shift S_1 on the first link $l_{i,j}$ by the necessary offset, in this case by the size of the occupied slots. S_1 gets an adjusted start offset $\phi^*_1 > 0$ consequently, and the no-wait principle is met.

The remaining delays are regarded as static delays and hardware-dependent. Consequently, the end-to-end transmission delay $e2e_k$ of s_k neglecting the stream offset ϕ_k is calculated by:

$$\forall l_{i,j} \in pt_k:$$
$$e2e_k = \sum_{l_{i,j} \in pt_k \setminus \{l_{talk(s_k),m}\}} (D^i_{proc} + D^{l_{i,j}}_{trans,k} + D^{l_{i,j}}_{prop}) + D^{l_{talk(s_k),m}}_{trans,k} + D^{l_{talk(s_k),m}}_{prop}. \tag{1}$$

In Fig. 1, the talker does not consider a processing delay at its end-station. We also neglect the clock deviation that must be considered in real systems. A schedule is feasible if Fig. 2 applies to all scheduled streams.

$$\forall s_k \in \mathcal{S}:$$
$$\phi^*_k + e2e_k \leq d_k. \tag{2}$$

V. SCHEDULING ALGORITHMS FOR TAS

We will present a classification of scheduling algorithms for TAS-capable networks in Section V-A that follow the assumptions made in the system model. Implementation details will be presented in Section V-C. The main objective of the presented scheduling approaches is to comply with the stream end-to-end deadline and to minimize the *makespan*. Makespan defines the effective length of the schedule beginning from the transmission start of the earliest stream and terminating at the transmission end time of the latest stream in the resulting schedule. The goal of our work is not only to examine which algorithms provide the best makespan but also to investigate which scheduling variants provide the most efficient bandwidth utilization and GCL configuration. Therefore, we propose algorithm variants considering the input data properties and influencing the input data ordering, the scheduling procedure, and the GCL configuration.

A. SCHEDULING ALGORITHM CLASSES

The main part of this work is the design of scheduling algorithms that impact schedule quality regarding end-to-end stream delays and network resource utilization. Therefore, we have identified a hierarchical classification of TAS algorithms, as illustrated in Fig. 6. Each level of the hierarchy has unique distinguishing characteristics. The distinction was derived from TAS properties, other fields of applications, and use cases and influences the makespan and the network resource utilization. On the top level, we subdivide the scheduling algorithms into harmonic and non-harmonic variants that state how stream periods are selected. This classification refers to the characteristics of the input data. Harmonic periods are multiples of each other in contrast to non-harmonic periods. Harmonic periods are characteristic for industrial real-time communication systems, as stated

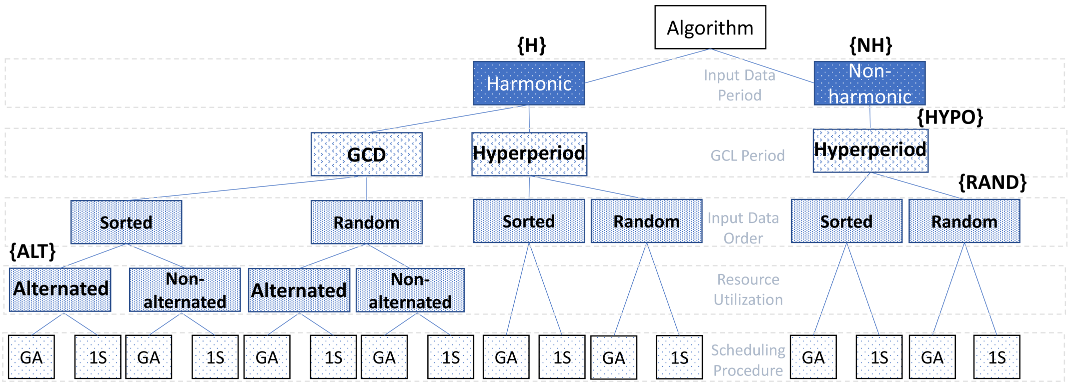
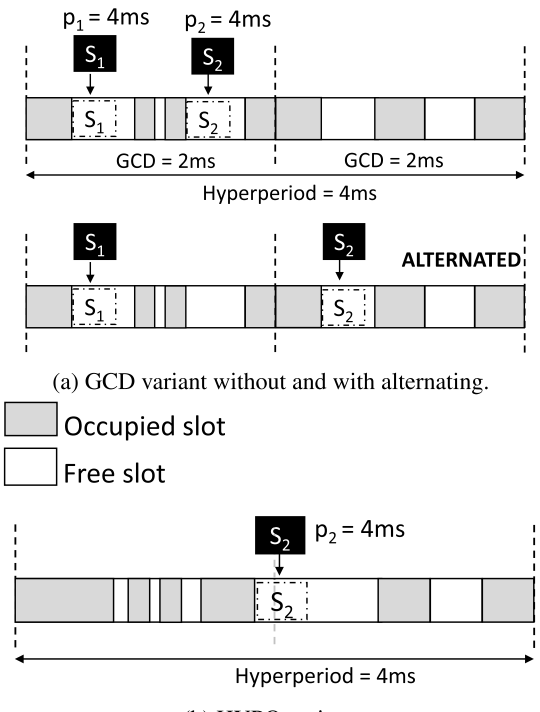


FIGURE 6. Algortihm classification with abbreviations {.} for Section VI.

in [21] and [22]. Notably, in automotive systems also non-harmonic periods are of relevance [23]. In general, all scheduling algorithms create a schedule with a length of the hyperperiod hp which is the least common multiple of all stream periods. Each stream s_k has to be scheduled hp/p_k times within the hyperperiod on each link of its path. The schedule is repeated after each hyperperiod. The final resulting GCL period for each TAS-capable egress port can be fitted into the size of the hyperperiod in case of harmonic and non-harmonic periods or it can be transformed and reduced to the GCD of the stream periods for harmonic periods. This subdivision is illustrated underneath the harmonic class in Fig. 6 and has an influence on the configuration of the GCL of the TAS. The GCD has the size of the smallest period, all other periods and the hyperperiod are multiples of it. The reduction of the GCL period to the GCD helps to lower the number of necessary GCL entries for critical traffic as detailed in Section V-D. To be able to reduce the GCL period, we have to subdivide the hyperperiod into several segments of the length of the GCD resulting in hp/GCD segments, see Fig. 7a (upper and lower scenario). The figure shows exemplary scheduling of S_1 and S_2 in an egress port where other streams have already been scheduled (occupied slots). In principle, planning a stream s_k in the GCD and hyperperiod variant (*HYPO*)[1] work similarly (Section V-C) with the significant exception that s_k is not planned beyond the boundaries of a GCD segment on a link. GCD algorithms do not plan stream beyond GCD boundaries since the GCL period will be set to GCD and the transmission behavior between GCL periods is not clearly defined in the hardware. Fig. 7b shows a HYPO algorithm that places S_2 between two GCD periods. The non-harmonic branch neglects the GCD branch since the GCD of non-harmonic periods can be significantly smaller than the stream periods and consequently lead to a significant planning and management overhead.

The following applies to harmonic and non-harmonic variants: Each variant can be executed with a random order of streams or with a sorted stream order. This classification refers to the input order of data. Sorting is done in the

[1]We will abbreviate the hyperperiod algorithm classes with HYPO in the following sections.



(a) GCD variant without and with alternating.

(b) HYPO variant.

FIGURE 7. Example of a schedule on a link/port for GCD and HYPO versions.

ascending order of the stream periods. Consequently, the first stream to plan is the one with the smallest period. The sorting approach can be related to the well-established rate monotonic algorithm for task scheduling, which plans the tasks with a lower period (higher rate) first [24].

We additionally consider the alternated and non-alternated versions of the GCD class. The alternation follows a load-balancing approach and means that streams with a period that is a multiple of the GCD are planned in the GCD segment with the least occupation first. The alternation branches represent an optimization process for bandwidth utilization that is integrated with the scheduling phase. Fig. 7a shows an example of an alternating and non-alternating approach. In both, S_1 with a period of 4 ms that is a multiple of the GCD duration is scheduled in the first GCD segment. Then S_2 is scheduled, also with a period of 4 ms. Since the

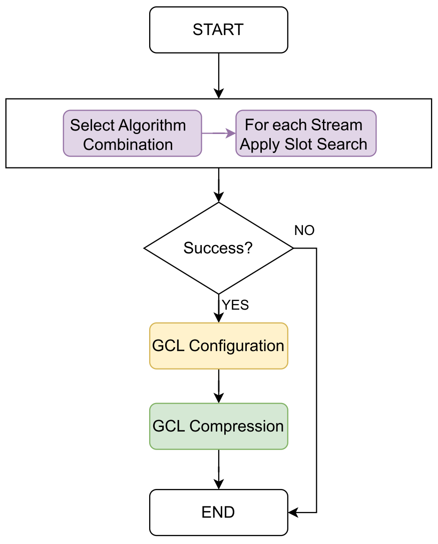

START
Select Algorithm Combination
For each Stream Apply Slot Search
Success?
NO
YES
GCL Configuration
GCL Compression
END

FIGURE 8. The whole scheduling procedure.

first segment has now a higher occupation, S_2 is scheduled in the second segment in the alternated approach. In the non-alternating scenario, S_2 is scheduled in the first fitting slot in the first segment.

The leaves of the classification tree describe the procedure that solves the scheduling problem. The input streams can be scheduled with a heuristic approach, that is specifically designed for the scheduling problem. It is executed and evaluated once, one-shot (1S), with a given order of streams. The other option is the GA. The difference to heuristics is that GAs investigate several permutations of the stream orders and evolve them over several generations to improve the makespan, which represents the fitness value of an individual.

B. OVERALL SCHEDULING PROCEDURE

One scheduling algorithm is defined as a combination of the algorithm classes that starts at the harmonic/non-harmonic subdivision and ends at the 1S/GA leave, which leads to 16 different combinations. One combination is, e.g., H_GCD_Sorted_ALT_GA. Fig. 8 illustrates the steps of the whole scheduling procedure. Regardless of the combination selected, we follow the same steps. The first major step is the selection of the algorithm. Initially, we are given a set of streams with either harmonic or non-harmonic periods for scheduling. Based on the periods of these streams, we then choose either the harmonic or non-harmonic variant. Subsequently, we determine whether to arrange the input data in order or leave it randomized, mapping the decision to a scheduling class. If we operate with a harmonic class and intend to fit the GCL period into the GCD, we choose a GCD class. In this case, we can apply a balancing effect by selecting the alternating approach. Then, the algorithm extracts one stream after the other from the given order of streams and tries to find proper scheduling slots according to the proposed slot search strategy in Section V-C with a 1S or GA procedure. If successful, the slot occupation of streams on all links is determined and forwarded to the GCL setup, where designated GCL entries on each utilized link are calculated (Section V-D). If desired, the GCL configuration can be optimized afterward using a schedule compression procedure that is introduced in Section V-E.

C. SLOT SEARCH ALGORITHM

The primary scheduling step consists of finding a feasible schedule using one of the 16 presented algorithms. All algorithms make use of the slot search algorithm that we introduced in [17]. The implementation presented in our former work [17] covers solely the H_GCD_Rand_1S approach. In this paper, we reuse the slot search algorithm from [17] and extend it with alternating, sorting, and HYPO variants. During the scheduling phase, we only consider slots for critical traffic and assume that the whole bandwidth is available for critical traffic.

In alignment with the slot search algorithm suggested in [17], the goal of the scheduling algorithms is to find suitable and non-overlapping slots for each stream s_k along its path pt_k such that it does not experience any queueing delay, only static network delays. The algorithm tries to place s_k on each link of pt_k starting with the initial offset ϕ_k at the talker. If there are no suitable slots ϕ_k is shifted to ϕ_k^* until appropriate free slots or no feasible solution can be found. We look for equidistant time slots that are one stream period apart on each link of pt_k and require that slots on consecutive links $l_{i,j}$ and $l_{j,m}$ are $D_{trans,k}^{l_{i,j}} + D_{prop}^{l_{i,j}} + D_{proc}^{j}$ time steps apart to maintain the no-wait approach. Equidistant slots help reduce the jitter of end-to-end times. Each egress port keeps track of free and occupied time slots. Whenever a stream is scheduled, it occupies new time slots and reduces free slots. If s_k fits multiple times in the hyperperiod we look for hp/p_k equidistant slots on each link. These steps are common for all variants.

Fig. 9 shows a HYPO-based planning process for S_1 with $\phi_1 = 0$ and $p_1 = hp/2$ on link $l_{i,j}$. S_1 has to be scheduled two times within the hyperperiod. The algorithm looks for free slots in the interval $[0;p_1)$ and $[p_1;hp)$. Here, we can place S_1 in the free slot of the first segment and the second segment such that the start times lie p_k apart. The position of S_1 within the free slots does not have to align with the beginning or the end times of the free slot. In some cases, ϕ_1 has to be shifted to find feasible slots. We proceed with the slot search on the next link starting at $\phi_1^* + D_{trans,1}^{l_{i,j}} + D_{prop}^{l_{i,j}} + D_{proc}^{j}$ (cf. Fig. 5).

The GCD-based approaches work similarly with the difference that we look for free slots in GCD segments that are one stream period apart. The segment-based search for non-alternating GCD classes has already been presented in our former work [17]. Fig. 10 and Fig. 11 show exemplary GCD planning with 4 GCD segments of the size 2 ms. A stream with a period of 2 ms, such as S_1, is scheduled in each GCD segment. The stream S_2 with a period of 4 ms has to be scheduled in two segments that are one stream period apart, e.g. in the first and the third segment (GCD index 0 and 2) or the second and the fourth. Non-alternating GCD and HYPO

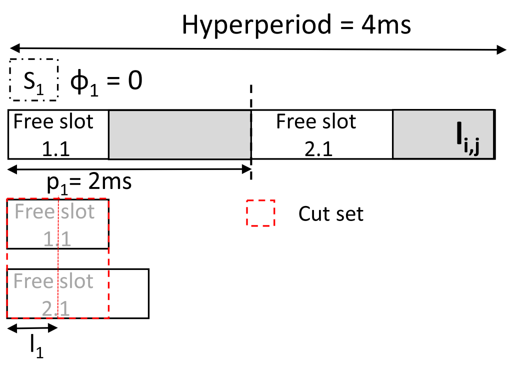


FIGURE 9. Scheduling S_1 with p_1 < hyperperiod.

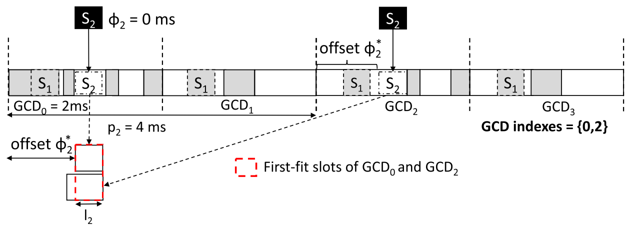


FIGURE 10. Scheduling S_1 and S_2 with non-alternating GCD.

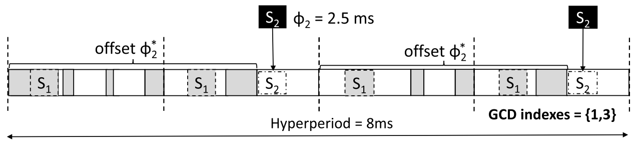


FIGURE 11. Scheduling S_2 with a different offset using GCD.

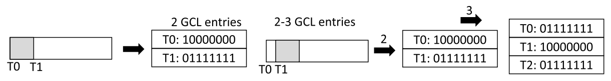


FIGURE 12. Example configuration of GCL entries on a link.

approaches apply the earliest and first fit solution, while the alternating versions implement a load-balancing strategy that starts the slot search in the least occupied GCD. In Fig. 10, S_2 has an initial offset $\phi_2 = 0\ ms$. The non-alternating approach searches for free slots that are big enough for l_2 in the earliest segments indexed with $\lfloor \phi_2/GCD \rfloor = 0$ and 2. In Fig. 11, $\phi_2 = 2.5\ ms$. Thus, the non-alternating algorithm starts the slot search at indexes 1 and 3. The alternating algorithm examines the bandwidth occupation in segments {0,2} and {1,3} and begins the slot search in the least occupied combination. Other algorithm classes do not have an influence on the slot search algorithm, but rather on the order and the overall procedure.

D. GCL CONFIGURATION

Successful schedules provide the final transmission offsets and the occupied slots for each stream and guarantee exclusive access to network resources. After, the configuration of GCL entries has to be accomplished. The network could set the GCL periods of all utilized egress ports to the hyperperiod of all streams and allocate one GCL entry for each stream along its path. The configuration times are given by occupied slots of the streams. However, the resulting allocation of

Algorithm 1 createCritGCLEntries($\mathcal{C}$)

```
1  $maxTrans \leftarrow$ setmaxTrans();
2  $normSlots \leftarrow$ normalize($\mathcal{C}$);
3  sort($normSlots$);
4  $refSlot \leftarrow$ popNext($normSlots$);
5  if $refSlot.start > 0$ and $refSlot.start < maxTrans$ then
6      sumUpWasted();
7      $refSlot.start \leftarrow 0$;
8  while $normSlots$ not empty do
9      $slot \leftarrow$ popNext($normSlots$);
10     $diff \leftarrow slot.start - refSlot.end$;
11     if $diff \geq maxTrans$ then
12         createGCLEntry($refSlot.start, refSlot.end$);
13         $refSlot = slot$;
14     else
15         sumUpWasted();
16         setSlotStart($refSlot$,
             min($refSlot.start, slot.start$));
17         setSlotEnd($refSlot$,
             max($refSlot.end, slot.end$));
18     end
19 end
20 $diff \leftarrow GCLPeriod - refSlot.end$;
21 if $diff < maxTrans$ then
22     sumUpWasted();
23     $refSlot.end \leftarrow GCLPeriod$;
24 createGCLEntry($refSlot.start, refSlot.end$);
```

GCL entries could have many gaps that cannot be used by other traffic classes. Also, the number of configurable slots for all traffic classes in the TAS is often strongly limited. It varies between around 128 GCL entries for smaller and experimental setups up to 1024 entries [16]. Thus, it might not be possible to create an exclusive entry for each stream. In Fig. 13a a HYPO algorithm allocates nine slots, while in Fig. 13b a GCD algorithm results in 5 slots for the same amount of critical network traffic.

The challenges are highlighted by an example in Fig. 12: If only one stream is scheduled in the network the GCL period of all traversed ports could be set to the stream period. Assigning a GCL slot to a stream s_k that starts its transmission at $t = 0$ means configuring at least two slots, one for the critical traffic and one for other traffic classes. If the stream is transmitted on a link at $t > 0$ then either another slot for non-critical traffic can be created before the transmission of s_k and one after, or the preceding or the succeeding time window is merged with the stream slot. Merging slots reduces the number of necessary GCL entries but can lead to unused bandwidth. In the case of more streams, we have more decisions to make about configuration, merging, and GCL period.

Our slot configuration method in Algorithm 1 configures and optimizes the GCL slots for critical streams. The slots in between are reserved for best-effort traffic classes. The algorithm receives the effective set of occupied stream slots $\mathcal{C}$ that results from one of the scheduling algorithms.

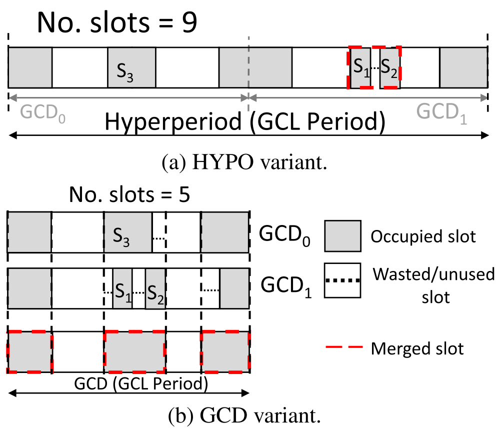


FIGURE 13. Merging GCL entries.

First, we determine *maxTrans* that represents the transmission time of the MTU-sized frame with 1542 Byte on wire depending on the link rate (line 1). If the scheduling algorithm is a GCD variant we have to normalize the effective transmission times to create entries in the range of the GCD (line 2). The slot normalization is denoted by:

$$\begin{aligned} newSlotStartTime &= oldSlotStartTime\%GCLPeriod \\ newSlotEndTime &= oldSlotEndTime\%GCLPeriod. \end{aligned}$$

A normalized view is presented in Fig. 13b for GCD algorithms, while Fig. 13a illustrates the non-normalized view for GCD and at the same time the normalized representation for HYPO approaches. The normalization does not affect HYPO variants. We sort the normalized stream slots in the ascending order of the transmission slot start times (line 3). The GCL period is set to GCD or the hyperperiod depending on the algorithm classification. Starting with the earliest slot, we first check if there is a gap between the start of the GCL period and the start of the earliest slot *refSlot* (lines 5-7). If the gap is smaller than *maxTrans* we merge the unused window with the stream slot and add up wasted bandwidth. We assume that the guard bands have a fixed size of *maxTrans*, as discussed in Section II-A. This chosen window size guarantees that at least one best-effort Ethernet frame can pass if it arrives before the guard band time in contrast to smaller window sizes. In lines 9-10, we remove the next slot (*slot*) and calculate the gap between *slot* and the reference slot *refSlot*. If the difference is greater than or equal to *maxTrans* we create a GCL entry with the times and size of *refSlot* and leave the gap between *slot.start* and *refSlot.end* for other traffic classes (lines 11-13). In this case, *slot* becomes the reference slot in the next iteration. If the difference between the slots is smaller than *maxTrans* (line 14) we merge the *refSlot* and *slot* by taking the minimum of both start times (line 16) and the maximum of both end times (line 17). Exemplary slot merge approaches are illustrated in Fig. 13. In GCD variants, the start time of *slot* is greater than or equal to the start time of *refSlot*, but the end time could be smaller since we normalize and times and sort by start times. The merged slot is the reference slot in the next iteration under this condition.

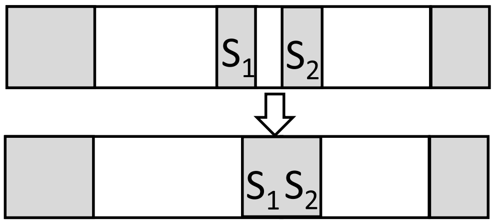


FIGURE 14. Schedule compression.

We repeat this procedure over all slots (lines 8-18) and finally examine the gap between the last slot and the GCL period end (lines 20-24). During the whole procedure, we also calculate the wasted bandwidth by summing up all gaps that are located in merged slots (lines 6,15). In HYPO variants, the unused slots only occur between two adjacent critical slots (Fig. 13a) and between one critical slot and the beginning or the end of the GCL period, see Fig. 12. In the GCD-based variants, a merged slot can be used by several streams running in different GCDs, see Fig. 13b. If one normalized stream slot overlaps with another, e.g. S_1 with S_2 and S_3 in Fig. 13b, we take the average of the wasted slot sizes over all affected GCD segments, which are GCD_0 and GCD_1 in our example. Particularly alternating approaches support the re-usage of merged slots by different streams [16]. We do not include a specific limit of GCL entries in the algorithms, but we can extend the algorithm to do so.

E. SCHEDULE COMPRESSION

Another approach that is applied in this paper is schedule compression. It was shortly introduced in [5]. The schedule compression tries to shift the scheduled streams such that the wasted bandwidth is reduced and the transmissions of streams are closer to each other, as depicted in Fig. 14. After the slot configuration part, we examine if schedule compression is useful. Therefore, we sort the scheduled streams in descending order of their resulting end-to-end times. We pick the first stream with the latest end-to-end time (reference stream) and look for streams in the sorted list that cross its paths. Then, we try to shift the crossing streams closer to the reference stream so that the maximum end-to-end deadline of the crossing streams is not missed. The shift move consists of increasing the resulting offset ϕ_k^* of s_k. Therefore, we have to find a shift value that is feasible on the whole path of the crossing streams without overlapping with other scheduled streams. We continue the procedure with the next stream in the sorted list as the reference stream, etc. If the shift operations lead to contiguous streams being torn apart and resulting in worse bandwidth utilization we neglect the shift step. In the end, we finally agree on whether to use the previous plan or the shifted one, which depends on if the shifted version improves the bandwidth utilization or not.

F. ADJUSTMENT OF THE GENETIC ALGORITHM

In Section II-B, we have presented how to encode an input order of streams in a chromosome and which crossover and mutation operators can be applied. These characteristics suit

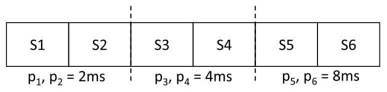


FIGURE 15. **Chromosome encoding creating sorted groups.**

TABLE 1. **GA parameters.**

Parameter	**Setting**
Population size	30
Generation number	20
Crossover method	PBX, PBX-Sort
Mutation method	Swap, Swap-Sort
Crossover rate	0.7
Mutation rate	0.1

well for random order algorithm classes. However, sorted algorithm classes need a slight adjustment to the encoding of the chromosome and the evolution operators to be able to maintain the sorted order of streams (by their periods). The sorted approaches create subgroups in the chromosome. Each subgroup is identified by a stream period. Streams with the same period are assigned to the same subgroup, see Fig. 15. The subgroups are sorted in the ascending order of the subgroup period within the chromosome. The chosen crossover operator for this algorithm is the *PBX-Sort* that is a modified PBX version, and the selected mutator is *Swap-Sort*, which is also a modified version of the Swap mutator. The difference is that the modified operators work on subgroups and not on the whole chromosome. In summary, they are applied to randomly drawn subgroups and can lead to changes within the subgroups.

The selected parameters for GA algorithm classes are summarized in Fig. 1. In general, a bigger population size and a higher number of generations might lead to a better solution for space exploration and exploitation. But to avoid a runtime explosion, we have chosen a population size of 30 and a generation number of 20. The choice of the crossover and mutation operators and rates has been experimentally determined.

VI. EVALUATION

We have developed the set of scheduling algorithms using the Java-based GCD method from our earlier work [17], which solely included the H_GCD_Rand_GA variant. We have extended the implementation with the sorted, alternating, GA, and all HYPO variants. The GA variants have been implemented with the help of the *Jenetics* framework[2] that is also written in the Java language. The evaluation was performed on an Intel(R) Core(TM) i7-8550U CPU with a 1.88 GHz clock rate and 24 GB RAM. To test all variations of the scheduling algorithms, we have varied the following parameters:

- number of switches: {3, 5, 10, 20, 30}
- number of end-stations: ⌈ [1;2] * number of switches ⌉
- number of streams: {50, 150, 200, 300, 500, 800}
- topologies: {star, ring, meshed}
- stream length: 84-1542 bytes, one frame per stream
- transmission mode: unicast

We group a set of switches and streams to small- to middle-sized and larger industrial networks, resulting in (cf. [25]):

- small/middle networks (SMN): {3, 5, 10} switches and {50, 150, 200} streams
- large networks (LN): {20, 30} switches and {300, 500, 800} streams

The link speed was set to 1 Gbps (Gigabit per second), which is well represented in today's networks. The network topologies are randomly generated and also the streams are randomly allocated. The selection of the stream periods has a significant impact on the following investigations. The set {2, 4, 8, 16, 32} ms identifies the harmonic periods. Non-harmonic periods are {2, 4, 5, 10, 20} ms, as identified for the vehicular domain in [23]. We have executed all combinations of algorithms from Fig. 2, which defines the 16 algorithm classes, 50 times for each combination of a number of switches, number of streams, and experiment. The periods of the streams differ for harmonic and non-harmonic variants but are the same within each group. The harmonic and non-harmonic use cases are not directly comparable. But we would like to explain how the choice of periods affects the experiments. The main purpose of the evaluation is to observe the time performance, schedulability, scalability, and bandwidth utilization based on the numerical results derived from the execution of the scheduling algorithms.

A. TIME PERFORMANCE

A reasonable runtime is especially important for rapid prototyping and reconfiguration scenarios. Fig. 16 shows the average time performance of the different algorithms in seconds in dependence on the number of streams. Only feasible schedules are taken into account. It becomes apparent that the runtime increases with the number of streams. While the heuristic approaches perform linearly and under 30 seconds, the GAs show a high runtime increase for more than 200 streams. GAs perform in a range of several minutes for a high number of streams. Also, the figure reveals that HYPO-based algorithms perform worse on average. GAs using the GCD and alternating the segments perform better than other GA variants. A similar conclusion can be drawn for the time performance in dependence on the increasing number of bridges, see Fig. 16b. Experiment NH_HYPO_Sorted_GA improves for 30 switches. One reason could be that the streams are better distributed across the network. Thus, the slot search algorithm finds fewer conflicting streams and a faster solution.

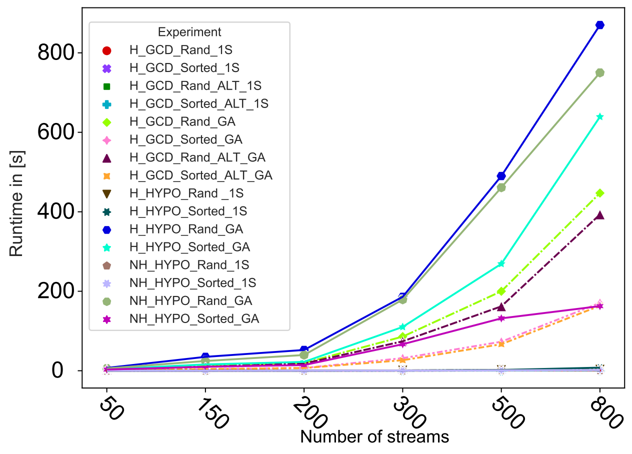
B. SCHEDULABILITY

Especially in relation to 1S heuristic algorithms, the success rate becomes relevant. While GAs evaluate and permutate hundreds of different solutions, 1S algorithms only have one

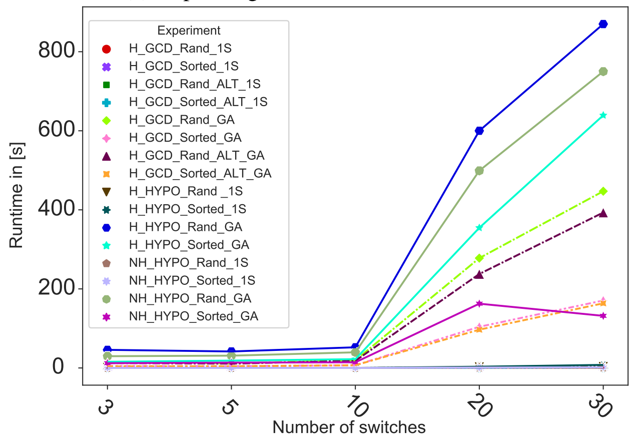
[2]https://jenetics.io/

TABLE 2. Experiments classification.

Experiment	Periods	GCL length	Algorithm	Sorted	Alternating
H_GCD_Rand_1S	harmonic	GCD	one-shot	no	no
H_GCD_Sorted_1S	harmonic	GCD	one-shot	yes	no
H_GCD_Rand _ALT_1S	harmonic	GCD	one-shot	no	yes
H_GCD_Sorted_ALT_1S	harmonic	GCD	one-shot	yes	yes
H_GCD_Rand_GA	harmonic	GCD	GA	no	no
H_GCD_Sorted_GA	harmonic	GCD	GA	yes	no
H_GCD_Rand_ALT_GA	harmonic	GCD	GA	no	yes
H_GCD_Sorted_ALT_GA	harmonic	GCD	GA	yes	yes
H_HYPO_Rand_1S	harmonic	hyperperiod	one-shot	no	no
H_HYPO_Sorted_1S	harmonic	hyperperiod	one-shot	yes	no
H_HYPO_Rand_GA	harmonic	hyperperiod	GA	no	no
H_HYPO_Sorted_GA	harmonic	hyperperiod	GA	yes	no
NH_HYPO_Rand_1S	non-harmonic	hyperperiod	one-shot	no	no
NH_HYPO_Sorted_1S	non-harmonic	hyperperiod	one-shot	yes	no
NH_HYPO_Rand_GA	non-harmonic	hyperperiod	GA	no	no
NH_HYPO_Sorted_GA	non-harmonic	hyperperiod	GA	yes	no



(a) Depending on the number of streams.



(b) Depending on the number of switches.

FIGURE 16. Average runtime of the scheduling algorithms.

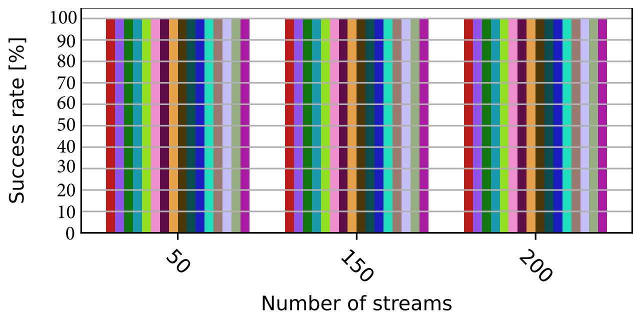
trial. The success rate is composed of the number of successful executions (#s), that found feasible solutions, divided by the sum of successful executions and failed executions (#f):

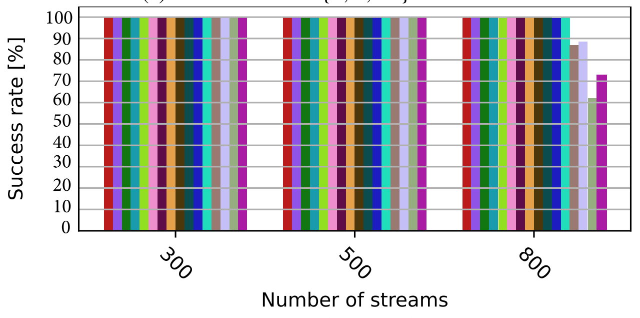
$$\frac{\#s}{\#s + \#f}.$$

Fig. 17 shows the success rate for all algorithms related to the number of streams and switches. For the set of {3, 5, 10} switches and {50, 150, 200} all algorithms show a 100% success rate. For the combinations of {20, 30} switches and {50, 150, 200} streams, all non-harmonic variants perform worse, especially for 800 streams, while the others provide a (nearly) 100% success rate. The GA classes present a higher success rate for non-harmonic experiments. The illustrations show that sorting streams does not bring any advantages for non-harmonic use cases regarding the success rate.

C. GCL AND BANDWIDTH CONSIDERATION

Since network devices have limited memory and often a simple logic, the number of resulting GCL entries is significant. At this point, we remind the reader that the number of entries calculated for critical traffic is only one part of the actual configuration. To evaluate the configuration, we have observed the maximum number of the resulting critical slots among all egress ports. This number determines the necessary dimensioning of the hardware. After the scheduling and configuration step, the number of GCL entries can be extracted for each egress port from the results. Fig. 18a shows that the number of entries rises with the number of switches and streams for GCD and Fig. 18b for HYPO classes. All GCD-based variants provide lower numbers than HYPO algorithms. Foremost, alternating variants, particularly in conjunction with the GA, show the best performance regarding maximum GCL entries. Non-harmonic algorithms outperform the harmonic and HYPO-based variants. One reason could be that they can better exploit unused slots due to


(a) Success rate for {3, 5, 10} switches.


(b) Success rate for 20 switches.

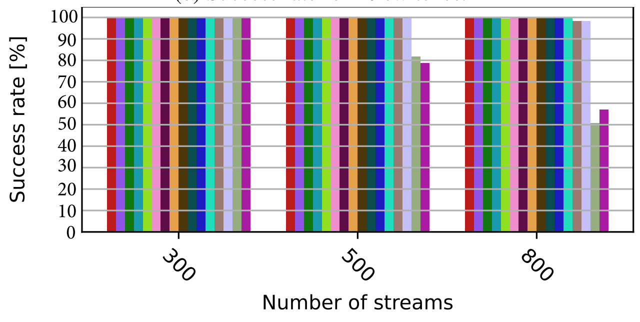
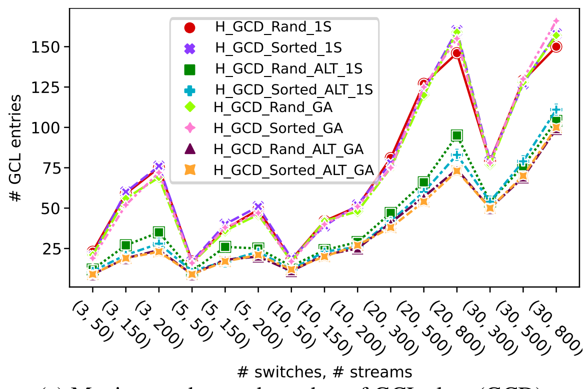
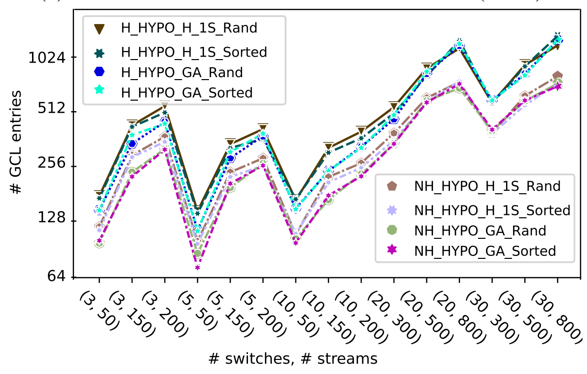
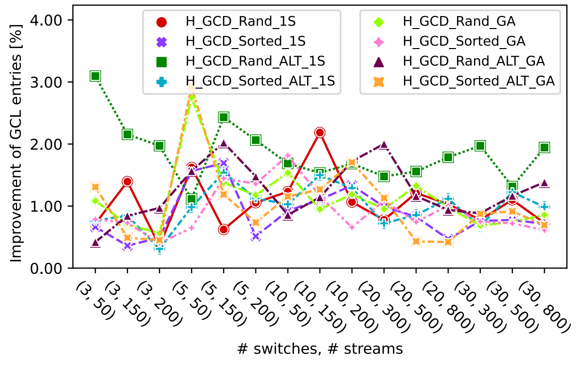

(c) Success rate for 30 switches.

FIGURE 17. Success rates of the experiments in dependence on the number of switches and streams.


(a) Maximum observed number of GCL slots (GCD).


(b) Maximum observed number of GCL slots (HYPO).


(c) Percentage decrease of the number of GCL slots (GCD).

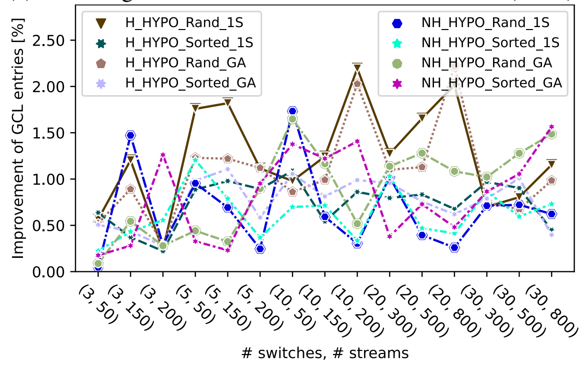

(d) Percentage decrease of the number of GCL slots (HYPO).

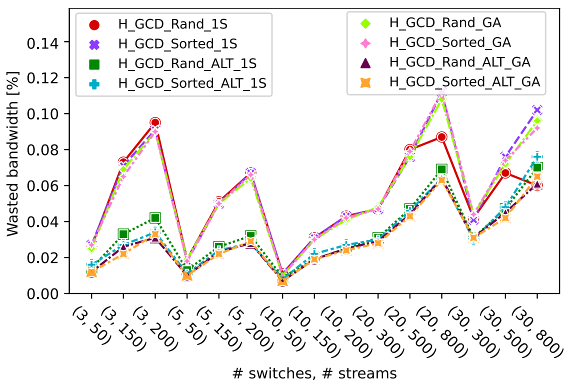
FIGURE 18. GCL metrics for a varying number of switches and streams.

odd periods in some cases. Sorting variants perform similarly to non-sorting variants. Moreover, we have investigated the percentage improvement/decrease of the maximum number of necessary GCL slots in an egress port without ($gclMaxWC$) and with schedule compression ($gclMaxC$):

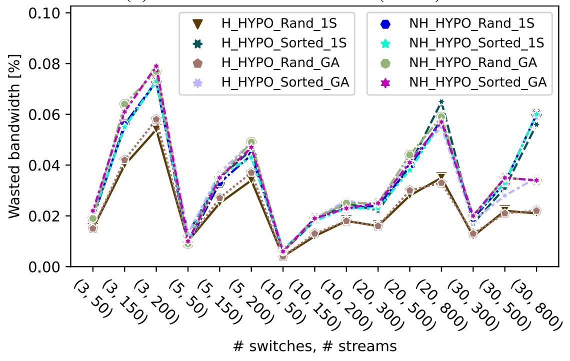
$$impGclMax\% = \frac{gclMaxWC - gclMaxC}{gclMaxWC} * 100\%.$$

Fig. 18c illustrates $impGclMax\%$ for GCD and Fig. 18d for HYPO variants. The compression algorithm can improve GCD-based approaches by up to 4%, which means that less critical transmission slots are needed. The improvement of the number of GCL slots has more effect on algorithms using hyperperiods because the number of GCL entries is significantly higher for them.

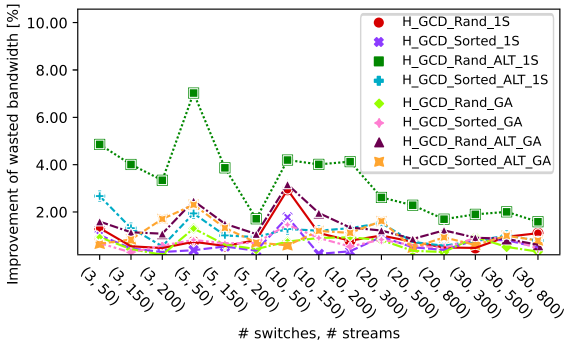
Fig. 19a shows the bandwidth wastage for GCD and Fig. 19b for HYPO classes. This value represents the bandwidth wastage in the whole network over all used ports, calculated in the $sumUpWasted$ step in Fig. 1. Since we have several similar experiments, we take the maximum of all recorded values for similar tests. In this case, all HYPO variants perform better than GCD variants without alternating, particularly NH_HYPO_Rand_1S and H_HYPO_Rand_1S. Other HYPO algorithms and GCD alternating 1S and GA variants are located in the middle.



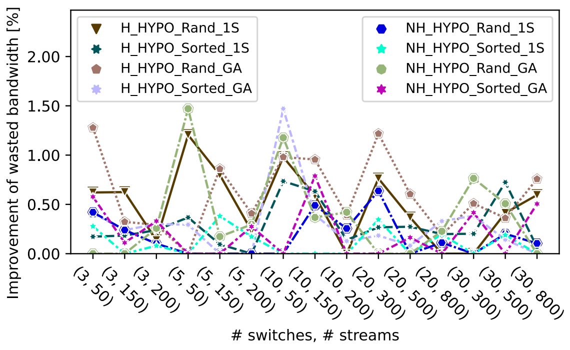
(a) Results for *wastedWC* (GCD).



(b) Results for *wastedWC* (HYPO).

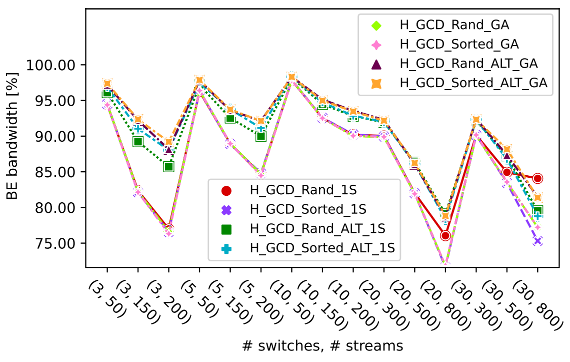


(c) Results for *impWasted* (GCD).

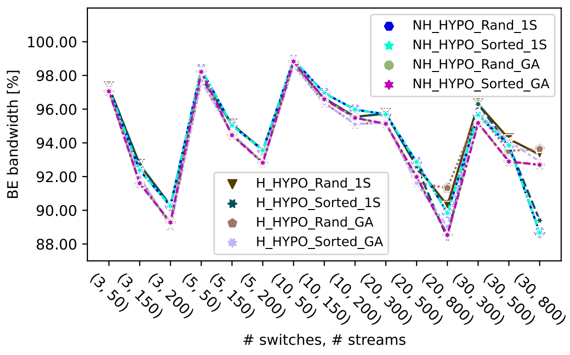


(d) Results for *impWasted* (HYPO).

FIGURE 19. Examination of bandwidth wastage for a varying number of switches and streams.



(a) GCD variant.



(b) HYPO variant.

FIGURE 20. Left-over bandwidth for other traffic classes for a varying number of switches and streams.

Furthermore, we considered the impact of the compression algorithm on the wasted bandwidth. We denote the wastage without compression as *wastedWC* and the wastage after compression as *wastedC*. The improvement (*impWasted*%) is described with:

$$impWasted\% = \frac{wastedWC - wastedC}{wastedWC} * 100\%.$$

Fig. 19c illustrates that the compression algorithm can improve the wasted bandwidth by up to 7%, particularly for H_GCD_Rand_ALT_1S. The compression technique shows only little merit for HYPO classes, as illustrated in Fig. 19d.

Another interesting metric is the left-over bandwidth for other traffic classes. Therefore, we have calculated the left-over bandwidth in the whole network. In general, all HYPO-based algorithms show a higher available bandwidth followed by the H_GCD_Rand_ALT_GA and H_GCD_Sorted_ALT_GA approaches, see Fig. 20. The non-alternating GCD algorithms H_GCD_Rand_1S, H_GCD_Rand_GA, H_GCD_Sorted_1S, and H_GCD_Rand_GA show the worst performance. Also in this scenario, alternating approaches show a high benefit. However, with a higher number of critical GCL slots, we have a higher number of non-critical slots and more guard bands that might prevent the transmission of non-critical traffic. The actual available bandwidth has to be observed analytically or by using simulation. The maximum observed utilization on a link for 800 streams and 20-30 switches was about 73%, while we received a low utilization of up to 12% for small networks. The average utilization was lower.

Last but not least, we have compared the resulting makespans of feasible solutions on the one hand for all

TABLE 3. The percentage increase of the makespan compared to the best solution H_GCD_Sorted_GA.

Experiment	Percentage degradation
H_GCD_Rand_GA	0.51 %
H_HYPO_Sorted_GA	0.13 %
H_HYPO_Sorted_1S	0.34 %
H_HYPO_Rand_GA	0.49 %
H_HYPO_Rand_1S	0.71 %
H_GCD_Sorted_1S	1.31%
H_GCD_Rand_1S	2.56%
H_GCD_Sorted_Alt_GA	30.43%
H_GCD_Rand_Alt_GA	31.29%
H_GCD_Sorted_Alt_1S	32.67%
H_GCD_Rand_Alt_1S	33.72%

TABLE 4. The percentage increase of the makespan compared to the best solution NH_HYPO_Sorted_GA.

Experiment	Percentage degradation
NH_HYPO_Rand_GA	0.88%
NH_HYPO_Sorted_1S	1.45%
NH_HYPO_Rand_1S	3.43%

harmonic classes (Fig. 3) and on the other hand for non-harmonic (Fig. 4). The tables illustrate the percentage increase of the makespan compared to the best results. H_GCD_Sorted_GA provides the smallest makespan over all harmonic experiments and represents the baseline for Fig. 3. The alternating versions provide the worse makespan, which is due to the load-balancing strategy and not the first-fit selection of the GCD. For non-harmonic periods, NH_HYPO_Sorted_GA features the best makespan and results in the baseline for Fig. 4.

D. DISCUSSION

In our research, we operate under the assumption of ideal clocks. Yet, drifts between device clocks are typical in practical settings. The IEEE 802.1AS-2020 specification [9], designed for TSN, guarantees a clock deviation no greater than 1 µs between the master and slave clocks. By simply adjusting the device-to-device delays, this variance can be seamlessly integrated into the system model, ensuring the relevance and applicability of our proposed algorithms in real-world contexts.

Furthermore, we have not taken into account the time precision of the devices. At present, our implementation assumes a one-nanosecond precision. However, we can incorporate the respective time precision into the system model by adjusting the time scale.

Moreover, our study has primarily focused on the unicast mode of transmission, overlooking multicast streams. While our algorithms are capable to manage multicast streams, we chose to emphasize unicast streams to facilitate the description of the system model.

Additionally, our analysis focused on the wasted bandwidth within critical slots only. In practical scenarios, there's a likelihood of encountering further bandwidth wastage due to guard bands assigned for non-critical traffic classes. This introduces the argument that an increased number of configured GCL entries could amplify bandwidth wastage, potentially diminishing the efficiency of the HYPO variants. However, such an examination must be undertaken either through simulation or analytical methods, given our limited insights into the behavior of non-critical traffic, which also varies significantly across different application domains.

VII. CONCLUSION

We have presented a set of heuristic and meta-heuristic algorithms in this paper that schedule critical TSN streams in a no-wait approach, minimizing the jitter and end-to-end latency of the streams and the makespan of the schedule in GA-based approaches. The algorithms apply different modifications to input data ordering, the scheduling procedure, and GCL configuration. This has an impact on the number of necessary GCL slots and bandwidth utilization. We have evaluated the different scheduling approaches regarding the runtime, success rate, maximum number of necessary GCL entries, as well as bandwidth utilization and wastage. If we have only harmonic periods and compliance with the maximum end-to-end deadline is sufficient then alternating approaches are suitable to provide a low number of GCL entries and lower bandwidth wastage. Alternating applies a load-balancing approach that allows a better utilization of the bandwidth and a better re-usage of GCL slots. Furthermore, if runtime is not a crucial factor then GA variants in conjunction with alternating deliver the best results for bandwidth statistics and GCL configurations when using harmonic periods. In general, all GA variants in harmonic and non-harmonic branches perform better on average considering all metrics but the runtime. Sorting versions do not show significant merit compared to non-sorted approaches. Generally, GCD-based algorithms are helpful for reducing the necessary number of GCL entries. Harmonic and non-harmonic approaches combined with HYPO show significantly better results regarding the wasted and left-over bandwidth compared to GCD variants without alternating but can result in a high number of necessary GCL entries for larger topologies. In this case, the limitations of hardware have to be carefully considered and the slot merging approach has to be adapted to consider a limited number of GCL entries. Moreover, non-harmonic algorithm classes show a smaller success rate for large topologies than all other algorithms. Also, non-harmonic algorithms classes do not benefit much from sorting approaches, while GA additions help to reach a higher success rate for larger topologies. Furthermore, the schedule revealed a slight improvement in the results in individual cases.

In summary, the evaluations show that the modifications of scheduling algorithms have a high impact on the network resource usage, particularly for harmonic algorithm classes, and that accurately designed heuristic algorithms are well suited to solve scheduling problems and beyond. These considerations are often neglected in the literature.

In the future, we intend to embed the clock deviation and precision into our system model. Moreover, we plan to investigate bandwidth utilization with a simulation environment to analyze the impact of the TAS configuration resulting from our suggested algorithms on the non-critical network traffic.

REFERENCES

[1] *Time-Sensitive Networking (TSN) Task Group*. Accessed: Aug. 7, 2023. [Online]. Available: https://1.ieee802.org/tsn/

[2] M. Barreiros and P. Lundqvist, *QOS-Enabled Networks: Tools and Foundations*. Hoboken, NJ, USA: Wiley, 2011.

[3] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic*, IEEE Standard 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as Amended by IEEE Standard 802.1Qca-2015, IEEE Standard 802.1Qcd-2015, and IEEE Standard 802.1Q-2014/Cor 1-2015), Mar. 2016.

[4] C. Stadler, F. Montanari, W. Baron, C. Sippl, and A. Djanatliev, "A credibility assessment approach for scenario-based virtual testing of automated driving functions," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 45–60, 2022.

[5] 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.*, New York, NY, USA, Oct. 2016, pp. 203–212.

[6] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*, IEEE Standard 802.1Qcc-2018 (Amendment to IEEE Standard 802.1Q-2018 as amended by IEEE Standard 802.1Qcp-2018), Oct. 2018.

[7] L. James, *On-Board and Off-Board Data Platforms: Innovation Report*. Coventry, U.K.: Univ. Warwick, 2019. [Online]. Available: https://books.google.de/books?id=IYt-zwEACAAJ

[8] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks*, IEEE Standard 802.1Q-2005 (Incorporates IEEE Standard 802.1Q1998, IEEE Standard 802.1u-2001, IEEE Standard 802.1v-2001, and IEEE Standard 802.1s-2002), 2006.

[9] *IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications*, IEEE Standard 802.1AS-2020 (Revision of IEEE Standard 802.1AS-2011), 2020.

[10] 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.

[11] 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*. Berlin, Germany: Springer, Mar. 2020, pp. 99–117.

[12] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley, 1989.

[13] J.-Y. Potvin, "Genetic algorithms for the traveling salesman problem," *Ann. Oper. Res.*, vol. 63, no. 3, pp. 337–370, Jun. 1996.

[14] 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.*, New York, NY, USA, 2016, pp. 183–192.

[15] X. Jin, C. Xia, N. Guan, C. Xu, D. Li, Y. Yin, and P. Zeng, "Real-time scheduling of massive data in time sensitive networks with a limited number of schedule entries," *IEEE Access*, vol. 8, pp. 6751–6767, 2020.

[16] Q. Li, D. Li, X. Jin, Q. Wang, and P. Zeng, "A simple and efficient time-sensitive networking traffic scheduling method for industrial scenarios," *Electronics*, vol. 9, no. 12, p. 2131, Dec. 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/12/2131

[17] A. Arestova, W. Baron, K. J. Hielscher, and R. German, "ITANS: Incremental task and network scheduling for time-sensitive networks," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 369–387, 2022.

[18] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems," in *Proc. 19th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2014, pp. 119–124.

[19] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Berlin, Germany: Springer, Apr. 2012.

[20] L. Maile, K.-S. Hielscher, and R. German, "Network calculus results for TSN: An introduction," in *Proc. Inf. Commun. Technol. Conf. (ICTC)*, May 2020, pp. 131–140.

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

[22] 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.

[23] 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. (WATERS)*, 2015. Accessed: Oct. 17, 2023. [Online]. Available: https://www.ecrts.org/forum/download/WATERS15_Real_World_Automotive_Benchmark_For_Free.pdf

[24] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed. Berlin, Germany: Springer, 2011.

[25] A. Arestova, L. Maile, N. Halikulov, K.-S. Hielscher, and R. German, "Joint task and flow scheduling for time-triggered and strict-priority networks," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Feb. 2023, pp. 608–614.

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

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. His research interests include measurement, modeling, and simulation of distributed systems, and deterministic and stochastic network calculus. He is currently working as a Postdoctoral Researcher at the Department of Computer Science (Computer Networks and Communication Systems), University of Erlangen-Nürnberg.

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 currently a Full Professor at the Computer Networks Laboratory, Department of Computer Science, University of Erlangen-Nürnberg, Germany. He is also an Adjunct Professor at the Faculty of Information Technology, Monash University, Melbourne, VIC, 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.