2018 IEEE Real-Time and Embedded Technology and Applications Symposium

IEEE 802.1Qbv Gate Control List Synthesis using Array Theory Encoding

Ramon Serna Oliver Silviu S. Craciunas Wilfried Steiner
TTTech Computertechnik AG, Vienna, Austria
{rse,scr,wst}@tttech.com

***Abstract*—Time Sensitive Networks (TSN) emerge as the set of sub-standards incorporating real-time support as an extension of standard Ethernet. In particular, IEEE 802.1Qbv defines a time-triggered communication paradigm with the addition of a *time-aware shaper* governing the selection of frames at the egress queues according to a predefined schedule, encoded in so-called *Gate Control Lists* (GCL). Nonetheless, the design of compositional systems with real-time demands requires a proper configuration of these mechanisms to truly achieve the temporal isolation of communication streams with end-to-end timeliness guarantees. In this paper we address how the synthesis of communication schedules for GCLs defined in IEEE 802.1Qbv can be formalized as a system of constraints expressed via *first-order theory of arrays* ($\mathcal{T}_A$). We formulate the necessary constraints showing the suitability of the theory of arrays and discuss optimization opportunities arising from the underlying scheduling problem. Our evaluation using general-purpose SMT/OMT solvers proves the validity of the approach, scaling well for small- to medium-networks, and exposing trade-offs for the time needed to synthesize a schedule. Furthermore, we conduct a comparison against previous work and conclude the appropriateness of the method as the basis for future TSN scheduling tools.**

I. Introduction

Deterministic real-time communication has long been a requirement in the aerospace domain [1]. The strictness of certification and industry practices are only satisfied if sufficient proofs of evidence guarantee the deterministic behavior of static configurations, which are often deployed in production programs spanning over several decades. In recent years, fast-moving markets like automotive and industrial automation are increasingly joining the trend of deterministic networking albeit being reluctant to accept a detriment when it comes to generalized networking features, like high communication speeds, near-to-full bandwidth utilization, off-the-shelf component availability, or the capability of dynamic reconfiguration.

The *IEEE 802.1 Time Sensitive Networking (TSN) task group* [2] has been active standardizing time-sensitive capabilities for *Ethernet* networks ranging from distributed clock synchronization [3] and time-based ingress policing [4] to frame preemption [5], redundancy management [6], and scheduled traffic enhancements [7].

Two of these features, when combined, lay the foundation for a standardized time-triggered communication paradigm guaranteeing strict real-time communication and, at the same time, introducing stream isolation mechanisms enabling compositional system designs [8]. Namely,

- IEEE 802.1ASrev [3] defines a time-synchronization protocol implementing a global clock reference with basic fault-tolerance mechanisms.
- IEEE 802.1Qbv [7] specifies the *time-aware shaper* functionality implementing the time-triggered paradigm [9] at the egress ports of communicating nodes.

The *time-aware shaper* defined in IEEE 802.1Qbv [7] is essentially a gate mechanism dynamically enabling or disabling the selection of frames from egress queues based on a predefined cyclic schedule referred to as the *Gate Control List* (GCL). More precisely, 802.1Qbv defines one *timed-gate* on the egress side for each priority queue in a port, which at a given time can be in one of two defined states: *open* or *close*. When the gate is in the *open* state, frames may be selected from the respective queue for transmission to the physical link in first-in first-out (FIFO) order. If the gate is in the *close* state, frames from the respective queue are not selected. A priority-based arbitration or *credit-based shaper* is then applied among all opened queues.

State changes are calculated offline and configured with their predefined activation time via entries in the GCL. Later, in a deployed network, each egress port is timely configured at run-time following its own GCL, executed synchronously based on the global notion of time.

Figure 1 depicts a simplified logical representation of an 802.1Qbv-capable switch. In the example, streams coming from ports A and B (ingress) are routed to port C (egress). Internally, a switching fabric determines to which output port a frame is to be routed and assigns it to a queue (or traffic class) in the respective egress port. This assignment is based on criteria like the *priority code point* of the IEEE 802.1Q header or the *priority table* of the IEEE 802.1Qci *per stream filtering* functionality. All ports in an 802.1Qbv-enabled switch will have an equivalent logical composition as the one depicted, including a number (typically 8) of logical egress queues. In practice, a subset of the queues may be reserved at design-time for scheduled streams and the remainder used to isolate non-scheduled traffic [10].

We trace an equivalence between the problem of scheduling *timed-gates* and scheduling gate *windows* by noting that encoded in the GCL entries is an ordered list of transmission windows on the time domain, i.e., intervals in which a

978-1-5386-5295-4/18/$31.00 ©2018 IEEE
DOI 10.1109/RTAS.2018.00008

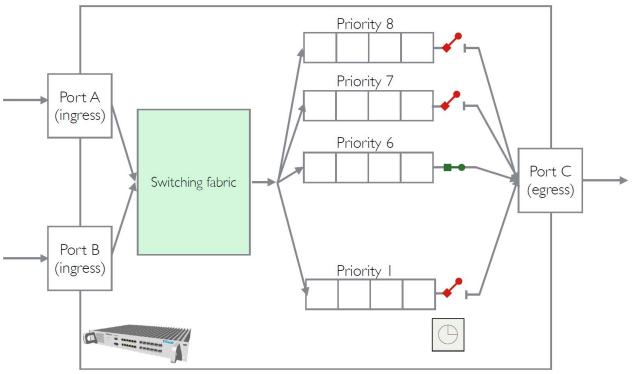


Figure 1. Simplified view of an 802.1Qbv-capable switch.

gate remains in the *open* state. Each window is in itself defined by a left boundary and a right boundary marking the time instants where the gate state transitions to *open* and, respectively, *close*.

The scheduling problem is approached via the formulation of constraints to compute window intervals which are directly mapped to entries in the GCL. These constraints express dependencies between variables denoting among others the *open* and *close* instants of time for scheduled windows as well as the *frame-to-queue* and *frame-to-window* assignments.

To the best of our knowledge, our work is the first to directly address the scheduling problem of windows. In [10] a transmission offset is computed for each individual frame in the network deriving the events encoded in the GCL in a post-processing step. While this is in principle a valid approach it introduces several limitations. Namely, that the schedule of individual frames leads in practice to a large number of events as they are freely placed in the timeline. Consequently, the post-processing step may easily exceed the length of a GCL, expected to range from 8 to 1024 entries for typical nodes implementing IEEE 802.1Qbv. In addition, the time granularity used by the scheduler (so-called *macrotick*) generates unnecessary gaps between scheduled frames, even when their offsets are computed to be one after the other, causing a detriment in the available bandwidth for non-scheduled traffic. Lastly, the strictly-periodic communication model may significantly reduce the solution space due to the global enforcement of 0-jitter transmissions.

The goal of this work is particularly concentrated on the suitability analysis of first-order theory of arrays for the formal specification of scheduling constraints allowing a direct encoding via the GCL of IEEE 802.1Qbv-capable nodes. The formalization theory applied in the specification of a system of constraints may enable the use of dedicated tools like specialized solvers to search for a satisfiable solution. In this paper, we particularly address the appropriateness of *first-order theory of arrays* as a suitable means of specifying the set of constraints for the gate operations (*open* and *close*) incorporating the size of the GCL as an input bound to the scheduler, hence showing how the satisfiability of the system can be solved with the use of general-purpose SMT solvers.

The first-order theory of arrays ($\mathcal{T}_A$) is built around two operations: *select*, used to return an element of an array from a certain index, and *store*, used to write an element into an array at a certain index. In addition to the usual operators from linear integer arithmetic, we use the syntax presented in [11] to introduce array theory and express the scheduler constraints in the following sections. Usually, the signature of $\mathcal{T}_A$ is defined as $\sum_A : \{\cdot[\cdot], \cdot\langle\cdot \leftarrow \cdot\rangle, =\}$. In [11], the sorts $array$, $elem$, and $index$ are used for arrays, elements, and indices, respectively. Furthermore, the syntax $a[i]$ is used for the select function of the element at index i from array a and $a\langle i \leftarrow e\rangle$ is used for the store operation of element e in array a at index i. The two main axioms of array theory are [12], [11]:

$$\forall a : array, \forall i, j : index, \forall x : elem$$
$$i = j \rightarrow a\langle i \leftarrow x\rangle[j] = x$$
$$i \neq j \rightarrow a\langle i \leftarrow x\rangle[j] = a[j]$$

Together with axioms of linear integer arithmetic these form the theory of integer-indexed arrays ($\mathcal{T}_A^{\{\mathbb{Z}\}}$). Although $\mathcal{T}_A^{\{\mathbb{Z}\}}$ has been shown to be undecidable, the quantifier-free fragment, which we use in this paper, is decidable (NP-complete) [11].

The resulting system of constraints builds on top of a relaxed timing model allowing communication with bounded jitter, yet guaranteeing determinism. We define this model based on reasonable assumptions over the distributed time synchronization capabilities. In particular, we assume that a global notion of time, provided by a protocol stack like e.g. IEEE 802.1AS-rev, allows nodes to synchronize their transmission events within a known bounded precision (δ), denoting the maximum difference between any two synchronized clocks in the network at any instant of time.

It is known from previous work (e.g. [13], [14], [15]) that approaches of similar complexity can be implemented by incremental scheduling algorithms yielding a significant decrease in the required runtime as well as improving scalability in comparison to the solutions based on direct SMT encoding. However, we explicitly opt to analyze the problem without the consideration of advanced scheduling algorithms and let the scalability concerns be addressed at a future time.

In section II we introduce the network and traffic model and formulate the scheduling constraints for IEEE 802.1Qbv using first-order theory of arrays ($\mathcal{T}_A$) (section III). In section IV we elaborate an SMT-based synthesis algorithm implementing the previously defined constraints together with a discussion and formalization of several optimization

opportunities arising from our model. We evaluate the scalability of our approach in section V and survey related work in section VI. We conclude the paper in section VII.

II. System Model

We model a network as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is a set of vertices, $\mathcal{E}$ is a set of directed edges connecting vertices to each other. If there exists a physical link between v_i and v_j then $(v_i, v_j), (v_j, v_i) \in \mathcal{E}$, where the first vertex in the pair description defines the source node and the second vertex defines the destination node. Nodes may be the source or destination of messages (end systems) or may forward messages to other nodes (switches).

Communication requirements are modeled through the concept of streams. A stream (or flow) is a periodic multicast data transmission (the message) from one talker (the sender node) to one or multiple listeners (the receiver nodes). Without loss of generality, we reduce the number of receivers to one (unicast) and the message size to one Ethernet frame in order to simplify the formalism, noting that extending the model to the general case is a trivial step [15]. We denote the set of streams in the network with $\mathcal{S}$. Similar to [13], we denote the route of a stream $s_i \in \mathcal{S}$ from talker v_1 to listener v_n routed through the intermediary nodes (i.e. switches) $v_2, v_3 \ldots, v_{n-1}$ as $\mathcal{R}_i = [(v_1, v_2), \ldots, (v_{n-1}, v_n)]$.

We assume that the routing of a stream from source v_1 to destination v_n is computed beforehand and therefore known.

A stream $s_i \in \mathcal{S}$ is defined by the tuple $\langle C_i, T_i, L_i, J_i \rangle$, denoting the message size in bytes, the period, the maximum allowed end-to-end latency, and the maximum allowed jitter of the stream, respectively.

The instance of a stream $s_i \in \mathcal{S}$ routed through link $(a, b) \in \mathcal{E}$ is defined by the frames $f_{i,j}^{(a,b)} \in \mathcal{F}_i^{(a,b)}$, where $\mathcal{F}_i^{(a,b)} \subset \mathcal{F}^{(a,b)}$ is the set of all frames of stream s_i that are to be scheduled on link (a, b). We denote the set of all frames routed through link (a, b) with $\mathcal{F}^{(a,b)}$. Since streams may have different periods resulting in an overall schedule cycle (hyperperiod) larger than any individual stream period, when constructing the GCL we must consider all instances of a specific stream repeating until the schedule cycle. Hence, a set $\mathcal{F}_i^{(a,b)}$ will have T_s/T_i frames, where T_s is the schedule cycle of all scheduled streams in the network, calculated as the *least common multiple* of the periods of all streams $s_i \in \mathcal{S}$. Additionally, each such periodic frame is characterized by a frame transmission duration $l_i^{(a,b)}$ calculated based on the data size C_i of the stream s_i and the speed of the egress port associated to the physical link (a, b). For example, a maximum- sized Ethernet frame of 1542 bytes (including the IEEE 802.1Q tag) has a duration of $12.336 \mu sec$ on a 1Gbit/sec link.

We denote the maximum number of scheduled windows per edge derived from the maximum length of the GCL with $\mathcal{W}^{(a,b)}$. In order to encode the scheduling problem in $\mathcal{T}_A^{\{\mathbb{Z}\}}$, we define for each link (a, b) two arrays, $\phi^{(a,b)}$ and $\tau^{(a,b)}$ over the sort $array$, containing the integer variables for, respectively, the open and close time instants of the windows indexed by the position in both arrays for the egress port associated to link (a, b). Furthermore, we define for each frame instance $f_{i,j}^{(a,b)} \in \mathcal{F}_i^{(a,b)}$ a window index $\omega_{i,j}^{(a,b)}$ over the sort $index$ representing the frame-to-window assignment index in both aforementioned arrays.

We adopt the definition in [10], where the queue configuration is expressed as a tuple $G(Q) = \langle \aleph, \aleph_{tt}, \aleph_{prio} \rangle$, where $\aleph$ is the total number of queues per egress port, of which $\aleph_{tt}$ is the number of queues for scheduled traffic and $\aleph_{prio}$ the remaining number of priority queues for non-scheduled traffic. In order to connect windows to egress queues, we define an additional array $\kappa^{(a,b)}$ over the sort $index$ denoting the assigned queue for each window.

III. Formal Scheduling Constraints

We formalize the constraints for gate operations (i.e. open, close) as well as for the frame-to-window and window-to-queue assignment variables such that the resulting gate control list correctly drives the deterministic time behaviour of frames.

A. Technology Constraints

Technology constraints are those derived from the functional specification of 802.1Qbv. They define a system of constraints that shall be fully satisfied in a feasible solution.

Well-defined Windows Constraints: We first formalize logical constraints for all windows of an egress port. Since each physical link connects one egress port to one ingress port, we assume an equivalence in the formalism between an egress port and the connected directed edge (physical link) for the remainder of the paper.

We constrain the open and close events of each window defined on that link to be greater than or equal to 0 and less than or equal to the schedule cycle of all streams in the network. Hence, we have the constraint:

$$\begin{aligned} &\forall (a,b) \in \mathcal{E} : \forall k \in \{1, \ldots, \mathcal{W}^{(a,b)}\} : \\ &\big(\phi^{(a,b)}[k] \geq 0\big) \wedge \big(\tau^{(a,b)}[k] < T_s\big), \end{aligned} \tag{1}$$

where, as defined above, T_s is the schedule cycle (hyperperiod) of all communication streams in the network.

Additionally, each window is assigned to an egress queue scheduled in the range $[0..\aleph_{tt} - 1]$, therefore we add the bounds for the queue assignment array

$$\forall (a,b) \in \mathcal{E} : \forall k \in \{1, \ldots, \mathcal{W}^{(a,b)}\} : 0 \leq \kappa^{(a,b)}[k] < \aleph_{tt}.$$

Stream Instance Constraints: Communication in system deployments rarely appear with a normalized period. Instead, streams are sourced at multiple rates which result in a hyperperiod defining the length of the schedule tables to be at least the *least common multiple* of all periods involved.

The assignment of frames, and as a consequence the length of each window, is a result of the scheduler. Streams routed through the same link having different periods will contribute a number of frame instances, each to be scheduled within each period instance occurring until the hyperperiod. As a result, the open and close bounds for each window is further constrained to set the window of a each frame instance within the corresponding period instance. Hence, for each stream s_i routed through (a, b) we construct the following constraint:

$$\forall s_i \in \mathcal{S} : \forall (a,b) \in \mathcal{E} : \forall j \in \left[0, \frac{T_s}{T_i} - 1\right] : \tag{2}$$
$$\big(\phi^{(a,b)}[\omega_{i,j}^{(a,b)}] \geq j \times T_i\big) \wedge$$
$$\big(\tau^{(a,b)}[\omega_{i,j}^{(a,b)}] < (j+1) \times T_i\big),$$

which for each frame instance scheduled at different period instances until the hyperperiod sets a lower bound for the window open event and upper bound to the window close event to, respectively, the beginning and the end of the respective period instance.

While we allow for greater flexibility resulting in an increased solution space than in previous approaches [10], we note that our model can also handle strictly periodic systems, i.e., systems in which frames belonging to the same stream have to arrive at exactly the same time in each period instance. For that, we define an optional constraint for the opening time of the assigned windows of each frame instance to be exactly one period apart and allow only one frame to be assigned to those windows, resulting in 0 jitter for those streams:

$$\forall s_i \in \mathcal{S} : \forall (a,b) \in \mathcal{E} : \forall j \in \left[0, \frac{T_s}{T_i} - 2\right] : \tag{3}$$
$$\big(\phi^{(a,b)}[\omega_{i,j+1}^{(a,b)}] - \phi^{(a,b)}[\omega_{i,j}^{(a,b)}] = T_i\big).$$
$$\forall s_i \in \mathcal{S} : \forall (a,b) \in \mathcal{E} : \forall j \in \left[0, \frac{T_s}{T_i} - 1\right] :$$
$$\big(\tau^{(a,b)}[\omega_{i,j}^{(a,b)}] - \phi^{(a,b)}[\omega_{i,j}^{(a,b)}] = l_i^{(a,b)}\big).$$

Ordered Windows Constraint: An essential constraint for determinism is that no two frames transmitted on the same egress link overlap in the time domain. Moreover, we explicitly forbid multiple windows to remain open at the same time in order to avoid the non-determinism introduced by contention. Note that guaranteeing determinism while allowing multiple opened windows simultaneously would require a complex step in addition to scheduling to analyze the resulting worst-case interference of different traffic classes. Methods like network calculus (e.g. [16]) allow such analysis at the cost of reducing the compositionality property provided by offline scheduling.

Conceptually, the formulation of this constraint does not allow any two windows on the same link to overlap, similar to [10], which we define as

$$\forall (a,b) \in \mathcal{E} : \forall i,j \in \{1, \ldots, \mathcal{W}^{(a,b)}\}, i \neq j : \tag{4}$$
$$\big(\tau^{(a,b)}[i] \leq \phi^{(a,b)}[j]\big) \vee \big(\tau^{(a,b)}[j] \leq \phi^{(a,b)}[i]\big)$$

Note that this formulation results in a large number of assertions with a disjunction operator, which has proved to be computationally intensive. However, since the assignment of frames to windows is not restricted beforehand and any frame may be assigned to any window, we can simplify this constraint if the order of windows on each link is predefined offline, hence fixing their respective open and close events to be sequential. We prefer the following alternative formulation performing significantly better in terms of resources:

$$\forall (a,b) \in \mathcal{E} : \forall i \in \{1, \ldots, \mathcal{W}^{(a,b)} - 1\} : \tag{5}$$
$$\tau^{(a,b)}[i] \leq \phi^{(a,b)}[i+1],$$

which enforces an ordered sequence for the open event of each window to be after the close event of its predecessor.

In addition, the latter formulation allows reducing the number of assertions in (1) by reducing the required bounds to the open event of the first window and, respectively, the close event of the last window of each link. All other events (open/close) will be bounded due to the imposed sequential order. Hence, we reduce constraint (1) to the following:

$$\forall (a,b) \in \mathcal{E} : \tag{6}$$
$$\big(\phi^{(a,b)}[1] \geq 0\big) \wedge \big(\tau^{(a,b)}[\mathcal{W}^{(a,b)}] < T_s\big).$$

Frame-to-Window Assignment Constraint: The frame-to-window assignment variable defines the index in the 3 arrays (open, close, and queue assignment) of the respective port. Thus, we restrict the variables to be no larger than the configurable maximum number of windows per port ($\mathcal{W}^{(a,b)}$):

$$\forall (a,b) \in \mathcal{E} : \forall f_{i,j}^{(a,b)} \in \mathcal{F}^{(a,b)} : \tag{7}$$
$$\big(\omega_{i,j}^{(a,b)} \geq 1\big) \wedge \big(\omega_{i,j}^{(a,b)} \leq \mathcal{W}^{(a,b)}\big).$$

Window Size Constraints: Since frames are assigned to windows by the scheduler, i.e., they are not known a-priori, the size of a window results from the accrued sum of the duration of all frames assigned to it. Hence, we must ensure that the close event of each window allows sufficient time to transmit the set of assigned frames. We note that this constraint is the first and only one in our formalism requiring the *store* operation of array theory.

We start by storing the uninterpreted term for each open variable in the respective position of the close array:

$$\forall (a,b) \in \mathcal{E} : \forall k \in \{1, \ldots, \mathcal{W}^{(a,b)}\} : \tag{8}$$
$$\tau^{(a,b)}\langle k \leftarrow \phi^{(a,b)}[k]\rangle.$$

This is equivalent to setting all close events equal to the open event at the same index, initializing the length of the window

to 0. Note that, the close event of all windows without any frame assigned will remain equal to the respective open event.

We now construct at each position in the close array the sum over the duration of all frames assigned to that window, using the frame-to-window assignment index:

$$\forall(a,b) \in \mathcal{E} : \forall f_{i,j}^{(a,b)} \in \mathcal{F}^{(a,b)} : \\ \tau^{(a,b)}\langle \omega_{i,j}^{(a,b)} \leftarrow \tau^{(a,b)}[\omega_{i,j}^{(a,b)}] + l_i^{(a,b)} \rangle. \tag{9}$$

The construct iterates for all frames, adding the frame duration to the previous uninterpreted value for the close event of the window to which the frame is assigned and storing the result at the same index as a new uninterpreted expression.

Stream Constraint: The stream constraints describe the sequential nature of communication from talkers to listeners. The generic condition is that frames belonging to the same stream must be scheduled sequentially with respect to time along the routed communication path. Hence, we have

$$\forall s_i \in \mathcal{S} : \forall (v_k, v_{k+1}) \in \mathcal{R}_i, k \in \{1, \dots, n-2\} : \\ \forall f_{i,j}^{(v_k,v_{k+1})} \in \mathcal{F}_i^{(v_k,v_{k+1})} : \forall f_{i,j}^{(v_{k+1},v_{k+2})} \in \mathcal{F}_i^{(v_{k+1},v_{k+2})} : \\ \tau^{(v_k,v_{k+1})}[\omega_{i,j}^{(v_k,v_{k+1})}] + \delta \leq \phi^{(v_{k+1},v_{k+2})}[\omega_{i,j}^{(v_{k+1},v_{k+2})}], \tag{10}$$

where δ is the constant value representing the *precision*.

In other words, the propagation of frames of a stream follow the sequential order along the path. Therefore, the window open event of each frame has to be greater than or equal to the close event of the window assigned to the predecessor frame, plus the network precision constant to compensate for clock differences between the two hops.

Stream Isolation Constraint: We only briefly present here the isolation problem in TSN networks and refer the reader to [10] for a more in-depth description of the general problem. IEEE 802.1Qbv [7] specification controls the opening and closing of the timed gates and not the sending and receiving of individual frames, like TTEthernet. It is plausible that at runtime, a network may experience frame losses or streams showing differences in their periodic payload size. Therefore, to ensure that the execution of the schedule during runtime conforms to the offline planning we need to compute a schedule providing guarantees on the deterministic state of each queue at any given instant of time.

Consider the case in which two streams s_i and s_j are received from different links, (x, a) and (y, a), respectively, on device a and both forwarded to the same egress port on link (a, b). If their frames are put in the same queue, the order of frames in that queue may differ during runtime depending on minimal variations on the exact order of arrival, or the processing mechanism for ingress ports in the switch fabric. Moreover, frame losses in one or the other stream may equally introduce differences in the queue state at each period instance. Hence, the offline scheduled opening and closing of the egress queue may effectively cause a different behavior at runtime induced by the non-deterministic state of the queue.

Guaranteeing determinism implies that all frames respect their computed window assignment throughout the lifetime of the system. Hence, we must either isolate them in the time domain, similar to [10], restricting that a stream is not received until the other stream has already been scheduled for egress, or assign the respective frames of the two streams to the same scheduled window, which was not possible in [10]. Alternatively, if multiple queues are available for scheduled traffic we can isolate the two frames in windows of different queues, in which case they may as well be received within overlapping intervals without altering the run-time behavior.

Hence, we formulate the stream isolation condition for streams s_i and s_j sent on link (a, b) as:

$$\forall k \in \left[0, \frac{T_s}{T_i} - 1\right] : \forall l \in \left[0, \frac{T_s}{T_j} - 1\right] : \\ \Big(\big(\tau^{(a,b)}[\omega_{i,k}^{(a,b)}] + \delta \leq \phi^{(y,a)}[\omega_{j,l}^{(y,a)}]\big) \vee \\ \big(\tau^{(a,b)}[\omega_{j,l}^{(a,b)}] + \delta \leq \phi^{(x,a)}[\omega_{i,k}^{(x,a)}]\big)\Big) \vee \\ \Big(\kappa^{(a,b)}[\omega_{i,k}^{(a,b)}] \neq \kappa^{(a,b)}[\omega_{j,l}^{(a,b)}]\Big) \vee \Big(\omega_{i,k}^{(a,b)} = \omega_{j,l}^{(a,b)}\Big), \tag{11}$$

where the three disjunctive conditions guarantee that either one of the two frames is received when the other one has already been forwarded (by comparing the sequence of the respective open and close events of the windows assigned to each frame), or each is assigned to a different queue (and hence to a different window), or both frames are assigned to the same window (and hence to the same queue).

B. User Constraints

User constraints are those denoting additional requirements on a particular property of the solution. They extend the system of constraints and shall be equally satisfied in a feasible solution but their exclusion does not imply an invalid schedule from the technology point of view. Following, we formalize two of the most industry relevant user constraints.

Stream End-to-End Latency Constraint: The end-to-end latency constraint states that the difference between the receiving instant of a stream on the listener side and the sending of the stream from the respective talker has to be smaller than or equal to a given maximum end-to-end latency.

As before, we construct the formula using the frame assignment variables in combination with the open and close events given that the frame-to-window assignment is not known a-priori. Therefore, we define the end-to-end latency

constraint for stream s_i as:

$$\forall j \in \left\{0, \cdots, \frac{T_s}{T_i} - 1\right\}: \quad (12)$$
$$\forall f_{i,j}^{(v_1,v_2)} \in \mathcal{F}_i^{(v_1,v_2)}, f_{i,j}^{(v_{n-1},v_n)} \in \mathcal{F}_i^{(v_{n-1},v_n)}:$$
$$\tau^{(v_{n-1},v_n)}[\omega_{i,j}^{(v_{n-1},v_n)}] - \phi^{(v_1,v_2)}[\omega_{i,j}^{(v_1,v_2)}] \leq L_i - \delta.$$

Note that we also include the precision δ in the constraint to compensate the possible synchronization error between the two nodes.

Stream Jitter Constraints: Real-time communication are typically sensitive to jitter introduced between the relative transmission or arrival times of the periodic frames of a stream. We base our jitter constraint on the observation that within the network, the jitter of individual frames of a stream is not relevant except for the sending and receiving nodes. The jitter becomes relevant on the sending side since the data may be produced by a periodic task requiring the transmission of that data with a bounded jitter. Similarly, on the receiver side the data has to be processed by a listener task which also may have requirements on the jitter for example in the case of control tasks [17]. Hence, we constrain the jitter of a stream for the sender and receiver nodes to be within a configurable maximum bound. Note, however, that for receiving the jitter constrain is applied to the scheduled window on the last hop before the listener.

In addition, we must also consider the relaxed periodicity on different period instances, i.e., the jitter has to be maintained between frame of any period instance of the stream until the hyperperiod.

We define the sender jitter constraint for stream s_i as:

$$\forall j, k \in \left\{0, \cdots, \frac{T_s}{T_i} - 1\right\}: \quad (13)$$
$$\forall f_{i,j}^{(v_1,v_2)}, f_{i,k}^{(v_1,v_2)} \in \mathcal{F}_i^{(v_1,v_2)}:$$
$$\left(\tau^{(v_1,v_2)}[\omega_{i,j}^{(v_1,v_2)}] - j \times T_i\right) -$$
$$\left(\phi^{(v_1,v_2)}[\omega_{i,k}^{(v_1,v_2)}] - k \times T_i\right) - l_i^{(v_1,v_2)} \leq J_i.$$

The constraint takes the relative offset of each variable being compared (open/close) with respect of the period instance to which it is bounded (see (2)). It then enforces that the difference between the latest transmission of a frame (i.e. the relative close instant of the window minus the frame duration) and the earliest possible transmission (i.e. the relative open instant of the window) of any two frame instances of the same stream is at most equal to the jitter. This also guarantees that the window duration for any stream is at most equal to the defined jitter plus the frame duration since, otherwise, the non-deterministic order of transmission within a window could already introduce a larger jitter than required. Moreover, it extends the condition to all frame instances of the same stream regardless of the period instance in which they are scheduled.

Similarly, we define the jitter constraint on the receiving side for stream s_i as:

$$\forall j, k \in \left\{0, \cdots, \frac{T_s}{T_i} - 1\right\}: \quad (14)$$
$$\forall f_{i,j}^{(v_{n-1},v_n)}, f_{i,k}^{(v_{n-1},v_n)} \in \mathcal{F}_i^{(v_{n-1},v_n)}:$$
$$\left(\tau^{(v_{n-1},v_n)}[\omega_{i,j}^{(v_{n-1},v_n)}] - j \times T_i\right) -$$
$$\left(\phi^{(v_{n-1},v_n)}[\omega_{i,k}^{(v_{n-1},v_n)}] - k \times T_i\right) - l_i^{(v_{n-1},v_n)} \leq J_i$$

Note that the formulation of (14) and (13) can be simplified in configurations with a single communication period where the only open and close events would refer to the same window.

If the jitter perception is globally relevant within the entire network, i.e. between intermediate nodes along the route of the stream, the constraint can be readily applied for all of those nodes.

IV. SMT/OMT-Based Schedule Synthesis

So far we have formalized a system of constraints denoting the scheduling requirements within the flexibility of the defined model. We chose to compute a schedule based on these constraints with the aid of *Satisfiability Modulo Theories* (SMT).

SMT solvers are used to determine the *satisfiability* or *unsatisfiability* of first-order logical formulas with respect to a certain background theory or a combination of background theories [18], [19]. A background theory may be for e.g. linear integer arithmetic ($\mathcal{LA}(\mathbb{Z})$), bit-vectors ($\mathcal{BV}$), or, as required in our case the theory of arrays ($\mathcal{T}_A$). If the set of constraints is satisfiable with respect to the defined theory, SMT solvers also provide a *model* which represents one solution for the given variables and constraints. Furthermore, a new branch called *Optimization Modulo Theories* (OMT) [20], [21] can provide optimal solutions with respect to given minimization or maximization objectives. Solving NP-complete scheduling problems with combinatorial characteristic defined through linear arithmetic constraints, like the one addressed in this paper, has proven to be suitable use-cases for SMT/OMT solvers, especially in the case of small and medium networks [22], [23]. In this paper we focus primarily on the suitability of array theory for encoding and solving the scheduling problem, leaving scalability improvements for exploration in future work. Therefore we do not claim the contribution of a new scheduling algorithm, and particularly, it is not the aim of this evaluation to include improvements to the constraint solver like, for example, the incremental scheduler in [13], which have been found beneficial to reduce the synthesis time for the average case. Our primary goal is, therefore, to evaluate the suitability of our approach, as well as to explore the trade-offs and optimization opportunities emerging from our model.

The aim of our scheduling algorithm for IEEE 802.1Qbv is to find solutions for the window open and close arrays on

each port as well as for the frame-to-window and window-to-queue index variables such that the constraints defined in section III are fulfilled. As background theory we use *quantifier-free integer-indexed arrays* ($\mathcal{T}_A^{\{\mathbb{Z}\}}$) over integer elements (QF_ALIA).

A. Optimization

Recently, SMT solvers like z3 [21] also offer the possibility to express optimization objectives binding the solver to provide an optimized solution with respect to given (single or multiple) minimization or maximization objectives.

Optimization objectives differ based on particular system requirement and deployment characteristics. Whereas a number of relevant objectives has been already discussed in [15], we prefer to concentrate on optimization opportunities arising from the characteristics of our model which were not possible in prior work. In particular, we address the trade-off between the defined number of windows per egress port and the resulting maximum jitter experienced by a stream. We focus here on a particular optimization objective that minimizes the receiving jitter for streams. This can be either expressed as minimizing the accrued jitter over all streams in the network or as a collection of objectives minimizing the individual jitter of each stream, which may result in local minima for some of the streams.

The jitter value for a stream s_i is defined in our model as J_i, denoting the maximum allowed jitter for the stream at the scheduled times of sending and receiving. Thus, we introduce an additional variable $j_i \leq J_i, \forall s_i \in \mathcal{S}$ replacing J_i in conditions (13) and (14). The optimization objective is then either defined as minimizing each j_i (using either *lexicographic* or *Pareto fronts combinations* [21]) or the sum over all j_i. Moreover, the solver allows introducing a weight j_i^w for each stream s_i representing the relative importance of minimizing the jitter of that particular stream. The optimization criteria is expressed as

$$minimize \sum_{s_i \in \mathcal{S}} j_i^w \times j_i,$$

subject to the constraints defined in section III.

Note that for strictly periodic streams or single period configurations we can directly express the minimization objective as the difference between the closing and opening times of the respective windows eliminating the need for an additionally variable per stream.

V. Evaluation

We have implemented a prototype tool for synthesizing schedules based on the system model and constraint definitions presented in the previous sections. We used z3 [24] v.4.5.0 as the underlying SMT/OMT solver with QF_ALIA as the background theory for the version without optimization and the built-in optimization solver of z3 for the experiments requiring optimization objectives. All experiments were run on an Intel(R) Core(TM) i7-2600 64bit CPU @ 3.40GHz with 12 GB of RAM. For convenience we have chosen a timeline granularity of $1\mu sec$ and a uniform communication speed for all physical links of 1Gbit/s. The message size for all experiments is fixed as one maximum-size Ethernet frame, resulting in a frame duration of $13\mu sec$ with the chosen time granularity.

A. Topology

As a baseline configuration for the evaluation we have chosen a simple line topology for switches with the same number of end systems connected to each of them. Only end systems act as senders (talkers) and receivers (listeners) of communication streams. An example is depicted in Figure 3. Note that the simple example includes three streams already illustrating scenarios which are relevant to the analysis. Among others, streams converging from different sources into one egress port (e.g. streams A and B at the first hop), incoming streams from the same ingress port diverging to different egress ports (e.g. streams A and B at the second hop), as well as other potential sources for jitter like unbalanced network load, cross traffic (e.g. stream C), etc.

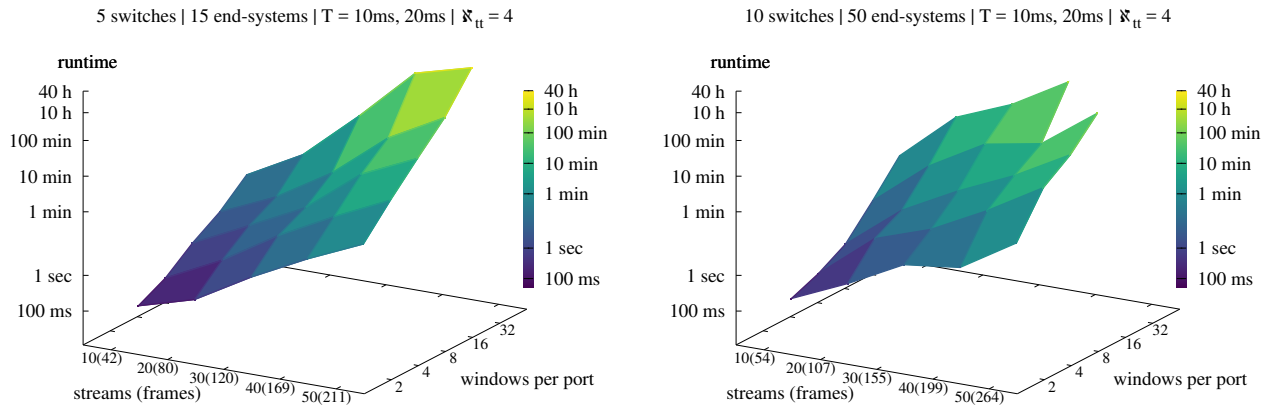
We denote the configuration setting for each experiment in terms of the number of switches (SW), the total number of end systems (ES) equally distributed along the switches, the set of periods for the streams (T), and the number of queues for each egress port ($\aleph_{tt}$). Streams are routed through the topology based on a random selection of a talker and a listener end systems. The network size as well as the chosen configurations are inspired by industrial use-cases.

B. Synthesis Time

Figure 2 plots the runtime of the scheduling synthesis when varying the number of streams and the maximum number of windows ($\mathcal{W}$) per egress port $(2, 4, 8, 16, 32)$. The periods of the streams are $(10, 20)ms$ resulting in a hyperperiod of $20ms$. The queue configuration is $\aleph_{tt} = 4$. On the x-axis we also indicate the total number of frame instances that are scheduled in the network as a result of a the streams being routed. The timeout for the solver was set to 40 hours. Note that both the y-axis (showing the number of windows per port) and z-axis (representing the synthesis time of the scheduler) are logarithmic.

Equivalent experiments were run with two alternative network sizes, shown in Figures 2(a) and 2(b), namely, a small-sized network consisting of 15 end-systems distributed through 5 switches (3 end-systems per switch), and a medium-sized network consisting of 50 end-systems distributed in a network of 10 switches (5 end-systems per switch).

The synthesis method shows to scale well for a small number of windows per egress port, even up to 50 streams (resulting in 211 and 264 frame instances) being scheduled



(a) 5 switches, 15 end-systems

(b) 10 switches, 50 end-systems

Figure 2. Synthesis time when varying the number of streams (frames) and maximum number of windows per egress port.

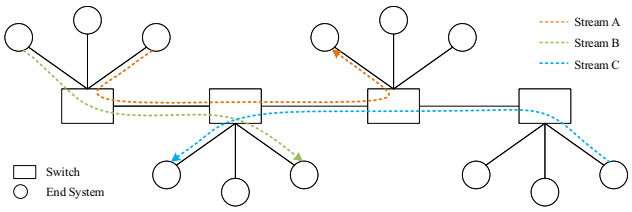


Figure 3. Example topology depicting four switches connected in a line, each switch with three end systems connected, and three streams (A, B, and C).

in small- and medium-scale networks. However, when increasing the number of windows, the synthesis time also increases from a few minutes to several hours, eventually timing out at 40 hours when scheduling 50 streams with 32 windows per port in the medium-sized topology.

While all cases were scheduled in under 1 minute with 2 or 4 windows per port, we note that the problem is NP-complete [10] resulting in an exponential runtime complexity with increasing input size. Several dimensions of the input, like period choice, macrotick, topology size, number of streams, etc., affect the runtime and have been studied for e.g. in [15] and [10]. We observe, for instance, that compared to related work (c.f. [10]) the number of windows per port has a greater impact on the synthesis time than for example the number of scheduled streams. Therefore we identify this variable as a metric particular to our solution.

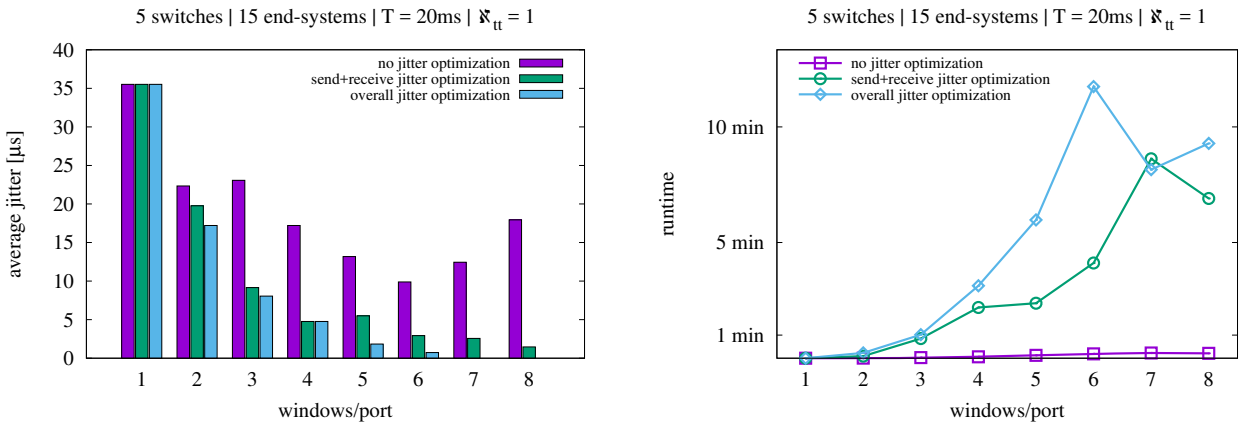
C. Jitter vs Window Trade-Off

We have identified a fundamental trade-off exposed by a low bound on the number of windows. Indeed, the synthesis time is directly affected by the window count, however we observe in Figures 4(a) and 4(b), discussed below, what intuitively seems plausible: the number of windows has an impact on the effective jitter experienced by the communication streams.

Constraining the jitter is straightforward (13) and (14), however, in this experiment we want to show the effect of the number of windows per port on the jitter (in contrast to synthesis time) when jitter minimization objectives are introduced. We chose a reference topology with 5 switches and 15 end-systems in which 25 streams are transmitted between random talkers and listeners, all streams with a period of 20 ms. The queue configuration is set to $\aleph_{tt} = 1$ to increase the confluence of frames on the same windows. We conduct the schedule synthesis with three alternative implementations of the algorithm. Namely, without optimization objective, with multiple optimization objectives minimizing the jitter for the sender and receiver of each stream, and, lastly, with multiple optimization objectives minimizing the jitter of all scheduled frames. The prioritization of the multiple optimization objectives is done using lexicographical ordering (c.f. [21]).

Figure 4(a) shows the average jitter when varying the number of windows per port from 1 to 8 and Figure 4(b) shows the synthesis time for the same configurations. The x-axis in both figures describes the number of windows per port and the y-axis shows the average jitter in the network and the synthesis time, respectively.

As expected, the jitter decreases when increasing the number of windows per port even in cases when no optimization is performed. However, optimizing the jitter either for senders/receivers or for all frames in the network reduces the average jitter, down to the minimum possible of 0 for cases with 7 and 8 windows per port. The trade-off becomes obvious when observing the trend in the synthesis time for the configurations, which scales rapidly in proportion to the number of windows for the configurations with jitter optimization. An observation worth noticing is the decrease



(a) average jitter

(b) runtime

Figure 4. Average jitter and synthesis time with and without optimization constraints.

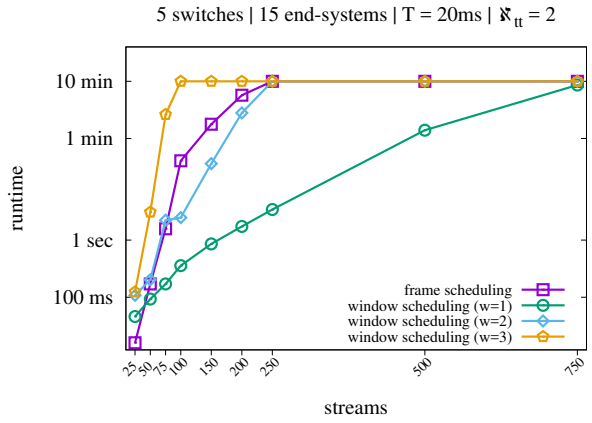


Figure 5. Scalability comparison between the frame-based and window-based synthesis algorithms.

in the runtime for the two experiments reaching 0 jitter. While this may not hold for all cases, we hypothesize that the effect is due to the inherent simplification of the constraint solving problem when each frame is assigned to an individual window. Even for the experiments optimizing only send and receive jitter we observe a similar decrease in the exponential trend, which could be equally explained by a partial assignment of frames to individual windows.

D. Frame-based vs Window-based Synthesis

We are interested in the scalability of this method in contrast to a frame-based approach. For this we replicated the synthesis constraints from [10] and compare the synthesis time of the frame-based approach against our window-based scheduler.

The reference configuration is based on 5 switches and 3 end-systems per switch, resulting in 15 end-systems in total. For the window-based method we select the number of windows from the set $\{1, 2, 3\}$ and for both implementations we set the queue configuration $\aleph_{tt} = 2$. A time-out aborts the synthesis if no schedule is found within 10 minutes. We try to schedule with each configuration 25, 50, 75, 100, 150, 200, 250, 500, and 750 streams within this time and evaluate the success of each algorithm. The period of all streams is set to 20 ms.

The results in Figure 5 show that the window-based method scales significantly better for the configuration with one window per port and even for two windows per port it reflects a slight improvement. However, with three windows per port it already shows a larger synthesis time, timing out at 10 minutes for 100 streams. Nevertheless, we argue that the relaxed jitter model and increased solution space provide a considerable benefit for the window-based scheduling method over the frame-based approach.

We demonstrate our claim of increased schedulability using a simple experiment in which streams with a small period of $150\mu s$ are scheduled in a topology of 5 switches and 10 end-systems with $\aleph_{tt} = 1$. With the frame-based method we can schedule up to 25 streams. When using the window-based method with 2 windows per port we can schedule 2 additional streams while with 3 windows per port an additional 7 streams are schedulable, leading to a significantly higher bandwidth utilization given the high rate of the streams ($T = 150\mu s$).

VI. Related Work

Traditionally, asynchronous networks are analyzed in terms of worst-case end-to-end communication latencies through methods like network calculus [25], [26]. One of the major drawbacks of this approaches is that the worst-case latency is analyzed based on the pre-assigned traffic

priorities and the arrival patterns of competing periodic and sporadic streams. This has the effect that compositional system design and temporal isolation of communication streams become very difficult. Time-triggered technologies on the other hand enable compositional system design as well as deterministic behavior and isolation of streams in the time domain.

Time-triggered scheduling has been initially formulated as a static cyclic task scheduling problem by Baker et al. [27]. Creating time-triggered communication schedules for deterministic networks (TTEthernet) using SMT solvers was first proposed in [13] and extended in [14], [15] to include the application layer on end-systems. Network scheduling problems for other proprietary technologies (e.g. PROFINET, FlexRay, TTP) have been studied in [28], [29], [30], [31].

Different aspects of TSN networks has been treated in existing literature. For example, in [32], Gutiérrez et al. analyze the synchronization quality of IEEE 802.1AS in large networks typically found in the industrial domain.

Our previous work [10] formally defined necessary constraints to compute deterministic schedules that could be mapped to TSN-compliant multi-hop switched networks providing jitter-free transmission and deterministic end-to-end latency guarantees for strictly-periodic scheduled frames. However, such stringent requirements on jitter and latency came at a high cost. On one hand, fully deterministic communication constraints restrict the solution space for valid schedules due to the isolation of streams in the time domain. On the other hand, the focus was given to finding exact timing for each transmitted frame, which was then mapped on a second step into a GCL reproducing the expected behavior. This made it difficult to optimize and tailor the output to device-specific properties, like the length of the GCL or the minimum distance between consecutive open and close gate events.

In [33], the authors introduce a new asynchronous traffic class to TSN and a mechanism for shaping that provides low delay guarantees. While the strictness of the new traffic class is lower than the time-triggered one, similar to our model, the paper does not consider the scheduling problem assuming that the timed-gates remain permanently in the open state.

Meyer et al. [34] analyze the interference effects of higher-priority time-triggered communication on AVB traffic converging in the credit-based shaper of TSN devices. Alderisi et al. [35] introduce a new traffic class, called Scheduled Traffic (ST), which has real-time guarantees and is strictly isolated from AVB streams via hardware mechanisms and not via the schedule of the timed-gates. Both papers assume that isolation of critical and non-critical streams is done via non-standard mechanisms whereas we enforce isolation and real-time behavior through the standard timed-gate schedule on the egress ports. Additionally, none of the papers address the underlying scheduling problem for Time-Sensitive Networks.

Heuristic approaches to schedule frames in TSN networks that are based on the constraints defined in [10] have been discussed in [8] and [36]. In [37], the authors define the TSN scheduling problem as a no-wait packet scheduling problem (NWPSP) and use a Tabu search algorithm for finding near-optimal solutions. Furthermore, the paper presents an optimization heuristic for reducing the number of gate open events by compressing the schedule such that multiple frames are transmitted in the same window. The reduction of the number of gate events is done after a schedule has been found, thus not having any guarantee that the resulting schedule will fit within the constraints of the hardware implementation. In our work, the number of gate events is an input and can be set to the respective hardware limit, thus ensuring that the resulting schedule can also be executed in hardware.

Array theory has been primarily used in software model checking and verification for sequential and concurrent programs [38], [39]. To the best of our knowledge, our work is the first to use array theory for encoding scheduling problems in distributed systems.

VII. Conclusion

We have presented a novel approach to synthesize the communication schedules for TSN based on the requirements defined in IEEE 802.1Qbv. Our model is based on the definition of constraints for gate windows defining the open and close instants for the timed gates of the egress ports. We have shown the suitability of the first-order theory of arrays ($\mathcal{T}_A$) to express these constraints and we have conducted a number of experiments to evaluate the performance as well as the underlying trade-offs. We have additionally compared our method to prior work and explored the scalability and schedulability dimensions of the proposed solution.

The results allow us to argue that our model and the formulation of constraints is suitable for the synthesis of schedules, even exhibiting potential to solve large networks with the aid of efficient scheduling algorithms. In particular, we conclude that our work is a valuable reference for the conception of synthesis tools for TSN with the application of incremental scheduling techniques like those presented in [13] and [14], where the incremental step is built around the number of windows, which we identify as a metric reflecting the most time-consuming factor in the system of constraints.

For even larger networks exceeding the reasonable size for offline configured systems, like some envisioned in the context of smart cities and the Internet of Things (IoT), a combination of heuristic algorithms and SMT-based methods implementing the presented constraints remain, in our opinion, the most promising approach.

ACKNOWLEDGMENT

This work has been funded by the European Commission through the FoF-RIA Project AUTOWARE: Wireless Autonomous, Reliable and Resilient Production Operation Architecture for Cognitive Manufacturing (No. 723909).

REFERENCES

[1] H. Kopetz and G. Grunsteidl, "TTP - A time-triggered protocol for fault-tolerant real-time systems," in *Proc. 23rd IEEE International Symposium on Fault-Tolerant Computing (FTCS-23)*, June 1993, pp. 524–533.

[2] Institute of Electrical and Electronics Engineers, Inc, "Time-Sensitive Networking Task Group," http://www.ieee802.org/1/pages/tsn.html, 2016, retrieved 06-Jul-2017.

[3] Institute of Electrical and Electronics Engineers, Inc, "802.1AS-Rev - Timing and Synchronization for Time-Sensitive Applications," http://www.ieee802.org/1/pages/802.1AS-rev.html, 2017.

[4] Institute of Electrical and Electronics Engineers, Inc, "802.1Qci - Per-Stream Filtering and Policing," http://www.ieee802.org/1/pages/802.1ci.htmll, 2017.

[5] Institute of Electrical and Electronics Engineers, Inc, "802.1Qbu - Frame Preemption," http://www.ieee802.org/1/pages/802.1bu.html, 2017.

[6] Institute of Electrical and Electronics Engineers, Inc, "802.1CB - Frame Replication and Elimination for Reliability," http://www.ieee802.org/1/pages/802.1cb.html, 2017.

[7] Institute of Electrical and Electronics Engineers, Inc, "802.1Qbv - Enhancements for Scheduled Traffic," http://www.ieee802.org/1/pages/802.1bv.html, 2016, draft 3.1.

[8] P. Pop, M. Lander Raagaard, S. S. Craciunas, and W. Steiner, "Design optimization of cyber-physical distributed systems using IEEE time-sensitive networks (TSN)," *IET Cyber-Physical Systems: Theory and Applications*, vol. 1, no. 1, pp. 86–94, 2016.

[9] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[10] 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.

[11] A. R. Bradley, Z. Manna, and H. B. Sipma, "What's Decidable About Arrays?" in *Proc. VMCAI*. Springer-Verlag, 2006.

[12] J. Mccarthy, "Towards a Mathematical Science of Computation," in *In IFIP Congress*. North-Holland, 1962, pp. 21–28.

[13] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Proc. RTSS*. IEEE, 2010.

[14] 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.

[15] 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.

[16] J. A. R. De Azua and M. Boyer, "Complete modelling of AVB in network calculus framework," in *Proc. RTNS*. ACM, 2014.

[17] A. Cervin, "Improved scheduling of control tasks," in *Proc. ECRTS*. IEEE, 1999.

[18] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*. IOS Press, 2009, vol. 185.

[19] R. Sebastiani, "Lazy Satisfiability Modulo Theories," *JSAT*, vol. 3, no. 3-4, pp. 141–224, 2007.

[20] R. Sebastiani and P. Trentin, "OptiMathSAT: A Tool for Optimization Modulo Theories." in *Proc. CAV*, ser. LNCS, vol. 9206. Springer, 2015.

[21] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νZ - An Optimizing SMT Solver," in *Proc. TACAS*. Springer-Verlag New York, Inc., 2015.

[22] L. De Moura and N. Bjørner, "Satisfiability modulo theories: Introduction and applications," *Commun. ACM*, vol. 54, no. 9, pp. 69–77, 2011. [Online]. Available: http://doi.acm.org/10.1145/1995376.1995394

[23] E. Ábrahám and G. Kremer, "Satisfiability checking: Theory and applications," in *Proc. SEFM*. Springer, 2016.

[24] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. TACAS*. Springer-Verlag, 2008.

[25] F. Frances, C. Fraboul, and J. Grieu, "Using network calculus to optimize the AFDX network," in *Proc. ERTS*, 2006.

[26] L. Zhao, P. Pop, Q. Li, J. Chen, and H. Xiong, "Timing analysis of rate-constrained traffic in TTEthernet using network calculus," *Real-Time Systems*, vol. 53, no. 2, pp. 254–287, 2017.

[27] T. P. Baker and A. Shaw, "The cyclic executive model and ada," *Real-Time Systems*, vol. 1, no. 1, pp. 7–25, 1989.

[28] Z. Hanzalek, P. Burget, and P. Šucha, "Profinet IO IRT message scheduling," in *Proc. ECRTS*. IEEE, 2009.

[29] J. Huang, J. O. Blech, A. Raabe, C. Buckl, and A. Knoll, "Static scheduling of a time-triggered network-on-chip based on SMT solving." in *Proc. DATE*. IEEE, 2012.

[30] H. Zeng, W. Zheng, M. Di Natale, A. Ghosal, P. Giusto, and A. Sangiovanni-Vincentelli, "Scheduling the flexray bus using optimization techniques," in *Proc. DAC*. ACM, 2009.

[31] P. Pop, P. Eles, and Z. Peng, "Schedulability-driven communication synthesis for time triggered embedded systems," *Real-Time Syst.*, vol. 26, no. 3, pp. 297–325, 2004.

[32] 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.

[33] J. Specht and S. Samii, "Urgency-Based Scheduler for Time-Sensitive Switched Ethernet Networks," in *Proc. ECRTS*. IEEE Computer Society, 2016.

[34] P. Meyer, T. Steinbach, F. Korf, and T. Schmidt, "Extending IEEE 802.1 AVB with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic," in *Proc. VNC*. IEEE Computer Society, 2013.

[35] G. Alderisi, G. Patti, and L. L. Bello, "Introducing support for scheduled traffic over IEEE audio video bridging networks," in *Proc. ETFA*. IEEE Computer Society, 2013.

[36] M. L. Raagaard, "Algorithms for the optimization of safety-critical networks," Master's thesis, DTU, 1 2017.

[37] F. Dürr and N. G. Nayak, "No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN)," in *Proc. RTNS*. ACM, 2016.

[38] K. L. McMillan, "Interpolants from Z3 Proofs," in *Proc. FMCAD*. FMCAD Inc, 2011.

[39] L. Cordeiro, B. Fischer, and J. Marques-Silva, "SMT-Based Bounded Model Checking for Embedded ANSI-C Software," in *Proc. ASE*. IEEE Computer Society, 2009.