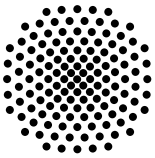


Technical Report 2016/03

Time-sensitive Software-defined Networks for Real-time Applications

Naresh Ganesh Nayak
Frank Dürr
Kurt Rothermel

Institute of Parallel and Distributed Systems
University of Stuttgart

Universittsstr. 38
705679 Stuttgart
Germany

May 2016

Time-sensitive Software-defined Networks for Real-time Applications

Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel

Institute of Parallel and Distributed Systems
University of Stuttgart, Germany

Abstract

Cyber-physical systems (CPS), like the ones used in industrial automation systems, are highly time-sensitive applications demanding zero packet losses along with stringent real-time guarantees like bounded latency and jitter from the underlying network for communication. With the proliferation of IEEE 802.3 and IP networks, there is a desire to use these networks instead of the currently used field-buses for time-sensitive applications. However, these networking technologies, which originally were designed to provide best effort communication services, lack mechanisms for providing real-time guarantees. In this paper, we present Time-Sensitive Software-Defined Networks (TSSDN), which provide real-time guarantees for the time-triggered traffic in time-sensitive systems while also transporting non-time-sensitive traffic. TSSDN provides these guarantees by bounding the non-deterministic queuing delays for time-sensitive traffic. To this end, TSSDN exploits the logical centralization paradigm of software-defined networking to compute a transmission schedule for time-sensitive traffic initiated by the end systems based on a global view. In particular, we present various Integer Linear Program (ILP) formulations for computing high-quality transmission schedules. Moreover, we show that end systems can comply with a given schedule with high precision using user-space packet processing frameworks. Our evaluations show that TSSDN has deterministic end-to-end delays ($\leq 14\,\mu$s on our benchmark topology) with low and bounded jitter ($\leq 7\,\mu$s) for packets of size 1500 bytes transmitted with a frequency of 10 kHz.

1 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 to and from the CPS controllers, respectively. Typically, such CPS are time-sensitive systems where the network delay (including the delay stemming from packet loss) and jitter impacts the quality of control of the CPS. For instance, machines in automotive shop floors might fail, when two consecutive packets are lost [3]. Another example from industrial automation are isochronous motion control systems, which require extremely bounded jitter in the order of microseconds for stability [30]. Many

more examples of time-sensitive CPS can be found in the area of Industry 4.0, tele-robotics, smart grid, etc.

Consequently, in order to ensure a deterministic behaviour of CPS, *deterministic real-time* networks with bounded delay and delay variance (jitter) are desirable. Traditionally, such guarantees have been provided by dedicated fieldbus networks. However, with the proliferation and steadily growing performance along with shrinking costs of IEEE 802.3 and IP networks, there is a strong desire to also utilize these technologies, initially designed to provide best-effort communication services, also for implementing time-sensitive CPS. Ideally, both, time-sensitive and non-time-sensitive applications, should be able to communicate over one converged IP-based IEEE 802.3 network.

The requirement to provide deterministic real-time networks is also in the focus of two major standards bodies in networking, namely IETF and IEEE. The IEEE 802.1 AVB Task Group, now rechristened as the Time-Sensitive Networking (TSN) Task Group (TG) [24][11], which develops standards for time-synchronized low latency streaming services for 802 networks, and the IETF DetNets Working Group [4], which targets time-sensitive communication over Layer 2 bridged and Layer 3 routed networks.

Looking at the initial discussions of these groups, we can identify the elimination of non-deterministic queuing delays in network elements as an essential requirement to achieve deterministic network delay and jitter for time-sensitive traffic. This effectively also eliminates packet loss occurring due to overflowing queues. One basic concept targeting highly time-sensitive periodic communication—e.g., a constant bit-rate sensor data stream—in local area networks (LAN) is to schedule the transmission of packets at the end systems using *time-triggered communication*. This concept leverages the possibility to precisely synchronize clocks of hosts using time synchronization protocols like the IEEE 1588 [8] Precision Time Protocol (PTP). Packets can then be assigned to time-slots based on a global transmission schedule such that in-network queuing is avoided. Additionally, time-sensitive traffic is assigned the highest priority in the network to isolate it from non-time-sensitive traffic.

Although this concept of time-triggered communication is well-known [26][31], several challenges remain, which we target in this paper. Firstly, we need suitable algorithms to compute a global transmission schedule. These algorithms should assign time-slots to time-sensitive communication flows such that in-network queuing is avoided (constraint) while maximizing the number of such flows in the network (optimization objective). Thus, we need to solve a constrained optimization problem to compute a transmission schedule. Secondly, we need to ensure that the end systems comply with the calculated schedules precisely enough to avoid queues on switches. Adverse effects that need to be considered are the variable delays of the network stack on the end systems, imperfect clock synchronization and timers, and the inevitable varying network delay of packets following network paths of different length.

To facilitate the calculation of schedules, we utilize software-defined networking (SDN), an emerging networking paradigm that enables deploying network applications executing centralized algorithms with a global view onto the network. To this end, we introduce a network controller, with a global view onto all time-sensitive flows (we refer to all packets belonging to a given stream as a flow) and the network topology, to compute the transmission schedules. This logically centralized architecture of our *Time-sensitive Software-defined Network*

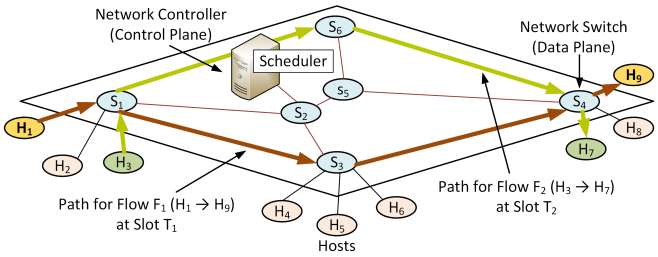


Figure 1: TSSDN. Network controller routing flows F_1 & F_2 and allocating them slots T_1 & T_2, respectively

(TSSDN) is also consistent with the initial architecture of the IETF DetNet WG, which considers logical centralization as one promising option. Further, to ensure schedule adherence by the hosts, we use user-space packet processing frameworks.

In detail, we make the following contributions:

- We introduce the scheduling problem in TSSDN (an NP-hard problem) and propose various Integer Linear Program (ILP) formulations to compute high-quality transmission schedules. These ILP formulations leverage the efficiency of modern ILP solvers and lend themselves well to a logically centralized implementation.
- We present a proof-of-concept implementation showing that the source hosts of time-sensitive flows can accurately comply with the computed transmission schedule using user-space packet processing frameworks.
- We show through evaluations that our ILP formulations can generate transmission schedules for networks of realistic sizes within seconds. Moreover, we show that adherence to these schedules results in deterministic network delays of $\leq 14\,\mu$s on our benchmark topology with ultra-low jitter ($\leq 7\,\mu$s) for packets-sizes of 1500 bytes transmitted with a frequency of 10 kHz.

The remaining paper is structured as follows. In Section 2, we present the system model of TSSDN and the corresponding problem statement. In Section 3, we present the ILP formulations to solve the scheduling problem in TSSDN. We present the usage of packet processing frameworks for schedule adherence in Section 4. We evaluate our work and present the related work in Section 5 and 6 respectively. Finally, we conclude in Section 7.

2 System Model & Problem Statement

In this section, we present the system model of TSSDN and the problem statement corresponding to its scheduling.

2.1 System Model

Our TSSDN (cf. Fig. 1) is based on the principles of software-defined networking (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 packet forwarding. 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 network elements to a network controller (not to be confused with the CPS controller) hosted on standard servers and communicating with the switches through a so-called southbound interface like the OpenFlow protocol [27]. The network controller is logically centralized, i.e., it has a global view onto the network elements, topology, traffic, etc., which facilitates the implementation of network control logic (routing, schedule configuration). Note that the logically centralized SDN controller can be physically distributed to several servers to increase scalability and availability. However, in this paper we do not consider the problem of control plane distribution.

Besides the network controller and switches, our system consists of end systems (hosts). End systems execute userspace processes, which are the sources and destinations for time-sensitive traffic. 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. We assume that the sources of time-sensitive traffic unicast packets with a constant bit-rate to the destination with time-periods that are an integral multiple of a "base-period", the minimum transmission period that can be supported. A time-triggered pattern is well suited for using sensors with fixed sampling periods or actuators requiring inputs within given time intervals. Time-sensitive traffic is transmitted in high priority UDP packets. Priority mechanisms like IEEE 802.1Q (VLAN) or Differentiated Services (DiffServ) can be used for this. We assume that every application layer data unit (e.g., sensor sample or actuator command) fits into a single maximum transfer unit (MTU) sized UDP packet. Further, we assume that all end systems have precisely synchronized clocks using the Precision Time Protocol (PTP). In this paper, we limit the maximum network diameter, e.g., to 8 hops between any pair of hosts, which is consistent with the IEEE 802.1D standard, i.e., we restrict our approach to local area networks. It may be noted that all time-sensitive traffic has the same priority, and, thus, an additional scheduling mechanism is required to handle conflicting time-sensitive traffic, which is the basic focus of this paper.

2.2 Problem Statement

The goal of TSSDN is to achieve deterministic network behavior with bounds on network delay and jitter for time-sensitive traffic to support real-time communication. Network delay comprises propagation delay, processing delay, transmission delay, and queuing delay. Propagation delay in a LAN with predefined maximum diameter is bounded, thus, deterministic, and very small (order of nanoseconds). Moreover, our measurements with commodity Ethernet switches have shown that their processing delays are in the range of microseconds or below and almost constant for a given set of matching tuples [19]. Thus, processing delay can also be considered to be deterministic. The transmission delay is also bounded and deterministic for constant bit-rate traffic. Therefore, the challenge for TSSDN is to bound the non-deterministic queuing delay for time-sensitive traffic.

Generally, queuing occurs in switches when packets from multiple input ports

attempt to transmit over the same output port simultaneously [21]. Queuing can be eliminated if no two inputs ports contend for transmitting over the same output port, i.e., the source host initiates transmission only when the entire network path over which the flow traverses is exclusively reserved for it. For instance, in the topology shown in Fig. 2, simultaneous transmissions of packets belonging to flows $F_i : A_i \rightarrow B_i; i \in [1 \ldots 5]$ will result in queuing at the output port of switch S_1. In such cases, the network delay for these packets would depend on the length of queues they encounter, i.e., the flows are affected by jitter.

TSSDN bounds queuing delays by scheduling the transmission of time-sensitive packets on source hosts in addition to routing them such that they will always find an empty queue (for high priority traffic) on each switch along their path, i.e., it isolates time-sensitive flows either spatially or temporally. To this end, we implement a time-triggered time division multiple access (TDMA) scheme, where every time-sensitive flow has well-defined time-slots allocated by the network controller during which its source can transmit. The scheduling algorithm in TSSDN uses its global knowledge of the network topology and the time-sensitive flows gathered using the southbound interface to define a suitable transmission schedule and route for all time-sensitive flows. For instance, in the benchmark topology shown in Fig. 2, each of the flows, F_i, is allocated a different time-slot to sufficiently skew their transmissions and avoid queuing on the bottleneck link (from switch S_1 to S_2).

In the following, we address two pressing problems with respect to scheduling in TSSDN. Firstly, *how to compute a transmission schedule that maximizes the number of scheduled time-sensitive flows?* By maximizing the time-sensitive flows that can be carried over the network, a larger number of real-time applications can be supported. This maximization problem is similar to the static light path establishment problem encountered during routing and wavelength assignment in optical networks [16]. In optical networks, light connections are established between nodes by allocating wavelengths for communication. However, light connections traversing over a given optical link cannot share the same wavelength. Analogously, time-sensitive flows with overlapping paths cannot share the same time-slot. The decision problems with respect to minimizing

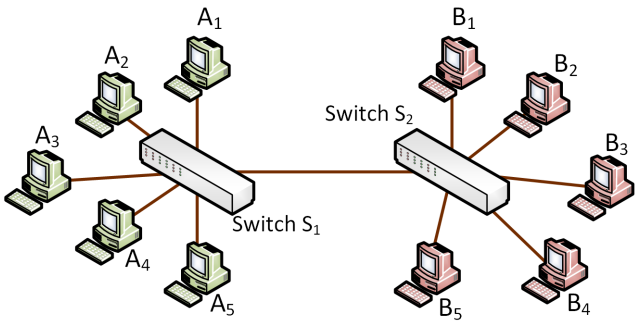


Figure 2: Benchmark topology with 10 hosts (A_1–A_5, B_1–B_5) connected to 2 switches (S_1 and S_2) with 10 Gbps links and 5 time-sensitive flows ($F_i : A_i \rightarrow B_i$; $i \in [1 \ldots 5]$)

the number of wavelengths used in an optical network or maximizing the number of established light connections are proven to be NP-complete [13]. In this paper, we present scheduling algorithms in the form of ILP formulations that compute transmission schedules for a static set of time-sensitive flows known a priori.

Secondly, *how can the source hosts precisely comply with a given transmission schedule?* The communication primitives offered by the operating systems, for instance the socket API's, are inadequate for source hosts to comply with a given transmission schedule with sufficient precision. They introduce non-deterministic delays in the network stack of the end systems that render the computed schedule useless. In Section 4, we present the usage of userspace packet processing frameworks for precisely adhering to a given transmission schedule.

3 Transmission Scheduling in TSSDN

3.1 Overview

The transmission schedule in TSSDN is modelled as a cyclic schedule of duration equalling the base-period, as shown in Fig. 3. It is divided into smaller time-slots, numbered from 0 to t_{max}, each wide enough for an MTU-sized packet to travel across the longest network path (restricted to 8 hops). The network controller can determine t_{max} based on the base-period and slot length, both being system parameters. The scheduler disburses time-slots, $T \equiv \{0, 1 \ldots, t_{max}\}$, to the sources of time-sensitive flows while also routing them. To avoid queuing, the scheduler is restrained from allocating the same time-slot to multiple flows that have overlapping paths. The sources then compute the exact transmission instants using the base-period, the slot length, its own sending period, and the allocated time-slot (cf. Section 4). The sources utilize their slots based on their individual period, for instance, a flow with period twice the base-period will use its slot in alternate cycles only. If a suitable slot is unavailable, then the source is prohibited from sending time-sensitive traffic.

In the following, we present three ILP formulations with varying degrees of constraints on routing for computing transmission schedules for TSSDN. The first one, *Scheduling with Unconstrained Routing* (S/UR), allows the ILP solver to explore all possible paths for routing the time-sensitive flows. The subsequent formulations, *Scheduling with Pathsets Routing* (S/PR) and *Scheduling with Fixed-path Routing* (S/FR), restrict the possibilities for routing the flows, thereby reducing the execution times, while compromising on the quality of the schedules in terms of number of scheduled flows.

3.2 Scheduling with Unconstrained Routing (S/UR)

In this approach, the scheduler is free to route the time-sensitive flows over any available path. The network topology and the set of desired time-sensitive flows are the inputs. Variables are the time-slots and paths for the flows. The optimization objective is to *maximize the number of flows that are allocated a time-slot.*

For the ILP formulations, we denote the network topology as a directed

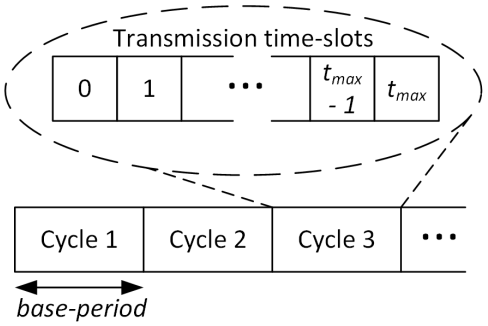


Figure 3: Transmission schedules in TSSDN

graph $G \equiv (V,E)$, where 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. Further, $V \equiv (S \cup H)$, where S and H are sets of switches and end systems, respectively. The time-sensitive flows are denoted as a tuple, $ts_i \equiv (s_i,d_i)$, where $s_i,d_i \in H$. Here, s_i and d_i is the source and the destination of the flow, respectively. The set of time-slots available for disbursement is denoted as $T \equiv \{0,1\ldots,t_{max}\}$. Additional functions needed to model the topology and time-sensitive flows are listed in Table 1.

3.2.1 ILP Inputs

The inputs required for the ILP formulation are as follows:

- Network Topology, $G \equiv (V,E)$.
- Set of time-sensitive flows to be scheduled, TS.
  $TS \equiv \{ts_i\};\ i \in [1\ldots NumFlows]$
  Here $NumFlows$ represent the number of flows to be scheduled.

3.2.2 ILP Variables

The decision variables used for formulating the ILP are as follows:

- Mapping of flows to network links, SL.
  $SL \equiv \{f_{i,j}\}\ \forall i \in TS,\ \forall j \in E$
  $f_{i,j} = 1$, if the flow i traverses over link j, else 0.

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(ts)	Flow ts, $ts \equiv (s,d)$	s
dst(ts)		d

Table 1: Helper functions for modeling network topology and time-sensitive flows

- Mapping of flows to time-slots, ST.
  $ST \equiv \{t_{i,k}\}\ \forall i \in TS, \forall k \in T$
  $t_{i,k} = 1$, if the flow i is allocated time-slot k, else 0.

- Helper variables, SLT. These enable the formulation of the scheduling problem as an ILP.
  $SLT \equiv \{m_{i,j,k}\}\ \forall i \in TS, \forall j \in E, \forall k \in T$
  $m_{i,j,k} = 1$, if the flow i traverses over link j and is allocated a time-slot k, else 0.

3.2.3 Objective Function

The objective function is formulated so as to primarily maximize the number of flows that are allocated time-slots. In some situations different solutions might exist with the same number of scheduled flows but where some solutions contain loops in their paths. Obviously, in such cases we would prefer the solutions without loops. Therefore, we define a secondary objective for weeding out solutions that route flows over paths with loops. This term in the objective keeps the path length at a minimum, thus, eliminating paths with loops, and is factored such that its total contribution to the objective is less than one. This ensures that the ILP solver gives priority to maximizing the number of flows that can be scheduled rather than minimizing the length of the individual paths allocated to them.

Maximize:

$$\underbrace{\sum_{\forall i \in TS} \sum_{\forall k \in T} t_{i,k}}_{\text{Primary Objective}} - \underbrace{\frac{1}{(|TS| \times |E|) + 1} \times \sum_{\forall i \in TS} \sum_{\forall j \in E} f_{i,j}}_{\text{Secondary Objective}}$$

3.2.4 Constraints

The constraints for the ILP formulation are as follows:

- Every flow shall be allocated at most one time-slot as they carry only one MTU-sized packet during their corresponding period.

$$\sum_{\forall k \in T} t_{i,k} \leq 1 \qquad \forall i \in TS$$

- The path for a given flow, i, starts at its source host and ends at its destination host, i.e., the source host has only one outgoing link with no incoming links while the destination host has one incoming link with no outgoing links. For all the other network nodes, the number of incoming links is equal to the number of outgoing links.

$$\sum_{\forall j \in in(src(i))} f_{i,j} = 0 \qquad \sum_{\forall j \in out(src(i))} f_{i,j} = 1$$

$$\sum_{\forall j \in in(dst(i))} f_{i,j} = 1 \qquad \sum_{\forall j \in out(dst(i))} f_{i,j} = 0$$

$$\sum_{\forall j \in in(n)} f_{i,j} = \sum_{\forall j \in out(n)} f_{i,j} \quad \forall n \in V \setminus \{src(i), dst(i)\}$$

This constraint is valid for all flows, i.e., $\forall i \in TS$.

- Multiple flows cannot be routed over a given link during any of the time-slots. This constraint ensures that the entire path for each flow is reserved for the flow exclusively during its allocated time-slot.

$$\sum_{\forall i \in TS} m_{i,j,k} \leq 1 \qquad \forall j \in E, \forall k \in T$$

- Finally, we need additional constraints to ensure that the ILP solver provides consistent values for the variables, i.e., for a flow i, edge j and time-slot k, the variable $m_{i,j,k}$ can be 1, only if variables $f_{i,j}$ and $t_{i,k}$ are both 1. Hence, the following constraint is required:

$$m_{i,j,k} = f_{i,j} \times t_{i,k} \qquad \forall i \in TS, \forall j \in E, \forall k \in T$$

  Although this constraint is non-linear, it can be transformed into purely linear constraints as follows:

$$\left.\begin{aligned} m_{i,j,k} &\leq f_{i,j} \\ m_{i,j,k} &\leq t_{i,k} \\ m_{i,j,k} &\geq f_{i,j} + t_{i,k} - 1 \end{aligned}\right\} \forall i \in TS, \forall j \in E, \forall k \in T$$

The ILP solver sets values for the variables SL and ST corresponding to the computed schedule. The network controller configures the flow-tables in the switches for routing flows based on SL, and disburses the time-slots based on ST.

This ILP formulation results in an optimal schedule, i.e., maximum number of time-sensitive flows are scheduled from a given set of flows, if the sources of all flows transmit with a period equalling the base-period. Presence of flows with periods higher than the base-period might result in sub-optimality, the extent of which depends on the number of such flows and the difference between its individual periods and the base-period. This is similar to the most field-bus systems, like Sercos-III [34], TTP [25] etc., catering to the needs of time-triggered systems in industrial automation. Accounting for the individual periods of the flows increases the complexity of the scheduling problem manifold. Moreover, the communication flows in these systems transmit with base-period or periods very close to the base-period. Thus, it is reasonable for TSSDN to also ignore the actual transmission periods of the time-sensitive flows for scheduling.

The runtime for computing transmission schedules with this ILP formulation is quite high because it has two degrees of freedom, viz., the routes for the flows and the corresponding time-slots. However, with respect to paths it seems reasonable to prefer short paths as it would result in fewer possibilities of slot collisions along paths sharing the same links. This leads us to other approaches that restrict the search space to explore only the shortest paths to reduce the runtime. This may, however, result in a lower number of flows being scheduled in comparison to *S/UR*. With our subsequent approaches—*S/PR* and *S/FR*—we strive to achieve results approximating those generated by *S/UR* with lower execution costs.

3.3 Scheduling with Pathsets Routing (S/PR)

For *Scheduling with Pathsets Routing*, we extend the model of time-sensitive flows to additionally include a set of "candidate" paths that it may use. The ILP formulation is restricted to route the flow through one of the paths in this set instead of searching the complete solution space for arbitrary paths. We use the set of all shortest paths between the source and destination of a given flow as its candidate paths. However, this approach will have a lower runtime only if the penalty to calculate the set of shortest paths for each flow is amortized by the savings in the execution time of the ILP solver. This is the case, as as we show in the evaluations.

3.3.1 ILP Inputs

The inputs for the ILP formulation are the set of flows to be scheduled and the paths through which each of the flows may be routed.

- Set of flows to be scheduled, TS.
  $TS \equiv \{ts_i\}$; $i \in [1 \ldots NumFlows]$
  Here, $NumFlows$ represent the number of flows to be scheduled.

- Set of possible paths through which the flows may be routed, P.
  $P \equiv \{p_l\}$; $l \in [1 \ldots NumPaths]$
  This set contains all the shortest paths from the source to the destination for each flow $ts \in TS$.

- Mapping between flows and paths, SP.
  $SP \equiv \{sp_{i,l}\}$; $\forall i \in TS$, $\forall l \in P$
  $sp_{i,l} = 1$, if flow i can traverse over path l, else 0.
  It must be noted that while a flow has multiple candidate paths over which it may be routed, a given path can be used by only one flow, i.e., the path identifies the flow.

- Mapping between paths and links, PL.
  $PL \equiv \{pl_{l,j}\}$; $\forall l \in P$, $\forall j \in E$
  $pl_{l,j} = 1$, if path l includes link j, else 0.

3.3.2 ILP Variables

In this formulation, we allocate time-slots to paths instead of a flows.
$PT \equiv \{pt_{l,k}\}$; $\forall l \in P$, $\forall k \in T$
$pt_{l,k} = 1$, if path l is allocated time-slot k, else 0.

3.3.3 Objective Function

The objective for this ILP formulation is to maximize the number of paths with assigned time-slots.

$$\text{Maximize} \sum_{\forall k \in T} \sum_{\forall l \in P} pt_{l,k}$$

3.3.4 Constraints

The constraints for this ILP formulation are enumerated as below:

- Each path may be allocated at-most one time-slot.

$$\sum_{\forall k \in T} pt_{l,k} \leq 1 \qquad \forall l \in P$$

- Each flow can be allocated at-most one time-slot, i.e., for a given flow, only one of its candidate paths can be allocated a time-slot.

$$\sum_{\forall k \in T} \sum_{\forall l \in P} pt_{l,k} \times sp_{i,l} \leq 1 \qquad \forall i \in TS$$

- To avoid collisions, no two paths with overlapping links will be assigned the same time-slot.

$$\sum_{\forall l \in P} pt_{l,k} \times pl_{l,j} \leq 1 \qquad \forall k \in T, \forall j \in E$$

The ILP solver sets values for PT based on which the network controller can disburse the time-slots for the flows and accordingly route them as well.

3.4 Scheduling with Fixed-path Routing (S/FR)

Another approach further reducing the execution time for computing transmission schedules is the *Scheduling with Fixed-path Routing Approach*. This approach extends the idea of *S/PR*. Here, we take a radical approach by choosing the path for a given flow randomly from the set of all shortest paths between its source and destination similar to Equal Cost Multi Path (ECMP) [12] routing. Then, the ILP formulation only deals with the time-slot allocation. While this approach is faster than *S/PR*, the computed schedule might be of even lower quality relative to *S/UR*.

3.4.1 ILP Inputs

The inputs for the ILP formulation is the set of flows to be scheduled and the path through which each of the flow is routed (selected at random from the set of all shortest paths between the source and destination of the flow).

- Set of flows to be scheduled, TS.
  $TS \equiv \{ts_i\}$; $i \in [1 \ldots NumFlows]$
- Mapping of flows to links, SL, indicating the links that belong to the path that a flow must traverse.
  $SL \equiv \{f_{i,j}\}$; $\forall i \in TS$, $\forall j \in E$:
  $f_{i,j} = 1$, if flow i traverses over link j, else 0.

3.4.2 ILP Variables

Decision variables are required only for mapping a flow to time-slots. ST indicates the time-slot that is allocated for a flow.
$ST \equiv \{t_{i,k}\}$; $\forall i \in ST$, $\forall k \in T$
$t_{i,k} = 1$, if flow i is allocated time-slot k, else 0.

3.4.3 Objective Function

The objective function is formulated so as to maximize the number of flows that are allocated time-slots.

$$\text{Maximize} \sum_{\forall i \in TS} \sum_{\forall k \in T} t_{i,k}$$

3.4.4 Constraints

The constraints for this ILP formulation are enumerated as below:

- Each flow may be allocated at most one time-slot.

$$\sum_{\forall k \in T} t_{i,k} \leq 1 \qquad \forall i \in TS$$

- To avoid collisions, no two flows can be allocated the same time-slot if they have overlapping paths.

$$\sum_{\forall i \in TS} t_{i,k} \times f_{i,j} \leq 1 \qquad \forall j \in E, \forall k \in T$$

The ILP solver allocates time-slots to the flows through T. Based on these values, the slots can be disbursed by the network controller.

4 Schedule Adherence in TSSDN

To reap the benefits of TSSDN—deterministic network delay and jitter—sources of time-sensitive flows must adhere to the computed schedule as precisely as possible. In high-speed networks, with high bandwidth links (exceeding 1 Gbps) and high performance cut-through switches, the duration of time-slots (time required for an MTU-sized packet to traverse 8 hops) is in the order of microseconds. Deviation beyond 1–2 μs will render the entire schedule useless. Further, CPS are mainly affected by the end-to-end delays (measured between the userspace applications) which comprise network delay and the delay incurred in the network stacks at the source and destination hosts. This implies that the delays incurred by the packets in the network stack must also be deterministic.

We evaluated the socket API's in Linux (CentOS, kernel version 3.10) to determine if they provide these properties for communication. For our evaluations, we deployed two userspace applications that act as source and destination of time-sensitive traffic on nodes A_1 and B_1, respectively, of our benchmark topology (cf. Fig. 2). We measured the end-to-end latency (between the applications) for 10,000 packets (each of size 1500 bytes), one packet sent every 10 ms. The results (cf. Fig. 4a) show the latency varying between 37–117 μs, with an average latency of 63.58 μs and a standard deviation of 4.88 μs, in absence of any cross traffic. Such high jitter is attributed to the variable delays (10–100 μs) that packets incur while traversing the network stack of the operating system [15], i.e., invoking *send()* on a socket does not place the packet on the network interface with deterministic delay, nor does *receive()* return with bounded delay after the network interface receives a packet. Thus, with socket

API's it is impossible to adhere to the transmission schedules with required precision and provide tight bounds on the delays incurred in the network stack of the end systems.

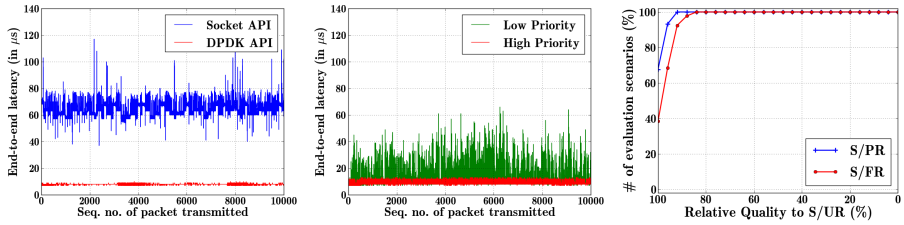
Userspace packet processing frameworks, like Intel's Data Plane Development Kit (DPDK) [5] or netmap [33], bypass the network stack by using custom device drivers and hand the complete control of communications to userspace applications. These may be used to get around the problem of variable delays in the network stack of the end systems. On the flip side, they are then unable to exploit the abstractions for communications, i.e., the userspace application is now responsible for the creation and interpretation of the data packets. To evaluate the feasibility of using these frameworks, we developed two DPDK applications, one as the source and the other as the destination of time-sensitive traffic and measured the end-to-end latency between them, similar to our evaluation of socket applications.

Algorithm 1 Source - Userspace DPDK application

```
1:  function SRC(basePeriod(bp), slotLength(sl), timeSlot(ts))
2:      init NIC and sending queues
3:      intervalAlrm ← flowPeriod
4:      firstAlrm ← now() + (bp − now() % bp) + sl × ts
5:      firstAlrm ← firstAlrm − pktCreationTime
6:      timer_settime(firstAlrm, intervalAlrm)
7:      while True do
8:          if alarm is triggered then
9:              Create payload by executing required tasks
10:             pkt ← dpdk.createPkt()
11:             dpdk.sendPkt(pkt)
```

The destination application simply receives the packet from the network interface bypassing the network stack and parses the packet to decode the information sent by the source. DPDK provides high performance packet processing API's for this purpose. The source application (pseudo-code in Algorithm 1) plays an important role with respect to TSSDN scheduling. It is responsible for configuring timers suitably to trigger packet transmissions. For this we used Linux interval timers (*timer_settime()* [9]) that generate an alarm at fixed intervals based on the base-period, slot length, flow period, and the allocated time-slot (Lines 3–5). The source can use the generated alarm for transmitting the time-sensitive packet prepared beforehand, or use it as a trigger to also create the packet (generate the payload by executing the sensing or CPS-control tasks). We use the latter approach (Lines 9–10) and hence advance the interval timer by *pktCreationTime* (profiled beforehand) to compensate for the time required to generate the payload and create the corresponding packet.

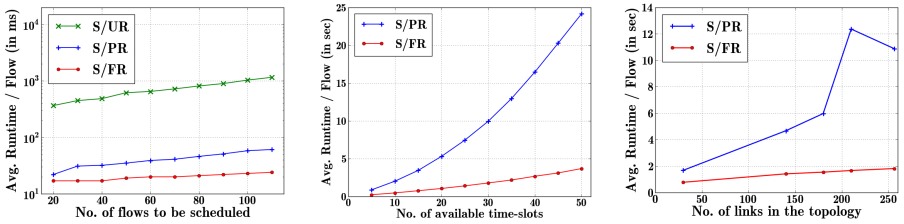
With DPDK API's, the end-to-end latency varied between 7–10 μs with an average latency of 7.94 μs and a standard deviation of 0.4 μs. This implies that the packets incur almost constant delays in the network stack with the use of userspace packet processing frameworks. The low end-to-end latency between the source and destination applications indicate that packets are placed on the network interface with minimal delay after the corresponding API is invoked. Hence, we use DPDK for precisely complying with the transmission schedules.



(a) Comparison of latencies

(b) Impact of packet prioritization

(c) Quality comparison of the ILPs



(d) Scheduling runtime vs No. of flows

(e) Scheduling runtime vs No. of slots

(f) Scheduling runtime vs Topology size

Figure 4: Evaluations Results

Our evaluations showed similar results for other frameworks also.

5 Evaluations

We evaluated TSSDN on two fronts. Firstly, we measured the end-to-end latency for time-sensitive traffic on the data plane of TSSDN under various scenarios to determine if it provides the promised real-time guarantees. Secondly, we evaluated the ILP formulations, executing on the control plane, to compute transmission schedules for random graphs created using different models to exhibit its correctness and scalability.

5.1 Data Plane Evaluations for TSSDN

To evaluate the real-time properties provided by TSSDN on the data plane, we implemented the benchmark topology, shown in Fig. 2, using five commodity machines (Intel Xeon E5-1650) each equipped with an Intel XL710 quad 10 GbE network interface [7] and an Edge-Core cut-through "bare-metal" switch (AS5712-54X) [6] running PicOS (ver 2.6.1) [10]. The switch was partitioned into virtual switches to create the topology, while each machine hosted two end systems, for instance, Host A_1 and B_1 were placed on the same machine but used different network interfaces. We used the Precision Time Protocol (PTP) for synchronizing clocks on all machines. To this end, we used a separate network infrastructure using a third network interface on each machine (two interfaces are used by the end systems hosted on the machine) dedicated to PTP synchronization. This was basically necessary because of two reasons:

First, our switch did not support PTP. Thus, high priority time-sensitive packets could potentially impact the accuracy of PTP latency measurements. With a switch which can measure the port-to-port residence time of PTP packets, the precision of clock synchronization would not be affected. Secondly, DPDK exclusively allocates a network interface to a process, so we cannot easily run a PTP daemon over the same port. Sharing a port between different processes is a common problem of current userspace packet processing frameworks and a separate research problem.

5.1.1 In-network Prioritization

As mentioned in Section 4, the end systems in TSSDN use userspace packet processing frameworks to adhere with the computed schedules (cf. Figure 4a). However, this alone is insufficient as TSSDN is also meant to additionally transport non-time-sensitive traffic. In this section, we experimentally show the importance of tagging time-sensitive packets as priority traffic, while also motivating the need for transmission scheduling in TSSDN.

To determine the impact of non-time-sensitive traffic, we loaded the bottleneck link (link from switch S_1 to S_2) of our benchmark topology with random traffic (random packet sizes and variable bitrate) initiated by end systems A_2–A_5. It may be noted that the link was never subscribed beyond 80 % of its total capacity. With this cross traffic, we measured the end-to-end latencies for 10,000 packets sent from $A_1 \rightarrow B_1$ with a period of 10 ms. As shown in Fig. 4b, the end-to-end latency fluctuates drastically between 7–66 μs, if the packets are not marked as priority packets by the source despite the spare capacity in the bottleneck link. End systems may tag time-sensitive packets as high priority packets so that its delivery would be expedited by the data plane. We used the IEEE 802.1Q priority scheme and marked time-sensitive packets with highest possible priority class (priority 7). With prioritization of time-sensitive traffic, the end-to-end latency with cross traffic varies in a narrower band of 7–13 μs. However, the standard deviation of end-to-end latency has increased from submicrosecond range (in absence of any interference from non-time-sensitive traffic) to 1.68 μs. This is because our switch does not support frame preemption (IEEE 802.1Qbu [1]), and hence time-sensitive packets, though higher in priority, must queue till the current non-time-sensitive packet is transmitted. With support for frame preemption, higher priority time-sensitive packets will not be affected by the lower priority cross-traffic. It may be noted that the upcoming standards for frame preemption will also soon make its way into commodity switches.

The impact of prioritizing time-sensitive packets is, however, nullified, if time-sensitive flows are not temporally or spatially isolated. In absence of scheduling, no guarantees can be provided with respect to the bounds on end-to-end delays and jitter, even if the time-sensitive packets are tagged as high priority packets.

5.1.2 Impact of Scheduling

To show the impact of scheduling, we deployed a varying number of time-sensitive flows on our benchmark topology. We used a slot-length of 15 μs, considering the end-to-end delay in traversing the network diameter of our bench-

# Flows	Avg. (μs)	Std. (μs)	Min (μs)	Max (μs)
1 Flow	7.99	0.62	7	13
2 Flows	8.09	0.57	7	14
3 Flows	8.04	0.49	7	14
4 Flows	8.07	0.48	7	14
5 Flows	8.06	0.54	7	14

Table 2: End-to-end latency for time-sensitive flows when scheduled in adjacent time-slots

mark topology. We assume a base-period of 1 ms and that all flows use their slots completely, i.e., transmit one packet every 1 ms. The flows are allocated adjacent slots to demonstrate that schedules can be adhered precisely by the end systems. It may be noted that we evaluate our system in the toughest scenario with adjacent slots occupied on a 10 Gbps link as this would amplify any consequence of non-adherence to schedules. We measured end-to-end latencies for 10^5 packets per flow and summarized the results in Table 2. As can be seen, the end-to-end delays for the time-sensitive flows vary in a narrow band of $\leq 7\,\mu$s, irrespective of the number of flows in the network. Further, the standard deviation for time-sensitive flows is also in sub-microsecond range indicating minimal communication jitter. In networks with lower bandwidth links, the performance would be equally good or even better. Thus, we conclude that suitable transmission schedules impart real-time properties for communication over the data plane of TSSDN.

Further, to emphasize the importance of transmission scheduling, we measure the end-to-end latencies for a varying number of time-sensitive flows when they are assigned the same transmission slot. Our ILP formulations would never allow time-sensitive flows to interfere, however, in absence of scheduling such a scenario cannot be ruled out. Hence, we repeated the above experiment but allotted the same slot to the flows instead of adjacent ones. The results summarized in Table 3 show that end-to-end latency of time-sensitive flows are affected if more than one flow is assigned the same slot. The average end-to-end delay and the standard deviation steadily increases with the number of time-sensitive flows sharing the time-slot. Moreover, the jitter goes beyond $7\,\mu$s when more than 3 time-sensitive flows contend for traversing a network link. This scenario also shows that in absence of scheduling, the time-sensitive traffic could end up impeding each other in the network. The reason for this is that TSSDN prioritizes time-sensitive traffic over non-time-sensitive-traffic but does not resolve priorities between them.

We observed that the jitter depends on the transmission frequency of the DPDK application and size of the packets being transmitted. For instance, jitter of $\leq 3\,\mu$s was observed at a frequency of 100 Hz for 64-byte sized packets, while it increased to $\leq 7\,\mu$s at a frequency of 10 kHz for 1500-byte sized packet. We infer that a part of this jitter (1–2 μs) originates from the interval timers in Linux, while the rest is a result of process preemptions or delayed availability of computing slice for the userspace applications (despite executing them with highest priority, i.e., nice level -20 in Linux) at source and destination hosts. In our future work, we will explore using real time kernel patches to further reduce the residual jitter.

# Flows	Avg. (μs)	Std. (μs)	Min (μs)	Max (μs)
2 Flows	8.63	0.86	7	14
3 Flows	9.19	1.14	7	14
4 Flows	9.75	1.42	7	15
5 Flows	10.2	1.71	7	17

Table 3: End-to-end latency for time-sensitive flows when scheduled in the same time-slot

5.2 Control Plane Evaluations for TSSDN

In this section, we evaluate the various ILP formulations, presented in Section 3, with respect to the quality of schedules they compute and their scalability.

We use the commercial ILP solver CPLEX from IBM [2] to solve our ILP formulations which are specified using PuLP [28], a Python-based tool-kit to specify ILPs. Moreover, we created different network topologies (different sizes and different network models) using NetworkX [22], a Python library for creating complex networks. In detail, we used the Erdős-Rényi (ER) model [20] (random graphs where nodes have *similar* degree), random regular graphs (RRG) (random graphs where nodes have same degree), the Barabási-Albert (BA) model [14] (scale-free networks where few nodes have high degree and many have small degree), and the Waxman model [37] (geographic model favoring short-distance links over long links). Together, these models for randomized graphs comprehensively test the limits of our ILP formulations. The sizes of these topologies and the number of time-slots and flows used as input are specified with the concrete evaluations.

We used two machines for evaluating our ILPs. The first is a high performance multi-processor machine with 2×8 cores (Intel Xeon E5-2650) and 128 GB RAM, while the second is a commodity machine with 2 cores and 8 GB RAM.

5.2.1 Qualitative Evaluations

To evaluate the quality of the schedules generated by the ILP formulations S/PR and S/FR with respect to S/UR, we computed the transmission schedules in 160 evaluation scenarios using 8 different topologies (3 RRG, 2 ER, and 3 BA), each with 24 hosts and 6 switches. Note that we had to choose a smaller topology to be able to compute the schedule using S/UR as reference since it has a very high runtime. Limiting the number of components in the topology also limited the number of topologies we could examine. Each scenario consisted of 20–110 flows with random source and destination hosts to be scheduled with 3–5 available time-slots in the network. We have deliberately chosen a smaller number of slots to create challenging scenarios for our ILP formulations even for smaller numbers of flows. As performance metric, we calculate the relative quality of the schedules computed by S/PR and S/FR, i.e., the ratio of the number of flows scheduled by them to the number of flows scheduled by S/UR.

Fig. 4c shows the cumulative distribution of the relative quality achieved by S/PR and S/FR. This figure shows that the quality of the solutions they generate closely approximate the quality of the ones computed using S/UR. For

instance, for S/PR, 80 % of the scenarios have at least a relative quality of 98 % or better. In detail, S/PR and S/FR generated schedules with 100 % relative quality in about 67 % and 38 % of the evaluation scenarios, respectively, with average qualities of 99 % and 97 %.

5.2.2 Scalability Evaluations

Knowing the quality of the different approaches, we next evaluate their scalability, i.e., the time to calculate solutions for different scenario sizes. The runtime for computing the transmission schedule depends mainly on three factors: the number of flows to schedule, the number of available time-slots, and the size of the topology. Therefore, we vary these parameters and measure the corresponding runtime for computing the schedule.

First, we vary the number of flows for scheduling using the ILP formulations. We use a small scenario, an ER topology consisting of 24 hosts and 6 switches (38 network links) with 5 time-slots for disbursement, to measure the runtime for computing schedules using our various approaches. We measure the runtime for computing the schedules with a varying number (20–110) of flows on our high performance machine. As shown in Fig. 4d, the runtime for computing the schedules using S/PR and S/FR is at least an order of magnitude lower than that for computing it using S/UR. As per our evaluations, S/PR and S/FR could compute schedules for over 100 flows in approximately 7 s and 3 s, respectively, while computing the schedule using S/UR required over 2 m. This translates to an average scheduling time of 1.1 s, 61 ms, and 24 ms, per flow for S/UR, S/PR, S/FR, respectively. We observed similar or worse results with execution times running into hours for computing schedules using S/UR on other topologies with similar set-up. An interesting observation with respect to the faster ILP formulations, S/PR and S/FR, is that its runtime is dominated by the time required to specify the ILP. The solver takes only a few milliseconds to solve it. This is an interesting insight for developing ILP formulations for incremental scheduling in the future.

Next, we vary the number of available time-slots to evaluate its impact on the runtime of the ILP formulations. For this and subsequent evaluations, we execute the ILP solver on the commodity machine and do not use the S/UR approach as the scenarios are too large for computing a schedule with it. Here, we use a topology with 200 hosts and 10 switches (256 network links) based on the Waxman model. We scheduled 300 flows on this topology using the ILP formulations and measured the average time to schedule a flow. The number of time-slots were varied between 5–50. As shown in Fig. 4e, the runtime increases rapidly for the S/PR approach with increasing number of available time-slots in contrast to the S/FR approach, which scales much better (approximately linearly with number of time-slots). It may be noted that a network with 1 Gbps links and a network diameter of 8 hops provides only about 50 slots (considering MTU as 1500 bytes) for a base-period of 1 ms. Moreover, assuming that a CPS comprise of two flows (one from sensor to the CPS controller and other from the CPS controller to the actuator), schedules for supporting up to 150 CPS can be calculated by our ILPs. Thus, we can conclude that our ILP formulations scale well for realistic scenarios.

Finally, we evaluated the impact of topology size (number of network links) on the runtime of the ILP formulations. For this evaluation, we used different

topologies (30–256 network links) and scheduled over 100 flows on them with 50 time-slots for disbursement. Fig. 4f summarizes the measured runtimes for S/PR and S/FR. We observe that the runtime of S/FR increases linearly with the size of the topology and takes on an average less than 2 s to schedule a flow in a topology containing 256 links. For the S/PR approach, the runtime is not directly related to the topology size. It, rather, depends on the number of shortest paths between the sources and the destinations of the flows, i.e., the path diversity of the network. Nonetheless, the worst case average time to schedule a flow was just over 12 s for this ILP formulation.

5.3 Evaluation Summary

In summary, our evaluations showed:

1. TSSDN provides virtually constant end-to-end latency (std. dev. $< 1\,\mu$s) with worst case jitter $\leq 7\,\mu$s for the time-sensitive traffic on our benchmark topology.

2. The S/PR and S/FR approaches for computing transmission schedules closely approximate the solution computed by S/UR (which provides optimal solutions in most practical cases), despite having runtimes that is orders of magnitudes lower.

3. Our ILP formulations, S/PR and S/FR, scale well to compute schedules for networks with over 200 network links with a data-rate of 1 Gbps (≈ 50 time-slots assuming a base-period of 1 ms) with over 300 flows.

6 Related Work

For long, real-time applications have used only field-bus networks for communications as they provide the required hard real-time guarantees. They avoid collisions by use of mechanisms like token parsing (in Profibus), in-network arbitration (in CAN bus), and scheduling transmissions (in Sercos III). However, field-bus networks suffer from scalability issues and can provide real-time guarantees only in static scenarios [30].

Therefore, there is a strong trend to make widely adopted IP-based networks and IEEE 802 networks ready for real-time traffic. These developments are driven, in particular, by the IEEE 802.1 Time-Sensitive Networking Task Group [24], 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 [4]. We presented our vision of TSSDN to the scientific community [29], however, in the initial step we only spatially isolated time-sensitive flows. With this paper, we make a step forward by bringing in the temporal aspect. We intentionally base our system on basic principles conforming with the initial proposals of these standards bodies like synchronized end systems and logically centralized configuration. This directly makes our contributions like the scheduling algorithms—which have not been considered by these groups so far—applicable to upcoming standard networks.

Scheduling of transmission at the hosts in time-triggered networks to impart real-time properties is a well-researched problem. Approaches using Satisfiability Modulo Theories (SMT) and Resource Constrained Project Scheduling (RCPS) has been applied to compute static schedules in multi-hop Ethernet-like networks [35][23]. Further, approaches to combinedly compute transmission schedules along with task schedules have been proposed [18][17]. However, all these approaches assume advance information of routes for the time-triggered flows ignoring the possibility of influencing them while scheduling. This may result in accommodation of fewer time-triggered flows in the network, for e.g., when several flows are routed over a single bottleneck link instead of distributing them over redundant links. Moreover, the approach in [35] cannot benefit from the modern high-speed cut through switches as they compute link schedules necessitating store-and-forward actions. In contrast, we address the combined problem of routing and scheduling of time-sensitive flows in TSSDN in this work. To the best of our knowledge, we are the first in using commodity hardware with software-defined networking technologies to impart real-time properties for communication over IEEE 802.3 networks while also transporting non-real-time traffic. Our evaluation shows excellent results with ultra-low jitter for communication while supporting high transmission rates.

The attempts of the networking community to impart real-time properties to IEEE 802 networks have been mainly directed at data-center architectures which are inherently soft real-time. Hedera uses SDN to spatially distribute the traffic to prevent hotspots in the network [12]. Fastpass [32] and Datacenter TDMA [36] are other attempts at minimizing queuing delays by means of scheduling. Fastpass uses a centralized controller to allocate time-slots like in TSSDN, while Datacenter TDMA uses 802.3x flow control mechanism to enforce schedules. However, these systems strive to provide soft-real-time guarantees for the unpredictable nature of traffic in data-centers, while TSSDN exploits the time-triggered nature of time-sensitive traffic to support hard-real-time systems.

7 Conclusion & Outlook

In this paper, we motivated the need for integrating mechanisms in IEEE 802.3 and IP networks to transport time-sensitive traffic with bounded end-to-end latency and jitter along with non-time-sensitive traffic. For this, we presented Time-sensitive Software Defined Networks, which provide real-time guarantees for communication of time-sensitive traffic by means of transmission scheduling. As a first step, we presented a set of ILP formulations that compute transmission schedules given a set of pre-defined time-sensitive flows in a network topology. Our evaluations showed that the ILPs could calculate high quality schedules efficiently and that adherence to the schedule results in deterministic network behaviour.

As a part of future work, we are going to develop *lightweight* and *fast* scheduling algorithms that can rapidly schedule time-sensitive flows incrementally. Another interesting open question which we also left for future work is, how much jitter can be reduced with hardware support (specialized NICs or NetFPGAs) or real-time operating systems.

References

[1] 802.1Qbu - Frame Preemption. http://www.ieee802.org/1/pages/802.1bu.html. Accessed: 2016-05-04.

[2] CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/. Accessed: 2016-05-04.

[3] Deterministic Networking - BOF Status. http://www.ieee802.org/1/files/public/docs2014/tsn-nfinn-Deterministic-Networking-BOF-0914-v1.pdf. Accessed: 2016-05-04.

[4] Deterministic Networking Charter. https://datatracker.ietf.org/wg/detnet/charter/. Accessed: 2016-05-04.

[5] DPDK: Data Plane Development Kit. http://dpdk.org/. Accessed: 2016-05-04.

[6] Hardware Switch Edge-Core AS5712-54X. http://www.edge-core.com/. Accessed: 2016-05-04.

[7] Intel Ethernet Controller XL710 and X710 Families. http://www.intel.com/content/www/us/en/embedded/products/networking/ethernet-controller-xl710-family.html. Accessed: 2016-05-04.

[8] Introduction to IEEE 1588. http://www.nist.gov/el/isd/ieee/intro1588.cfm. Accessed: 2016-05-04.

[9] Linux Programmer's Manual - TIMER_SETTIME(2). http://man7.org/linux/man-pages/man2/timer_settime.2.html. Accessed: 2016-05-04.

[10] PicOS Version 2.6. http://www.pica8.com/documents/pica8-datasheet-picos.pdf. Accessed: 2016-05-04.

[11] IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic. *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pages 1–57, March 2016.

[12] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10)*, 2010.

[13] Javier Aracil and Franco Callegati. *Enabling Optical Internet with Advanced Network Technologies*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[14] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[15] A. Beifus, D. Raumer, P. Emmerich, T.M. Runge, F. Wohlfart, B.E. Wolfinger, and G. Carle. A study of networking software induced latency. In *International Conference and Workshops on Networked Systems (NetSys), 2015*, pages 1–8, March 2015.

[16] Imrich Chlamtac, Aura Ganz, and Gadi Karmi. Lightpath communications: An approach to high bandwidth optical WAN's. *IEEE Transactions on Communications*, 40(7):1171–1182, 1992.

[17] Silviu S Craciunas and Ramon Serna Oliver. SMT-based task-and network-level static schedule generation for time-triggered networked systems. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, page 45. ACM, 2014.

[18] Silviu S. Craciunas and Ramon Serna Oliver. Combined Task- and Network-level Scheduling for Distributed Time-triggered Systems. *Real-Time Syst.*, 52(2):161–200, March 2016.

[19] Frank Dürr and Thomas Kohler. Comparing the Forwarding Latency of OpenFlow Hardware and Software Switches. Technical Report Computer Science 2014/04, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, University of Stuttgart, Institute of Parallel and Distributed Systems, Distributed Systems, July 2014.

[20] P. Erdős and A. Rényi. On random graphs I. *Publicationes Mathematicae 6*, pages 290–297, 1959.

[21] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand, and Jon Crowcroft. Queues Don't Matter When You Can JUMP Them! In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 1–14, Oakland, CA, May 2015. USENIX Association.

[22] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[23] Z. Hanzalek, P. Burget, and P. Sucha. Profinet IO IRT Message Scheduling With Temporal Constraints. *IEEE Transactions on Industrial Informatics*, 6(3):369–380, Aug 2010.

[24] M.D. Johas Teener, A.N. Fredette, C. Boiger, P. Klein, C. Gunther, D. Olsen, and K. Stanton. Heterogeneous Networks for Audio and Video: Using IEEE 802.1 Audio Video Bridging. *Proceedings of the IEEE*, 101(11):2339–2354, Nov 2013.

[25] H. Kopetz and G. Grunsteidl. TTP - A time-triggered protocol for fault-tolerant real-time systems. In *The Twenty-Third International Symposium on Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers.*, pages 524–533, June 1993.

[26] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The time-triggered ethernet (TTE) design. In *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005.*, pages 22–33. IEEE, 2005.

[27] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[28] Stuart Mitchell, Michael OSullivan, and Iain Dunning. PuLP: A Linear Programming Toolkit for Python. 2011.

[29] N.G. Nayak, F. Durr, and K. Rothermel. Software-defined environment for reconfigurable manufacturing systems. In *5th International Conference on the Internet of Things (IOT), 2015*, pages 122–129, Oct 2015.

[30] Peter Neumann. Communication in industrial automation - What is going on? *Control Engineering Practice*, 15(11):1332–1347, 2007.

[31] P. Pedreiras, P. Gai, L. Almeida, and G.C. Buttazzo. FTT-Ethernet: a flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems. *IEEE Transactions on Industrial Informatics,*, 1(3):162–172, Aug 2005.

[32] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: a centralized zero-queue datacenter network. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 307–318. ACM, 2014.

[33] Luigi Rizzo. netmap: A Novel Framework for Fast Packet I/O. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 101–112, Bellevue, WA, August 2012. USENIX Association.

[34] E. Schemm. SERCOS to link with ethernet for its third generation. *Computing Control Engineering Journal*, 15(2):30–33, April 2004.

[35] W. Steiner. An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In *IEEE 31st Real-Time Systems Symposium (RTSS), 2010*, pages 375–384, Nov 2010.

[36] Bhanu Chandra Vattikonda, George Porter, Amin Vahdat, and Alex C Snoeren. Practical TDMA for datacenter ethernet. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 225–238. ACM, 2012.

[37] Bernard M Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.