Traffic Planning for Time-Sensitive Communication

Wilfried Steiner, Silviu S. Craciunas, and Ramon Serna Oliver

Abstract

Many industries, including the automotive industry and industrial automation, have a need for reliable real-time communication. To satisfy this need, industry-specific, often proprietary, solutions have been developed in recent decades. Time-sensitive networking (TSN) is addressing a grand unification of these existing technologies to leverage cross-industry cost efficiency as well as to guarantee a stable growth path of communication capabilities.

While the functionality of time-sensitive networking is exhaustively standardized in IEEE 802.1, configuration standards for TSN are largely missing. Thus, in this article we review the IEEE TSN standards, in particular 802.1Qbv, and continue to illustrate a reference model for traffic planning for time-sensitive communication. Furthermore, the article references a detailed technical report describing the complete model, which is ideally suited as a starting point for standardization and ensures the interoperability of future TSN traffic planning tools.

Introduction

Many modern cyber-physical systems, such as automobiles, airplanes, or industrial automation systems, are distributed computer systems that more and more rely on the capabilities of a real-time communication network. A typical use case of real-time communication networks are closed-loop control systems in which a physical entity and/or its environment is measured by a set of sensors, which communicate their sensor readings to one or many processors. The processors then calculate correcting actions and communicate these actions to a set of actuators for execution. This triple [sensing, calculating, acting] is often executed periodically with application-specific frequencies that vary widely with the application area. For example, motion control operates in the kilohertz range and higher while some automotive systems operate at a few tens of hertz or lower. Timely delivery of messages in such control systems is essential to ensure proper system operation. Sometimes timely message delivery is also safety critical. In those cases, the network must even give a guarantee for the quality of service that it claims to provide. Often such a guarantee is at the level of a mathematical correctness proof, e.g., in fly-by-wire systems in airplanes.

Since the traffic in control systems is highly regular, it can be statically planned at system design time. Furthermore, this plan may not only define the communication paths and bandwidth reservations, but also particular points in a network-wide reference time at which messages are to be sent and forwarded. Such a plan including temporal aspects is called a "communication schedule" (or short, schedule) and the synchronized execution of the schedule by network participants is often referred to as "time-triggered communication." The IEEE 802.1 Time-Sensitive Networking Task Group (TSN[1]) has standardized a specific form of time-triggered communication in the IEEE 802.1Qbv project that developed "Amendment 25: Enhancements for Scheduled Traffic." The principle of operation is described in clause 8.6.8.4 of the amendment to the IEEE 802.1Q standard [1] as well as later in this article. Thus, TSN enables the use of standard Ethernet as a real-time communication network. Furthermore, TSN allows sharing an Ethernet network between multiple time-critical applications as well as non-time critical applications. This is a significant advantage to the current state-of-the-art where systems frequently install independent networks for different applications. While there exists a wide variety of real-time network solutions, many of which are Ethernet-based, the standardization within IEEE 802.1 promises broad market acceptance of TSN and a grand unification of proprietary protocols toward a set of open standards.

In this article we focus on the time-triggered communication aspects of TSN, in particular on the scheduling problem formulation. We are concerned with configuring IEEE 802.1Qbv to minimize transmission latency. For this, we assume that end stations and switches are able to execute the time-triggered paradigm with a sufficient level of quality. This may require hardware mechanisms to be in place to ensure timely accuracy of the execution of transmission and forwarding events in end stations and switches. Although we do not cover end stations/switches with relaxed timing guarantees in this article, future work will address how the formalism presented in this article can be adopted to also cover less stringent timing guarantees.

We continue with a review of the IEEE 802.1 model of communication including time-triggered communication. We define the scheduling problem in more detail and reference an in-depth technical article for the complete formal specification. One approach to solving the scheduling problem is by means of general-purpose tools, like SMT-solvers, which we discuss. We give a brief

[1] http://www.ieee802.org/1/pages/tsn.html

Digital Object Identifier:
10.1109/MCOMSTD.2018.1700055

The authors are with TTTech Computertechnik AG.

2471-2825/18/$25.00 © 2018 IEEE

overview of related work and then conclude the article.

TSN Time-Triggered Communication Principle

System Model

A network can be represented by a graph $\mathcal{G} = \mathcal{V}, \mathcal{E}$, where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of non-directed edges as well as directed edges connecting nodes to each other. Each undirected edge $(v_i, v_j) \in \mathcal{E}$ between two nodes $v_i, v_j \in \mathcal{V}$ defines two directed edges $[v_i, v_j], [v_j, v_i] \in \mathcal{E}$ between the two nodes, where the first node in the pair description defines the source node and the second node defines the destination node. In a TSN network nodes will typically be switches and end stations. The undirected edges, on the other hand, could be the physical Ethernet links, while the directed nodes represent the full-duplex communication capablity of each Ethernet link. An example graph $\mathcal{G}$ with eight nodes and seven undirected edges resulting in 14 directed edges is depicted in Fig. 1.

Nodes communicate with each other by the concepts of streams and frames. A stream (or flow) is a periodic multicast data transmission from one talker (the sender) to one or multiple listeners (the receivers). Typically, the senders and the receivers will be end stations, while switches will just operate as forwarding nodes. We denote the set of streams in the network with $\mathcal{S}$. A stream $s_i \in \mathcal{S}$ from talker node v_1 to listener node v_n routed through the intermediary nodes (i.e. switches) $v_2, v_3 \ldots, v_{n-1}$ is expressed as

$$s_i = [[v_1, v_2], \ldots, [v_{n-1}, v_n]].$$

We assume that for each stream the sender and receiver nodes v_1, v_n, as well as the routed communication path that connect the sender and receiver nodes are known and given. A stream is defined by the tuple $\langle s_i.e2e, s_i.jitter, s_i.size, s_i.period \rangle$, denoting the maximum allowed end-to-end latency, the maximum allowed jitter, the data size in bytes, and the period of the stream, respectively.

While the stream defines the overall end-to-end communication between sender and receivers, the concept of a frame (also called a frame instance) identifies a particular message communicated between any two nodes. In particular, each stream $s_i \in \mathcal{S}$ defines a frame $f_i^{[v_a,v_b]}$ on each edge $[v_a, v_b]$, where $[v_a, v_b]$ is part of the routed communication path of stream s_i. The set of all frames (from all streams) communicated on $[v_a, v_b]$ is defined by $\mathcal{F}^{[v_a,v_b]}$. Each such frame is characterized by a frame length $f_i^{[v_a,v_b]}.L$ and a frame period $f_i^{[v_a,v_b]}.T$. The period of the frame is equal to the period of the stream while the length of the frame is calculated based on the data size of the stream and the link speed.

Classical Time-Triggered Communication

In classical time-triggered communication, the sender of a frame is configured to transmit the frame at a specific point in time $f_i^{[v_a,v_b]}.offset$. This transmission point in time relates to a network-wide reference time, which can be established by an appropriate synchronization protocol as, for example, IEEE 802.1AS or other IEEE 1588 profiles. Such synchronization protocols ensure that any two non-faulty nodes in the network maintain synchronized local clocks C_a, C_b which are synchronized to each other with a known upper bound: $|C_a - C_b| \leq \Pi$. Π is called the "precision" and can be determined at system design time. Key parameters for the calculation of the precision are the hardware characteristics of the nodes, in particular the quality of the oscillators, the network topology, the communication medium, as well as the algorithmic properties of the synchronization protocol in use.

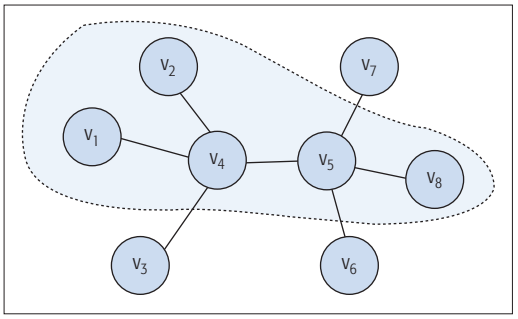

Figure 1. Example network.

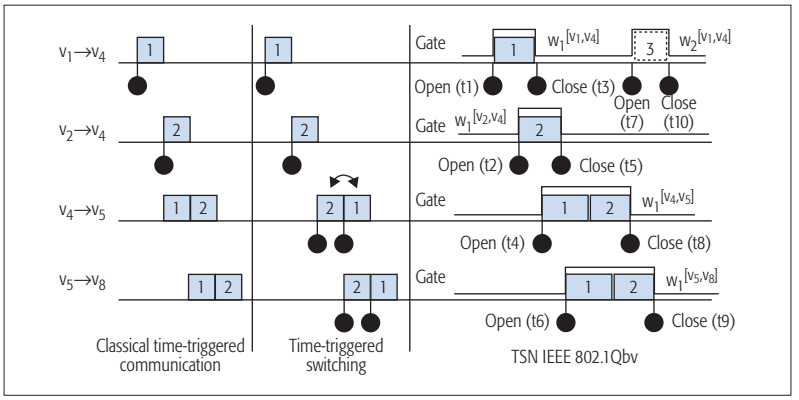

Figure 2. Communication scenarios in classical TT, TT switching, and TSN

In classical time-triggered protocols, the scheduling problem is to find the transmission points in time $f_i^{[v_a,v_b]}.offset$, given the frame properties, topology information, and precision as an input. Figure 2 (left side) depicts an example communication scenario in the example network from Fig. 1. In this example end stations v_1 and v_2 both send messages 1 and 2 to end station v_8. They are doing so by transmitting their respective messages at the transmission points in time indicated by black circles to switch v_4. Switch v_4 then forwards the messages to v_5, which in turn delivers the messages to end station v_8. Thus, in this specific example the scheduling problem is to find concrete values for the black circles.

Since classical time-triggered communication has been invented for networks with physical bus connections (later on for networks with hubs) [2], rather than for switched networks, the schedule of the transmission points in time has been mandatory and sufficient. Adding time-triggered forwarding capabilities in the switches eases the integration of time-triggered (i.e., scheduled) traffic with event-triggered traffic (i.e., standard non-time-triggered traffic), and provides more scheduling options to improve network efficiency.

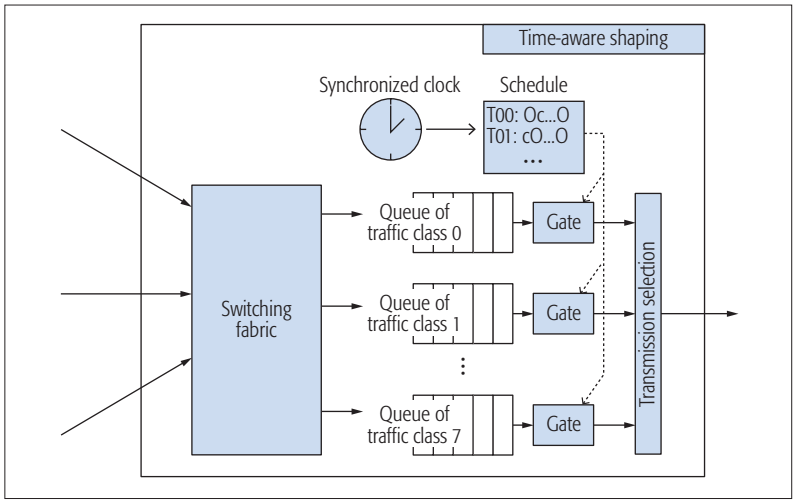

FIGURE 3. Overview of a TSN switch.

There are different options for time-triggered forwarding in a switch. We briefly discuss time-triggered switching as a direct extension of classical time-triggered communication next, and then focus on TSN IEEE 802.1Qbv in the remainder of the article.

Time-Triggered Switching

While in the classical time-triggered communication approach only the sender node has a scheduled transmission point in time $f_i^{[v_a,v_b]}.offset$, switch-based network architectures provide more capabilities for scheduling [3]. In time-triggered switching each switch can schedule the forwarding points in time per frame as well. An example is depicted in the middle of Fig. 2. Again, end stations v_1 and v_2 send messages 1 and 2 to end station v_8. The messages are dispatched at the same points in time as in classical time-triggered communication. However, in contrast to classical time-triggered communication, time-triggered switching allows to schedule the points in time when the messages are forwarded in the switches v_4 and v_5.

Again, an example is presented in Fig. 2 (middle scenario). End stations v_1 and v_2 dispatch frames to switch v_4. Switch v_4 now has the capability to schedule new forwarding points in time for the frames. In particular, switch v_4 reorders the two messages and sends message 2 before message 1.

Thus, the scheduling problem for time-triggered switching extends classical time-triggered communication from finding exactly one transmission point $f_i^{[v_a,v_b]}.offset$ for each message f_i (v_a being the sender of a stream) to finding *a set* of points in time $f_i^{[v_x,v_y]}.offset$ (v_x being any node in the path of the stream except the last one).

TSN Variation of Time-Triggered Communication

While time-triggered switching provides full control of the timing of each frame in a switch, it needs hardware support that is typically not available in standard IEEE 802.1 switches. In particular, frame memory management in a switch would need a major re-design as the communication schedule could require arbitrary frame reordering when forwarding frames (i.e., a frame received at a point in time t_1 could be scheduled for forwarding earlier than another frame that has been received before t_1). This is not the case in IEEE 802.1 switches that follow a queue-based model of communication that handles frames according a first-in first-out (FIFO) paradigm.

This queue-based model of communication is depicted in Fig. 3. Here, an overview of a TSN switch with four ports is shown. The three ingress ports on the left-hand side receive frames that are to be forwarded on the egress port on the right-hand side. As depicted, a switching fabric identifies the frames on the ingress, typically based on information in the Ethernet frame header, and assigns them to the egress port. Furthermore, the frames will be specifically assigned to one of potentially many queues associated with the same port. Usually, a defined priority field in the Ethernet frame is used for queue selection, although other possibilities are currently finalizing standardization.

We assume that each node (i.e., end station and switch) implements at least one queue q for each directed edge that it sources. For example, v_4 implements a queue $q_i^{[v_4,v_5]}$ for the edge $[v_4, v_5]$. For simplicity, in this article we assume only a single queue for scheduled traffic. Thus, all scheduled frames to be communicated from one node to another one will traverse the same queue. The generalization of the approach to multiple queues is straightforward.

To realize time-triggered communication, TSN defines a specific shaping mechanism on how frames are selected for transmission from a queue (also depicted in Fig. 3). In particular, each queue is assigned to a *gate* and this gate is at any time in one of the two states *open* or *close*. When the gate of a respective queue is in the open state, then frames can be selected for transmission on the directed edge associated with the queue in first-in first-out (FIFO) order. In case the gate of a respective queue is in the close state, frames from this queue are not selected. The state of a gate may change from open to close and vice versa from close to open. These state changes are statically scheduled with respect to a synchronized time and defined at design time of the network (or through proper re-configuration events in the network). Thus, as synchronized time proceeds, a node continually checks whether a state change for one of its gates is scheduled. If yes, then the state change is executed. An example communication scenario is depicted in Fig. 2 (right-most scenario).

Altering states of a queue's gate define an ordered list of transmission windows on a timeline, i.e., windows during which the gate is in the open state. Each window $w_k^{[v_i,v_j]}$ is defined by a left boundary $w_k^{[v_i,v_j]}.open$ and a right boundary $w_k^{[v_i,v_j]}.close$. As we will see later, the maximum number of windows $\mathcal{W}_{max}^{[v_i,v_j]}$ per edge will be an essential parameter in the performance of the schedule synthesis.

The TSN IEEE 802.1Qbv also contains clauses that standardize end station behavior for TSN time-triggered behavior. This behavior is analogous to the window-based approach described above for switches.

Figure 2 (right) depicts the communication scenario as discussed before for classical time-trig-

gered communication and time-triggered switching, but this time using window-based scheduling as defined in TSN (plus an additional frame 3). As depicted, end stations v_1 and v_2 define windows during which messages 1 and 2 are selected for transmission to switch v_4. The switches v_4 and v_5, then, do not assign forwarding points in time to the individual messages, but rather define points in time when to change the state of the gate of a respective queue from open to close and from close to open.

Thus, the scheduling problem in TSN time-triggered communication is to find the points in time for the open $w_k^{[vx,vy]}.open$ and closing events $w_k^{[vx,vy]}.close$ of the gates.

TSN Scheduling Problem Formulation

Open $w_k^{[vx,vy]}.open$ and closing events $w_k^{[vx,vy]}.close$ need to be in certain relations to each other. These relations can be mathematically expressed in the form of (mostly) linear inequalities that we call "constraints." Each constraint by itself has a simple form, only relating a couple of parameters to each other. However, the number of such constraints grows with the number of nodes in the topology and quadratic with the number of messages. Thus, although each individual constraint is trivial, the overall scheduling problem will grow in complexity because of the high number of constraints. For example, industrial-sized case studies can easily imply a million constraints and beyond. In this section we only briefly illustrate the formal specification approach by means of examples. The full specification of the scheduling constraints that formulate the scheduling problem of a TSN network can be found in [4].

Figure 2 depicts the transmission of frames 1 and 2 from end stations v_1 and v_2 to end station v_8 by passing through the switches v_4 and v_5. The frames are transmitted by the end stations and switches only when the gate of the associated queue is in the open state. As depicted, the phases when gates are in the open state form windows w_i and the traffic scenario in Fig. 2 defines five such windows: two for the transmission from v_1 to v_4 ($w_1^{[v1,v4]}$, $w_2^{[v1,v4]}$) and one window for each other link in the transmission paths of frames 1 and 2. Thus, each of the windows is defined by an open and a close event of the respective gate, and it is the purpose of the scheduling tool to generate concrete values for these events.

In this example, we assume that we executed the scheduling tool and the tool returned the concrete values $t1$ ... $t10$ which are given in the brackets next to the open and close events. Of course, in a real scenario, these values would likely be integer numbers representing concrete points in a synchronized time (e.g., as generated by IEEE 802.1AS).

The values $t1$ ... $t10$ are generated by the scheduling tool by searching for a solution that satisfies constraints such as described in [4]. By reference to Fig. 2 we can explain some of these constraints. One rather simple constraint defines that the point in time of the close event of a window is higher than the open event of the very same window. Thus, when the tool returns two integers t_1, t_3 and $t_1 < t_3$, then this constraint is said to be "satisfied" (i.e., it is true). Some more complex constraints define how messages are assigned to windows, constraints on the messages' end-to-end transmission latencies, as well as constraints on the sizes of the windows. For example, since window $w_1^{[v4,v5]}$ will transmit two messages, the difference between the open event and the close event of this window needs to be sufficiently large. Thus, the scheduler needs to return values t_4 and t_8 such that $(t_8 - t_4)$ is high enough.

Similar to the examples above, the remaining constraints that formulate the overall TSN scheduling problem do not get much more complicated. However, since there will be quite a lot of such constraints, the task of the scheduling tool becomes very difficult. We will discuss some classes of scheduling tools in the next sections.

Scheduling Performance and Results

The mathematical formulation of the TSN scheduling problem as discussed in the previous section enables us to directly encode concrete networks and communication needs as input to general-purpose tools. For example, SMT-solvers (satisfiability modulo theories) are such general purpose tools. This encoding serves two main purposes. First, schedules can be synthesized in some configurations (as we will see later) merely by pushing a button. Second, in the case that a TSN schedule has been calculated by other means (e.g., by a heuristic), this solution together with the constraints can both be used as input to a tool, like an SMT-solver. In this second case the SMT-solver will analyze whether the TSN schedule is consistent with the constraints, or not. This analysis is a perfect tool for standardization and interoperability tests of TSN schedules. Once the constraints are standardized, each vendor can produce schedules by whatever means and tricks available; a successful consistency check with the standardized constraints guarantees interoperability. The analysis time of such a consistency check is typically in seconds even for large networks [3].

To also demonstrate the schedule synthesis (and to discuss trade-offs) we have encoded some example networks in SMT formulation and synthesized schedules for small to medium sized networks. The results of the schedule synthesis is presented in Fig. 4 and Fig. 5.

The figures present the runtime of the window scheduler when varying the number of streams (10, 25, 50) and the number of windows per queue (1, 2, 3, 5). The z-axis is logarithmic. The period of all streams is set to 20 ms and the length of the frames to 1 MTU size (13 microseconds on a 1 Gbit/s link); the granularity of the timeline is 1 microsecond. The numbers were obtained on an Intel(R) Core(TM) i7-5600U CPU @ 2.60GHz with 8 Gb of RAM.

As expected, the figures demonstrate two trade-offs in generating TSN schedules. The first trade-off is between the scheduling time (i.e., the time duration it takes the tool to generate a schedule) and the number of windows per queue. As indicated in Fig. 4, the synthesis time of 50 streams considering a single window is about 10 milliseconds, while it grows up to about an hour for five windows per queue. Thus, the tool does

Once the constraints are standardized, each vendor can produce schedules by whatever means and tricks available; a successful consistency check with the standardized constraints guarantees interoperability. The analysis time of such a consistency check is typically in seconds even for large networks.

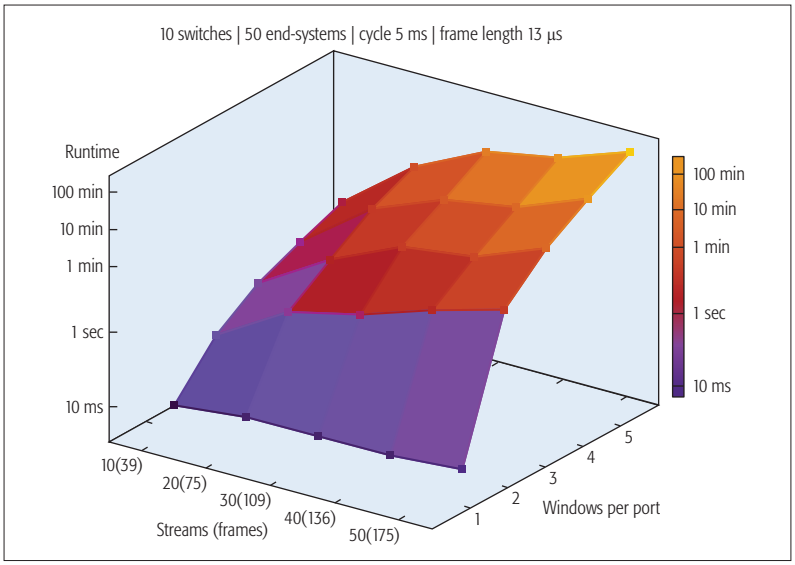

FIGURE 4. Schedule synthesis time.

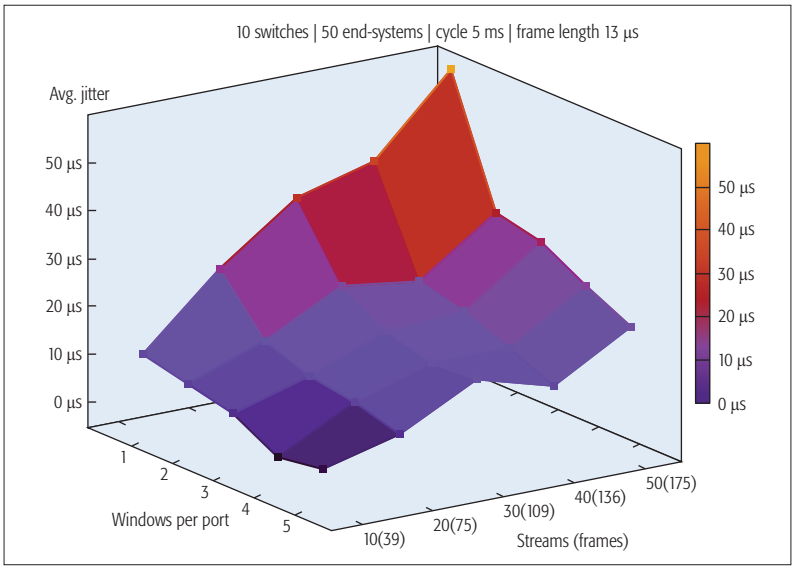

FIGURE 5. Schedule quality.

find solutions for one window much faster than for multiple windows. However, fewer windows also cause higher transmission latencies and jitter as indicated by the second trade-off depicted in Fig. 5. While the jitter for the scenario with five windows is around 10 microseconds, it grows beyond 50 microseonds when only a single window is used. From the two trade-offs we can also conclude that schedules with relaxed latency and jitter requirements are synthesized faster than those with stringent timing constraints.

While the number of streams is relatively low in the examples, scheduling time-triggered systems is an active field of research and much larger instances can be routinely solved today. We will briefly discuss some of the approaches and their relation to TSN scheduling in the following section.

Related Work

In general, the time-triggered scheduling problem has been studied in various forms for many decades. In an early incarnation the scheduling problem was formulated as static cyclic scheduling of a task set by Baker *et al.* in the late 1980s [5]. More recently, the implementation of time-triggered communication in Ethernet has been addressed in scheduling research by Hanzalek *et al.* [6] and by Steiner [3].

This article continues the approach presented in [3] which demonstrated that the core time-triggered configuration problem is a satisfiability problem and that SMT-solvers (satisfiability modulo theories) are, therefore, an effective tool to develop solutions. Indeed, in [3] scheduling instances in complex network topologies with up to a thousand messages have been solved in minutes by means of an incremental scheduling strategy. Following this strategy instead of solving the complete schedule at once, the set of messages is partitioned and solved in portions incrementally. Furthermore, in recent years this solution has been improved by orders of magnitude in the number of messages [7] and can now solve scheduling problems with hundreds of thousands of messages in a reasonable time span.

Craciunas *et al.* [8] have demonstrated how the scheduling results from time-triggered switching can be translated to the TSN time-triggered communication principle. In contrast to the formalization presented in this article, Craciunas *et al.* directly scheduled frame transmissions, like in time-triggered switching, and generated a window schedule for TSN in a post-synthesis step. While this allows the immediate application of existing time-triggered switching scheduling techniques, it causes restrictions in the generated schedules. In particular, the number of windows becomes high in relation to the number of frames. The direct encoding of the window-based constraints removes these restrictions.

The typical use case of TSN will be converged networks, which maintain time-triggered communication side by side with regular and unsynchronized real-time communication. There is also an increasing body of research that addresses scheduling and calculation of timing bounds for unsynchronized traffic in such converged settings. For example, Tamas-Selicean *et al.* [9] solve the time-triggered scheduling problem by means of meta-heuristics and include transmission latency requirements for unsynchronized communication in the schedule synthesis process. Thus, while the schedule is generated it is guaranteed that timing guarantees for the unsynchronized parts of traffic are also met. Zhao *et al.* [10] extend network calculus, a well known technique to determine timing bounds on unsynchronized traffic, to converged networks. Thiele *et al.* [11] provide a formal analysis to analyze the worst-case timing of TSN networks. Finally, Ashjaei *et al.* [12] address the behavior of IEEE AVB traffic (IEEE 802.1Qav) in the presence of scheduled traffic.

Conclusion

IEEE 802.1 TSN defines a set of open standards for real-time communication and there are well-developed adoption strategies in both the automotive and industrial industries. While the standards cover all functional aspects of TSN, the configuration challenge has not been addressed so far. Thus, in this article we have illustrated a formalization of the scheduling problem for the TSN

variation of time-triggered communication and referenced a technical report with a full formal reference model. We furthermore encoded small and medium network instances as an input for an SMT-solver and synthesized schedules by using the SMT-solver as a black box. We also reviewed the current body of research in time-triggered communication scheduling and discussed a clear growth path for performance improvements for TSN scheduling.

As discussed, the formalization of the scheduling problem in the form of mathematical constraints, as presented in this article, furthermore enables automated interoperabilty tests. Thus, this formalization ideally serves as a first step toward the standardization of TSN traffic planning, to ensure the compatibility of upcoming planning tools with each other.

References

[1] "IEEE Draft Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks," *IEEE P802.1Q-REV/D2.0*, July 2017, Jan. 2017, pp. 1–1946.

[2] F. Consortium *et al.*, "Flexray Communications System-Protocol Specification," *Version*, vol. 2, no. 1, 2005, pp. 198–207.

[3] W. Steiner, "An Evaluation of SMT-based Schedule Synthesis for Time-Triggered Multi-Hop Networks,"*Real-Time Systems Symposium (RTSS)*, 2010 IEEE 31st, 2010, pp. 375–84.

[4] S. S. Craciunas, R. Serna Oliver, and W. Steiner, "Formal Scheduling Constraints for Time-Sensitive Networks," Sep. 2017, https://doi.org/10.5281/zenodo.997996 https://arxiv.org/abs/1712.02246.

[5] T. P. Baker and A. Shaw, "The Cyclic Executive Model and Ada," *The J. Real-Time Systems*, vol. 1, 1989, pp. 120–29.

[6] Z. Hanzalek, P. Burget, and P. Sucha, "Profinet IO IRT Message Scheduling with Temporal Constraints," *IEEE Trans. Industrial Informatics*, vol. 6, no. 3, 2010, pp. 369–80.

[7] F. Pozo *et al.*, "Period-Aaware Segmented Synthesis of Schedules for Multi-Hop Time-Triggered Networks," *2016 IEEE 22nd Int'l. Conf. Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016, pp. 170–75.

[8] S. S. Craciunas *et al.*, "Scheduling Real-Time Communication in IEEE 802.1 Qbv Time Sensitive Networks," *Proc. 24th Int'l. Conf. Real-Time Networks and Systems. ACM*, 2016, pp. 183–92.

[9] D. Tămaş-Selicean, P. Pop, and W. Steiner, "Design Optimization of TTEthernet-based Distributed Real-Time Systems," *Real-Time Systems*, vol. 51, no. 1, 2015, pp. 1–35.

[10] L. Zhao *et al.*, "Timing Analysis of Rate-Constrained Traffic in TTEthernet Using Network Calculus," *Real-Time Systems*, vol. 53, no. 2, 2017, pp. 254–87.

[11] D. Thiele, R. Ernst, and J. Diemer, "Formal Worst-Case Timing Analysis of Ethernet TSN's Time-Aware and Peristaltic Shapers," *2015 IEEE Vehic. Net. Conf. (VNC)*, 2015, pp. 251–58.

[12] M. Ashjaei *et al.*, "Schedulability Analysis of Ethernet Audio Video Bridging Networks with Scheduled Traffic Support," *Real-Time Systems*, vol. 53, no. 4, 2017, pp. 526–77.

Biographies

Wilfried Steiner (wilfried.steiner@tttech.com) is a corporate scientist at TTTech Computertechnik AG and Leader of the research team TTTech Labs. He holds a degree of Doctor of Technical Sciences from the Vienna University of Technology, Austria. His research is focused on the design of systems and network protocols with real-time, dependability, and security requirements. Target areas are automotive, space, aerospace, and more recently the Industrial Internet of Things.

Silviu S. Craciunas is a core architect at TTTech Computertechnik AG, Vienna, working on scheduling algorithms for time-triggered networks and real-time operating systems. He received his Ph.D. in 2010 from the University of Salzburg, Austria, and his Ing. Dipl. from the Politechnica University of Timisoara, Romania in 2005. His research interests include real-time and safety-critical systems, real-time scheduling algorithms, power-aware computing, and deterministic real-time networks.

Ramon Serna Oliver works as core architect in the Deterministic Ethernet Core group in TTTech Computertechnik AG, Vienna, Austria. He received his Ph.D. (Dr.-Ing) in 2010 from the Chair of Real-Time Systems in TU Kaiserslautern, Germany. He also holds a computer engineer diploma from the University of Balearic Islands, Spain. His research interests include scheduling algorithms for time-triggered systems, deterministic wired and wireless communications, and network synchronization aspects and quality of service trade-offs.

The formalization of the scheduling problem in the form of mathematical constraints, as presented in this article, furthermore enables automated interoperabilty tests. Thus, this formalization ideally serves as a first step toward the standardization of TSN traffic planning, to ensure the compatibility of upcoming planning tools with each other.