Leveraging Flexibility of Time-Sensitive Networks for dynamic Reconfigurability

Christoph Gärtner*, Amr Rizk¶, Boris Koldehofe‡, Rhaban Hark*, René Guillaume§ and Ralf Steinmetz*
*Technical University of Darmstadt, Germany, {christoph.gaertner, rhaban.hark, ralf.steinmetz}@kom.tu-darmstadt.de
¶University of Duisburg-Essen, Germany, amr.rizk@uni-due.de
‡University of Groningen, Netherlands, boris.koldehofe@rug.nl
§Robert Bosch GmbH, Corporate Research, rene.guillaume@de.bosch.com

***Abstract*—In Time-Sensitive Networks (TSN) applications with the highest real-time flow requirements are deployed using the Time-Aware Shaper which requires careful planning and scheduling of flows before deployment. Such deployments lack support for dynamic industrial scenarios such as modular machine assembly and reconfiguration, which require a flexible transition between real-time tasks. In contrast, state-of-the-art techniques rely on flow rescheduling and deployment in conjunction with undesired network downtime. Existing works on adapting schedules to traffic admissions are limited in their ability to choose suitable flows to account for future tasks.**

In this paper, we aim to leverage the flexibility of scheduler configurations to enable TSN dynamic reconfigurability at run-time. We propose a notion of flexibility for TSN Time-Aware Shaper schedules which we utilize to decide the admissibility of consecutive real-time tasks.

***Index Terms*—TSN, IEEE802.1Qbv, flexibility, scheduling**

I. Introduction

Networks supporting real-time requirements are becoming increasingly important in industrial scenarios, e.g., to couple and control manufacturing processes. Since those processes span and control end-devices which physically interact, stringent real-time requirements are essential, but also challenging to meet. In the context of Time-Sensitive Networks (TSN) real-time guarantees on packet flows are enforced by mechanisms that explicitly avoid conflicts in the use of network resources in form of real-time schedules. One such mechanism is the IEEE 802.1Qbv Time-Aware Shaper (TAS), which supports time division multiplexing at egress ports for so-called *scheduled traffic* through specifying times at which the port is exclusively open for certain traffic class queues. This cyclic schedule is often referred to as gate control list (GCL), which contains the times the traffic queue gates are open. Thus, Time-Aware Shaping allows no-jitter isochronous traffic streams.

Deploying real-time applications with Time-Aware Shaping requires a careful planning before deployment. For each flow, properties need to be specified ahead such as the path between source/destination, the periodic behavior of the traffic bursts in form of a cycle time and the size of the payload. The feasibility of schedules is hence tested ahead and appropriate schedules are determined by offline techniques for solving variants of the constraint satisfaction problem. These static schedules are not designed to be adapted at runtime.

ISBN 978-3-903176-39-3

In this paper, we propose TSN support for dynamic industrial scenarios, where the flow characteristics change over time and the path characteristics may also be subject to change. Changes in flow characteristics are naturally given by changes in the tasks carried out by the real-time application. Here, flows may only exist for a fixed period of time in the network or flow properties such as cycle or frame size may also change upon certain events. Examples of dynamic industrial scenarios are given in [1], e.g., for dynamic plugging and unplugging of machines or modular machine assembly. A particular difficulty that we tackle in this paper is considering TAS for dynamic reconfigurations at runtime [2]. This is challenging as providing no-jitter isochronous flow guarantees in the form of TAS in a dynamic scenario is hard.

Existing approaches to support dynamic industrial scenarios update real-time flows at run-time by taking snapshots of flow properties/requirements and solving the entire scheduling/routing optimization problem before redeploying the GCLs. A key drawback is that such changes to the industrial network are usually implemented through halting/updating and restarting the network. Hence, in addition to solving the scheduling/routing optimization problem (which can be computed as soon as changes become known) the associated network downtime cannot be circumvented. Given a-priori known flow changes and requirements, the alternative approach carried out by the industry is to assume the worst case in terms of requirements and hence, allocate static *gate control entries* to all scheduled traffic flows even if these should only exist for a finite duration in the network. The key drawback is the unnecessary and potentially unbounded waste of capacity in addition to the difficulty of modelling the worst case in scenarios with complex machine interactions. New flows or changes to the statically configured network can only be integrated through a new schedule embedding which is associated with a down time.

In this paper, we aim to improve the flexibility at which the configuration of TSN-networks can be adapted to dynamic changes of real-time applications. Our contribution in this paper is the introduction of a flexibility notion (flexibility curve model and algorithms) for TAS schedules that allows

1) dynamic reconfiguration of TAS schedules at runtime
2) admitting multiple flows across a TSN network at once

without the need to recompute these schedules.

II. Related Work

In the following, we briefly review the related work specifically with respect to *(i)* calculating and reconfiguring schedules for TSN, *(ii)* models that allow obtaining performance guarantees for real-time traffic flows and finally *(iii)* notions of flexibility and adaptivity in resource allocation problems.

Creating static schedules to populate the gate control lists of TSN switches before deploying any real-time applications in the network can be carried out by solving a constraint satisfaction problem such as in [3], [4]. These works use Satisfiability Modulo Theories (SMT) solvers to generate schedules that meet a-priori specified flow requirements. Similarly, the authors of [5] use integer linear programming (ILP) to find a valid gate schedule. They introduce the notion of a flowspan, which indicates the time until all flows are delivered within each cycle, which grows when new flows are admitted on the limiting path. The authors search for schedules that minimize the flowspan using an ILP solver. Note that the assumption in [5] is that all flows possess the same cycle time. The approach is designed to focus GCL events at the beginning of each cycle, thus not directly applicable to answer the question of embedding one or more flows at once.

Changing flow characteristics can require the incremental addition of a set of flows. The work in [6] is closest to ours as it enables TSN reconfiguration at runtime using a heuristic. A requested admission of a new flow invokes the execution of a scheduling heuristic, which tries to find a valid schedule for *individually added* flows. If this is unsuccessful, the heuristic builds an entirely new schedule, e.g. through invoking classical scheduling approaches, which may discard existing flow timings. Note that the heuristic allows for only one flow-admission at a time, if multiple flows are to be added, another heuristic is needed that orders flows for an incremental admission. The work in [7] aims at the reconfiguration of GCLs by deciding if scheduled traffic gate windows can be populated with additional flows or are allowed to be extended. In contrast to our work added flows change the behavior within the time window for scheduled traffic, therefore affecting *all flows* within the traffic window.

Network Calculus based models such as [8]–[10] capture various properties of different TSN mechanisms such as IEEE 802.1 Ba/AS/Qat/Qav/Qcr very well. Service Curves for TSN schedulers enable the derivation of deterministic delay bounds given known constraints on the flow burstiness. Complementary to these works we consider here isochronous flows that are scheduled using the Time-Aware Shaper as given in IEEE 802.1Qbv. The key differences that make modeling such a scenario using Network Calculus hard are that *(i)* flows require contiguous service and *(ii)* flows need to arrive in an isochronous form at the destination. The first key difference becomes clear when considering the so-called strong service guarantee [11] (also known as strict service curve [8]) that describes the service using a function $S(t)$ that fulfills the relationship $D(t) \geq D(\tau) + S(t-\tau)$ for all $0 \leq \tau \leq t$ with $D(t)$ being the cumulative departures of a system (e.g. buffered link or scheduler). The argument made by a strict service curve is not directly applicable to the TAS flow admission, i.e., the value the service curve at $t-\tau$ does not solely decide whether a flow that requires service of length less than $t-\tau$ is contiguously admissible. The second key difference relates to the type of guarantee that can immediately be obtained from network calculus models. While obtaining delay bounds is well understood given a deterministic network calculus model [8], guarantees on isochronicity are not.

As we define a notion of flexibility in this paper, we note that previous works that analyze resource allocation algorithms in the context of shop floor/production [12] or network virtualization scenarios [13] provide viable starting points for this goal. In a nutshell, these works measure flexibility as a ratio of what we call weighted execution of tasks to the weight of a measurable task set. Now, the interpretation of this task set as well as its execution depends on the context of the flexibility notion. In [12] the flexibility of a production system is measured in terms of the ratio of the integral of the machine-task efficiency rating with respect to the weight-density function of tasks normalized by the overall task weight. In [13] flexibility is generalized towards network virtualization with a "weighted execution" that depends on the time and cost of introducing changes to virtualized network resources with a corresponding appropriate normalization. These works, however, do not lend themselves easily to model the queueing behavior that we allow at the TAS. Essentially, the *computation* of such a flexibility measure under a combination of queueing and scheduling remains very hard.

III. Problem description

In this paper, we consider a dynamic network centric real-time application, comprising a set of dynamically deployable real-time tasks $T = \{t_1, \ldots, t_l\}$. We use the number of tasks $l \in \mathbb{N}$ to define the index set $[l] := \{1, 2, \ldots, l\}$. Each task t_i will impose a set of flows $F_i = \{f_{i1}, \ldots, f_{im}\}$, which defines the network traffic necessary to perform the task. The task set T may be given as a directed acyclic graph that describes the temporal behavior of the application. In the following, we consider a simple totally ordered task set T.

Each flow f_{ij} is associated with a specific real-time requirement and task t_i can execute only if all of its flows can be admitted to the real-time network. Hence, the network needs to admit a sequence of flow sets $(F_i)_{i\in[l]}$ in order for the real-time tasks of the real-time application to make progress.

As depicted in Fig. 1 we consider real-time applications that dynamically introduce new flows f_{ij} or new flow sets F_i depending on the current application state and goal. Such real-time applications abstract dynamic use cases in industrial automation such as dynamic plugging and unplugging of machines or modular machine assembly that are mentioned in [1]. Fig. 1 shows a controller that is responsible for calculating the schedules at every output port at every switch and translating these to gate control lists that control the exclusive mapping of switch ports to output queues. Admitting a new flow f_{ij} of t_i

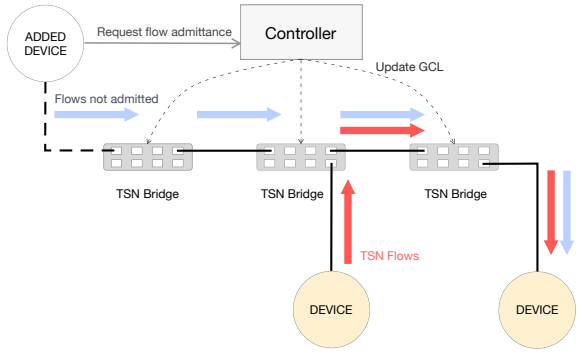


Fig. 1: New Devices request the admittance of new flows at the central controller that calculates a possible embedding and updates the global and local schedule of the network.

to the network requires a careful analysis whether a resulting schedule is meeting the real-time requirements of all admitted flows. To understand the overhead let's consider the following scenarios engaged in the admission process:

Consider first, there is sufficient capacity available. Recall that flows require contiguous transmission slots and cannot be split in time. In this scenario the controller can aim to determine a complimentary schedule to integrate the new flow. If not sufficient capacity is available then the controller may analyze whether a new schedule will allow to integrate the flow. Note that deploying the new schedule requires stopping the running tasks which leads to a down time of the network. Alternatively, the task t_j could be suspended until sufficient network resources of the network become available.

Overall, we assume in this paper that a controller of the real-time application aims to maximize the throughput in executing real-time tasks over the real-time network. A critical issue, required in all discussed scenarios, is to support the controller in detecting that one or more flows can be admitted. In this paper we propose a model that efficiently supports this decision and reduces the time until new flows can be admitted.

IV. System Model

In this section, we present a model of a time-sensitive network with TAS that lends itself to the derivation of a notion of flexibility for TAS schedules. We also provide a short review of schedule calculation for IEEE 802.1Qbv.

A. Network model

We model a given time-sensitive network as a graph $G(V, E)$. The real-time capable devices in V comprise output queued switches and end-devices, each with at least one Time-Sensitive Networking (TSN) scheduling mechanism enabled. An example of such scheduling mechanism is IEEE 802.1Qbv. The real-time capable devices are connected by bi-directional symmetrical links in E. Cyclic or isochronous real-time traffic belonging to a task t_i is described by the set of flows F_i where each flow $f_{ij} \in F_i$ is associated with a specification that characterizes the real-time behavior in terms of requirements and properties of end-to-end communication. Flow properties include the cycle time, source and destination node pair and payload size. The requirements describe the allowed packet loss per time-period, allowed jitter and an end-to-end delay bound. A scheduled TSN network using TAS (IEEE 802.1Qbv) has at each vertex $v \in V$ a set of output ports where each port i has a schedule s_i resulting in an set of network schedules $S = \{s_1, \ldots, s_N\}$. Overall there exists N scheduled output ports in the TSN network. As mandated by IEEE 802.1Qbv we assume time synchronization at all vertices and consider time to be slotted in slots of size Δ. The set of schedules is computed in a way that each flow constraint, described by F is met. Each schedule $s_i \in S$ is a sequence of pairs where a pair consists of an isochronous flow and an associated departure time on the port i within the *scheduling hyperperiod* $K\Delta$, i.e., the least common multiple of all flow cycle times. For a clear exposition we omit the transmission and propagation delays in the following.

B. Schedule Calculation for IEEE 802.1Qbv

IEEE 802.1Qbv controls the opening times of up to eight separated queues for each controlled output port. In our network model we assume a common network cycle time, which is equal to the flow-hyperperiod $K\Delta$. Since the GCL's granularity is limited we can assume a time slotted behavior for opened and closed queues. Therefore, each GCL encodes a part of the global schedule, which opens the corresponding queue gate when a flow is scheduled for egress.

In this work, we use state-of-the-art approaches to calculate TAS schedules. Given an empty network with a set of initial flows, we create a schedule, that satisfies the flow requirements, including a predefined routing, and the switch/port limitations, by solving a constraint satisfaction problem, similar to existing related work reviewed in Sect. II.

V. A Notion of Flexibility for Schedules

Given the tuple (G, F_i, S) describing a valid mapping of F_i to G using schedules in S. We denote L the set of paths that aggregate local port schedules between every distinct sender/receiver device pair, i.e., the path $l_p \in L$ is a sequence of one or more ports that are along the same end-to-end path between a given sender/receiver pair. Here p is the index of the path. Hence, we consider paths that carry one or more flows between given sender/destination pairs. The schedule of port i is given by $s_i \in S$ where s_i denotes a sequence of time points $(s_{ij})_{j \in [\chi_i]}$ for which the flows obtain exclusive use of the output port. Here, χ_i denotes the number of flows in schedule s_i. We define for each schedule s_i the number of free-slots after the last transmission as ϕ_i (each of width Δ) and the end-shifted sequence of departure times as

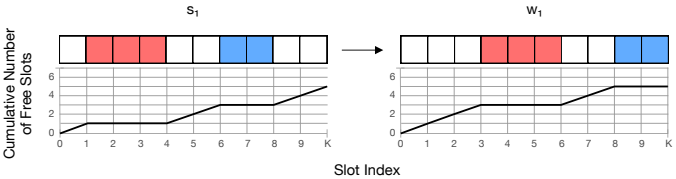


Fig. 2: (Left) An example schedule s_1 with $C_p(n)$ underneath. (Right) The end-shifted schedule w_1 of s_1.

$w_i = (s_{i1} + \phi_i, \ldots, s_{i\chi_i} + \phi_i)$. Fig. 2 depicts an example of such a sequence.

Given a schedule length, i.e., a hyperperiod of length $K\Delta$, we denote the cumulative capacity for port p up to slot index n by $C_p(n) = \sum_k \Delta 1_{\{n \geq a_{pk}\}}$ with a_{pk} being the time point of the k-th free slot at p with slot index $n \in [K]$. Note we use the end-shifted schedule w_i at each port p for the cumulative capacity. This allows considering scheduling wraps.

Next, we propose the following flexibility notion. We define the *flexibility curve* (flexcurve) for path l_p as

$$h_p(n) = \min_{k \in l_p} \sum_{\tau=0}^{K-n} 1_{\{C_k(n+\tau) - C_k(\tau) = n\}}. \quad (1)$$

Note that the flexibility curve is limited by the minimum value of possible flow arrangements for new flows along its path.

The rationale behind the flexcurve: It provides the number of possible arrangements for a flow of size $n\Delta$ at the corresponding bottleneck schedule along a given path between two endpoints. The flexibility curve denotes the "embeddable" flow sizes along the path l_p for cycles of size $K\Delta$ and unconstrained end-to-end delay *without* recalculating the schedules for the existing flows along that path. Note that it does not reflect potential flow queue isolation requirements.

Note that the flexcurve considers all potentially overlapping arrangements at each schedule. Restricting the flexcurve to only reflect non-overlapping arrangements, which would directly allow parallel flow embedding, would reduce the encoded information. Through counting overlaps the flexcurve encodes information on the capacity leftover after embedding a flow. The flexcurve encodes various information such as $h_p(1)$ which denotes the overall *residual free capacity* along a path or $n_{\max,p} = \arg\min_n h_p(n)|_{h_p(n)>0}$, which is denoted the maximum embeddable flow size on path p.

Numerical Example

Considering the topology of Fig. 3a as a store and forward network we define the following flows:

Flow	Size	Cycle	Max Delay
Flow 0 (red)	4Δ	20Δ	$\infty\Delta$
Flow 1 (green)	2Δ	20Δ	10Δ
Flow 2 (blue)	2Δ	20Δ	10Δ
Flow 3 (orange)	2Δ	10Δ	10Δ

Assume first that Flow 0 is not present yet. Without Flow 0 (red), a scheduler computed the output port schedules in Tab. I with regard to individual flow and queue availability constraints with a resulting hypercycle of 20Δ.

Port	Schedule
4	
5	
6	
7	

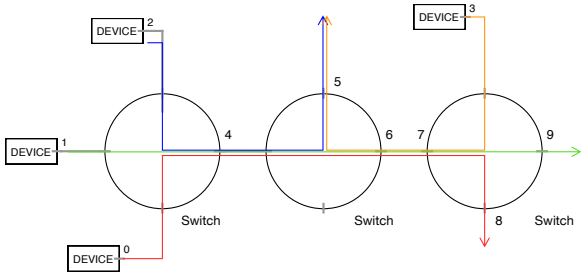
TABLE I: Port schedules for the numerical example.

With the given port schedules, we can calculate the flexibility curve for each current and future (Flow 0) path. Note that flexcurves for different non-disjoint paths might be identical if the corresponding bottlenecks are located in the non-disjoint part. In this example, this is the case for Flow 2 (blue) and Flow 3 (orange) where both have identical flexcurves for all embeddable flow sizes because both paths are constrained by the same port (port 5). For Flow 1 (green), the limiting schedule is located at Port 4.

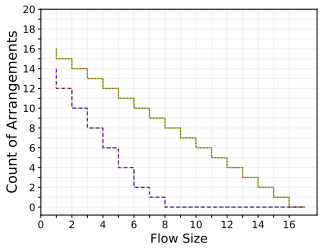
Given the schedule in Tab. I, the resulting flexcurves are visualized in Fig. 3b. There are three flexcurves, with two overlapping. Comparing flexcurves directly allows us to compare the immediate *embeddability* of additional flows along the path of each flexcurve. For instance, there are 10 possible arrangements for an additional flow of size 7Δ along the path $(1, 4, 6, 9)$. However, only two along the paths $\{(2, 4, 5), (3, 7, 5)\}$. Embedding additional flows might impact the flexcurves of all intersecting paths. Affected flexcurves have reduced flexibility because additional flows limit the capacity somewhere along the path.

Calculating the flexcurve along the red flow's path $(0, 4, 6, 8)$ we directly see that it allows for the immediate embedding of Flow 0 (red). Depending on the actual embedding, i.e., where the flow is assigned at the schedule of each port, the flexcurves are differently affected. Intuitively, this is because additional fragmentation within a schedule reduces the ability to deploy larger flows after the addition.

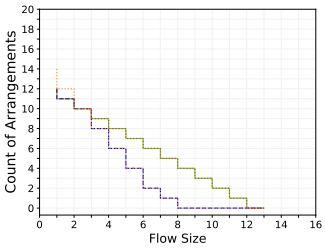
The path of Flow 0 (red) has its bottleneck at port 4. Scheduling the red flow on port 4 directly after Flow 1 (green) in Tab. I, reduces $h_{(0,4,6,8)}(1)$ to 12Δ, which is equal to the



(a)



(b)



(c)

Fig. 3: (a) Example routes and topology with four flows (red, green, blue, orange) and three switches. Output port numbers are labeled. (b) The resulting flexibility curves for all given paths of each schedule. The flexcurve for path of Flow 2/3 and Flow 0/1 overlap. (c) Flexibility curves after the addition of Flow 0 (red) directly after Flow 1 (green). The capacity loss is shifting the flexibility curve of the path of Flow 1 (green) downwards, and creating new limits for the path of Flow 2 (blue).

	Runtime (in ms)					
Added Flows	initial	$+1$	$+2$	$+3$	$+4$	$+5$
Flexcurve	2.48	2.37	2.32	2.47	2.76	2.81
SMT	48.30	62.00	77.72	106.36	137.90	171.09

TABLE II: Given Tab. I we compare the runtime of the creation of the flexcurve against a constraint solving approach (SMT) with an increasing number of added flows.

path's limiting residual capacity. However, the introduction of the red flow also changes the blue path's flexcurve, since the residual capacity and therefore limiting schedules for it also change. Until $n \leq 3$ the blue flexcurve is limited by port 4, larger flows have their bottleneck at port 5, from there the flexcurve follows values of orange's flexcurve.

VI. Applications

In this section we consider two applications of the notion of flexcurves in the context of dynamic reconfiguration of TSN networks with TAS. The first application is a basic admission control of one or multiple new flows at once while the second application revolves around using the flexcurve as an optimization constraint when calculating flow routing in case multiple paths between sender/destination pairs exist.

A. Admission Control

Deciding admissibility is required when a real-time application transitions from one task to the next in Sect. III. Being able to quickly decide which tasks can be admitted reduces computational waiting times. As the residual capacity $h_p(1)$ does not reflect contiguous free slots it is not sufficient to state the admissibility of a flow of size $n\Delta$ with $n > 1$.

In the following, we sketch how to use the flexibility curve metric to assess the possibility of future flow admissions. The flexibility curve is defined such that it immediately shows a number of arrangements for flow-admissions along a path for flows with unconstrained end-to-end delay, and with a cycle time equal to $K\Delta$. After the initial schedule creation, the flexibility curve can be precomputed providing information on the embeddability of future flows. This renders resource exhausting recalculations of the schedule unnecessary and allows rapid insertions of new flows. Note that the creation of the flexibility curve possesses time and space requirements that increase with the hyperperiod and the number of hops. Tab. II shows the runtime requirements of creating all four flexibility curves from the example in Sect. V, and creating a corresponding schedule by solving a SMT problem.

A naive and straightforward approach to check flow admissibility compared to fully scheduling or using flexibility curves is to apply Alg. 1 with a given path and size of the future flow. Alg. 1 checks every port schedule along a flow's path whether it has enough capacity to admit the flow. A minimal modification also returns a possible embedding. A main drawback of this simple approach is that it only applies to tasks with single flows as it is not able to decide admissibility for two or more flows simultaneously. For a simultaneous flow admission, a different search is required which warrants the usage of classical scheduling/optimization approaches.

Algorithm 1: Naive flow admissibility check

```
Data: Flow = (path, size)
Result: True/False : Flow is admissible
forall schedule in path do
    if size is not embeddable in schedule then
        return False;
return True;
```

The flexibility curve, however, is able to answer the admissibility question for multiple flows simultaneously. Here, we only need to regard flows that share common output ports along their path. The possibility for flow admission of path independent flows is given by the path's flexcurve.

A simultaneous embedding of multiple flows given by the set F_e sharing at least one output port is also possible, if all flexibility curves given by the paths of F_e are able to accommodate all flows $f \in F_e$. Hence, we check if

$$\forall f \in F_e : h_p(\sum_{f \in F_e} \text{size}(f)) > 0 \tag{2}$$

is true, where p is the path of f, i.e., whether a continuous gap of free-slots for the sum of all flow sizes F_e in every flow's path exists. Note that the condition above is only sufficient.

Now, if (2) does not hold we can still obtain parallel embedding through considering the set of flows to be embedded F_e and a set of disaggregated flexcurves as detailed in Alg. 2. In the following, we explain how to obtain disaggregated flexcurves to embed a set of flows at once: Considering each flow $f \in F_e$ in a decreasing order of flow sizes we match every flow f_i with a disaggregated flexcurve $h'_{p,i}(n)$. We obtain the ith disaggregated flexcurve $h'_{p,i}(n)$ by recursive subtraction as

$$h'_{p,i}(n) = h'_{p,i-1}(n) - \tilde{h}_p(n) \tag{3}$$

for $i > 1$. The first disaggregated flexcurve $h'_{p,1}(n)$ is calculated by considering the largest embeddable flow size $n_{\max,p}$ on path p and subtracting the canonical flexibility curve

$$\tilde{h}_p(n) = \begin{cases} n_{\max,p} - n + 1 & \text{if } n \leq n_{\max,p}, \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

from the flexcurve $h_p(n)$ as $h'_{p,1}(n) = h_p(n) - \tilde{h}_p(n)$. Observe that we perform the subtraction $h_p(n_{\max})$ times to remove the count of arrangements for all gaps of flow size $n_{\max}$. Note that this method is only applicable if the number of disaggregated flexcurves is larger or equal to the number of flows in F_e. If there exists more flows than disaggregated flexcurves, an embedding might still be possible, but we cannot decide on admissibility without combinatorial approaches. A fallback to classical scheduling approaches is required.

B. Flexibility Optimization and constrained Routing

In the previous subsection, we considered the application of the flexcurve to answering the question whether one or multiple flows can be admitted to a TSN network. Answering the admissibility question entails concretely scheduling the flow,

Algorithm 2: Flow admissibility using flexcurves

```
Data:
// Flows sorted descending by size Flows ← [size];
// Flexcurves of distinct paths Flexcurves ← [h()];
Result: True/False/null: Flows are admissible
// Single Flows are directly applied (omitted)
// Apply Eq. (2) on Flows (omitted)
// Further, flexcurves failing Eq. (2):
FlexcurvesUnsatisfied ← [];
Flowcount ← |Flows|;
for h in FlexcurvesUnsatisfied do
    // Disaggregate flexibility curve
    Disaggregations ← 0;
    repeat
        // Attempt flow placement
        // Create a decoupled schedule
        n_max ← Flexcurve.nmax;
        if Flows[0] > n_max then
            return False;
        Create h(n_max) schedule with K = n_max;
        while Flows can be placed in decoupled schedule do
            Place Flows[0] in decoupled schedule;
            Remove Flows[0];
            if |Flows| = 0 then
                return True;
        // Subtraction of canonical flexcurve
        forall i ∈ {1...K} do
            h(i) ← h(i) − n_max h̃_p(i);
        Disaggregations ← Disaggregations + 1;
    until h(1) = 0;
    if Flowcount > Disaggregations then
        return null;
    else
        return False;
```

i.e., deciding which slots should the flow occupy. Next, we consider this question in the light of the flexcurve definition.

Given the flexcurves $h_p(n)$ that describe the available arrangements for flows of various sizes in a network with flow set F, it is natural to aim at maximizing the flexibility of the entire network. Note that a flow admission on a given path decreases the flexcurves that share the same bottleneck schedule on that path.

To account for future flow embeddings, we propose to schedule the newly admitted flow to maximize flexibility for all distinct paths of a priori known flows, by maximizing the sum of the area under the flexibility curves as to

$$\text{maximize} \sum_{p \in F} \sum_{n} h_p(n) \tag{5}$$

subject to satisfying all individual flow constraints. Certainly, this criterion can also be used at the initial scheduling of flows.

Note that if flow paths are fixed, a flexcurve is maximal, when the fragmentation of each schedule along its path is minimal. This implies that the largest embeddable flow size $n_{\max,p}$ is maximized. An empty path possesses a canonical flexibility curve, cf. Eq. (4), with $n_{\max,p} = K\Delta$.

Building on the flexibility optimization above, a subset of flows belonging to some task that are being admitted may be allowed to be routed along different paths, i.e. different sequence of ports between two given end-points. This joint flexibility optimization and routing problem can be solved using the flexcurve concept to obtain the routes that provide the least loss of flexibility in terms of the criterion from (5).

VII. Conclusion

Transitions between real-time tasks in dynamic industrial scenarios at runtime require flexibility within the deployed TAS schedules. We proposed a network path-based flexibility notion, denoted flexcurve, that is deduced from schedules along a given path. Using the flexcurve we can decide the admissibility of multiple flows at once and assess their impact on the admissibility of future flows. We obtain a notion of the network flexibility by appropriately aggregating the flexcurves of different paths. Flexcurves enables online reconfiguration of schedules, without rescheduling existing flows. In future work we will generalize this concept to arbitrary flow cycle times, queue isolation and different delay guarantees.

VIII. Acknowledgment

This work has been funded in parts by the German Research Foundation (DFG) as part of projects T3, B4 within the Collaborative Research Center (CRC) 1053 – MAKI as well as the DFG project SPINE.

References

[1] R. Belliardi, J. Dorr, T. Enzinger, F. Essler, J. Farkas, M. Hantel, M. Riegel, M.-P. Stanica, G. Steindl, R. Wamßer, K. Weber, and S. A. Zuponcic, "Use Cases IEC/IEEE 60802 v1.3," Sep. 2018.

[2] B. Alt, M. Weckesser, C. Becker, M. Hollick, S. Kar, A. Klein, R. Klose, R. Kluge, H. Koeppl, B. Koldehofe, W. R. KhudaBukhsh, M. Luthra, M. Mousavi, M. Mühlhäuser, M. Pfannemüller, A. Rizk, A. Schürr, and R. Steinmetz, "Transitions: A Protocol-Independent View of the Future Internet," *Proc. IEEE*, vol. 107, no. 4, pp. 835–846, 2019.

[3] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," in *Proc. of ACM RTNS*, 2016, pp. 183–192.

[4] A. Santos, B. Schneider, and V. Nigam, "TSNSCHED: Automated schedule generation for time sensitive networking," in *Proc. of Formal Methods in Computer Aided Design (FMCAD)*, pp. 69–77.

[5] F. Dürr and N. G. Nayak, "No-Wait Packet Scheduling for IEEE Time-Sensitive Networks (TSN)," in *Proc. of ACM RTNS*, 2016, pp. 203–212.

[6] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling Fog Computing for Industrial Automation Through Time-Sensitive Networking (TSN)," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 55–61, 2018.

[7] A. Nasrallah, V. Balasubramanian, A. Thyagaturu, M. Reisslein, and H. ElBakoury, "Reconfiguration Algorithms for High Precision Communications in Time Sensitive Networks: Time-Aware Shaper Configuration with IEEE 802.1Qcc," Jun. 2019, arXiv: 1906.11596v1.

[8] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[9] E. Mohammadpour, E. Stai, M. Mohiuddin, and J. Le Boudec, "Latency and backlog bounds in time-sensitive networking with credit based shapers and asynchronous traffic shaping," in *Proc. of International Teletraffic Congress (ITC)*, 2018.

[10] J. A. R. De Azua and M. Boyer, "Complete modelling of avb in network calculus framework," in *Proc. of ACM RTNS*, 2014, p. 55–64.

[11] R. Agrawal, R. L. Cruz, C. M. Okino, and R. Rajan, "A framework for adaptive service guarantees," in *Proceedings of Allerton Conference on Communication, Control, and Computing*, 1998, pp. 693–702.

[12] P. H. Brill and M. Mandelbaum, "On measures of flexibility in manufacturing systems," *International Journal of Production Research*, vol. 27, no. 5, pp. 747–756, 1989.

[13] P. Babarczi, M. Klügel, A. Martínez Alba, M. He, J. Zerwas, P. Kalmbach, A. Blenk, and W. Kellerer, "A mathematical framework for measuring network flexibility," *Computer Communications*, vol. 164, pp. 13–24, 2020.