Window-Based Schedule Synthesis for Industrial IEEE 802.1Qbv TSN Networks

Niklas Reusch*, Luxi Zhao*, Silviu S. Craciunas†, Paul Pop*
*Technical University of Denmark Kongens Lyngby, Denmark
†TTTech Computertechnik AG, Vienna, Austria

***Abstract*—Time-Sensitive Networking (TSN) introduces standardized mechanisms that add real-time capabilities to IEEE 802.1 Ethernet networks. In particular, the Time-Aware Shaper (TAS) can be used to send frames in a deterministic fashion according to a predefined global schedule. Existing methods for generating the global communication schedule enforce isolation either in the time or in the space domain. This extended abstract presents a novel, more flexible window-based scheduling algorithm which removes the previously required isolation constraints for Scheduled Traffic (ST) by integrating worst-case delay analysis to guarantee bounded latency.**

I. Introduction

Time Sensitive Networking (TSN) is a set of amendments to IEEE 802.1 designed to bring real-time capabilities into Ethernet-based networks. The network-wide clock synchronisation protocol (802.1ASrev [1]) together with the Time-Aware Shaper (TAS) mechanism (802.1Qbv [2]) allow real-time scheduled traffic (ST) to coexist with standard best-effort (BE) within the same multi-hop switched Ethernet network [3]. The TAS defines a timed gate for each queue (traffic class) at the egress ports of network devices that enables or disables the transmission of frames according to a global communication schedule implemented in so-called Gate Control Lists (GCL).

The GCL schedule synthesis [3], [4], [5] aims at enforcing end-to-end latency guarantees for ST streams through strict temporal isolation not only from BE streams but also from other ST streams. More specifically, [3] introduces an Satisfiability Modulo Theories (SMT)-based method that creates schedules with 0-jitter and defined end-to-end latencies for individual ST flows. The method enforces a complete isolation of critical flows from each other either by not allowing them to be in the same queue at the same time or scheduling them in different queues (i.e., temporal or spatial isolation). This results in a strictly-periodic communication model with 0-jitter transmission which may significantly reduce the solution space and may, additionally, generate a large number of GCL events exceeding the hardware capabilities of existing TSN devices. In [4] the 0-jitter assumption is relaxed by allowing a bounded interference between ST frames scheduled in the same queue at the same time. However, both approaches enforce that gates of different scheduled queues are opened and closed in a mutually exclusive fashion, i.e., the priority mechanism is essentially circumvented, eliminating potential delays due to higher priority ST frames. The main reason for the mutually exclusive behavior is that the worst-case delay analysis required ST flow interference cannot be easily expressed in SMT-formulation. The isolation may however over-constrain the schedule generation for certain applications. Moreover, both papers require that end-systems have TSN capabilities which restricts the application domain significantly since a lot of use-cases rely on off-the shelf end-nodes without TSN mechanisms (e.g. those providing sensor data).

In this extended abstract we present a novel heuristic window-based approach for GCL synthesis that relaxes the mutually exclusive gate opening requirement of prior work and allows two or more ST gates to be enabled at the same time. Hence, our method does not individually schedule ST frames and flows but rather schedules open gate windows for individual scheduled queues, and does not impose that scheduled queues are opened and closed in a mutually exclusive fashion. In order to still provide real-time guarantees and calculate the ensuing delays between ST frames of different priorities we incorporate the worst-case delay analysis presented in [6]. Furthermore, this new approach also works for networks in which end-systems do not have TSN capabilities. We define the analysis-driven window optimization problem resulting from our more flexible approach with the goal to increase the *porosity* of the schedule, a metric introduced by Steiner [7], expressing the degree by which a schedule has empty intervals that allow lower priority traffic to be transmitted. We evaluate the proposed approach on both synthetic and real-world test cases, validating the correctness and the scalability of our implementation and compare it to existing solutions for TSN schedule generation.

We introduce the system model in Sect. II and outline the problem formulation in Sect. III. In Sect. IV we present the novel scheduling mechanism and the optimization strategy followed by an experimental evaluation in Sect. V. We conclude the paper in Sect. VI.

II. System Model

Network Model. We model the network as a directed graph G, where the vertices are end systems (ES) and switches (SW)—also called *nodes*, and edges are bi-directional full-duplex physical links. Fig. 1 shows an example.

In order to reduce the network equipment and configuration requirements, we relax the constraint that ESs are also part of the synchronization. Hence, ESs are allowed to transmit unsynchronized traffic according to a strict priority (SP) scheme. SW are assumed to have 802.1Qbv capability (explained in detail below) and are synchronized via 802.1ASrev.

Fig. 2 depicts a simplified TSN switch. The switching fabric decides to which egress port(s) an incoming frame is routed. According to the 802.1Q standard [8], each output port has eight priority queues, which store the received frames in FIFO order. A subset of the queues is used for ST traffic while the rest are used for other, less critical, communication.

802.1Qbv specifies that there is a timed gate associated to each traffic class (queue), which can be either opened or closed

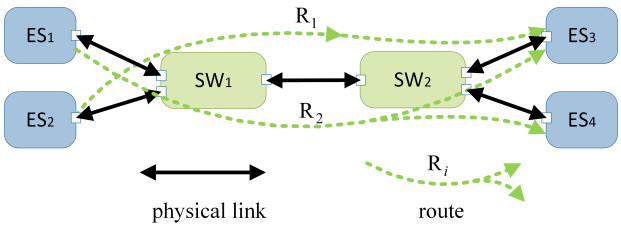

Figure 1. TSN network topology example

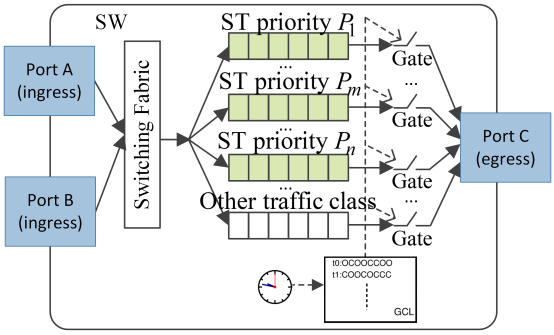

Figure 2. TSN Switch Internals

according to a predefined Gate Control List (GCL). Traffic is sent from the respective queue only if the gate is in the open state. When multiple gates are open at the same time, the highest priority queue can transmit frames, blocking others until it is empty or the corresponding gate is closed.

The 802.1Qbv standard defines a lookahead mechanism for each traffic class to check whether there is enough time to send the entire frame before the closing time of the gate. If not, the frame cannot be forwarded until the next open window, and there will be an idle time (guard band) at the end of the current open window. We assume a non-preemption policy (no 802.1Qbu support) if two or more are gates open at the same time, which means that an ST frame already in transmission cannot be interrupted by a higher priority ST frame.

The transmission approach advocated in this paper is "window"-based, as it controls traffic based on the open windows defined in the GCLs and not by individual frames as in previous work, e.g., [3]. Hence, the GCL configuration is defined as a tuple containing, for each queue in an output port, the window offset, window length, and window period.

Application Model. The traffic class we focus on in this paper is scheduled traffic (ST) also called time-sensitive traffic. ST traffic can have requirements on bounded end-to-end latency and/or minimal jitter. Communication requirements of ST traffic itself are modeled with the concept of flows (also called streams). An application is modeled as a set of ST flows. A flow is characterized by its frame size, the period in the source ES, the priority, and the deadline, i.e. the maximum allowed end-to-end latency. The route for each flow is statically defined (for example the dot-dash green arrows in Fig. 1) as an ordered sequence of directed links.

III. Problem Formulation

The problem addressed in this paper can be formulated as follows. Given a set of flows $\mathcal{F}$ with fixed routes, priorities and deadlines, determine the offset, length and period for each window in each priority queue on the routes of all flows, such that the overall still available bandwidth of all ports in the network is maximized and the ST flows are schedulable, i.e., the worst-case delay (WCD) of each ST flow is smaller than or equal to the respective deadline.

In order to evaluate and validate a solution, we need to know the upper bounds of WCDs for the ST flows. In previous work, the WCD of each ST flow did not depend on other ST flows with different priority due to the opening of gates being mutually exclusive. Since we want to remove this assumption, the fully-deterministic pattern for each ST flow no longer exists. Thus it is necessary to perform a schedulability analysis

Table I
Stream Definition

	Size	D (μs)	T (μs)	P	Route
tt1	1500	3000	200	1	ES1-SW1-SW4-SW6-ES9
tt2	2500	3000	200	1	ES1-SW1-SW2-ES4
tt3	1500	3000	200	2	ES4-SW2-SW1-ES10
tt4	4200	3000	200	2	ES5-SW4-SW2-SW3-ES3
tt5	3000	3000	200	1	ES5-SW4-SW6-ES9
tt6	1500	3000	200	3	ES5-SW4-SW5-ES6
tt7	1000	3000	200	1	ES7-SW5-SW4-ES8
tt8	1500	3000	200	3	ES8-SW6-SW4-SW2-SW3-ES2
tt9	2763	3000	200	2	ES10-SW1-SW2-SW3-ES2

(a) Initial Windows

(b) Optimized Windows

Figure 3. Example configuration solution with optimized windows

for ST traffic when multiple gates are open simultaneously [6].

ST flows passing through output ports of switches are shaped by the GCL which is modeled by windows assigned for the corresponding priority queue. According to the overlapping window-based network calculus model [6], the upper bounds of worst-case end-to-end delay of a ST flow is the sum of all queue delays on its route.

In the following example we have given the nine flows in Table I. A solution that maximizes the available bandwidth for lower-priority traffic is shown in Fig. 3(a). Such a solution could be obtained using the initial solution algorithm that will be presented later. Ports are denoted as $[v1, v2]$ and their queues are shown below them. The colored boxes represent the windows, with the dotted line showing their periods. Initially, a lot of free space is available in every port (low cost) and the periods of all ports are very uniform.

However, this schedule does not fulfill the deadline constraint, as three flows (tt4, 8 and 9) will miss their deadlines. An optimization algorithm should refine the solution such that all of these deadlines are fulfilled while not sacrificing too much free bandwidth. A possible solution, generated with the iterative optimization algorithm from Sect. IV, is presented in Fig. 3(b). In this solution several queue periods have been shorted, for example in the highly frequented port [SW4, SW2], to reduce the worst-case delay, at the expense of a higher cost for that port. We now meet all the deadlines, while on average there is only 4.1% less time available in the ports to send lower-priority traffic.

IV. Optimization Strategy

We propose a solution based on an iterative optimization algorithm, which, starting from an initial solution, determines the windows such that the cost function (defined in Sect. IV-A) is optimized. The impact of windows of the schedulability

of flows is determined with the analysis from [6], which is not *exact* as it is fit for the analysis for a single window-based node but not sensitive to relative positions of windows on consecutive nodes. Fig. 4 depicts the components of our optimization strategy and their interaction. In the following, we will explain each component.

Initial Solution. We developed a constructive heuristic to find an initial solution which minimizes the cost, the box "Generate initial solution" in the figure. Note that we can *look at ports individually* because changes in one port do not have any impact on the possible window configurations of other ports, i.e., decreasing the cost of one port will always have a positive impact on the overall cost. Although changes in a port impact the WCDs, we do not consider this in the initial solution. If the initial solution does not fulfill the deadline constraints, the optimization algorithm from the box "Optimize ports of flow" will take over, taking the global impact of ports on worst-case delay into consideration.

Looking at an individual port, the goal is to create cost-optimal windows. For each queue, the first step is to determine the necessary length of the window. That length can be deduced from the flows passing through the queue. We can then determine the lower bound on the window length as the longest sending time of any flow plus the guard band. If the window would be smaller than that, the largest flow could not be transmitted.

Based on a given window length without guard band, we can then calculate the period of the window. The goal is to find the maximum period of the window for which no flow has an infinite worst-case delay. This maximum period can be deduced from the period percentages pp of the flows, i.e., the proportion of the flows sending time to the flows period. The cumulative period percentage of all flows in the queue then give us the proportion of the windows length, excluding the guard band gb, to its maximum period. That means with a given length w we can determine the maximum period for a queue q as $q.T = (q.w - q.gb)/q.pp$.

Such a solution is not sufficient because the periods of different queues are not aligned. Hence, the initial solution algorithm also scales up all window periods, and proportionally the window lengths, to the highest existing period in the port. Based on those new window lengths it then calculates offsets such that windows start right after a previous window has ended, with the highest priority window starting at offset 0 and the other windows following in order of priority.

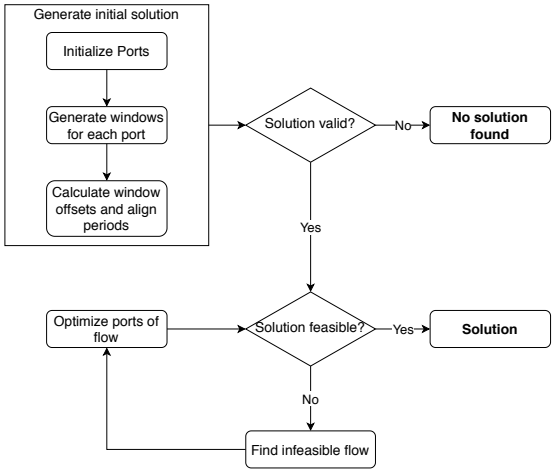

Figure 4. Overview of our optimization strategy

Optimizing Windows. We define a solution to be *valid* if no flow has an infinite worst-case delay, i.e. the windows are appropriately chosen. If a solution also fulfills the deadline constraint it is said to be *feasible*. It is possible to find an initial solution that is near-optimal in terms of cost, but which may not meet the deadlines of every flow, thus further optimization becomes necessary. The window optimization algorithm, as presented, starts from an initial solution and then it checks the feasibility of the solution with the deadline of all flows. If the initial solution is feasible already, there no need to further optimize it, see Fig. 4.

If the solution is not feasible we have to optimize it. For that we first determine the subset of infeasible flows, which we sort by exceeding percentage in descending order. The exceeding percentage of a flow f_i is defined as $ep(f_i) = WCD(f_i)/D_i - 1$, capturing how much a flow exceeds its deadline. The rationale is to improve the windows of the worst flows first, as by doing that, other flows might improve as well.

Optimizing a flow is done iteratively, in a loop: it either increases or decreases the period of all ports on the route of the flow. Increasing/decreasing the period of a port means increasing/decreasing the period of every window in the port by the same amount. It does that for as long as at least one period of any port changed. The goal is to approximate the maximum period in each port for which the flow is barely feasible. If feasibility is not yet reached, the periods get decreased in every port along the flow route. Otherwise, they are increased until the solution is infeasible again or the period did not change in any port.

A. Cost Function

Inspired by [7], we want to evaluate the porosity of our window schedule. We say that an egress port has *high porosity* if the windows are evenly spaced allowing long and frequent intervals for lower priority traffic. To measure the porosity we calculate the *occupation percentage* of the hyperperiod: the percentage of the hyperperiod that at least one window is active. A low occupation percentage means high porosity.

Determining the cost is challenging because windows may overlap. Each time interval where overlap occurs means a reduction of occupation percentage. Thus, simply taking the sum of all window lengths would lead to a higher occupation than in reality. Thus, for each output port p, the problem of finding the occupation of a hyperperiod through windows can be solved mathematically by calculating the sum of a set of intervals, which is formally defined as follows,

$$Cost(p) = \frac{l(p.I)}{l(0, p.T_{hyper})}, \tag{1}$$

where the function $l(x)$ helps determine the length of an interval or a set of intervals, $l([a,b]) = b - a$. The union operation makes sure overlapping intervals are merged into one. Calculating the union of intervals is solved using a line sweep algorithm [9]. Then we define the overall cost function $Cost(\mathcal{P})$ as the sum of all port costs $Cost(p)$, i.e., $Cost(\mathcal{P}) = \sum_p Cost(p)$.

V. Evaluation

A good solution is determined by a low value of the cost function and by worst-case delays that fulfill the respective deadline requirements. We evaluated our proposed flexible

Table II
COMPARISON OF WND TO RELATED WORK: 0GCL, SP AND AVB

	Mean e2e Delay (μs)				Mean Cost (%)				Calculation Time (s)				Infeasible Flows			
	0GCL	WND	SP	AVB	0GCL	WND	SP	AVB	0GCL	WND	SP	AVB	0GCL	WND	SP	AVB
TC1	821	4382	2520	6834	47.8	100	44.1	44.1	600	8.2	0.46	0.2	0	7/13	4/13	8/13
TC2	190	1914	597	1890	46	75.7	42.5	42.5	600	5.6	0.44	0.22	0	1/15	0	4/15
TC3	124	1265	419	1798	39.5	74.8	36.5	36.5	600	7.5	0.22	0.21	0	1/16	0	4/16
TC4	126	1589	318	1877	32.8	44.1	30,3	30.3	600	3	0.41	0.21	0	1/14	0	1/14
TC5	125	1421	367	2239	22.3	63.5	20.6	20.6	600	7.5	0.21	0.21	0	0	0	2/8
TC6	1277	7420	2499	14286	23.2	48.6	63.4	21.5	600	17.8	0.2	0.2	0	4/14	2/14	9/14
TC7	338	5910	1534	8723	24.4	91.2	22.6	22.6	600	10.52	0.21	0.2	0	0	5/11	4/11

window-based GCL scheduling method (called WND) on 7 synthetic test cases inspired from industrial application requirements.

Several scheduling methods have already been proposed [3], [4], [5]. The choice of which scheduling approach to use is highly dependent on the specific use-case and application domain as well as by the TSN hardware mechanisms available in the devices.

Table II shows the compared results from four aspects, the mean upper bounds of end-to-end latency (WCDs of flows), mean cost, calculation time and number of infeasible flows. In order to compare our method to previous work, we first need to harmonize the test-cases proposed in [10], which miss two crucial parameters: the stream priority and deadline. Hence, we first run the scheduling algorithm from [10] and extract the priorities from the resulting schedule. We then set the deadline to 10 times the period for each stream. The comparison was made with the greedy randomized adaptive search procedure (GRASP) metaheuristic from [10]. The results from [10] are presented in the columns named "0GCL" while the results for our method are presented in the columns named "WND". In addition, we are interested in comparing the results with strict priority (SP) traffic and audio video bridging (AVB) traffic. The classes for AVB traffic correspond to previously mentioned flow priorities. The results for the SP and AVB algorithms are shown in the columns named "SP" and "AVB".

0GCL, i.e. deriving schedules for individual flows that are separated, has the best performance in supplying ultra-low latency, and is able to make all flows schedulable (no infeasible flows). The schedule synthesis problem of 0GCL is intractable, and requires exponential time to find a solution. Similar to [10], we have used a time limit, 10 minutes in our case. In addition, 0GLC will have a lower fault-tolerance and higher complexity in implementation for online configuration or reconfiguration problems. The SP scheduling policy performs better in the end-to-end delay compared with the scheduling for AVB traffic, as well the flexible window-based WND method proposed in this paper, and with the similar mean cost with the 0GCL method. Nevertheless, with SP it is difficult to protect against babbling-idiot faults thus leading to starvation for lower priority traffic. Moreover, the SP policy causes highly uncertain delays on all classes except the highest. AVB has the policy of bandwidth reservation allocation, thus avoids starvation for lower priority traffic, but it leads to larger worst-case delays compared with SP. Both SP and AVB have a comparable cost to 0GCL.

VI. CONCLUSION

We have presented a novel heuristic scheduler for TSN networks which uses a window-based approach where the previously required isolation of open gate windows is not necessary anymore. In order to still provide real-time guarantees, we combine the scheduling step with a worst-case delay analysis method based on previous work. We have evaluated our approach using both synthetic and real-world test cases and conducted a comparison with related work.

We are currently working on improving the analysis from [6] to make it sensitive to relative window positions on the whole network. Based on such an analysis, we will update the optimization approach presented in this paper. Our preliminary work indicates improvements in WCDs of over 50%, compared to the WND results in Table II. With these improvements, WND can be a promising alternative to the other methods of time-sensitive message transmission in TSN.

REFERENCES

[1] IEEE, "802.1ASrev—Timing and Synchronization for Time-Sensitive Applications," http://www.ieee802.org/1/pages/802.1AS-rev.html, 2017.

[2] IEEE, "802.1Qbv—Enhancements for Scheduled Traffic," [Online] http://www:ieee802:org/1/pages/802:1bv:html, 2015.

[3] S. S. Craciunas, R. Serna Oliver, M. Chmelik, and W. Steiner, "Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks," in *Proc. RTNS*, ACM, 2016.

[4] R. Serna Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1 Qbv gate control list synthesis using array theory encoding," in *Proc. RTAS*, 2018.

[5] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks," IET Cyber-Physical Systems: Theory & Applications, 1(1), 2016.

[6] L. Zhao, P. Pop, and S. S. Craciunas, "Worst-case latency analysis for ieee 802.1qbv time sensitive networks using network calculus," IEEE Access, vol. 6, 2018.

[7] W. Steiner, "Synthesis of static communication schedules for mixed criticality systems," in *Proc. ISORC*, 2011.

[8] IEEE, "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks," in IEEE Std 802.1Q-2014, pp.1-1832, 2014

[9] J. D. Boissonnat, and F. P. Preparata, "Robust plane sweep for intersecting segments," SIAM Journal on Computing, 29(5), 2000.

[10] M. L. Raagaard and P. Pop, "Optimization algorithms for the scheduling of IEEE 802.1 Time-Sensitive Networking (TSN)," Technical Report, Technical University of Denmark, 2017.