An Online Scheduler for Reconfigurable Time-Sensitive Networks

Fabian Kummer, Frank Golatowski
Institute of Applied Microelectronics and Computer Engineering
University of Rostock
18051 Rostock, Germany
firstname.lastname@uni-rostock.de

Willi Brekenfelder, Helge Parzyjegla, Peter Danielis, Gero Mühl
Institute of Computer Science
University of Rostock
18051 Rostock, Germany
firstname.lastname@uni-rostock.de

***Abstract*—Time-Sensitive Networking (TSN) facilitates the implementation of realtime data traffic with deterministic delay and jitter in Ethernet networks. In its time-triggered communication variant, network paths as well as transmission time slots along the paths need to be planned and reserved for all data streams, respectively. Conventionally, such a feasible (i. e., realtime compliant) network schedule is computed and optimized offline, installed upon deployment, and cannot be changed anymore at runtime without stopping the communication. The latter becomes unsuitable for future industrial use cases with a growing demand for flexibility including schedule adjustments. In this paper, we present an online scheduler capable to incrementally integrate new streams at runtime and to adapt the existing schedule, if necessary, while retaining the realtime guarantees given to active streams. For this purpose, the scheduling heuristic identifies streams whose adjustment requires minimal reconfiguration efforts. In a thorough evaluation, we analyze the scheduler's trade-off between resource usage and schedule quality showing its suitability for diverse application scenarios.**

***Index Terms*—Time-Sensitive Networking (TSN), Time-Aware Shaper (TAS), Reconfigurable Online Scheduling**

I. Introduction

In recent years, factory automation has become increasingly important for industrial production as part of Industry 4.0. To guarantee the realtime requirements of machine communication, highly deterministic networks with limited latency and jitter are needed. With *Time-Sensitive Networking* (TSN), a series of standards [1] has been published that extends Ethernet by mechanisms for traffic isolation, time synchronization, deterministic forwarding, and resource management. For time-triggered traffic, TSN enables the reservation of exclusive transmission slots for selected data frames on communication links that is enforced by the time-controlled release of the frames from output queues of the bridges along the network path. To guarantee minimal latency and jitter, the reserved transmission slots of different time-triggered data streams need to be coordinated by the network's communication schedule. Conventionally, the realtime requirements of these data streams are gathered at design time, the schedule is computed and optimized offline, and then installed upon deployment. Furthermore, the schedule is usually static and cannot be changed without stopping the ongoing communication which is not suitable anymore for future industrial use cases that demand flexible schedule adjustments at runtime.

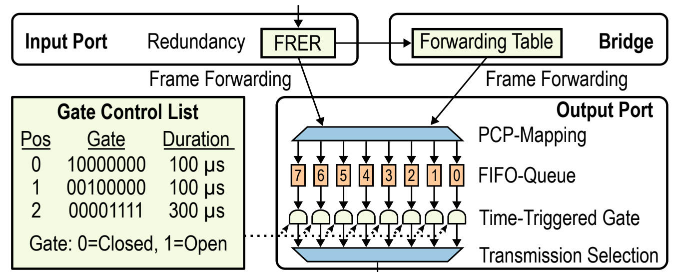

Figure 1. Time-triggered frame forwarding on TSN devices with Frame Replication and Elimination for Reliability (FRER) [2] and Time-Aware Shaper [1]. Gate states in the GCL are indicated by a corresponding bit vector.

In this work, we present a TSN online scheduler that is capable to incrementally integrate new time-triggered streams at runtime, adjusting and reconfiguring the original schedule if necessary. During the reconfiguration process, all realtime guarantees once given to active streams are upheld. Furthermore, the scheduling heuristic can be configured to produce optimized schedules w.r.t. an objective function (e. g., average or maximum stream latency) or to save resources such as CPU and RAM flexibly suiting different application scenarios.

The remainder of the paper is structured as follows: Section II explains basic TSN mechanisms leveraged for deterministic forwarding of time-triggered traffic, whereas Sect. III introduces the scheduling and reconfiguration problem to be solved. In Sect. IV, we give an overview about the scheduler's architecture before diving into the details of the devised heuristic in Sect. V. Section VI presents the evaluation results w.r.t. schedule quality and resource usage. Finally, we review related work in Sect. VII and conclude the paper in Sect. VIII.

II. Time-Triggered Forwarding

In TSN, there are two mechanisms for a network bridge to forward an incoming frame to the output port. Both are illustrated in Fig. 1. Conventionally, the forwarding decision is taken by a lookup in the *forwarding table* (FT) based on MAC address and VLAN identifier. With *Frame Replication and Elimination for Reliability* (FRER) [2], TSN also supports the redundant transmission of data streams on different network

Table I
FORMULA NOTATION

Notation	Characteristics	Definition
t_F	$t_F \in \mathbb{R}, t_F > 0$ ns	processing delay
t_L	$t_L \in \mathbb{R}, t_L > 0$ ns	propagation delay
t_Q	$t_Q \in \mathbb{R}, t_Q \geq 0$ ns	queuing time
D	$D \in \mathbb{R}, D > 0$ bit/s	data rate
N	$N = (id, t_F)$	topology node
E	$E = (id, N_{src}, N_{dst}, D, t_L)$	outgoing link
G	$G = (\{N\}, \{E\})$	topology
t_B	$t_B \in \mathbb{R}, t_B > 0$ ns	basetime
Z_H	$Z_H \in \mathbb{R}, Z_H > 0$ ns	hypercycle time
z_H	$z_H \in \mathbb{N}_0$	hypercycle number
B_H	$B_H = t_B + z_H * Z_H$	hypercycle start time
o_S	$o_S \in \mathbb{R}, 0 \text{ ns} \leq o_S < Z_H$	start time offset
B_S	$B_S = B_H + o_S$	stream start time
l_S	84 Byte $\leq l_S \leq$ 1542 Byte	frame size (Layer 1)
t_S	$t_S = l_S / D$	transmission duration
Z_S	$Z_S \in \mathbb{R}, Z_S \mid Z_H$	stream cycle time
z_S	$z_S \in \mathbb{N}_0, 0 \leq z_S < \frac{Z_H}{Z_S}$	stream cycle number
B_Z	$B_Z = B_S + z_S * Z_S$	stream cycle start time
p_S	$p_S = \{(N, E)\}$	stream unicast path
q	$q \in \mathbb{N}_0, 0 \leq q \leq 7$	queue assignment
T	$T = (t_{\text{Start}}, t_{\text{End}}, q)$	time slot
T_S	$T_S = \{T\}, \forall E \in p_S, \exists T$	time slot allocation
$L_{\max}$	$L_{\max} \in \mathbb{R}, 0 < L_{\max} \leq Z_H$	max. latency
$L_{\min}$	$L_{\min} \in \mathbb{R}, \forall E \in p_S, t_Q = 0$	min. latency
L	$L \in \mathbb{R}, L_{\min} \leq L \leq L_{\max}$	stream latency
S_R	$S_R = (N_{src}, N_{dst}, Z_S, l_S, L_{\max})$	stream requirements
S_C	$S_C = (B_S, Z_S, l_S, p_S, T_S, L)$	scheduled stream
scd	$scd = \{(S_R, S_C)\}$	schedule

paths and, thus, requires additional means prior to the FT to identify and filter duplicate frames.

Independent of the forwarding mechanism used, the frame is transferred to one of eight FIFO queues of the output port. In the simplest case, the queue assignment is based on the frame's priority given by the PCP value in the VLAN header [1]. By opening and closing the gates of the queues, the *Time-Aware Shaper* (TAS) can then release the frames in a time-controlled manner. For this purpose, the TAS follows a *Gate Control List* (GCL) whose entries define the state of the gates (i. e., 1 for open and 0 for closed) for a specified duration. The GCL is repeated periodically starting from a given base time. A *Transmission Selection Algorithm* (TSA) [3] finally determines which frames from open queues are transmitted, usually preferring the queue with the highest priority.

III. SCHEDULING AND PROBLEM STATEMENT

In TSN, traffic is divided in different traffic classes w.r.t. realtime requirements [1] roughly summarized as best-effort, rate-constrained, and time-triggered streams. In this work, we focus on periodic unicast time-triggered streams with hard realtime requirements that do not allow any packet loss. Time-triggered streams thus require forwarding on a fixed path with exclusively allocated time slots on the outgoing ports of the bridges to guarantee a fully deterministic transmission behavior. This makes time slot adjustments as well as path adaptations particularly challenging at runtime. Table I gives an overview about the formal notations used in the following.

A TSN topology consists of nodes N and links E. Nodes include a certain propagation delay t_F, whereas the links connecting those nodes are defined by a given datarate D and propagation delay t_L. A node can either be a bridge or an endpoint, whereas endpoints are divided into talkers and listeners [1]. The data traffic is sent by talkers, forwarded by bridges, and received by listeners. The upper part of Fig. 2 shows a TSN topology with a bridge and two endpoints. Senders and receivers refer to the applications that send data to or receive data from a talker or a listener, respectively.

A periodic time-triggered stream S is defined by its source node N_{src} and its destination node N_{dst} as well as by the cycle time Z_S and its frame size l_S. Furthermore, it has a start time B_S in the cycle and a deadline specified as the maximum allowed latency $L_{\max}$. The stream is planned by reserving exclusive transmission time slots in the GCLs of all outgoing ports along the designated communication path from source to destination. To schedule the time slots the propagation delays t_L and processing delays t_F along the path have to be taken into account, whereas the time slots need to be set at least for the transmission duration of the frame t_D.

A network schedule is generated by planning all streams. As streams may have individual cycle times Z_S, a schedule is usually computed for the hyperperiod Z_H, i. e., the *least common multiple* (LCM) of all cycle times, after which the schedule repeats periodically. Because of the LCM, each stream has one or more cycles in the hyperperiod, for which time slots can be assigned individually. The bottom part of Fig. 2 illustrates the time slots of a stream for two cycles that are scheduled slightly differently. For example, in the first cycle at the talker and at the bridge, if the frame cannot be forwarded immediately as the time slot is not available upon arrival, the frame remains in the queue resulting in additional queuing times t_Q. This can also happen during the transfer from the listener to the receiver process as shown in the second cycle. Thus, if there are varying latencies at the listener for different cycles, these differences can be balanced by buffering the data from the listener to the receiver in order to prevent jitter.

Scheduling can be done offline before runtime or online at runtime. Beyond offline scheduling, which calculates a static schedule from stream requirements prior to deployment, online scheduling additionally allows for the dynamic integration of new streams and the adjustment of existing ones at runtime. New streams can be integrated by utilizing gaps between allocated time slots or, if that is not possible, by displacing and rescheduling existing streams. When reconfiguring active streams in TSN at runtime, two main issues arise.

In contrast to GCL entries, the forwarding rules of FT and FRER cannot be set or changed in a time-controlled manner. Hence, it is not possible to change the entire path of a stream at once. Instead, the forwarding rules have to be changed for each individual bridge stepwise along the path in such a way that deterministic forwarding remains guaranteed. This can be achieved by an intermediate step in which redundant paths are temporarily set up with FRER for both the current and the new configuration. While the two redundant paths are active,

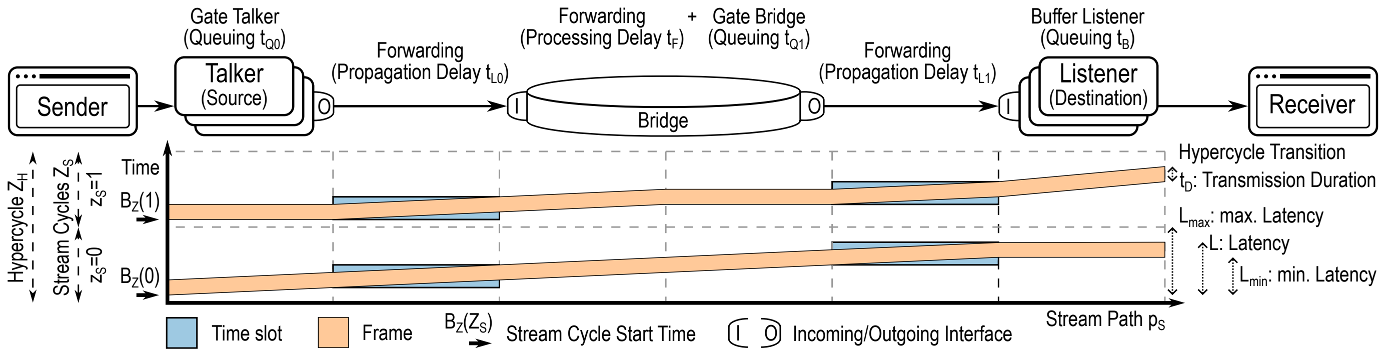

Figure 2. Scheduling of time-triggered streams in the fully centralized model [1] with occurring latencies, queuing times and time slot allocation.

the rules in the FT can be switched over. Once the FT has been switched, FRER can be disabled and only the new path is used afterwards.

The second issue arises when changing GCL entries while frames are still in transit. This happens for frames of streams that are sent at the end of a cycle and received at the beginning of the next cycle. When reconfiguring the GCL at the end of a hyperperiod, frames in transit are thus risking to miss their reserved time slots along the remaining path if the stream is scheduled earlier in the new configuration and all time slots have been preponed correspondingly. This issue can also be solved by introducing an intermediate step in which both time slots, the slot of the current setup and the designated slot after reconfiguration, are reserved simultaneously for the same stream. The frames in transit thus can still use the old slots, whereas frames sent subsequently are using the new slots afterwards.

IV. Scheduler Architecture

In this section, we present the developed online scheduler that can integrate new streams into an existing schedule at runtime, while keeping the realtime guarantees of all active streams. The scheduler's objective is to minimize the reconfiguration effort that arises due to rescheduling of displaced streams. In particular, rescheduling with path changes and conflicts at the hypercycle transition are mostly avoided as those usually results in higher reconfiguration efforts requiring intermediate steps with both the current and the new configuration being active simultaneously. Instead, the scheduling process primarily aims to integrate a new stream without displacing existing streams by using unallocated time slots in the schedule. To enable flexible scheduling, the scheduler is capable to delay the forwarding of frames by queuing as well as to reserve different time slots for the stream for each cycle in a hyperperiod. If it is not possible to integrate a new stream this way, existing streams are displaced. The streams to be displaced are selected based on the least reconfiguration effort required to reschedule, which favors rescheduling them on their current paths.

Figure 3 gives an overview of the scheduler's planning process. The input required by the scheduler includes the stream requirements of the new stream req_S, the topology top, and the existing schedule scd_{ext} as defined in Table I. The output of the scheduler is the adapted schedule scd_{new} which contains a

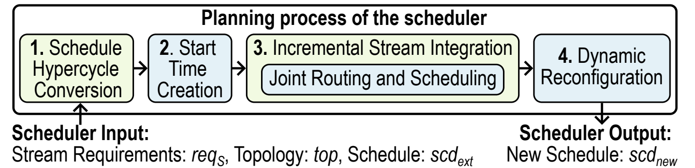

Figure 3. Overview of the scheduler planning process with in- and output.

list of schedules that, when executed sequentially, enable a reconfiguration that keeps the realtime guarantees for all streams. The actual scheduling process is divided into four steps.

The first step is the conversion of the schedule's hypercycle that is only necessary if the new stream's cycle time does not fit the current hypercycle. In such a case, the least common multiple (LCM) of the schedule's hypercycle and the stream's cycle time is calculated and used as new schedule hypercycle. The time slot reservations of all existing streams are adjusted accordingly by rolling out iterations of the original schedule until it completely fills the longer hypercycle.

In the second step, potential start times B_S for the new stream are generated, evaluated, and selected as they are required for subsequent steps. Start time candidates are time slots right before and after already allocated slots at the talker as well as in regular intervals of configurable size within any free gap. For these start times the reconfiguration effort is estimated that is required to integrate the stream without queuing into the existing schedule. The reconfiguration effort is computed by a cost heuristic as explained in Sect. V that is applied in the vicinity of the talker up to a predefined path length. Start times with low reconfiguration efforts are usually preferred.

In the third step, the new stream is planned and integrated in the schedule including the rescheduling of any displaced stream if necessary. A *joint routing and scheduling* (JRS) algorithm is used to route the stream and schedule its time slots along the path based on a given start time B_S. The JRS algorithm is explained in Sect. IV-A, whereas the incremental integration and rescheduling of displaced streams is discussed in Sect. IV-B. Deleting a stream is also supported by simply removing all its reserved time slots from the schedule. Due to its trivial implementation, the deletion process is not discussed any further. Finally, the fourth step is the dynamic

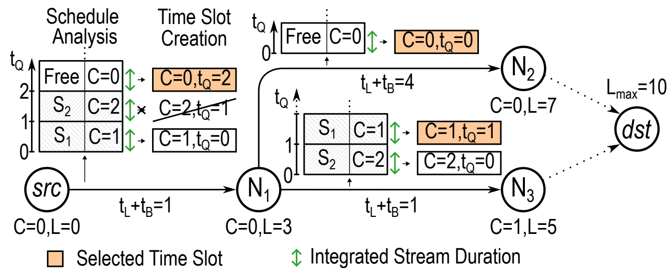

Figure 4. Joint routing and scheduling process with an exemplary time-triggered stream, topology, and time slot allocation of an existing schedule. C: Cost, L: Latency, S: Stream, t_L: Propagation Delay, t_B: Processing Delay, t_Q: Queuing

reconfiguration of the existing schedule to the new schedule, which is presented in Sect. IV-C.

A. Joint Routing and Scheduling

A *joint routing and scheduling* (JRS) algorithm is developed to integrate the new stream with a given start time into an existing schedule. The scheduling objective is to plan the new stream while minimizing the reconfiguration effort. This is primarily achieved by leveraging existing gaps between already allocated time slots. However, if no gap is available, a displacement and rescheduling of existing streams is triggered.

The scheduling of any stream is done stepwise for each node, starting from the source, and following the basic idea of Dijkstra's algorithm [4]. Figure 4 illustrates the beginning of the algorithm for integrating a new stream into an existing schedule starting at src and continuing with nodes N_1, N_2, and N_3. In contrast to the Dijkstra algorithm, which uses fixed weights for each edge, our algorithm uses heuristically calculated costs (see Sect. V) representing the reconfiguration effort together with the stream latency. For each created start time a separate instance of the algorithm is created, hereafter referred to as a *JRS instance*. As with the Dijkstra algorithm, a JRS instance has an open and a closed list. The open list contains nodes that have already been reached via a path with a time slot allocation from the source. An entry in the open list for the source with a specific start time is initialized in each JRS instance. The closed list contains nodes that have already been visited and processed by the algorithm. To prevent cycles, these nodes are no longer considered by the algorithm.

In each algorithm step, a node is taken from the open list, processed, and eventually added to the closed list. For the selected node, all time slots on all outgoing edges between the frame's arrival time and its deadline are considered and weighted according to our cost heuristic as detailed in Sect. V. The heuristic returns a cost value C for each slot that is zero for a free slot or, otherwise, proportional to the estimated reconfiguration effort required for displacing and rescheduling the stream who occupies the slot. At the slot boundaries and at the frame's arrival time, a candidate time slot is created for the new stream to be integrated. The duration of the candidate time slot is determined by the frame size of the new stream and the data rate of the outgoing port. Its cost is computed by summing up the costs of slots that overlap with the candidate and, thus, have to be displaced. In addition, each candidate time slot induces a queuing time t_Q that is only zero if the frame is immediately forwarded upon arrival. Candidate slots that have *both* higher costs and queuing times than other candidates are not worth pursuing any further and are discarded. Please note that this processing needs to be performed for each stream cycle in the schedule's hypercycle. The example in Fig. 4 shows the analyzed time slots together with their estimated reconfiguration costs above each node. The created candidate time slots are represented by green arrows and listed with their induced costs and queuing times to the right. The second candidate slot at src is eventually removed as it is less favorable than the first candidate.

Nevertheless, there may be still more than one suitable candidate time slot for each outgoing port that each needs to be developed further within an individual JRS instance. New JRS instances are created by cloning the current instance state. After selecting a candidate time slot for each outgoing port, cost and latency values are updated accordingly for all neighbors. The cost of the neighboring node is determined by adding the cost of the current node and the cost of the displaced time slots along the link. The neighbors latency value is given by the sum of latency value of current node, the node's processing delay t_B, any induced queuing time t_Q, and the propagation delay t_L on the link to the neighbor. Furthermore, all displaced streams are marked so that their time slots are considered as free on subsequent nodes. If a neighboring node is not already in the open list, it is added. In Fig. 4, the free time slot is selected at src and the cost and latency values for N_0 are updated. However, there is still another alternative candidate slot for which a new JRS instance is created.

The above process is repeated for each node in the open list until the open list is empty or the stream's maximum latency L_{max} is surpassed. Hence, Fig. 4 continues the algorithm for N_1. In fact, the next node selected is the node with the lowest cost and, in case of equal cost values, the node with the lowest latency is chosen. A feasible time slot allocation for the new stream is reached when the stream's destination node is removed from the open list. Due to the selection criteria, this solution has minimal estimated reconfiguration costs.

B. Incremental Stream Integration

To manage the integration of a new stream and the rescheduling of displaced streams, a tree structure is created whose root node is the new integrated stream and whose branches represent the various integration and rescheduling options. Fig. 5 displays the computation tree for integrating a new stream S_N. The objective of the computation tree is to sequentially reschedule the existing streams that have been displaced during the integration of the new stream. The computation tree consists of calculation nodes, which perform scheduling and routing for individual streams, and solution nodes, which cache the adapted schedule after a successful stream integration or rescheduling. For each start time created, a separate calculation node is initialized for the new stream to be integrated, which manages the routing and scheduling of the stream with the JRS

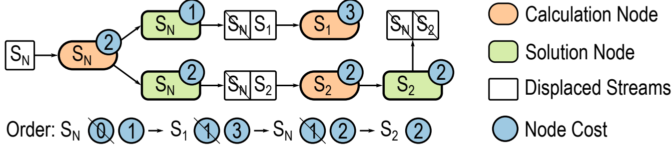

Figure 5. Computation tree of calculation and solution nodes for integrating a new stream S_N and rescheduling displaced streams S_1 and S_2.

instances as described in Section IV-A. Each calculation node contains a cost-based queue for the JRS instances, in which a JRS instance with the source node in the open list is initialized.

To find the solution with the least reconfiguration effort for integrating the new stream and rescheduling the displaced streams, a cost-based selection of calculation nodes is performed, which is an effective combination of breadth-first and deep-first search in the computation tree. To capture the total cost of integration and rescheduling, costs are inherited in the tree. The calculation nodes are assigned a cost value corresponding to the lowest cost of a JRS instance from its internal queue plus the cost value of its predecessor, which is zero at the root. Because the cost value of calculation nodes can change, solution nodes inherit a fixed cost value when they are created. At each computation step in the tree, the calculation node with the lowest cost is selected and scheduling continues with its JRS instances from the internal queue.

When a solution is found for integrating the stream using the JRS instances, a solution node is created as a child, storing the cost, the displaced streams, and the adapted schedule. This is necessary because the calculation nodes may find alternative solutions for stream integration after a solution has been created, and therefore the state at the moment a solution is created must be saved. If the solution computed by the JRS instances in the calculation node contains displaced stream, as described in Section IV-A, it is necessary to reschedule them in the adapted schedule. To do this, a new calculation node is created as a child of the solution node that performs the rescheduling of a displaced stream, as shown in Fig. 5 for S_1 and S_2 as children of solution nodes of S_N. If multiple streams are displaced, the displaced streams must be integrated sequentially. To avoid a cycle in this sequence, each existing stream can be displaced a maximum of once. To prevent deadlocks caused by the order of integration, a calculation node is created as a child for each displaced stream. If a solution node does not contain displaced streams, such as the end of the bottom path of the tree in the Fig. 5, a complete and valid schedule has been found that integrates the new stream and reschedule all displaced streams. This cost-based node selection ensures that the least-cost solution is always found first.

C. Dynamic Reconfiguration

Once a complete schedule for the integration of the new stream is found, dynamic reconfiguration from the existing schedule to the new schedule must be performed in a sequence of reconfiguration steps while guaranteeing deterministic transmission behavior of the streams at all times. For any rescheduled streams that have a path change or hypercycle conflict, the existing and new slots must be allocated simultaneously in a separate reconfiguration step. Once this reconfiguration step is complete, the allocated time slots from the existing schedule can be released. To calculate the reconfiguration step sequence, the dependencies between the streams that allocate the time slots in the existing and new schedule must be considered. This dependency between the streams can be calculated by comparing the time slots of the existing and new schedule.

The reconfiguration sequence can be computed by reducing the dependencies stepwise, starting with those streams that do not require a time slot allocated by another stream in the existing schedule. After the first stream is switched, time slots from the existing schedule are released and the next stream can be switched until all streams have been switched. If there is a cyclic dependency between streams, the streams must be switched at the same time. If there are streams in this cyclic dependency that require time slots that another stream requires for simultaneous allocation of the existing and new time slots, reconfiguration cannot occur because the time slot is required multiple times in the reconfiguration step.

V. HEURISTIC

To determine the cost of a time slot to be displaced, we present an evaluation heuristic based on estimating the reconfiguration effort to reschedule the time slots of the stream on the existing path. The notation used for the heuristic is defined in the Table I. We assume that shifting time slots without changing the path between the start time and deadline for reconfiguration is significantly easier than changing the path of the stream. To estimate the reconfiguration effort, we consider the free spaces between allocated time slots with

$$\mathrm{Free}(t,E) = \begin{cases} 1, & \text{for free time slot on } E \\ 0, & \text{for allocated time slot on } E \end{cases} \quad (1)$$

which indicates whether a time slot is allocated on the outgoing port E at a given time t. To evaluate the reconfiguration effort, we introduce the measure flexibility. Flexibility indicates how much free time is available between the start time and the deadline of a stream. When a time slot is displaced, only the rescheduling of the time slots for one cycle is necessary, so the flexibility of a stream is calculated only for the corresponding cycle of the displaced time slot. Flexibility F_E of a stream S for an outgoing port E and a cycle z_S is defined as follows.

$$F_E(S,z_S,E) = \int_{B_Z(z_S)+L_{\min}(N_{src},E)}^{B_Z(z_S)+L_{\max}-L_{\min}(E,N_{dst})} \mathrm{Free}(t,E)dt \quad (2)$$

When rescheduling a stream, the port with the least flexibility is the most demanding, so the minimum flexibility along the path p_S for the corresponding cycle z_S is used to determine the flexibility of the entire stream S, defined as

$$F(S,z_S) = \min(F_E(S,z_S,E)), \forall E \in p_S \quad (3)$$

This method can be used to estimate the current reconfiguration effort of a stream, but if a new stream is integrated after the displacement, the free time at the port may change. The

integration of the new stream can be taken into account in the port flexibility as follows.

$$F_{\text{est}}(S_1, z_{S_1}, E, l_{S_{new}}) = F_E(S_1, z_{S_1}, E) - \frac{l_{S_{new}}}{D} \quad (4)$$

By subtracting the transmission duration of the new stream, the flexibility after the integration of the new stream and the rescheduling of the displaced stream can be constructed. As a result of the deducted time, the flexibility can also fall below zero, which indicates that after the integration of the stream, the displaced stream cannot be rescheduled on the existing path without further displacement. Since less flexibility means more reconfiguration effort, this value is not suitable as an edge weight in the scheduling algorithm. Therefore, a cost value C is determined from the flexibility F, which also increases with increasing reconfiguration effort, defined as follows.

$$\mathrm{C}(F) = \begin{cases} C_{\text{max}}, & F \leq 0 \\ \frac{F_{\text{max}} - F}{F_{\text{max}}} * (C_{\text{max}} - C_{\text{min}}) + C_{\text{min}}, & 0 < F < F_{\text{max}} \\ C_{\text{min}}, & F_{\text{max}} \leq F \end{cases} \quad (5)$$

To calculate the cost value, the minimum cost C_{min}, the maximum cost C_{max} and the maximum flexibility F_{max} must be set. Between 0 and F_{max} the cost increases linearly with decreasing flexibility, when flexibility reaches zero the cost stagnates at C_{max}, when flexibility is higher than F_{max} the cost stagnates at C_{min} to ensure that the cost is always higher than that of a free slot.

VI. Evaluation

We prototypically implemented our scheduler in Python providing rich configuration options to fine-tune the scheduling heuristic and study the effects. Table II lists the most important parameters with the default settings highlighted in bold. Whether *store-and-forward* or *cut-through* switching can be used is determined by the capabilities of the networking equipment, whereas the decision how many bridge output queues and which of them are dedicated to time-triggered traffic (i. e., *time-triggered queues*) lies with the network administrator. *Queuing limitation* allows to bound the scheduled buffering time in bridges, for example, with a *maximum queuing delay* of zero, the scheduler plans a continuous transmission along the whole path from talker to listener without any delays as preferred by certain scheduling methodologies. Without allowing *stream displacements* the scheduler incrementally adds new streams without ever adjusting streams already planned. Furthermore, the schedule adaptations can be restricted to time slot adjustments or may also include *path changes*. The *schedule reconfiguration* option tells the scheduler to also generate a valid reconfiguration sequence that transforms the old schedule into the new one. Finally, *restricted start times* aligns the start time, when a stream's frame is planned to be released by its talker, to the boundaries of free or existing time slots leading to a more compact allocation in bigger blocks.

A. Evaluation Setup

We evaluated the scheduling heuristic using the network depicted in Fig. 6. The network consists of eight bridges connected in a ring topology of which each bridge is also linked to an end station. All network links have a data rate $D = 1$ Gbps. There are up to 82 time-triggered streams to be scheduled with their sources and sinks randomly distributed to the end stations. Each stream has a cycle time $Z_S = 250\,\mu s$ and a frame size $l_S = 1500$ bytes, thus, occupying $4.8\,\%$ of a link's capacity. The maximum stream latency $L_{\text{max}} = 100, \ldots, 150\,\mu s$ is set proportionally to the minimum path length between talker and listener that varies between three and six links. The streams are incrementally scheduled one after the other until all 82 streams have been processed, no feasible schedule is found anymore, or a maximum runtime of 15 minutes is exceeded. The scheduler is executed on a virtual machine with four assigned AMD EPYC 7502 cores and 64 GB of RAM. We measure the scheduler's runtime, its memory usage, the number of scheduled streams and how many streams are adapted at which step. In addition to quantify the schedule quality, we also determine the average latency and the maximum latency of the planned streams.

Table II
Scheduler options with default settings in bold

Features	Setting Options
Packet Switching System	Store-and-Forward, **Cut-through**
Time Triggered Queues	0, 1, 2, 3, **4**, **5**, **6**, **7**
Queuing Limitation	Enabled, **Disabled**
Maximum Queuing	Nanosecond Accuracy (**0 ns**)
Stream Displacement	**Enabled**, Disabled
Stream Path Changes	**Enabled**, Disabled
Schedule Reconfiguration	**Enabled**, Disabled
Restricted Start Time Creation	Enabled, **Disabled**

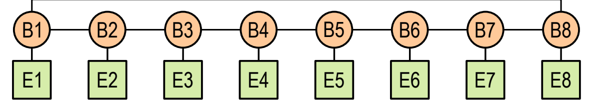

Figure 6. Ring topology consisting of eight bridges, each with an endpoint, which are connected via duplex 1 Gbps Ethernet.

B. Evaluation Results

The evaluation is conducted for four different scheduler configurations. Besides the *default settings* (DF) as listed in Table II, we also analyze a *no queuing* (NQ) variant that does not consider buffering at the bridges and a setting with *restricted start time creation* (RSTC) in which the start time of new streams is always aligned to the boundaries of already existing time slots. Furthermore, in the *reconfiguration disabled* (RD) variant, we have switched off both the generation of and the check for a proper reconfiguration sequence for the schedule adjustment. All results are shown in Fig. 7.

As displayed in Fig. 7(a), RD is able to schedule all 82 streams, whereas DF can only integrate 80 streams. This is due to high network utilization and missing free gaps that are required to simultaneously allocate a stream's old and new time slot in an intermediate step during reconfiguration. Interestingly, RSTC is also able to schedule all 82 streams, but with the reconfiguration check enabled. This is because

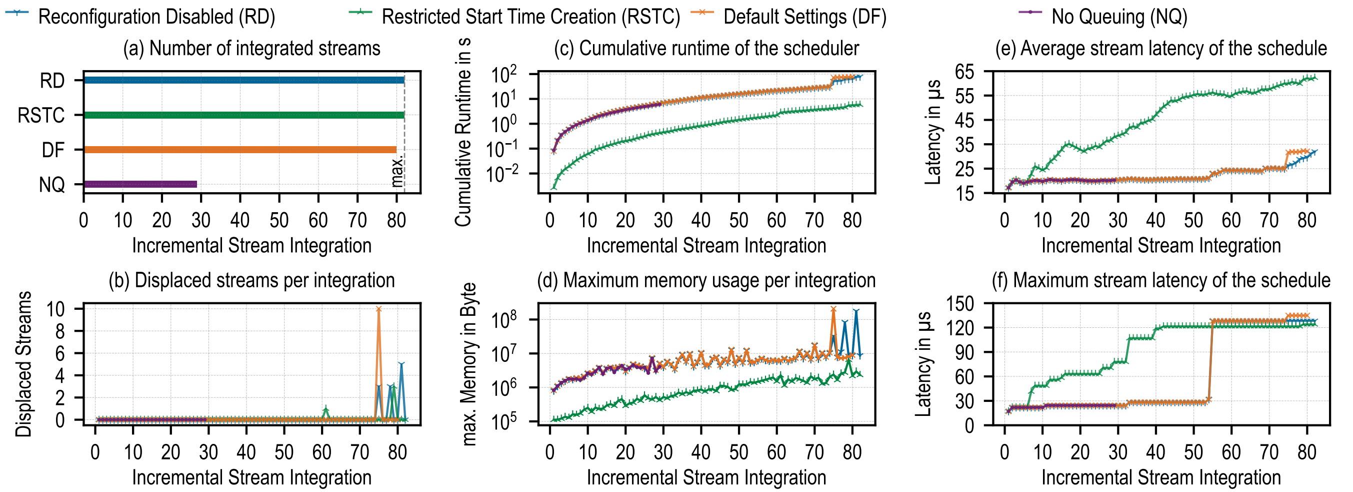

Figure 7. Incremental integration of 82 streams with a cycle time of 250 µs and a frame size of 1500 bytes in a ring topology of eight bridges, each with an endpoint to which the sources and destinations of the streams are distributed. The abscissa of all six diagrams displays the incremental integration of streams. (a) displays the number of streams that were successfully integrated, (b) shows the number of streams that were displaced during each integration, (c) displays the maximum memory usage per execution step of the scheduler, (d) shows the cumulated runtime of the scheduler since the first step, (e) shows the average latency, and (f) displays the maximum latency of all streams in the output schedule for the execution step.

the streams' start times are set without gaps, which leads to a tighter time slot allocation in earlier schedules and saves reconfigurations later. This can be observed in Fig. 7(b) when comparing the number of displaced streams at each integration step. In addition, the effects of the reconfiguration check are made apparent at step 75, when DF and RD start to diverge. DF has to continue to search for a more complex solution adapting ten already planned streams because the simpler solution of RD cannot be reached without violating given realtime guarantees during the reconfiguration sequence. For NQ, only 28 streams are integrated as the heuristic cannot find a feasible solution without buffering at bridges afterwards. Reconfiguring a stream without queuing basically boils down to using the alternative path on the ring which is usually longer and whose time slots are already occupied by other streams. In fact, NQ only added new streams and was never able to adapt a previous schedule.

Fig. 7(c) shows the scheduler's cumulated runtime growing nearly linearly until the first schedule adaptations occur. When stream displacements are necessary, the runtime increases significantly at these steps, cf. Fig. 7(b). This is due to the additional work required to also schedule the displaced streams alongside with the new one. Likewise, the memory usage as graphed in Fig. 7(d), is much higher at these steps caused by larger computation trees required in these cases. The minor fluctuations in memory usage, however, are due to the different path length of the streams to integrate. In comparison to the other configurations, RSTC has both a significantly reduced runtime and memory usage because of its more compact planning and fewer start times that lead to fewer and less branched computation trees. Please note that RSTC scheduled all 82 streams within 10 s requiring at most 10 MB of RAM.

However, this advantage is accompanied by a significant increase in stream latencies, as shown in Fig. 7(e) and (f). Due to the reduced number of start times created with RSTC, it happens that all of these start times do not fit a suitable gap between time slots on the network ring leading to high queuing delays at the links from the source to the bridge. Unfortunately, this effect grows with an increasing number of streams starting from the same source and causes higher stream latencies. In contrast, the other three scheduler configurations show only a slight deviation in stream latency. The stream latencies either correspond to the minimum latency possible or are slightly above it. They only start to grow noticeably in cases of high network utilization.

In summary, the evaluation quantified the effects of different configuration options on the scheduler's resource usage and quality of created schedules. Without queuing (NQ), resulting stream latencies are minimal. But allowing bridges to buffer frames increases the solution space drastically without significant effects on stream latencies (DF and RD). Especially in cases of high network utilization, the reconfiguration check is important (DF) as the heuristic may find solutions otherwise (RD) for which a corresponding reconfiguration sequence cannot be derived. Arranging transmission time slots in a very compact format (RSTC) leads to huge resource savings in terms of both CPU runtime and RAM usage facilitating the scheduler's deployment even on resource-constrained embedded devices. Although this comes at the cost of increased stream latencies, all generated schedules are still feasible.

VII. Related Work

The survey [5] and the reviews [6] and [7] provide a comprehensive and systematic analysis and comparison of the concepts for scheduling and reconfiguration in TSN that have been published since 2016. There are already many approaches to scheduling in TSN that differ in optimization objectives, supported traffic classes, and TSN mechanisms, but they mainly focus on offline scheduling. According to the TSN scheduler classification in [7], no approach uses heuristic

JRS with allowed queuing like the scheduler of this work. The only scheduler that uses JRS with allowed queuing according to the classification in [7] is the scheduler in [8] and its evolution for multicast in [9], but it uses with an ILP model an exact approach instead of a heuristic one in this work.

To the best of our knowledge, there are currently only a few online scheduling approaches in TSN. In [10], a heuristic online scheduler based on fixed routing and scheduling with allowed queuing is presented. In contrast to this work, the existing streams are not displaced or adapted, which on the one hand makes the reconfiguration from an existing to a new schedule trivial, but on the other hand severely restricts the integration of streams and excludes possible configurations that can only be found by reschedule existing streams. An ILP-based online scheduler with allowed queuing is presented in [11]. The new streams are also scheduled exclusively in the free time slots of the existing schedule. This is done by defining constraints on the schedule and the streams in the ILP model. If the ILP solver fails to integrate multiple streams, backtracking is performed to change the order of integration of new streams, possibly resolving the conflict. In addition, [11] describes a two-part reconfiguration process that integrates the new streams into the existing schedule without interfering with or causing packet loss to the existing streams.

In [12] and [13], a measure of stream flexibility called *flexcurve* is introduced that is used during scheduling to facilitate integration and reconfiguration of streams in the future. For a given path, the number of potential combinations of time slots for the integration of the stream is examined. If a reconfiguration is necessary, it is assumed that paths with a high degree of flexibility, as indicated by a high combination of possible time slot allocations, can be reconfigured more effectively in future adjustment steps of the schedule. The objective is to make better use of the gaps in the existing schedule when scheduling new streams, thus allowing efficient allocation in the entire schedule. Similar to the flexibility of this work, the free time slots of the outgoing ports of a given path are analyzed. However, instead of examining and evaluating an existing stream in its limits between start time and deadline, possible paths for the integration of a new stream are analyzed. Furthermore, granular time slots are used, where flexibility corresponds to a number of combinations, whereas in this work flexibility corresponds to the minimum free time between time slots on a continuous time base.

VIII. Conclusions

In this paper, we developed a novel online scheduler for TSN networks that is able to integrate new time-triggered streams into an existing schedule at runtime. If necessary, one or more existing data streams are displaced in order to create space to accommodate new streams. In addition, we generate a series of stepwise reconfigurations to transform the current schedule into the new one while keeping the realtime guarantees (e. g., bounded latency and jitter) of all active streams during the process. The scheduler leverages a heuristic to identify streams that are likely to be displaced successfully requiring minimal reconfiguration efforts. In particular, we prefer a modest reassignment of communication slots along a network path rather than changing the network path and computing new time slots for a displaced stream.

In an evaluation, we thoroughly analyzed the behavior of our scheduling heuristic and devised parameter configurations suited for different application scenarios. These configurations trade resource utilization (i. e., CPU runtime and RAM usage) for schedule quality w.r.t. an optimization objective such as average or maximum stream latency. This way, our scheduler is able to quickly deliver feasible network reconfigurations when implemented in a network controller running on an embedded system at a factory workshop. It could also be used to compute optimized solutions when outsourced to more powerful servers. By enabling both the seamless integration of new realtime traffic and the reconfiguration of active communication schedules at runtime, our online scheduler makes a significant step towards meeting the growing demands for dependability and flexibility within a future smart factory.

References

[1] “IEEE standard for local and metropolitan area networks – bridges and bridged networks,” *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)*, 2022, doi: 10.1109/IEEESTD.2022.10004498.

[2] “IEEE standard for local and metropolitan area networks – frame replication and elimination for reliability,” *IEEE Std 802.1CB-2017*, 2017, doi: 10.1109/IEEESTD.2017.8091139.

[3] “IEEE standard for Ethernet,” *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, 2022, doi: 10.1109/IEEESTD.2022.9844436.

[4] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959, doi: 10.1007/BF01386390.

[5] T. Stüber, L. Osswald, S. Lindner, and M. Menth, “A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (TSN),” *IEEE Access*, vol. 11, pp. 61 192–61 233, 2023, doi: 10.1109/ACCESS.2023.3286370.

[6] H. Chahed and A. Kassler, “TSN network scheduling – challenges and approaches,” *Network*, vol. 3, no. 4, pp. 585–624, 2023, doi: 10.3390/network3040026.

[7] C. Xue, T. Zhang, Y. Zhou, M. Nixon, A. Loveless, and S. Han, “Real-time scheduling for 802.1Qbv time-sensitive networking (TSN): A systematic review and experimental study,” in *IEEE RTAS*, 2024, pp. 108–121, doi: 10.1109/RTAS61025.2024.00017.

[8] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, “ILP-based joint routing and scheduling for time-triggered networks,” in *RTNS*. ACM, 2017, pp. 8–17, doi: 10.1145/3139258.3139289.

[9] E. Schweissguth, D. Timmermann, H. Parzyjegla, P. Danielis, and G. Mühl, “ILP-based routing and scheduling of multicast realtime traffic in time-sensitive networks,” in *IEEE RTCSA*, 2020, doi: 10.1109/RTCSA50079.2020.9203662.

[10] M. L. Raagaard, P. Pop, M. Gutiérrez, and W. Steiner, “Runtime reconfiguration of time-sensitive networking (TSN) schedules for fog computing,” in *IEEE FWC*, 2017, doi: 10.1109/FWC.2017.8368523.

[11] Z. Pang, X. Huang, Z. Li, S. Zhang, Y. Xu, H. Wan, and X. Zhao, “Flow scheduling for conflict-free network updates in time-sensitive software-defined networks,” *IEEE Trans. Ind. Inform.*, vol. 17, no. 3, pp. 1668–1678, 2021, doi: 10.1109/TII.2020.2998224.

[12] C. Gärtner, A. Rizk, B. Koldehofe, R. Guillaume, R. Kundel, and R. Steinmetz, “On the incremental reconfiguration of time-sensitive networks at runtime,” in *IFIP Networking*, 2022, doi: 10.23919/IFIPNetworking55013.2022.9829815.

[13] C. Gärtner, A. Rizk, B. Koldehofe, R. Guillaume, R. Kundel, and R. Steinmetz, “Fast incremental reconfiguration of dynamic time-sensitive networks at runtime,” *Comput. Netw.*, vol. 224, no. C, Apr. 2023, doi: 10.1016/j.comnet.2023.109606.