Out-of-sync Schedule Robustness for Time-sensitive Networks

Silviu S. Craciunas
TTTech Computertechnik AG, Vienna, Austria
silviu.craciunas@tttech.com

Ramon Serna Oliver
TTTech Computertechnik AG, Vienna, Austria
ramon.serna.oliver@tttech.com

***Abstract*—Time-Sensitive Networks (TSN) enable real-time communication guarantees over Ethernet by defining a timed-gate mechanism (802.1Qbv), binding frame transmissions to a global schedule, and a synchronization protocol (802.1AS), which aligns the time of each node to a reference clock. When the reference clock is lost, the individual clocks drift apart until a new master clock is elected (re-synchronization interval). This may lead to a complete loss of determinism due to an inconsistent alignment of the transmission patterns between nodes.**

In this paper, we address the problem of creating time-triggered schedules for 802.1Qbv with an enhanced robustness against the temporary loss of synchronization. We analyze the protocol behavior upon a loss of synchronization and derive new constraints for the computation of schedules able to tolerate the worst-case drift of individual clocks during the re-synchronization interval. Moreover, we formulate the problem as a design space exploration to synthesize schedules with the maximum tolerance towards loss-of-sync issues. We have developed a scheduler tool based on the Z3 SMT/OMT solver and conducted a series of experiments with a set of synthetic use-cases. Our evaluation exposes a series of trade-offs between the robustness of the schedule and the feasibility and computation run-time of the scheduler as well as the end-to-end communication latency.

I. Introduction

Many new areas, e.g., industrial automation, fog computing, and distributed control applications, require real-time communication guarantees, albeit not with the same degree of criticality as applications in, e.g., the aerospace domain. Time-Sensitive Networks (TSN) [1] introduce standardized mechanisms to enhance and guarantee the timely behavior of communication streams across Ethernet networks. Providing hard real-time guarantees for critical traffic is possible through the use of the time-aware gating mechanism defined in [2] in combination with a common global time provided through the use of a clock synchronization protocol [3]. The global schedule for the time-aware gates, encoded in so-called Gate-Control Lists (GCLs), specifies at which points in time frames are transmitted and forwarded out of the egress queues of the TSN devices in the network. This schedule is computed offline and relies on a global and common notion of time, i.e., it assumes that all clocks in the network are synchronized to one another within some bounded and known precision.

Schedule synthesis algorithms for TSN networks (e.g. [4]–[6]) take as an input the network precision and generate schedules that determine the transmission and forwarding times of all critical frames in the network. Since the local clocks of the network devices drift away from each other at a certain rate, they need to be synchronized to one another using a synchronization protocol (c.f. [7]). If the synchronization of the clocks is done at regular (pre-defined) intervals, the deterministic behavior of the communication satisfying the end-to-end latency requirements is maintained. However, if the synchronization fails at runtime and the local clocks start drifting away from each other uncontrolled, the real-time guarantees can no longer be ensured. This is particularly problematic since, in IEEE 802.1Qbv, the static schedule encoded in the GCL determines the timing of individual egress queues (traffic class) at which enqueued frames are eligible for transmission. This differs from other time-triggered protocols (e.g., TTEthernet), in which the schedule directly relates to the transmission of individual frames (c.f. [8]).

Synchronization loss can occur due to many reasons, either because of a device or link failure or due to a targeted attack on the reference node, which supplies the corrected time to the network's nodes (c.f. [7]). Adding fault-tolerance to the clock synchronization alleviates the problem; however, the feature introduced in the IEEE 802.1AS-rev [9] revision is not always supported. Moreover, a targeted attack can invalidate the fault assumption by (temporarily) eliminating all redundant clock grandmasters via, e.g., a DoS attack against the grandmasters or the synchronization protocol [7].

In this paper, we aim to create TSN schedules with enhanced resilience to synchronization loss. With our method, the created schedules will still guarantee deterministic frame behavior in the temporal domain in the case of a sporadic synchronization loss. The duration for which the critical frames maintain their real-time behavior (which we call the *re-synchronization interval*) is sufficiently long to allow the IEEE 802.1AS protocol to reestablish synchronization within the network. We call the property that schedules guarantee real-time behavior over the re-synchronization interval *out-of-sync robustness*. This robustness to synchronization loss is achieved by considering the maximum time interval that a network can be out of sync and computing an additional component to the nominal network precision. This extended precision is then used to generate robust schedule tables. Since the robustness comes at the expense of schedulability, we also present a design-space exploration that guides the IEEE 802.1AS parameters' configuration space. We show the feasibility of our approach using a set of synthetic experiments.

In Sect. II we introduce the TSN timed-gate and sync mechanisms, followed by our fault and network models (Sect. III), and discuss standard-compliant mitigation strategies for sync

978-1-6654-2478-3/21/$31.00 ©2021 IEEE

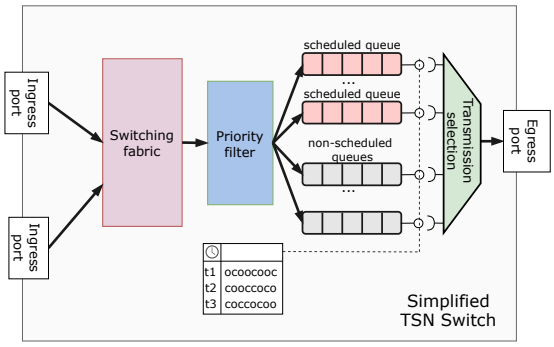


Figure 1. Simplified representation of an IEEE 802.1Qbv-capable switch.

loss scenarios in Sect. IV. We present our schedule synthesis approach for out-of-sync robustness as well as the design-space exploration problem in Sect. V. We show the feasibility of our approach using several synthetic experiments in Sect. VI, followed by a related work overview (Sect. VII), and concluding remarks in Sect. VIII.

II. Time-Sensitive Networks

A. Scheduled traffic in TSN networks

The time-aware gate mechanism defined in 802.1Qbv [2] enables or disables the transmission of frames from egress queues according to a pre-computed and pre-configured cyclic schedule referred to as a Gate Control List (GCL). Each egress queue of a port in an 802.1Qbv device has an associated timed-gate that can be either in an open (o) or closed (c) state defined in the GCL configuration. At run-time, each gate's state change is triggered by the local clock of the device.

Enqueued frames are sent in FIFO order when the associated gate opens. If multiple gates are open simultaneously, frames are sent according to the queue priority. Fig. 1 shows a model of an 802.1Qbv switch where frames ingress from the right-side port and are routed to one of the left-side egress ports via the switching fabric. A priority filter determines the assignment of frames to one of the 8 queues (traffic classes) based on criteria like the PCP of the 802.1Q header or the 802.1Qci filtering tables. Similar to [4], we assume that some queues are dedicated to scheduled (critical) streams, and the rest enqueue non-critical traffic (best-effort).

In TSN networks, time-aware nodes that participate in transmitting or forwarding critical real-time frames via 802.1Qbv must have their local clocks synchronized to a common time. Since the synchronization is never perfect, the clocks are synchronized so that the maximum difference between any two clocks is at most a predefined value, called *precision*. The main scheduling problem for time-aware networks is: given the network topology, device properties, communication requirements, and synchronization quality (precision), generate static schedules for the transmission and forwarding of critical frames so that the required temporal behavior is guaranteed.

In TSN, as opposed to other time-triggered networks like TTEthernet (SAE AS6802) [10], the transmission schedule refers to traffic classes (i.e., egress queues and their associated time-gates) instead of individual frames [8]. Hence, if

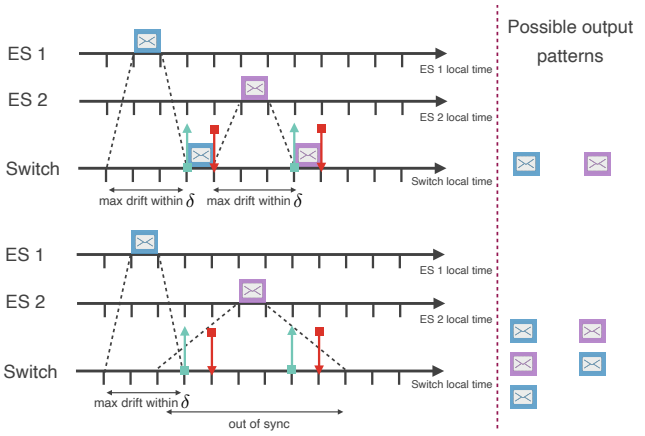


Figure 2. Non-deterministic frame transmission for out-of-sync devices.

frames are lost, payload sizes change, or the synchronization quality degrades beyond the configured precision, the order of enqueuing frames in the individual traffic classes may differ from the intended state (c.f. [4]). In other words, at design-time, the schedule is constructed to ensure that the order and timing of ingress (i.e. enqueuing) and egress (i.e. transmission) of frames is deterministic. However, when certain assumptions taken at design-time, like the synchronization precision, are not fulfilled at run-time, the temporal behavior may not be deterministic anymore due to the variation in queue states.

In Fig. 2 we show an example of this effect. Two end-systems (ES1 and ES2) are connected to a switch (SW). Under the precision assumption (top timeline), the transmission of a frame from ES1 will arrive within some bounded interval with respect to the SW local clock. The same situation occurs for the frames sent by ES2. Regardless of how the local clocks of ES1, ES2, and SW drift in relation to each other, as long as they stay within the assumed precision, the opening and closing of gates in the switch queue (shown as green and red arrows) lead to the same pattern of transmission, even when both frames are stored in the same SW queue. However, when the difference between the local clocks of ES2 and SW drift further apart than the pre-configured precision, shown in the bottom timeline, the time interval in which a frame from ES2 may arrive with respect to the switch's local clock can be much larger than planned. If the local clock of ES is faster than the local clock of SW, the frame sent from ES2 may arrive before the one sent from ES1. Since the opening and closing of the queue gate in the SW does not depend on the queue state, the frame from ES2 will be sent first. On the contrary, if the local clock of ES2 is slower than the one of SW, the frame from ES2 may arrive after its planned time slot and hence never be sent. These patterns may occur at run-time when the assumption on the bounded precision does not hold, hence breaking the determinism and real-time properties of the streams.

B. Synchronization and IEEE 802.1AS

In IEEE 802.1AS [9], the term precision refers to the deviation of a local clock with respect to an ideal clock reference, usually referred to as the clock grandmaster (GM).

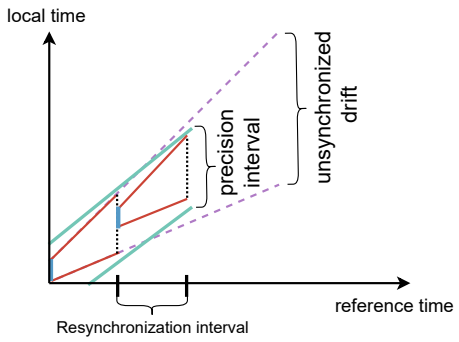


Figure 3. Clock synchronization (inspired by [11, p.59]).

In order for the local time of each device to not drift in relation to the reference clock further away than the desired precision, all clocks need to be synchronized with a certain rate, also called the *synchronization interval* [11, p.59], as shown in Fig. 3. Maintaining the synchronization requires a correction that adjusts the value of the clocks in such a way that the worst-case drift in relation to each other over the subsequent synchronization interval will not cause the relative offset to exceed the desired precision [11, p.60]. A precision of $< 1\mu s$ relative to the reference clock is possible for networks with under 5 hops and 1 Gbit/s links [12]; however, this precision is not achievable for nodes that are more than 30 hops away from the reference clock node [13]. In order to provide the necessary synchronization, the IEEE 802.1AS standard mandates that at least one node is defined as a clock grandmaster (GM). The GM constitutes the root node of the synchronization spanning tree, which is constructed via the Best Master Clock Algorithm (BMCA) [14]. For each clock domain, the BMCA constructs one spanning tree consisting of the time-aware nodes in the respective domain and paths from the GM to these nodes [12]. Once the tree is constructed, the GM sends its time value to the connected neighboring nodes within the spanning tree for the given synchronization domain. Each switch then corrects the received time by adding the propagation delay and residence time in the switch and forwards the corrected time to the next nodes in the spanning tree [14]. We refer the reader to [12], [13] for a detailed description of 802.1AS.

Within the same clock domain, all nodes use the same GM. If the GM node fails, a new GM has to be elected, and the spanning tree has to be recreated via the BCMA. In the reelection phase, the first step of the process is detecting the loss of a GM via a time-out value specified by the standard (default is 3s [9]), which, in essence, implements a watchdog timer checking for *announce* messages from the GM. In a second step, the BMCA runs, and each node sends its clock value until the spanning tree is recomputed.

The IEEE 802.1AS-rev [9] is a revision of 802.1AS within the TSN context, introducing fault-tolerance for clock synchronization. The upgraded protocol allows multiple clock domains to be simultaneously active and overlapping, therefore enabling multiple GMs to be active at the same time [15]. These GMs deliver their time values to independent clock domains, and if the domains overlap, a hot-standby GM mechanism is enabled. Hence, whenever a GM fails, the transition to another GM is seamless. Additionally, the revision offers improved accuracy of the time measurement [16]. However, the hot-standby features are not yet supported in most currently available TSN devices on the market or may not always be configured for the given network.

III. Fault and Network Models

We represent the TSN network as a finite, unweighted, connected, undirected graph $G = (V, E)$ consisting of the set of time-aware nodes (vertices) V and the set of bi-directional, full-duplex physical links (edges) E. If nodes are not time-aware or are not part of the considered synchronization domain, they are not part of the set V, and their connected edges (links) are not part of the set E.

When synchronization is lost, the clocks of individual devices begin to drift apart, eventually exceeding the bounds of the assumed precision. The synchronization loss will be detected after the configured timeout time (see below), and the BMCA will execute to recompute the spanning tree and select a new GM. This re-computation takes a certain amount of time and is proportional to the number of hops in the spanning tree. After this, the new time will be propagated to the devices, which will be synchronized again. In the case of a GM failure, we assume an arbitrary runtime selection of the next GM out of a defined set of eligible GM nodes. The resulting spanning trees have the respective new GM node as the root.

A subgraph $T_v = (V, E' \subseteq E)$ is a rooted spanning tree of G, denoted with $T_v \overset{\text{r}}{\mapsto} G$, if it is a connected, acyclic subgraph of G where all paths to leaf nodes originate in the root node v. We denote the set of nodes that can take the role of grandmaster with $GM(V)$. The longest path in terms of the number of edges of a tree T is denoted by $l(T)$. The number of hops for the longest path out of all the possible spanning trees for a given network G is denoted with

$$N_G = \max\{l(T_v) \mid v \in GM(V), T_v \overset{\text{r}}{\mapsto} G\}. \tag{1}$$

We define the *out-of-sync interval* t_{resync} as the time span between the instant an out-of-sync occurs until the moment when the affected clocks are again resynchronized. We assume the following model for computing the *out-of-sync interval* of a given network G:

$$t_{resync} = \delta_t + \delta_{hop} \times N_G, \tag{2}$$

where δ_t is the timeout bound for out-of-sync detection, δ_{hop} is an upper bound for the BMCA to run and propagate information on each node, and N_G is derived from Eq. 1. Typically, a node waits for 3 announce cycles (each cycle being by default 1s) until it assumes that sync has been lost. This is defined in [9] as the *announceReceiptTimeoutTimeInterval* which is *announceReceiptTimeout* $\times$ *announceInterval*, both of which can be configured. The δ_{hop} duration is not defined in the standard, but we extrapolate a safe upper bound of 1s per hop, supported by our empirical observations and in-house development know-how. However, please note that this value

needs to be configured as a safe upper-bound for the given network and devices.

We cannot trust the timeliness of critical messages during the out-of-sync interval until all nodes are re-synchronized. However, we can compute the maximum deviation between any two clocks at the end of the out-of-sync interval. We introduce the *out-of-sync drift*, denoted by Δ_s, to represent the maximum drift between any two clocks at the end of the *out-of-sync interval*. We bound the worst-case clock drift rate ρ_{max} for all clocks which means that between any two clocks, the drift is at most $\rho = 2 \times \rho_{max}$ $[s/s]$ [17]. The value for ρ_{max} depends on the quality of the individual clock oscillators. Typical values of ρ_{max} are between 50 and 100ppm, meaning that the maximum deviation between any two clocks is between 50 and $100\mu s$ per second. The *out-of-sync drift*, Δ_s $[\mu s]$, is computed as

$$\Delta_s = 2 \times \rho_{max} \times t_{resync} \tag{3}$$

If, for e.g., we consider the worst-case configuration with $N_G = 3$, $\delta_t = 3s$, $\delta_{hop} = 1s$ and $\rho_{max} = 100ppm$, the upper bound on the deviation between any two clocks following a GM failure at the point of re-synchronization is $1200\mu s$.

IV. Mechanisms for out-of-sync mitigation

An approach to add robustness in cases of synchronization loss is to mitigate non-deterministic behavior by introducing an *out-of-sync mode* in case of sync loss. The out-of-sync mode aims to contain the damage that arises from the arrival of frames outside the expected time bounds, for example, by triggering a priority regeneration within the affected devices (e.g., via 802.1Qci [1]), causing a redirection of scheduled frames to low-priority, non-scheduled queues. If a segment of the network (e.g., being in a different time domain) is not affected by the synchronization loss, we want to prevent frames coming from the faulty devices (or time domain) from interfering with the deterministic queue states of the synchronized devices. Critical frames received by an out-of-sync device will be re-assigned into a non-scheduled queue at the next (potentially in-sync) node. Hence, subsequent nodes will not have scheduled queues polluted by out-of-time frames.

Despite limiting the impact on the critical frames being forwarded to synchronized devices, there may still be some frames within an out-of-sync device that were enqueued before the priority regeneration became effective or even ahead of the sync-loss detection. Therefore, as soon as the respective timed-gate opens, potentially at an incorrect time, the transmitted frames may not correspond to the original schedule.

A possible strategy to prevent the propagation of these types of incorrect frames is to use the Per-Stream Filtering and Policing mechanism defined in IEEE 802.1Qci [1]. The standard provides mechanisms to control the admission of frames on ingress based on timing parameters driven by the local clock. The mechanism allows rejecting frames if their arrival time is not equal to the one specified in the local schedule. Hence, nodes can filter frames from devices that have experienced a synchronization loss and thus maintain their queues in a deterministic state. A benefit of using time-based policing is that it is independent of the detection of a sync loss. If one device is out of sync on the critical stream route, its frames will be dropped at the subsequent device since they will inherit the wrong timing behavior. Therefore critical frames will only be transmitted from sender to receiver(s) if all forwarding devices are synchronized.

With the first method, a sync loss will still lead to critical frames being received at the listener devices, albeit with the wrong timeliness, while the filtering method will not let critical frames arrive at the receiver.

V. Schedule-based sync loss robustness

The main problem with the mitigation strategies presented above is that until the synchronization loss is detected (e.g., up to 3s), the clocks' drift can be significant and exceed the precision specified at design-time. Hence, while we may be able in some situations to prevent the impact of ill-timed frames, the partial loss of synchronism will lead to compromising a subset of the critical communication. We propose a mitigation strategy consisting of enhancing the synthesis of schedules to feature robustness during the re-synchronization intervals, i.e., we create the schedules so that, even when synchronization is lost, the state of the queues will be deterministic until synchronization is restored. Through this, we ensure that deterministic queue states are preserved and that all critical frames reach their destinations within their allowed end-to-end latency constraint.

Based on our previous work on scheduling TSN networks [4], we extend the scheduling constraints used to synthesize GCL schedules.The schedule events (i.e., the frame transmission and forwarding times) will be placed so that an extended deviation from the nominal network precision is tolerated, which accounts for the worst-case drift in the network over the re-synchronization interval. In this way, even when a GM fails, the schedule will still be deterministic, and the real-time properties of streams will still be fulfilled until a new GM is elected and the synchronization is restored. The new scheduling method entails a trade-off between a reduction of the solution space in exchange for robust schedules tolerating synchronization losses. In [4], we have presented the necessary correctness constraints for generating valid TSN schedules and formalized them based on a generic model of TSN networks. We use a similar notation as in our previous work [4], [8] but introduce certain simplifications without loss of generality, e.g., the macrotick is the same for all devices, there is one frame per stream, and the link propagation delay is 0. Please note that the extensions presented in this paper can be readily generalized for the more general model presented in [4], [6].

The TSN network, encoded as a graph $G = (V, E)$, comprises a set of time-aware nodes (vertices) V and a set of bi-directional, full-duplex physical links (edges) E, where a link from node v_a to node v_b is expressed as $(v_a, v_b) \in E$. We denote the set of critical streams with $\mathcal{S}$ where a stream $s_i \in \mathcal{S}$ is defined by the tuple $\langle l_i, T_i, D_i \rangle$, which contains the frame size, the period, and the end-to-end deadline. The route

for a stream s_i is computed at design-time and formalized as an ordered sequence of links, e.g., the communication route from a sender v_1 to a receiver v_n is represented via the set $R_i = \{(v_1, v_2), ..., (v_{n-1}, v_n)\}$. We formalize, similar to [4], [8], the frame of a stream s_i on a link (v_a, v_b) as $f_i^{(v_a,v_b)}$ with $\phi_i^{(v_a,v_b)}$ and $l_i^{(v_a,v_b)}$ representing the transmission time from node v_a through link (v_a, v_b) and the duration of the frame on the respective link, respectively. The set of all frames which are sent on a link (v_a, v_b) is denoted by $\mathcal{F}^{(v_a,v_b)}$.

For the sake of brevity, we will not repeat all of the constraints and refer the reader to [4], [8] for the complete formalism. We only adapt the constraints in which the precision of the network is included and extend them to account for the *out-of-sync drift* Δ_s over the *out-of-sync interval*.

Flow Transmission Constraint. The constraint ensures that a frame can only be forwarded from a device once it has been received on that device. Since the reception of a frame on a device is defined by the sending time on the previous hop, the constraint relates the transmission times in two different devices and hence needs to take into account the network precision, denoted with δ. We extend this constraint from [4] to include not only the precision but also the *out-of-sync drift* Δ_s over the *out-of-sync interval*:

$$\forall s_i \in \mathcal{S}, \forall (v_a, v_x), (v_x, v_b) \in R_i : \\ \phi_i^{(v_x,v_b)} - (\phi_i^{(v_a,v_x)} + l_i^{(v_a,v_x)}) \geq \delta + \Delta_s.$$

By including the maximum drift Δ_s, we ensure that the sending time of the frame instance of a stream from node v_x is placed at a safe distance from the sending time of the predecessor frame instance in node v_a. By safe, we mean that if there is a loss of sync and the two involved clocks drift further apart from the network's precision, the frame will still be forwarded by node v_x only after it has been received. The upper bound on the *out-of-sync drift* ensures that the interval is safe since by the time the drift reaches $\delta + \Delta_s$, the network will be re-synchronized with a precision below δ.

End-to-End Latency. The deadline of a stream, denoted by $s_i.D$, captures the requirement that the time from the sending of a stream at the sender until the stream's reception at the receiver is less than or equal to the desired bound. We use $src(s_i) \in R_i$ and $dest(s_i) \in R_i$ to represent the sender link and the last link before the receiving node, respectively. The maximum end-to-end latency constraint (extended from [4]) is

$$\forall s_i \in \mathcal{S} : \phi_i^{dest(s_i)} + l_i^{dest(s_i)} - \phi_i^{src(s_i)} \leq D_i - (\delta + \Delta_s).$$

Here we have to consider that the clocks of the sender and receiver nodes can drift beyond the precision of the network in case of a loss of synchronization. Again, the condition is safe since in the worst case, by the time the drift exceeds $\delta + \Delta_s$, the network will be re-synchronized. This constraint clearly shows that the solution space is constrained by the additional term subtracted from the deadline in order to achieve a safe bound in case of synchronization loss.

802.1Qbv Flow/Frame Isolation. The frame/flow isolation constraint imposes a deterministic state of the queues. We refer the reader to [4] for an in-depth explanation of the queue determinism problem and the complete formalization. Let $f_i^{(v_a,v_b)}$ and $f_j^{(v_a,v_b)}$ be the frame instances of streams $s_i \in \mathcal{S}$ and $s_j \in \mathcal{S}$ on link (v_a, v_b), respectively. The frames are both sent on link (v_a, v_b) of device v_a. Stream s_i arrives from device v_x connected to v_a via link $(v_x, v_a) \in R_i$ and stream f_j arrives from device v_y connected to v_a via $(v_y, v_a) \in R_j$. For any link (v_a, v_b) and any such frame pair $f_i^{(v_a,v_b)}$ and $f_j^{(v_a,v_b)}$, we extend the stream isolation constraint from [4] to include the *out-of-sync drift* Δ_s as follows:

$$\forall \alpha \in \left[0, hp_i^j/T_i\right), \forall \beta \in \left[0, hp_i^j/T_j\right) : \\ (\phi_j^{(v_y,v_a)} + \beta \times T_j - \phi_i^{(v_a,v_b)} - \alpha \times T_i \geq \delta + \Delta_s) \vee \\ (\phi_i^{(v_x,v_a)} + \alpha \times T_i - \phi_j^{(v_a,v_b)} - \beta \times T_j \geq \delta + \Delta_s).$$

The hyperperiod of the two streams is $hp_i^j = lcm(T_i, T_j)$ and $\alpha, \beta \in \mathbb{N}$ are index to the period instances. The constraint ensures that once a device receives a stream, no other stream sharing the same queue can enter the device until the first stream has been transmitted and the queue is empty. Since this constraint also relates the sending times of frames on different devices, we have to consider both the precision and the maximum drift in order to make the resulting schedule robust during the re-synchronization interval.

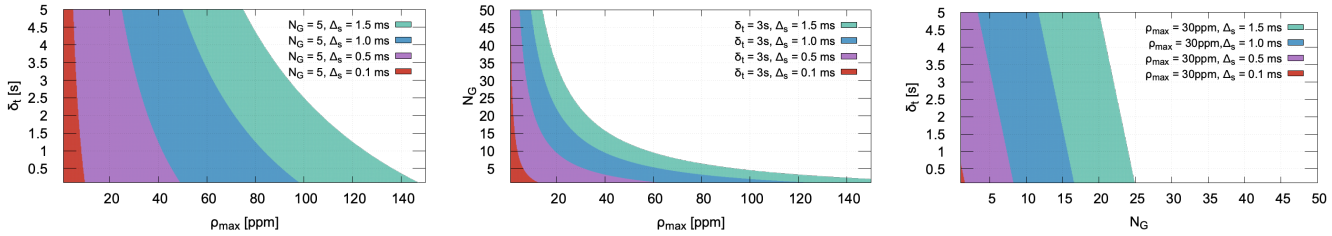
A. Design space exploration

When generating the schedule with the *out-of-sync drift* Δ_s, the solution space is reduced, and previously feasible use-cases may become unschedulable. For example, it may occur that the utilization on some links or the end-to-end deadline constraints of some streams is too tight to accommodate the maximum drift. In such cases, we cannot guarantee that the schedule is robust during the out-of-sync interval.

We can transform the problem into a design space exploration by configuring the maximum drift to be a variable that is computed by the scheduler rather than a fixed constant input. Thus, the scheduling problem becomes an optimization problem, with the optimization objective being to *maximize* the *out-of-sync drift* Δ_s subject to the constraints defined above and in [4]. We can use the resulting maximized value of the *out-of-sync drift* Δ_s to select the best configuration parameters in terms of δ_t[1], which is the timeout bound for out-of-sync detection, the maximum number of hops N_G, and the worst-case clock drift rate ρ_{max}. Selecting a value for one parameter will constrain the possible values for the other dimensions in the configuration space. The easiest parameter to change is δ_t since the devices' clocks cannot be easily changed, and the topology is often fixed. However, it may be necessary in some cases to select devices with better clocks or reconfigure the network such that the maximum number of hops is reduced.

We show the configuration space for different example networks where the maximized Δ_s is $100, 500, 1000, 1500\mu s$ in Fig. 4. Please note that configuration spaces with smaller

[1] δ_t is configured via the *announceReceiptTimeout* and the *announceInterval* values in the IEEE 802.1AS configuration.



(a) ρ_{max} vs. δ_t when $N_G = 5$. (b) ρ_{max} vs. N_G when $\delta_t = 3s$. (c) N_G vs. δ_t when $\rho_{max} = 30ppm$.

Figure 4. Different configuration space dependencies for example networks where $\Delta_s \in \{100, 500, 1000, 1500\}\mu s$.

areas are plotted over the ones with larger areas where they overlap. In Fig. 4(a) we show the configuration space for ρ_{max} and δ_t when $N_G = 5$. We can see that, for high values of Δ_s, the timeout bound for out-of-sync detection δ_t can be as high as $5s$ (for typical values of ρ_{max} between $50ppm$ and $100ppm$), whereas for low values of Δ_s, we either have to reduce δ_t or select devices with very accurate clocks (under $10ppm$). In Fig. 4(b) we show the dependency between ρ_{max} and N_G when $\Delta_s = \{100, 500, 1000, 1500\}\mu s$ and δ_t is set to the default value of $3s$. We can see that the quality of the clocks needs to be improved significantly if we have networks where the maximum number of hops in the spanning tree goes beyond 5. In Fig. 4(a) we plot the configuration space of δ_t as a function of N_G for networks with the same Δ_s as above and with with $\rho_{max} = 30ppm$ clocks. We can see that if the resulting *out-of-sync drift* Δ_s is very low, we can only guarantee out-of-sync robustness for very small networks with under 5 hops, while as the *out-of-sync drift* gets larger, we can support networks with up to 25 hops depending on the configured value of the out-of-sync detection interval δ_t.

Based on our fault model, we can generate robust schedules against failures of the GM and can tolerate the network being out of sync for the duration of the *out-of-sync interval*. After the *out-of-sync interval*, the network devices are again synchronized to below the originally configured precision. Hence, the schedule is again robust to a new loss of synchronization. However, if a device fails while the corrected time is being propagated through the spanning tree, i.e., if a failure occurs while the devices' clocks are not within the precision envelope, the robustness property does not hold anymore. In the case of such *cascading* failures, the *out-of-sync interval* may be longer than the one computed at design time (see Sect. III). Such types of failures are not fully tolerated by our method. However, in the design space exploration, we can maximize the *out-of-sync interval* by maximizing Δ_s and hence mitigate, to some degree, the impact of cascading failures. Furthermore, with the enhancements in IEEE 802.1AS-rev [3] introducing hot-standby grandmasters, it will be possible to tolerate the loss of all GMs before entering the re-synchronization phase.

VI. Experiments

We have developed a customized scheduler tool computing optimized GCLs with a configurable precision parameter. The tool is based on the frame-based scheduler in [4] using the Z3 [18] SMT/OMT solver v.4.8.10 running on a 64bit Cygwin

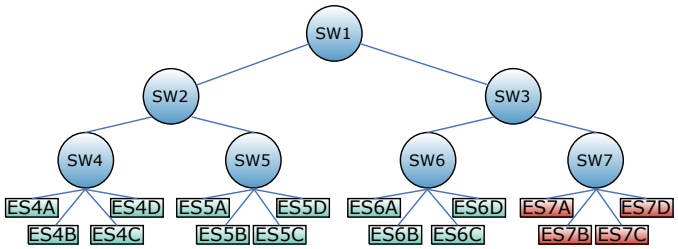


Figure 5. Test Topology.

environment within Windows 10. Unless otherwise stated, all experiments assume up to 2 dedicated queues for 802.1Qbv unicast scheduled traffic with the scheduler macrotick fixed at $1\mu s$, a constant link latency of $1\mu s$, and homogeneous link speeds of 1Gbps. The hardware platform is an Intel i7-8650U CPU @1.90GHz notebook with 16GB of main memory.

A. Schedulability

We assess the impact of our approach on the network schedulability by running a set of experiments on a synthetic scenario with the tree topology depicted in Fig. 5. The test network consists of 7 switches (SW1 to SW7), of which SW4 to SW7 connect 4 end nodes each, named ESiA, ESiB, ESiC, and ESiD, where i refers to the SW index ($4 \leq i \leq 7$). Each end node from SW4, SW5, and SW6 transmits full-sized Ethernet messages (i.e., a payload of 1500 byte) with a periodic data stream to each end node of SW7. Each node of SW7 transmits a response 100-byte payload message in opposite streams to all other nodes in SW4, SW5, and SW6. We adjust the period of communication between 10 ms and $500\mu s$ to increase the link utilization, with the stream deadline being equal to the period. Table I summarizes the configuration settings for the experiments and the schedulability results for different configurations. The permutation of these values result in $\Delta_s = \{1200, 800, 600, 400, 60, 40\}$. The results show how the choice of parameters affecting the schedule robustness impacts the schedulability when the utilization increases, particularly in the bottleneck link (SW3→SW7), where the utilization increased from 5.76% to 57.6%.

B. End-To-End Latency

We evaluate the impact of adding out-of-sync robustness on the achievable end-to-end latency of streams. We have instrumented our scheduler to minimize the end-to-end latency of each stream, combining the individual objectives as Pareto

Table I
SCHEDULABILITY RESULTS FOR DIFFERENT Δ_s CONFIGURATIONS.

ρ_{max} [ppm]	100	100	50	50	5	5	100	100	50	50	5	5	100	100	50	50	5	5
$\delta_t[s]$	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1
$\Delta_s[\mu s]$	1200	800	600	400	60	40	1200	800	600	400	60	40	1200	800	600	400	60	40
Max util. [%]	57.6	57.6	57.6	57.6	57.6	57.6	11.52	11.52	11.52	11.52	11.52	11.52	5.76	5.76	5.76	5.76	5.76	5.76
Runtime [ms]	218	249	203	234	343	390	219	312	391	407	406	422	437	359	343	390	389	422
Schedulability	false	false	false	false	true	true	false	false	true	true	true	true	true	true	true	true	true	true

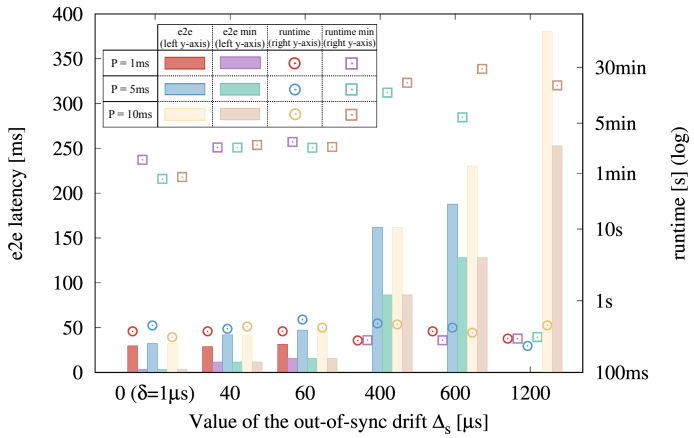


Figure 6. End-to-end latency with and without minimization objectives.

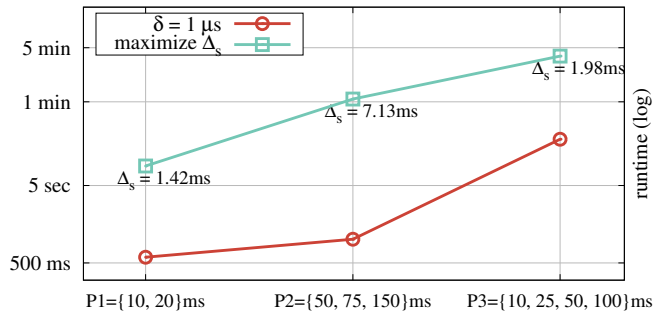


Figure 7. Scheduler runtime with and without Δ_s maximization objective.

fronts (c.f. [18]). Fig. 6 depicts the accrued end-to-end latency of all streams, with and without out-of-sync robustness, for the set of scenarios evaluated in Sect. VI-A. We depict the accrued end-to-end latency (left y-axis) when we add the minimization objective (e2e min) as well as without optimization (e2e) for streams featuring the 3 periods from before and for $\Delta_s = \{0, 40, 60, 400, 600, 1200\}\mu s$ (x-axis). Both the non-optimized and the optimized e2e latencies increase when the out-of-sync drift increases. For certain period and Δ_s values, the use-cases were not schedulable (bars are missing in the plot). We also plot the runtime to schedule each use-case (logarithmic right y-axis) using circles and squares for the non-optimized (runtime) and optimized (runtime min) variants, respectively. As expected, the scheduler runtime increases when adding optimization objectives and rises from 1min when $\Delta_s = 0$ to 28min for $\Delta_s = 600\mu s$ (16min for $\Delta_s = 1200\mu s$). For the non-schedulable cases, the scheduler returns an infeasible result in under $1s$.

C. Schedule Synthesis Time

Maximizing the precision as part of a design space exploration allows finding a schedule with the highest possible robustness against out-of-sync intervals. In Fig. 7 we compare the scheduler run-time of computing a schedule for 3 different period configurations, namely $P1 = \{10, 20\}$, $P2 = \{50, 75, 150\}$, and $P3 = \{10, 25, 50, 100\}$ [ms], when maximizing the precision in contrast to that of finding a schedule with a constant precision of $1\mu s$. For each run, we generate a set of 50 streams with randomly chosen end nodes as sender and receiver, a payload size randomly selected from the set $\{500, 1000, 1000\}$ [byte], as well as period, with an equivalent deadline, randomly selected from the respective set. Note that the scheduler tool used for these experiments attempt to schedule all streams at once and does not take advantage of incremental methods described in previous literature (c.f. [4]). Incremental scheduling algorithms with backtracking steps significantly improve the schedulability of SMT-based schedulers (c.f. [4], [19]–[21]. However, we remark that this is not relevant for comparing the relative difference between the two evaluated scheduling synthesis objectives (i.e., with vs. without maximized robustness). The figure shows a penalty due to the maximization objective. Moreover, we observe that the increase in run-time due to the precision maximization is significantly smaller than that of minimizing the end-to-end latency. We relate this to the complexity of the minimization objective itself, since in the former case, it relates to a single variable, the precision, while in the latter, there is a relation between the offset of two frames for each stream (i.e., the first sent and the last received). Additionally, each stream adds its own minimization objective resulting in a multi-optimization problem. Please note that once the maximized value of Δ_s has been computed for a scenario, it can be used as a constant value in certain re-scheduling iterations.

VII. RELATED WORK

Offline scheduling for TSN (or other deterministic networks) uses design-time upper bounds on the network's precision. Such methods (e.g., [4]–[6], [19], [21], [22]) have relied on the network never losing synchronization or there being a fault-tolerant clock synchronization algorithm in order to guarantee real-time behavior for critical communication. For TTEthernet networks [10], the upper bound on the synchronization precision has been proven using SMT-based model checking [23]. No such proof currently exists for IEEE 802.1AS. However, the work in [13] shows via simulations that a $1\mu s$ precision, as requested by the IEEE 802.1AS standard, is realistic for networks with a maximum spanning tree hop count of under 30 ($N_G \leq 30$), while in a $N_G = 100$ network the resulting precision is $2\mu s$. Another simulation result for

measuring the IEEE 802.1AS precision as a function of the network load in automotive networks is presented in [24].

Naturally, there has been a large body of work introducing fault-tolerance in the clock synchronization domain in order to prevent a loss of synchronization in the first place and guarantee the real-time behavior of the network (e.g., [17], [25]–[27]). However, if the fault assumption is not fulfilled, i.e., a scenario outside the failure model occurs, the deterministic behavior cannot be guaranteed.

Several papers propose extensions to IEEE 802.1AS that improve the synchronization accuracy (e.g., [28], [29]). Better accuracy of the synchronization at runtime increases the probability that TSN schedules that have been created under a worst-case assumption of the synchronization quality are robust in case of synchronization loss. However, unlike our approach, these methods do not provide a design-time guarantee that the communication behavior will remain deterministic and adhere to the real-time deadlines in case of synchronization loss.

Resilient time synchronization is complementary to our work and focuses on preventing synchronization loss or alleviating its effects at runtime. The work in [30] statistically evaluates the synchronization uncertainty and offers a prediction on the clock quality that may be used to maintain synchronization (at least to some degree) in case of synchronization loss. Other works focus on either making the synchronization protocol resilient towards external attacks [31] or on detecting attacks and reducing the consequences thereof within the synchronization protocol [32]. Similar to the previously mentioned approaches, these mechanisms cannot guarantee continued real-time behavior in case of synchronization loss.

VIII. Conclusion

An intermittent hardware failure or intentional sabotage of the 802.1AS grandmaster, among other possible issues, may trigger a reelection mechanism causing a critical disruption in the synchronization service. In this paper, we have analyzed the effect and worst-case duration of the resynchronization interval in 802.1AS and derived additional constraints for the synthesis of 802.1Qbv schedules with *out-of-sync robustness*. We have also shown how the schedule computation can be transformed into a design space exploration problem maximizing the communication scheme's resilience towards time synchronization anomalies. We have implemented a scheduling tool based on the Z3 OMT/SMT solver and conducted a series of experiments on a set of synthetic benchmarks. Our analysis shows the impact on schedulability, computation runtime, as well as the trade-off between robustness and the achievable end-to-end latency of the communication streams.

Acknowledgments

We want to thank Timo Koskiahde and Jürgen Wohlmuth for their invaluable assistance in understanding IEEE 802.1AS(-rev) and their helpful comments on the approach in this paper. The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 952702 (BIECO).

References

[1] IEEE, "Time-Sensitive Networking Task Group," http://www.ieee802.org/1/pages/tsn.html, 2016, retrieved 29.01.2021.
[2] IEEE, "802.1Qbv - Enhancements for Scheduled Traffic," https://standards.ieee.org/standard/802_1Qbv-2015.html, retrieved 29.01.2021.
[3] IEEE, "802.1AS-Rev," https://1.ieee802.org/tsn/802-1as-rev/, 2020.
[4] S. S. Craciunas, R. Serna Oliver, M. Chmelik, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv Time Sensitive Networks," in *Proc. RTNS*. ACM, 2016.
[5] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *Proc. ETFA*, 2018.
[6] R. Serna Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv Gate Control List Synthesis using Array Theory Encoding," in *Proc. RTAS*. IEEE, 2018.
[7] M. Lévesque and D. Tipper, "A survey of clock synchronization over packet-switched networks," *IEEE Commun. Surv. Tutor.*, vol. 18, 2016.
[8] S. S. Craciunas and R. Serna Oliver, "An overview of scheduling mechanisms for time-sensitive networks," Tech. rep. ETR, 2017.
[9] IEEE, "IEEE Std 802.1AS-2020," https://standards.ieee.org/standard/802_1AS-2020.html, 2020, retrieved 29.01.2021.
[10] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, "TTEthernet: Time-Triggered Ethernet," in *Time-Triggered Comm.* CRC Press, 2011.
[11] H. Kopetz, *Real-Time Systems - Design Principles for Distributed Embedded Applications*, ser. Real-Time Systems Series. Springer, 2011.
[12] M. D. Johas Teener and G. M. Garner, "Overview and timing performance of ieee 802.1as," in *Proc. ISPCS*. IEEE, 2008.
[13] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "Synchronization quality of IEEE 802.1AS in large-scale industrial automation networks," in *Proc. RTAS*. IEEE, 2017.
[14] H.-T. Lim, D. Herrscher, M. J. Waltl, and F. Chaari, "Performance Analysis of the IEEE 802.1 Ethernet Audio/Video Bridging Standard," in *Proc. SIMUTOOLS*, 2012.
[15] IEEE, "P802.1ASdm – Hot Standby," https://1.ieee802.org/tsn/802-1asdm/, retrieved 01.02.2021.
[16] A. Nasrallah *et al.*, "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G Ull research," *IEEE Commun. Surv. Tutor.*, vol. 21, no. 1, 2019.
[17] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Trans Comput*, vol. C-36, no. 8, 1987.
[18] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νz - an optimizing SMT solver," in *Proc. TACAS*. Springer, 2015.
[19] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Proc. RTSS*. IEEE, 2010.
[20] S. S. Craciunas and R. Serna Oliver, "SMT-based task- and network-level static schedule generation for time-triggered networked systems," in *Proc. RTNS*. ACM, 2014.
[21] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks," *IEEE Trans Industr Inform*, vol. 14, no. 5, 2018.
[22] S. S. Craciunas and R. Serna Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, 2016.
[23] W. Steiner and B. Dutertre, "Automated formal verification of the TTEthernet synchronization quality," in *Proc. NFM*. Springer, 2011.
[24] H. Lim, D. Herrscher, L. Völker, and M. J. Waltl, "IEEE 802.1AS time synchronization in a switched Ethernet based in-car network," in *Proc VNC*. IEEE, 2011.
[25] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *J. ACM*, vol. 32, no. 1, p. 52–78, 1985.
[26] D. Dolev, J. Halpern, B. Simons, and H. Strong, "Dynamic fault-tolerant clock synchronization." *J. ACM*, vol. 42, pp. 143–185, 1995.
[27] M. Paulitsch and W. Steiner, "Fault-tolerant clock synchronization for embedded distributed multi-cluster systems," in *Proc. ECRTS*, 2003.
[28] H. Baniabdelghany, R. Obermaisser, and A. Khalifeh, "Extended synchronization protocol based on IEEE802.1AS for improved precision in dynamic and asymmetric TSN hybrid networks," in *Proc. MECO*, 2020.
[29] H. Zhu *et al.*, "Measures to improve the accuracy and reliability of clock synchronization in time-sensitive networking," *IEEE Access*, 2020.
[30] A. Bondavalli *et al.*, "Resilient estimation of synchronisation uncertainty through software clocks," *IJCCBS*, vol. 4, no. 4, p. 301–322, 2013.
[31] K. Sun, P. Ning, and C. Wang, "TinySeRSync: secure and resilient time synchronization in wireless sensor networks," in *Proc. CCS*, 2006.
[32] E. Lisova *et al.*, "Protecting clock synchronization : Adversary detection through network monitoring," *J. Electr. Comput. Eng.*, 2016.