Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks

Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel

***Abstract*—Several networking architectures have been developed atop IEEE 802.3 networks to provide real-time communication guarantees for time-sensitive applications in industrial automation systems. The basic principle underlying these technologies is the precise transmission scheduling of time-triggered traffic through the network for providing deterministic and bounded latency and jitter. These transmission schedules are typically synthesized offline (computational time in the order of hours) and remain fixed thereafter, making it difficult to dynamically add or remove network applications. This paper presents algorithms for incrementally adding time-triggered flows in a time-sensitive software-defined network (TSSDN). The TSSDN is a network architecture based on software-defined networking, which provides real-time guarantees for time-triggered flows by scheduling their transmissions on the hosts (network edge) only. These algorithms exploit the global view of the control plane on the data plane to schedule and route time-triggered flows needed for the dynamic applications in the Industrial Internet of Things (Industry 4.0). The evaluations show that these algorithms can compute incremental schedules for time-triggered flows in subseconds with an average relative optimality of 68%.**

***Index Terms*—Industrial networks, integer linear programming, software-defined networking.**

I. Introduction

CYBER-PHYSICAL SYSTEMS (CPS) controlling physical processes through a set of distributed sensors, actuators, and CPS controllers rely on computer networks to transport sensor data and actuator commands. Typically, such CPS are time-sensitive systems, where the network delay and delay variance (jitter) impact the quality of control of the CPS. For instance, machines in industrial automation implementing time-sensitive control loops demand bounds on network delay and jitter to the order of a few microseconds [1]. Consequently, in order to ensure a deterministic behavior of CPS, *deterministic real-time* networks with bounded delay and jitter are desirable.

Traditionally, such guarantees have been provided by dedicated field-bus networks. With the proliferation and steadily growing performance along with shrinking costs of IEEE 802.3 networks, there is a strong incentive to also utilize these technologies, originally designed to provide only best-effort communication services, for implementing time-sensitive CPS. Ideally, both time-sensitive and nontime-sensitive applications should be able to communicate over one *converged* IP-based IEEE 802.3 network. Several real-time Ethernet variants, such as TT-Ethernet [2], ProfiNET [3], etc., have been proposed for this purpose. Moreover, the standardization of such real-time networks is also in the focus of two major standards bodies in networking, namely IETF targeting routed IP networks with the Deterministic Networking (DetNet) Working Group [4] and IEEE targeting switched real-time networks with the Time-Sensitive Networking (TSN) Task Group [5]. Given that the propagation, processing, and transmission delays in the state-of-the-art network elements are deterministic and constant [6], these network architectures strive to bound the nondeterministic queuing delays in order to achieve an overall deterministic end-to-end latency and jitter for time-sensitive traffic. To this end, they schedule the transmission of packets belonging to time-triggered flows (all packets of a given stream are referred to as a flow) at the hosts and the network elements (switches) by means of precisely synchronized clocks achieved using protocols such as the IEEE 1588 Precision Time Protocol (PTP). Packets belonging to time-triggered flows are then transmitted based on a global transmission schedule such that the incurred queuing delays are also deterministic.

While the concept of time-triggered communication is a well-established one, the focus has been on developing methods that compute transmission schedules for systems where the time-triggered flows are fixed and known *a priori* [7]–[9]. However, with the emergence of dynamic applications, e.g., in the Industrial Internet of Things (Industry 4.0) [10], the online flow scheduling problem has become highly relevant. Online flow scheduling algorithms are required for incrementally setting up applications in the network, for instance, adding "plug-and-produce" devices in the production lines [11]. These algorithms must avoid disturbances to the existing flows due to schedule modifications, as it may lead to a temporary violation of bounds on delays or jitter with undesirable consequences [12]. Moreover, to be deployable, these approaches are required to have low execution times (less than a few seconds), and hence, the existing methods for offline scheduling cannot be used due to their high runtime in the order of hours.

This paper introduces scheduling algorithms to incrementally compute schedules for time-triggered (periodic) flows in

Manuscript received April 26, 2017; revised July 27, 2017 and October 20, 2017; accepted November 16, 2017. Date of publication December 11, 2017; date of current version May 2, 2018. Paper no. TII-17-0829. *(Corresponding author: Naresh Ganesh Nayak.)*

The authors are with the Institute for Parallel and Distributed Systems, University of Stuttgart, 70174 Stuttgart, Germany (e-mail: nayaknh@ipvs.uni-stuttgart.de; duerrfk@ipvs.uni-stuttgart.de; rothermel@ipvs.uni-stuttgart.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org

Digital Object Identifier 10.1109/TII.2017.2782235

a time-sensitive software-defined network (TSSDN) [13]. The TSSDN is a network architecture based on software-defined networking (SDN), which provides deterministic real-time guarantees for time-triggered traffic by isolating it either spatially or temporally. The TSSDN exploits the logical centralization of the control plane in SDN to dynamically compute routes and schedules for time-triggered flows. Furthermore, transmissions in the TSSDN are scheduled on the hosts only, i.e., at the network edge, rather than also on the switches. This enables the TSSDN to provide lightweight mechanisms to swiftly update schedules online for the dynamically changing time-triggered traffic. In particular, the contributions of this paper are the following.

1) This paper introduces the scheduling problem and the challenges to incrementally compute schedules for time-triggered flows in the TSSDN.
2) Moreover, this paper also proposes various integer linear programming (ILP) formulations for computing incremental schedules, which approximate the optimal solutions of the corresponding static scheduling problem.
3) Through evaluations, the paper shows that the presented ILPs, along with the proposed optimizations, can compute incremental schedules for flows in networks of realistic sizes in the subsecond range. The resulting schedules have an average relative optimality of 68%.

This paper is structured as follows. The related work is presented in Section II. Section III introduces the TSSDN and the underlying scheduling problem. Section IV introduces ILP formulations for flow scheduling and relevant optimizations to reduce the computational runtime. The presented scheduling approaches are evaluated in Section V.

II. Related Work

There is a strong trend to make widely adopted IP-based and IEEE 802 networks ready for real-time traffic. These developments are driven, in particular, by the IEEE 802.1 TSN Task Group [5], which aims for time-synchronized low-latency streaming services through IEEE 802 networks, and the IETF DetNets Working Group targeting deterministic data paths with bounds on packet latency, loss, and jitter over Layer 2 bridged and Layer 3 routed networks, respectively. The IEEE has also recently finalized and published the enhancements (IEEE 802.1Qbv) for supporting scheduled traffic in IEEE 802.3 networks [14]. Additionally, there is also an impetus to provide real-time guarantees over wireless networks such as IEEE 802.11 and the emerging 5G networks [15], [16].

Transmission scheduling in time-triggered networks to impart real-time properties is a well-researched problem. Approaches using satisfiability modulo theories (SMTs) and resource constrained project scheduling have been applied to compute static schedules in multihop Ethernet-like networks such as TT-Ethernet and ProfiNET [7], [9], [17]. These approaches have also been extended to compute combined transmission and task schedules [8]. Furthermore, approaches using SMT and no-wait job-shop scheduling have also been used to compute schedules for IEEE 802.1Qbv networks [18], [19]. All of these approaches separate the flow routing problem from flow scheduling, requiring that flow routes are given before calculating the schedules. This strict separation prevents the joint optimization of routing and scheduling and the possibility to isolate flows spatially through different routes to avoid bottlenecks along routes. Moreover, these network architectures schedule transmissions of the flows also on the switches, and hence, schedule updates require coordination across multiple switches. This results in high execution time of the aforementioned scheduling approaches, effectively ruling them out for usage in online scheduling.

In contrast, the TSSDN handles the scheduling and routing problem combinedly to increase the number of time-triggered flows that may be supported in the network. In the TSSDN, transmissions are scheduled on the network edge only and not on the switches inside the network. Thus, schedule updates are limited to inserting flow-table entries in the switches and are designed to not influence the existing flows in the network. The existing work focuses on offline computation of static schedules and routes in the TSSDN [13]. This paper makes an important step forward by introducing optimized online scheduling algorithms to compute incremental schedules based on the capabilities of SDN to modify the data plane at runtime.

III. Time-Sensitive Software-Defined Network

A. Architecture

The TSSDN is based on the principles of SDN. SDN is an emerging networking paradigm based on the principle of control plane and data plane separation. The data plane consists of network elements (switches), which are responsible for forwarding packets. The control plane is responsible for configuring the data plane, e.g., by calculating routes and programming the forwarding tables (also called flow tables) of switches. With SDN, the control plane is moved from the switches to a network controller hosted on standard servers that communicates with the switches through a so-called southbound interface like the OpenFlow protocol [20]. The network controller is logically centralized, i.e., it has a global view onto the switches, topology, traffic, etc., which facilitates implementation of network control logic like routing and scheduling as in this paper.

The data plane in the TSSDN consists of full-duplex Ethernet switches and end systems (hosts). The TSSDN relies on fixed-priority first-in first-out queues as typically available in commodity switches (cut-through as well as store-and-forward). The maximum network diameter in the TSSDN is limited, e.g., to seven hops as also recommended by the IEEE 802.1D standard. End systems execute user space processes, which serve as the sources and destinations for the time-triggered flows (time-sensitive in nature). For instance, an end system can be a sensor transmitting a stream of samples, an actuator acting upon a stream of commands, or a CPS controller responsible for driving a physical process. The source hosts of time-triggered flows in the TSSDN transmit application layer data units encapsulated in a single user datagram protocol (UDP) packet (size limited by the network maximum transmission unit (MTU)) based on a global transmission schedule. This necessitates precisely synchronized clocks on end systems, which is achieved using the PTP. The hosts use packet-processing frameworks such as netmap [21] or Intel's DPDK for sending and receiving

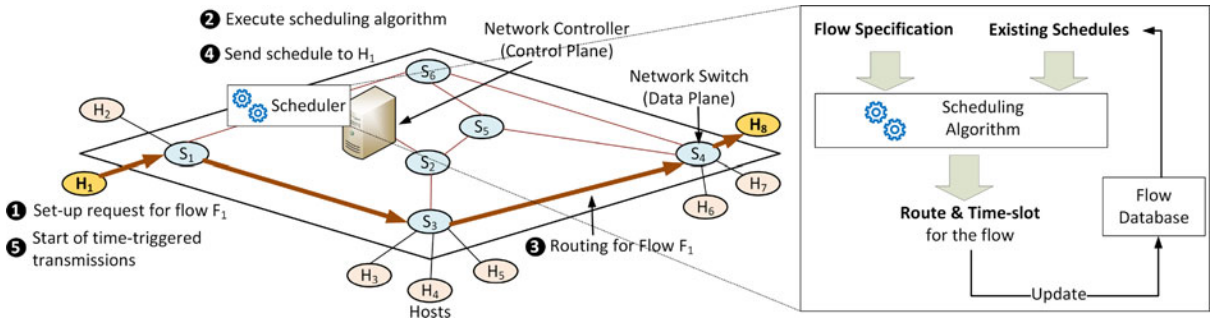

Fig. 1. Steps involved during incremental scheduling of flow F_1 ($H_1 \rightarrow H_8$) in the TSSDN.

packets bypassing the host network stacks to avoid nondeterministic processing delays of the host stacks. Moreover, this enables the hosts to precisely adhere with the global transmission schedule.

The control plane of the TSSDN is hosted on a standard host and is responsible for computing the routes and transmission schedules for the time-triggered flows in the network based on its global knowledge of the topology and traffic. Here, time-triggered flows refer to periodic flows, i.e., flows that initiate transmissions after a predefined period. For static scenarios, the network controller computes offline schedules and updates the flow tables of the switches for routing the time-triggered traffic while also sending the schedule information to the source hosts of the flows. For dynamic scenarios, the network controller computes the schedules online, as shown in Fig. 1. For instance, to schedule an additional flow (say Flow $F_1 : H_1 \rightarrow H_8$), the source host ($H_1$) sends a flow setup request containing the flow specifications (the source host, the destination host(s), the transmission period, etc.) to the network controller in Step 1. In Step 2, the network controller executes an online scheduling algorithm to determine if the flow can be accommodated in the network. If a suitable schedule (time slot and route) exists, then the network controller programs the flow-table entries in the switches for routing the packets belonging to the flow over the corresponding path (Step 3). In Step 4, the source hosts (in this case H_1) are informed about the schedule. The source hosts commence their time-triggered transmissions in Step 5. The network controller uses exact match semantics for the flow-table entries, i.e., no flow aggregation. Moreover, the network controller uses a transactional interface (available since OpenFlow v1.4) for this purpose, so as to ascertain that the updates are indeed applied before sending the schedules to the source hosts. This way, the controller ensures that existing flows are not affected and that network updates are consistent, i.e., no loops/blackholes are created in the data plane [22]. A source host requiring to send multiple packets per cycle must request for a corresponding number of time slots in Step 1. When flows are no longer required, the source host sends a flow removal request to the network controller. The resources freed up are used for scheduling new incoming flows. SDN protocols such as OpenFlow may be extended for the communication between the hosts and the network controller. Overall, the schedule update procedure is limited to inserting flow-table entries for new flows and sending schedule information to the source hosts. Since switches play no role in enforcing schedules, they need not be reconfigured, resulting in lightweight schedule updates.

The global transmission schedule of the TSSDN is a cyclic schedule divided into smaller time slots, numbered from 0 to $t_{\max}$, each long enough for an MTU-sized packet to traverse across the longest network path (restricted to seven hops). The length of this schedule is referred to as the base period, *bp*. The use of cyclic scheduling in the TSSDN restricts the transmission periods of the time-triggered flows to be integral multiples of the base period. To start transmissions, the source host of a flow must acquire a *suitable time slot* from the network controller. During the allocated time slot, the entire path over which the packet traverses is exclusively reserved for it. The reservation implies that no high-priority packets belonging to other time-triggered flows would interfere with it. The lower priority best-effort traffic is not restricted in any way and is free to traverse through the network. The network controller allocates a time slot to the time-triggered flow being scheduled and routes it such that no high-priority flows with overlapping paths are allocated the same time slot, i.e., flows are isolated either *spatially* or *temporally*. Multiple time-triggered flows can share the same time slot if they do not have overlapping routes. This bounds the nondeterministic queuing delays and results in minimal network delay. Time-triggered flows are temporally isolated from best-effort flows through standard priority scheduling.

B. Scheduling Problem

The TSSDN aims to bound end-to-end network delays and jitter for transporting packets of time-triggered traffic. A radical approach is to altogether eliminate the nondeterministic queuing delays in the switches on the path from the source host to the destination host. To this end, whenever packets belonging to time-triggered flows arrive at a switch, the ports on which they are to be forwarded must be free, so that the packets can be forwarded with *zero queueing delay*. The scheduling algorithms in the TSSDN are formulated to guarantee that no high-priority packets interfere and delay each other. Although priority queuing on switches mostly isolates time-triggered traffic from best-effort traffic, best-effort packets that are already in transit when a time-triggered packet arrives at the switch can delay time-triggered traffic. To minimize the variable delay induced by the best-effort traffic already in transit, frame pre-emption mechanisms from *IEEE 802.1Qbu* [23] can be used, or buffers can be added to the destination host for eliminating the resulting

jitter. Thus, zero queuing delay is a slightly idealized formulation of the goal. By (almost) eliminating the queuing delay in the network, the TSSDN delivers time-sensitive packets to their destinations as early as possible. Hence, the formulations of the scheduling problems in the TSSDN do not account for individual flow deadlines.

The scheduling model in the TSSDN results in larger (and hence fewer) time slots for allowing an MTU-sized packet to traverse the network diameter. Moreover, already allocated time slots/routes cannot be modified, as it may induce jitter and issues with consistent network updates in the data plane [22]. These challenges make the scheduling problem in the TSSDN highly relevant for efficiently utilizing the slots to accommodate more flows. To this end, the network controller can influence two variables, namely the allocation of time slots to flows and their routes. In the static scheduling problem, this optimization is NP-hard. In the dynamic version, the focus is on faster computation of schedules while approximating the results of the corresponding static scheduling problem.

This paper strives to reduce (not bound) setup times for time-triggered flows to subseconds by optimizing the scheduling algorithms. The minimization of latencies between the hosts, switches, and the network controller is an orthogonal problem and is out of the scope of this paper [24]. Furthermore, the delays in inserting flow-table entries in switches is hardware dependent. State-of-the-art SDN switches can insert entries with same priorities in less than 15 ms [25].

IV. Scheduling in the TSSDN

This section presents scheduling approaches for the TSSDN based on ILP. ILPs are a broadly accepted standard method for such optimization problems with readily available and highly optimized solvers. Alternatively, as mentioned in the related work, SMT-based approaches could also be used. However, since ILP and SMT showed similar performance [26], the formulations presented in this paper are limited to ILPs.

A. Terminologies and Notations

The network topology is modeled as a directed graph $G \equiv (V, E)$. Here, V is the set of nodes and $E \equiv \{(i,j) | \, i,j \in V$ and i,j are connected by a network link$\}$ is a set of tuples representing the network links. Furthermore, $V \equiv (SW \cup H)$, where SW and H are sets of switches and hosts, respectively. A time-triggered flow is denoted as a tuple, $F \equiv (s, D, p)$, where $s \in H$, $D \subseteq H$, and $p \in \mathbb{Z}^+$. Here, s is the source, D is the set of destinations of the flow, and p denotes the transmission period of the flow. The set of time slots for scheduling is denoted as $T \equiv \{0, 1 \ldots, t_{\max}\}$. $|D|$ and $|E|$ represent the number of elements in the respective sets. Additional functions needed to represent the topology and the flows are listed in Table I.

B. Static Scheduling Problem

The goal of the static scheduling problem is to maximize the number of time-triggered flows, which are accommodated in the network, i.e., the flows that are allocated time slots and routes.

TABLE I
Helper Functions for the ILP Formulations

Helper Function	Parameters	Output
in(n)	$n \in V$	$\{(u,v) \in E \mid v = n\}$
out(n)		$\{(u,v) \in E \mid u = n\}$
src(F)	Flow F,	s
dst(F)	$F \equiv (s, D, p)$	D

The general approach to this problem is to define for each time-triggered flow a set of candidate paths between its source and destinations. The solution to the scheduling problem requires routing flows over one of its candidate paths and allocating them a time slot, while maximizing the number of flows that can be allocated with a time slot and a route.

The choice of candidate paths for the flows determines the number of flows successfully scheduled. In general, the set of candidate paths for each flow can be made from all the shortest paths between its source and its destinations. This approach is referred to as scheduling with pathsets routing (S/PR). Including all paths that exist between the source and the destinations of the flow in the set of candidate paths yields the best solution for the static scheduling problem. Such an approach is named as scheduling with unconstrained routing (S/UR). The set of candidate paths may also be restricted to only one of the randomly chosen shortest paths for computing solutions faster. Such an approach with *a priori* routing is named as scheduling with fixed routing (S/FR). It must be noted that the S/UR and S/FR are specific cases of S/PR and are at the two extremes of the solution space.

The detailed ILP formulations for the approaches—S/PR, S/UR, and S/FR—are available in [13]. This paper uses the solutions generated by these approaches as a benchmark for comparing the approaches to the dynamic scheduling problem.

C. Dynamic Scheduling Problem

In the dynamic scheduling problem, flows are scheduled incrementally, one at a time. Obviously, scheduling and routing a flow with no prior knowledge of flows that would arrive in the future would yield suboptimal solutions in comparison to the corresponding static scheduling problem, where all the flows are known *a priori*. Hence, heuristics such as the number of hops, the number of occupied time slots on each link, etc., are used by the online scheduling algorithms to approximate the solutions generated by the static scheduling algorithm.

1) Shortest Available Path (D/SAP): The objective of this ILP is to *minimize the number of links over which a flow is routed* while allocating a time slot for it, during which all the constituting links are unoccupied. This results in a larger number of free links during a given time slot for future flows.

ILP Inputs: The inputs for D/SAP are the following:

1) network topology as *directed graph*, $G \equiv (V, E)$;
2) *set of time slots*, T. $T \equiv \{0, 1, \ldots, t_{\max}\}$;
3) *specification of the flow* to be set up, $F \equiv (s, D, p)$;
4) *existing schedules*, ES
   ES $\equiv \{\text{es}_{i,j}\}; \forall\, i \in E, \forall\, j \in T$.

$\text{es}_{i,j} = 1$, if any scheduled flow traverses link i at time slot j, else 0.

ILP variables: The variables for D/SAP are the following.

1) *Link occupancy*, LO. This indicates the set of links over which the flow F must be routed. $\text{LO} \equiv \{\text{lo}_i\}; \forall\, i \in E$.
   $\text{lo}_i = 1$, if flow F is routed over link i, else 0.
2) *Destination count*, DC. This indicates the number of destinations of flow F reachable over a given link i if the flow is routed over it. $\text{DC} \equiv \{\text{dc}_i\}; \forall\, i \in E$.
   dc_i is the number of flow destinations that may be reached over link i if flow F is routed over i, else 0.
3) *Slot occupancy*, SO. This indicates the time slot that must be allocated to flow F. $\text{SO} \equiv \{\text{so}_j\}; \forall\, j \in T$.
   $\text{so}_j = 1$, if the flow is allocated slot j, else 0.
4) *Auxiliary variables*, A. These variables enable the formulation of the scheduling problem using linear constraints. $A \equiv \{a_{i,j}\}; \forall\, i \in E, \forall\, j \in T$.
   $a_{i,j} = 1$, if flow F is routed over link i and allocated slot j, else 0.

Objective function: The objective function for D/SAP minimizes the number of links used for routing the flow

$$\text{Minimize} \sum_{\forall i \in E} \text{lo}_i$$

Constraints: The constraints for D/SAP are as follows.

1) *Time-slot constraint:* Flow F shall be allocated exactly one time slot

$$\sum_{\forall j \in T} \text{so}_j = 1.$$

2) *Routing constraint:* The route for flow F starts at the source host and ends at the destination host(s), i.e., the number of destinations reachable over the outgoing links of the source host is equal to the number of destinations of the flow, while the number of destinations reachable over the incoming links of the destination hosts is 1. For all the other nodes in graph G, the sum of destinations reachable over incoming links is equal to the sum of destinations reachable over outgoing links

$$\begin{aligned}
&\sum_{\forall i \in \text{in}(s)} \text{dc}_i = 0, && \sum_{\forall i \in \text{out}(s)} \text{dc}_i = |D| \\
&\sum_{\forall i \in \text{in}(n)} \text{dc}_i = 1, && \sum_{\forall i \in \text{out}(n)} \text{dc}_i = 0\ \forall\, n \in D \\
&\sum_{\forall i \in \text{in}(n)} \text{dc}_i = \sum_{\forall i \in \text{out}(n)} \text{dc}_i && \forall\, n \in V \setminus (\{s\} \cup D).
\end{aligned}$$

3) *Collision avoidance constraint:* Flow F must be routed and allocated a time slot such that all network links are occupied by at most one flow at any given time slot

$$a_{i,j} + \text{es}_{i,j} \leq 1 \qquad \forall i \in E, \forall j \in T$$

4) *Auxiliary constraints:* For consistency among the ILP variables, the following constraint is required:

$$a_{i,j} = \text{lo}_i \times \text{so}_j \qquad \forall i \in E, \forall j \in T.$$

This nonlinear constraint is modeled by the following set of linear constraints:

$$\left.\begin{aligned} a_{i,j} &\leq \text{lo}_i \\ a_{i,j} &\leq \text{so}_j \\ a_{i,j} &\geq \text{lo}_i + \text{so}_j - 1 \end{aligned}\right\} \qquad \forall i \in E, \forall j \in T.$$

Additionally, the following constraint is required to model the relation between DC and LO. For a link i, $\text{dc}_i > 0$ implies that the link is used for routing, i.e., $\text{lo}_i = 1$.

$$\text{lo}_i \times |D| \geq \text{dc}_i \qquad \forall i \in E.$$

A feasible solution to this ILP provides a suitable schedule for flow F. The values of LO and SO determine the allocated time slot and route, respectively. In the absence of a feasible solution, flow F cannot be accommodated in the network.

2) Mini-Max (D/MM): Solely minimizing the number of links for routing a flow may result in some scenarios in formation of bottleneck links, which are occupied during all time slots. The second approach for online scheduling, Mini-max (D/MM), aims to distribute spare capacity over all links in the network to improve the prospects of successfully accommodating any future flow at the cost of routing some flows over longer paths. This ILP builds on the formulation of D/SAP by modifying its objective and adding a constraint. An additional decision variable, *MaxSlots*, is also added. It defines the maximum number of time slots during which any of the network links may be occupied. Thus, $\textit{MaxSlots} \in \mathbb{Z}^+$.

Here, the primary objective is the *minimization of the maximum number of time slots, during which any of the network links may be occupied*, hence the name *mini-max*. The secondary objective minimizes the number of links over which the flow is routed to weed out undesirable solutions that route the flow being scheduled over paths containing loops

$$\text{Minimize} \underbrace{\text{MaxSlots}}_{\text{Primary Objective}} + \underbrace{\frac{1}{|E|+1} \times \sum_{\forall i \in E} l_i}_{\text{Secondary Objective}}$$

The additional constraint in D/MM compared to D/SAP does not allow any network link to be occupied for a number of time slots exceeding MaxSlots, which is being minimized

$$\sum_{\forall j \in T} \text{es}_{i,j} + \text{lo}_i \leq \text{MaxSlots} \qquad \forall i \in E$$

Similar to D/SAP, LO and SO determine the route of the flow and the allocated time slot, respectively.

D. Determining the Base Period in the TSSDN

In the TSSDN, the base period determines the number of time slots available in the network for accommodating time-triggered flows. While higher values for the base period result in a larger number of time slots, the base period must also be restricted to values lower than or equal to the lowest transmission period that a flow may have in the network. Moreover, the choice of the base period is important as the periods of time-triggered flows

are constrained to be an integral multiple of the base period. For flows violating this condition, the period can be *reduced to the nearest multiple of the base period*. However, this results in an equivalent increase in the network load (packets per unit time) on account of scheduled traffic. For a given set of flows, *FS*, and a base period, *bp*, the network load due to scheduled flows is

$$\text{Scheduled traffic load} = \sum_{\forall f_k \in FS} \frac{1}{bp \cdot \lfloor \frac{p_k}{bp} \rfloor}.$$

The network load on account of the time-triggered flows in the TSSDN varies based on the selected value of the base period. To minimize the load due to scheduled traffic, the base period is determined by evaluating the load for scheduled traffic across different values of possible base periods. A reasonable value is chosen such that enough time slots are available for scheduling, while the load on account of scheduled traffic is also minimal. It must be noted that modifying the base period at runtime results in high coordination overhead. Reducing the base period would result in a reduction in the number of available time slots. Already allocated time slots may cease to exist, requiring a reallocation for the affected flows. The relative difference between the positions of the new and old time slots in the overall schedule, for instance, when the time slot for a flow is moved from the beginning of the schedule to the end, introduces jitter during the transition. Moreover, such transitions may also need rerouting of flows over paths of lengths differing from the initial paths contributing to the induced jitter. Hence, the TSSDN does not allow runtime modifications to the base period, which is calculated offline and remains fixed. For the dynamic scheduling problem, the base period is computed using a set of flows that are expected to be scheduled in the network. The computed base period would be suboptimal if flows not belonging to this expected set are scheduled.

E. Dynamic Scheduling of Flows With Different Periods

The presented approaches, for the static as well as the dynamic scheduling problem, assume that the transmission periods for all flows equal the base period. One time slot is allocated to each of the flows irrespective of their individual transmission period. While this suffices for most of the cases of the static scheduling problem, e.g., in ProfiNET [9], it is beneficial to also appropriately handle flows that have transmission periods higher than the base period in the dynamic scheduling problem. To avoid overallocation of capacity, flows with equal transmission periods can be phased, i.e., for every flow being scheduled, a *phase* is allocated along with a time slot, i.e., a ⟨*time slot*, *phase*⟩ pair is allocated. *Phase* indicates the validity of the *time slot* during a given cycle of the schedule. A flow can utilize its allocated time slot only if the allocated phase matches the corresponding cycle of the schedule. Two flows with equal transmission periods can be allocated the same time slot despite traversing overlapping paths so long as they are allocated *different phases*, thus ensuring temporal isolation for the flows. For instance, in Fig. 2, flows F_1 and F_2 with transmission periods of $2 \cdot bp$ can use the same time slot but in alternate cycles. Flow F_1 uses the time slot during *Phase 0*, while flow F_2 uses it during *Phase 1*.

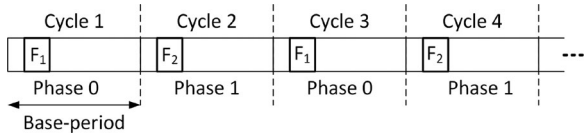

Fig. 2. Phasing of flows F_1 and F_2 with periods $2 \cdot bp$.

The total number of phases available for allocation on a link for a given time slot depends on the first flow that acquires this time slot and is routed to traverse the link. If a flow with transmission period $n \cdot bp$ is routed to traverse over link l_i at time slot t_j, then *n* phases ([0, ..., $n-1$]) are available on the (l_i, t_j) pair. A subsequent flow with period $n \cdot bp$ can use time slot t_j despite traversing link l_i provided that it is allocated a phase different from all the preceding flows.

The presented ILP formulations for the dynamic scheduling problem, D/SAP and D/MM, schedule flows over the set of time slots, T, allocating one time slot for each flow. To use phasing, a phase must be allocated to each of the flows. For this, an additional input is provided to the ILP formulation, namely the number of available phases, $P \equiv \{0, \ldots, n-1\}$, where $n \cdot bp$ is the period of the flow being scheduled. The ILP formulations must schedule over $(T \times P)$ instead of T and thus allocate a ⟨time slot, phase⟩ pair for each flow being scheduled. To this end, all references to T in the ILPs—D/SAP and D/MM—are replaced with $(T \times P)$. Additionally, the input *existing schedules*, ES, in the ILP formulations is redefined as $\text{ES} \equiv \{\text{es}_{i,(j,k)}\}$; $\forall i \in E, \forall \langle j,k \rangle \in (T \times P)$.

Here, $\text{es}_{i,(j,k)} = 0$, when no flow is scheduled to traverse link i at time slot j *or* the transmission period of the flow being scheduled is equal to the periods of flows traversing link i at time slot j, *and* none of these flows are allocated phase k. In all other cases, $\text{es}_{i,(j,k)} = 1$.

With these modifications, D/SAP and D/MM compute not only a time slot, but also a phase for the flow being scheduled, along with the route for the flow, and, thus, handle flows with different transmission periods.

F. Optimizations for Dynamic Scheduling Approaches

The presented approaches for dynamic scheduling, D/SAP and D/MM, have computational runtime in the order of several seconds (cf., Section V). The runtime mainly depends on the size of the topology ($|E|, |V|$), the number of available time slots ($|T|$), and the number of phases considered for scheduling ($|P|$). The following optimizations are introduced to reduce the execution time to the subsecond range.

1) Topology Pruning: With topology pruning, the ILPs consider only a subset of the network topology, $G' \equiv (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, for scheduling. To this end, the topology is pruned to remove network links along with nodes that do not play any role in routing a given flow, $F \equiv (s, D, p)$.

It may be observed that not all edge links (links connecting end systems to the network) play a role in routing flows. Only those edge links are used that connect the source and destination hosts of the flow with the network. The others may be removed without any impact on the computed solutions. Thus, the pruned topology is defined as follows.

Algorithm 1: Generate Time-Slot Slice, *TP*.

```
1:  function GenerateTP (T, P, G, SliceSize, Schedules)
2:      TP ← φ                          ▷ List of time slot, phase pairs
3:      for t in T do
4:          for ph in P do
5:                  ▷ Find number of free core links during ⟨t, ph⟩
6:              freeCoreLinks(G, Schedules, ⟨t, ph⟩)
7:          TP.append((t, getBestPhase()))
8:      ▷ Sort TP based on number of available core links
9:      TP ← sort(TP)
10:     return TP[0 : SliceSize]
```

1) $V' \equiv SW \cup \{s\} \cup D$, where SW is set of switches.
2) $E' \equiv \{(n_1, n_2) | (n_1, n_2) \in E;$
$(n_1, n_2 \in SW) \vee (n_1 = s) \vee (n_2 \in D)\}$.

2) Time-Slot Slicing: Another factor that affects the computational runtime is the number of time slots and phases over which the flow is to be scheduled. To reduce the execution time, only a subset of ⟨time slot, phase⟩ pairs referred to as time slot slice, $TP \subseteq (T \times P)$, may be considered.

Using of time slot slicing may lead to suboptimal solutions (longer paths in the case of D/SAP, or paths resulting in bottleneck links in case of D/MM) leading to fewer flows being accommodated overall. Worse, this may also introduce false negatives, i.e., the scheduler is unable to allocate a suitable ⟨time slot, phase⟩ pair and corresponding route for the flow as all the possible solutions lie in $(T \times P) \setminus TP$. It is thus imperative that *TP* is constructed such that the resulting suboptimality and number of false negatives are minimized. For this, the number of unoccupied core links (links between two switches) during a given ⟨time slot, phase⟩ pair is used as a heuristic to generate *TP* from $(T \times P)$. A higher number of unoccupied core links during a ⟨time slot, phase⟩ pair improves the prospects of yielding a solution to the scheduling problem. For instance, if all core links are unoccupied during a ⟨time slot, phase⟩ pair, it is certain that the optimal solution will be available, provided that the necessary edge links (connecting source and destinations of the flows to the network) are also unoccupied during the corresponding ⟨time slot, phase⟩ pair.

Using the aforementioned heuristic, the time slot slice, *TP*, of size *SliceSize* can be computed for scheduling as shown in Algorithm 1. The algorithm computes the heuristic for all $\langle t, ph\rangle \in (T \times P)$ (Lines 3–6). For each $t \in T$, a phase, ph, that provides the best chance for the scheduler to compute the optimal solution is shortlisted (Line 7). Phase ph must have the maximum number of unoccupied core links during the corresponding time slot, t. Furthermore, during $\langle t, ph\rangle$, all the edge links required for routing the particular flow, i.e., the links connecting the source and destination hosts to the network, must also be unoccupied. From all the shortlisted $\langle t, ph\rangle$ pairs, the best *SliceSize* number of pairs in terms of the computed heuristic form the time slot slice (Lines 9–10).

Here, the parameter *SliceSize* determines the execution time for scheduling along with the introduced suboptimality and false negatives. Low values of *SliceSize* result in lower execution time, but at the cost of an increased number of suboptimally routed flows and a higher number of false negatives. To reduce false negatives, scheduling can be reattempted with an increased value of *SliceSize*. In such cases, those ⟨time slot, phase⟩ pairs that did not yield any solution in the previous attempts must be excluded. Together with topology pruning, time-slot slicing realizes subsecond runtime for the dynamic scheduling algorithms in the TSSDN.

G. Network Utilization With the TSSDN

The scheduling model in the TSSDN allocates a time slot to each flow such that the entire path over which the flow traverses is exclusively reserved for it. The time slots are, hence, required to be long enough for guaranteeing this even for the worst case, i.e., traversal of an MTU-sized packet over the network diameter. Ideally, it would be sufficient to reserve time slots over network links, where the length of time slots is limited to the time required for transmitting the actual size of the packet plus some guard bands to isolate the scheduled traffic from best-effort traffic, e.g., using IEEE 802.1Qbv[18], [19].

The coarse-grained scheduling model of the TSSDN results in lower available bandwidth for scheduled traffic in comparison to the fine-grained scheduling in the ideal case. However, the TSSDN, being work-conserving in nature, makes available the remaining bandwidth for best effort traffic. Given that a sufficient amount of best-effort traffic is available in the network, the TSSDN can achieve full network utilization. In contrast, with fine-grained link scheduling (non-work-conserving in nature), like in IEEE 802.1Qbv networks, bandwidth is lost due to the presence of guard bands or when reserved time slots are not used by time-sensitive traffic, e.g., in event-based communication. During the guard bands, new packets are not taken up for transmission, even if the corresponding link is idle, to avoid interference with the next time slot for the scheduled traffic. Usually, the guard bands are configured to be long enough for a MTU-sized packet to be transferred over the corresponding link. The number of guard bands required and the resulting drop in network utilization is dependent on the topology of the network and the used scheduling approach [18], [19].

In absence of best-effort traffic, the network utilization would drop significantly for the TSSDN. The worst case for the TSSDN would be a line topology with six switches handling only scheduled traffic. Remember that the TSSDN restricts the network diameter to seven hops. In this case, the network utilization would drop to approximately one-seventh of what would be possible with fine-grained link scheduling.

It may be noted that the reservation for time-triggered traffic in the TSSDN does affect the best-effort traffic. However, transport protocols like transmission control protocol (TCP) will adapt to the available bandwidth automatically. A detailed quantitative analysis along with concepts to minimize the impact on best-effort traffic throughput is a subject for future work.

V. Evaluations

This section presents the results of the evaluations for the scheduling approaches in the TSSDN. The evaluations are

mainly focused on the computational time and quality of the dynamic scheduling approaches in comparison to the static scheduling approaches in the TSSDN. This paper refers to the existing literature for evaluations on the interaction between the data plane and the control plane in SDN, in particular, the insertion of flow-table entries in switches and the data plane performance of the TSSDN [13], [25].

The evaluations use CPLEX(v 12.5), a commercial ILP solver from IBM, for computing the schedules and routes by solving the presented ILPs. The ILPs were specified using PuLP, a Python-based tool kit to specify ILPs. The scheduling approaches were evaluated against a range of network topologies (various sizes and network models) created using NetworkX, a Python library for creating complex networks. In detail, topologies were created using the Erdős–Rényi (ER) model, random regular graphs (RRG), the Barabási-Albert (BA) model, and the Waxman model. Altogether, these graph models comprehensively test the scheduling approaches. The sizes of the topologies, the number of time slots, and the flows used as input are specified with the concrete evaluations.

A. Qualitative Comparison

For a given set of flows with transmission period equal to the base period, S/UR yields the best schedule in terms of the number of scheduled flows. Hence, S/UR is used as a benchmark for comparing the scheduling approaches.

For comparison, evaluations were carried out in 160 scenarios created using eight different topologies (three RRG, two ER, and three BA), each consisting of 20–110 unicast flows whose transmission periods were equal to the base period. These flows were scheduled over three to five time slots. The high runtime of S/UR restricted the evaluations to small scenarios only. The flow schedules were calculated using all the presented approaches in each of the scenarios. For the static scheduling approaches, all flows were collectively considered, while for the dynamic scheduling approaches, the flows were considered *one at a time* in a random order. As evaluation metric, the relative quality of the approach was determined with respect to S/UR, i.e., the ratio of number of flows scheduled by the approach to the number of flows scheduled using S/UR.

Fig. 3(a) presents the cumulative distribution of the relative quality for the evaluated scenarios. As expected, the static scheduling approaches yield better relative quality compared to the dynamic scheduling approaches. Overall, the solutions generated by S/FR and S/PR closely approximate solutions computed using S/UR with average relative qualities of 99% and 97%, respectively. Furthermore, S/PR and S/FR result in solutions with relative qualities of at least 92% and 84% or better, respectively. This drops to 72% and 64% for D/SAP and D/MM, respectively. On an average, the dynamic scheduling approaches produce solutions with a relative quality of 68% compared to the S/UR. It is, thus, imperative to ensure that the optimizations to reduce the runtime of the dynamic scheduling approaches do not degrade their quality further.

It is also worth to note that the evaluations show D/SAP to be slightly better than D/MM. Of the 160 evaluation scenarios, D/MM outperformed D/SAP in 30 scenarios with respect to the number of scheduled flows, D/SAP was better than D/MM in 49 scenarios, while in 81 scenarios, both of them scheduled an equal number of flows. The is because D/MM occasionally routes flows over longer paths resulting in consumption of time slots over a higher number of links.

B. Execution Time

The execution time of the scheduling approaches is mainly affected by the topology size and the number of available time slots for scheduling. Fig. 3 presents the evaluation results detailing the impacts of these parameters on the execution time, based on evaluations on a machine with Intel Xeon E5-1650 processor (2×6 cores) and 16-GB RAM.

Fig. 3(b) shows the impact of the number of available time slots on the execution time (amortized over one flow) for all the scheduling approaches. The figure shows the time to schedule 300 flows with periods equal to the base period on a Waxman topology (256 network links) for a varying number of time slots (5–50). The results show that the runtime of S/PR increase rapidly with the number of available time slots, requiring about 25 s per flow with 50 time slots. S/FR requires less than 4 s per flow for scheduling with 50 time slots. It is worth noting that the TSSDN provides only about 50 time slots for a base period of 1 ms (considering an MTU size of 1500 bytes). The dynamic scheduling approaches without the proposed optimizations require over 80 s to schedule a flow. In contrast, with optimizations, the approaches could determine the schedule in less than a second. Furthermore, to evaluate the effect of the network size, the time for scheduling over 100 flows on topologies of different sizes (30–256 network links) was measured. The results [see Fig. 3(c)] show a trend similar to results with varying time slots. The static scheduling approaches fare better than the dynamic scheduling approaches with an amortized cost of 12 s for S/PR to schedule a flow. The evaluations also show that the runtime of S/PR is dependent on the number of paths available for routing, i.e., the path diversity in the network. On the other hand, the runtime of dynamic scheduling approaches (without optimization) increases steeply with the size of the topology, with about 75 s runtime to schedule one flow for a topology with 256 links. Clearly, the raw ILPs are not practical for online scheduling without further optimizations. The evaluations show that the execution times of D/SAP and D/MM are similar. Furthermore, the execution time is independent of the type of the flow(s) being scheduled, unicast or multicast. In the case of multicast flows, the number of destinations also does not have an impact on the execution time.

The evaluations also showed that the computation of the base period for the flows expected to be scheduled in the network increases linearly with the number of flows. As shown in Fig. 3(e), the base period for up to 10^5 flows in the TSSDN can be determined in 150 s on commodity machines. Thus, it is reasonable to compute the base period for the flows upfront.

The dynamic scheduling approaches are also affected by the transmission period of the flow being scheduled. Due to use of phasing, flows with larger transmission periods have more options to be accommodated in the network schedule. Fig. 3(d) shows the impact that this has on the runtime (along with their standard deviations) to schedule 100 flow incrementally on an

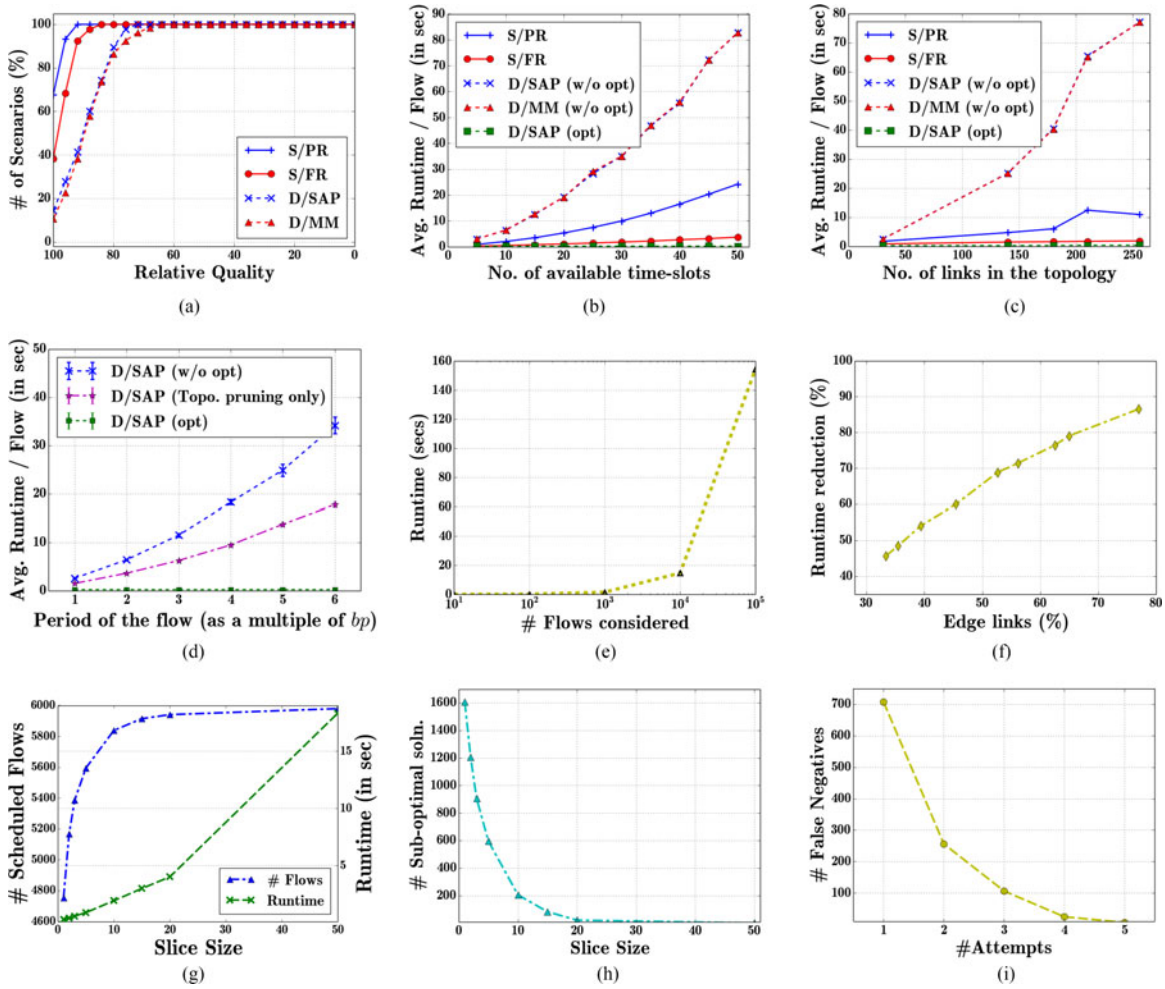

Fig. 3. Evaluations results for the ILP formulations presented in Section IV. (a) Relative quality to S/UR. (b) Runtime versus number of slots. (c) Runtime versus number of links. (d) Impact of optimizations on D/SAP. (e) Base-period computation. (f) Impact of topology trimming. (g) Time-slot slicing. (h) Slice size versus suboptimality. (i) Number of attempts versus false negatives.

ER topology (150 links). In absence of time-slot slicing, flows with larger transmission periods resulted in longer execution time with an increasing standard deviation. With time-slot slicing, the transmission period ceases to have an effect on the execution time which is then achieved in subseconds.

C. Impact of Optimizations on Dynamic Scheduling

This paper presented two optimizations for reducing the execution time of the dynamic scheduling approaches. The first of them is topology pruning to remove unnecessary edge links. To evaluate its impact, the execution time for scheduling flows using D/SAP and D/MM on several topologies (RRG, ER, and BA) was measured with and without topology pruning. As shown in Fig. 3(f), with an increasing proportion of edge links in the network, the reduction of runtime achievable also increases substantially. For instance, in a network with over 75% edge links, topology pruning results in 86% reduction of execution time for the scheduler, from 7.4 to 1 s. However, in realistic networks, the proportion of edge links is lower, for instance, in a fat-tree topology (found in data-centers), only 33% of the total network links are edge links. Here, topology pruning provides about 45% reduction in execution time.

While topology pruning is qualitatively nondestructive, time-slot slicing may result in suboptimal schedules or generates false negatives. It is, thus, not straightforward to evaluate the impact of time-slot slicing. For this, 10 000 time-triggered flows were scheduled on an ER topology (500 hosts and 20 switches with 131 core links) over 50 time slots. For each of these flows, the schedule was computed with and without time-slot slicing (using D/SAP and topology pruning) to evaluate its impact on optimality of the computed schedules and determine false negatives. Fig. 3(g)–(i) summarizes the results of this evaluation. Fig. 3(g) shows that the execution time of the scheduler increases linearly with the size of the time-slot slice with runtimes lower than 1 s for slice sizes less than 5. However, the number of flows scheduled in the network does not commensurately increase with it. The evaluations also show that the achieved reduction in the number of generated false negatives and suboptimal solutions is not substantial with an increase in the size of time-slot slices beyond a certain degree [cf., Fig. 3(h) and (i)]. Overall, *with a slice size of just three to five time slots, subsecond setup time is achieved for scheduling flows*. Of the total scheduled flows,

only 906 and 595 flows were suboptimally routed with slice size of 3 and 5, respectively. Furthermore, the number of generated false negatives also reduced substantially by reattempting to schedule the flow using the next best time-slot slice, in this case, from 709 to 106 by allowing just two additional attempts to schedule the flow in case a suitable schedule could not be computed. After deploying the optimizations (topology pruning and time-slot slicing with a time-slice size of 5 with up to three scheduling reattempts), the overall number of flows scheduled in the network decreases by less than 4% only.

D. Evaluation Summary

Overall, the evaluations show the following.

1) S/UR yields the best schedules in terms of the number of flows scheduled for the TSSDN and serves as a benchmark for other heuristic solutions.
2) S/PR and S/FR closely approximate the solutions generated by the S/UR with runtime that is orders of magnitude lower.
3) The dynamic scheduling approaches, D/SAP and D/MM, with optimizations can compute online schedules for time-triggered flows in subseconds.

VI. Conclusion and Future Work

This paper presented incremental scheduling algorithms for time-triggered flows in the TSSDN, a network architecture that provides real-time guarantees to the time-triggered flows using capabilities of SDN. These algorithms formulate the scheduling problem using ILP and compute transmission schedules for static and dynamic scenarios. Overall, the TSSDN achieves subsecond setup times for scheduling arbitrary time-triggered flows dynamically on account of the fast online scheduling algorithms and lightweight mechanisms for updating schedules.

References

[1] P. Neumann, "Communication in industrial automation—What is going on?" *Control Eng. Pract.*, vol. 15, no. 11, pp. 1332–1347, 2007.

[2] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered ethernet (TTE) design," in *Proc. 8th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput.*, 2005, pp. 22–33.

[3] E. Tovar and F. Vasques, "Real-time fieldbus communications using profibus networks," *IEEE Trans. Ind. Electron.*, vol. 46, no. 6, pp. 1241–1251, Dec. 1999.

[4] Deterministic Networking. 2017. [Online]. Available: https://datatracker.ietf.org/wg/detnet/charter/

[5] M. Johas Teener *et al.*, "Heterogeneous networks for audio and video: Using IEEE 802.1 audio video bridging," *Proc. IEEE*, vol. 101, no. 11, pp. 2339–2354, Nov. 2013.

[6] F. Dürr and T. Kohler, "Comparing the forwarding latency of openflow hardware and software switches," Univ. Stuttgart, Stuttgart, Germany, Tech. Rep. Comput. Sci. 2014/04, Jul. 2014.

[7] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Proc. 31st IEEE Real-Time Syst. Symp.*, Nov. 2010, pp. 375–384.

[8] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Syst.*, vol. 52, pp. 161–200, Mar. 2016.

[9] Z. Hanzalek, P. Burget, and P. Sucha, "Profinet IO IRT message scheduling with temporal constraints," *IEEE Trans. Ind. Informat.*, vol. 6, no. 3, pp. 369–380, Aug. 2010.

[10] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study," *IEEE Access*, vol. 5, pp. 9882–9910, 2017.

[11] L. Monostori *et al.*, "Cyber-physical systems in manufacturing," *CIRP Ann., Manuf. Technol.*, vol. 65, no. 2, pp. 621–641, 2016.

[12] R. S. Oliver, S. S. Craciunas, and G. Stöger, "Analysis of deterministic ethernet scheduling for the industrial internet of things," in *Proc. 19th IEEE Int. Workshop Comput. Aided Model. Des. Commun. Links Netw.*, Dec. 2014, pp. 320–324.

[13] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (TSSDN) for real-time applications," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, Oct. 2016, pp. 193–202.

[14] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic*, IEEE Standard 802.1Qbv-2015, Mar. 2016, pp. 1–57.

[15] Y. H. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "RT-WiFi: Real-time high-speed communication protocol for wireless cyber-physical control applications," in *Proc. IEEE 34th Real-Time Syst. Symp.*, Dec. 2013, pp. 140–149.

[16] L. Chiaraviglio *et al.*, "Optimal superfluid management of 5G networks," in *Proc. IEEE Conf. Netw. Softwarization*, Jul. 2017, pp. 1–9.

[17] C. Scholer, R. Krenz-Baath, A. Murshed, and R. Obermaisser, "Computing optimal communication schedules for time-triggered networks using an SMT solver," in *Proc. 11th IEEE Symp. Ind. Embedded Syst.*, May 2016, pp. 83–91.

[18] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, Oct. 2016, pp. 183–192.

[19] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, Oct. 2016, pp. 203–212.

[20] N. McKeown *et al.*, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[21] L. Rizzo, "Netmap: A novel framework for fast packet I/O," in *Proc. 21st USENIX Security Symp.*, Aug. 2012, pp. 101–112.

[22] T. Kohler, F. Dürr, and K. Rothermel, "Consistent network management for software-defined networking based multicast," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 447–461, Sep. 2016.

[23] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 26: Frame Preemption*, IEEE Standard 802.1Qbu-2016 (Amendment to IEEE Standard 802.1Q-2014), 2016, pp. 1–52.

[24] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[25] K. He *et al.*, "Measuring control plane latency in SDN-enabled switches," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, 2015, Art. no. 25.

[26] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks," *IET Cyber-Phys. Syst.: Theory Appl.*, vol. 1, no. 1, pp. 86–94, 2016.

Naresh Ganesh Nayak is currently working toward the Doctoral degree with the Distributed Systems Department, University of Stuttgart, Stuttgart, Germany.

His research interests include time-sensitive networking and software-defined networking.

Frank Dürr received the Doctoral and Diploma degrees in computer science from the University of Stuttgart, Stuttgart, Germany in 2010 and 2001, respectively.

He is a Senior Researcher and Lecturer with the Distributed Systems Department, Institute of Parallel and Distributed Systems, University of Stuttgart. His research interests include software-defined networking and time-sensitive (real-time) networking, mobile and pervasive computing, privacy, and cloud computing aspects overlapping with these topics such as mobile cloud computing, edge-cloud computing, or industrial and datacenter networks.

Kurt Rothermel received the Doctoral degree in computer science from the University of Stuttgart, Stuttgart, Germany, in 1985.

Since 1990, he has been a Professor in computer science with the University of Stuttgart. From 2003 to 2011, he was the Head of the Collaborative Research Center Nexus (SFB 627), conducting research in the area of mobile context-aware systems. He is the Director of the Institute of Parallel and Distributed Systems, University of Stuttgart. His current research interests include distributed systems, computer networks, and mobile systems.