TSNSCHED: Automated Schedule Generation for Time Sensitive Networking

Aellison Cassimiro T. dos Santos
Federal University of Paraíba
Paraíba, Brazil
Email: cassimiroaellison@gmail.com

Ben Schneider
fortiss
Munich, Germany
Email: schneider@fortiss.org

Vivek Nigam
fortiss
Munich, Germany
Email: nigam@fortiss.org

***Abstract*—Time Sensitive Networking (TSN) is a set of standards enabling high performance deterministic communication using different scheduling mechanisms. Due to the size of industrial networks, configuring TSN networks is challenging to be done manually. We present TSNsched, a tool for automatic generation of schedules for TSN. TSNsched takes as input the logical topology of a network, expressed as flows, and outputs schedules for TSN switches using an SMT-solver. The generated schedule guarantees the desired network performance (specified in terms of latency and jitter), if such schedules exist. TSNsched can synthesize IEEE 802.1Qbv schedules and supports unicast and multicast flows, such as, in Publish/Subscribe networks. TSNsched can be run as a standalone tool and also allows rapid prototyping with the available JAVA API. We evaluate TSNsched on a number of realistic-size network topologies. TSNsched can generate high performance schedules, with average latency less than 1000μs, and average jitter less than 20μs, for TSN networks, with up to 73 subscribers and up to 10 multicast flows.**

I. INTRODUCTION

Time Sensitive Networking (TSN) is a recent OSI layer 2 network protocol standard (IEEE 802.1Q [2]) that extends Ethernet. It is developed to address the increasing performance demands of, for example, industrial automation applications like motion control, where the movement of components has to be synchronized in a microsecond range [15]. Many of these applications require multicast flows as the same data, *e.g.*, position of the axis, is sent to different actuators, as well as combine high priority traffic with best-effort traffic.

TSN achieves high performance requirements by a set of sub-standards which provide, for example, high precision time synchronization and a time-triggered scheduler [1], [2]. Moreover, TSN allows the co-existence of best-effort flows and flows that take advantage of the high precision time synchronization, *e.g.*, flows that require low latency and jitter.

The major problem for TSN to be successfully adopted in the industry is the complex configuration of the different TSN schedules. The complexity caused by the growth of networks and the necessary domain-specific knowledge about the requirements and network standards makes manual network configuration infeasible.

We present the tool TSNSCHED[1], which generates schedules for TSN. It takes as input a network topology, including the network flows, and network performance requirements, including values for the maximum latency and jitter per flow. Flows may be unicast or multicast. TSNSCHED reduces the TSN scheduling problem to an SMT problem which is solved using the Z3 SMT solver [12]. For a modular translation, we split flows into *Flow Fragments*. If the scheduling problem is shown to be solvable by Z3, then TSNSCHED extracts, from the witnessing model, the TSN scheduling configuration for each switch in the network topology. This includes the TSN switch queues' time slots, cycle times, and priority queues associated with each flow.

The key motivation for developing TSNSCHED in JAVA is our on-going efforts to integrate TSNSCHED with the eclipse 4diac™ framework[2]. 4diac™ is used for the development of applications for distributed industrial automation systems. The integration of TSNSCHED with 4diac™ will allow to automatically deduce domain-specific requirements as input for the schedule synthesis. It will also mean that TSNSCHED will be used by a large and active set of users, leading to further tool improvements.

Some other tools have been proposed for a similar purpose, notably, the tool described in [11], which builds on the foundational work carried out in [10]. While we have been inspired by these works, TSNSCHED has some considerable differences to existing tools. Besides being publicly available, whereas the tool in [11] is proprietary, we enumerate below some of TSNSCHED's key features:

- **Variations of TSN Scheduling Problem:** One can specify a large number of aspects of the network, including the specification of ports, switches, travel times, etc. This allows for the combination of different variants of the TSN scheduling problem. For example, it is possible to combine the problems defined in [9] and in [10], [11]. The former [9] does not explicitly encode TSN time windows, thus not allowing to reason about its properties, while the latter works [10], [11] encode TSN time windows, but

We thank Tiziano Murano for fruitful disscussions. Nigam and Cassimiro have been partially supported by CNPq projects 425870/2016-2, 303909/2018-8 and and fapesp project 15/24516-1. This project received funding from the EUs Horizon 2020 research and innovation programme under grant agreement No 830892. Schneider was supported by the DEVEKOS project, funded by the German Federal Ministry of Economic Affairs and Energy (BMWi) (no.01MA17004F).

[1] TSNSCHED can be found at https://github.com/ACassimiro/TSNsched.
[2] See https://www.eclipse.org/4diac/

not the frame transmission time constraints. TSNSCHED is able to specify both TSN time windows and frame transmission time constraints.
- **Unicast and Multicast Flows:** Previous work [10], [11] consider unicast flows, TSNSCHED allows for the specification of both unicast and multicast flows.
- **Best-Effort and TSN flows:** TSNSCHED generates schedules satisfying the given requirements for high performance TSN flows as well as of flows using best-effort. This contrasts with previous work that puts emphasis on TSN flows;
- **Standalone and API for Rapid Prototyping:** While it is possible to run TSNSCHED as a stand-alone tool, TSNSCHED also provides a general API to enable Rapid Prototyping. Indeed, this was motivated so that we can easily integrate TSNSCHED with eclipse 4diac™.
- **Machine-Readable Schedule Outputs:** The output generated by TSNSCHED is machine readable. This means that it is, in principle, possible to generate configuration files from TSNSCHED. We have not done so, as there is an on-going discussion on the exact configuration format.

We believe that a reason for the increased expressiveness of TSNSCHED is that we have found a suitable abstraction for this problem. Instead of encoding given network flows as a end-to-end specification, we split flows into Flow Fragments. This modular specification of the problem improves the understandability of the code and of the specifications generated.

Our experiments demonstrate that TSNSCHED is capable of generating high performance TSN schedules with low average jitter even for relatively large networks (with up to 5 flows and up to 73 subscribers). Despite the admissible average jitter of 25μs, the schedules generated by TSNSCHED guaranteed a much lower average jitter of 4.85μs even in some larger scenarios. We also observed that TSNSCHED's performance worsens when dealing with larger flows, *i.e.*, flows with more switches, and with flows with performance close to the upper or lower limit of the duration of cycles.

Section II explains the TSN scheduling problem. Section III describes TSNSCHED execution process and in Section IV we present some details of the problem encoding. Then in Section V, we describe and evaluate our experimental results. We conclude in Section VI discussing related and future work.

II. Time Sensitive Networking (TSN) Scheduling Problem

The Time-Aware-Shaper (TAS) according to IEEE802.1Qbv [2] is designed to ensure determinism and real-time by reserving bandwidth via time slots. An example of a (trivial) scheduling problem is shown in Figures 1 and the correlating schedule in Figure 2.

The example topology contains Publishers Pub and Subscribers Sub and two switches SW_1 and SW_2. Lets assume that $Flow_1$, from Pub_1 to Sub_1, belongs to a safety critical application with hard real-time requirements sending periodic traffic. $Flow_2$, Pub_2 to Sub_2, periodically transmits process

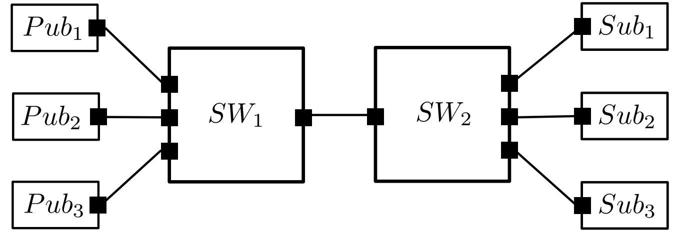


Fig. 1. Network topology with Publishers (Pub) and Subscribers (Sub) connected via a single link (SW_1 to SW_2)

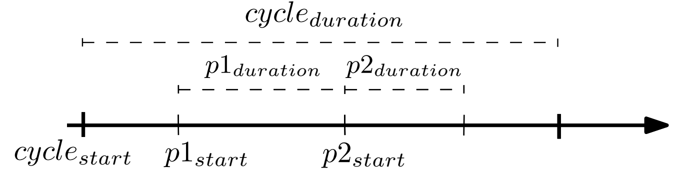


Fig. 2. IEEE802.1Qbv schedule for the port that connects SW_1 to SW_2

data into the cloud for analysis purposes and has deterministic requirements. $Flow_3$, Pub_3 to Sub_3, is a best effort stream which requires high bandwidth (*e.g.*, video data for visualization) but is not critical for the function or safety of the production plant. In this setup all streams compete for bandwidth on the link between SW_1 and SW_2. Without a proper scheduling mechanism it might happen that the low priority best effort $Flow_3$ consumes the bandwidth which is needed by the higher prioritized $Flow_1$ and $Flow_2$. The TAS [2] enables the time-triggered scheduling of flows using bandwidth reservation by assigning different flows to their appropriate priority (1-8) and finding time slot for each of the priority queues on a transmission port. The example TAS schedule shown in Figure 2 shows a valid configuration where $Flow_1$ is mapped to priority p_1, $Flow_2$ is mapped to priority p_2 and the rest of the cycle is open for transmitting best effort traffic.

The scheduling problem of the TAS can be summarized as finding the cycle duration, the time slot starting time and the time slot duration [16]. Moreover, these slots shall specify some specific TSN requirements, *e.g.*, be separated by a Guard-Band under some conditions, and many others.

This time-triggered scheduling problem is a bin-packing problem and known to be NP-complete [9]. Solving it manually is a time-consuming and error-prone task that needs detailed domain-specific knowledge about the communication system and application.

Moreover, current trends show that the complexity of scheduling will grow in the future. An example can be given from the industrial automation domain where topologies continue to grow (known as IIoT – Industrial Internet of Things). Therefore, the requirements of future industrial networks make a manual configuration infeasible.

III. TSNSCHED Description

TSNSCHED takes as input a scheduling problem for deterministic high performance periodic network traffic and

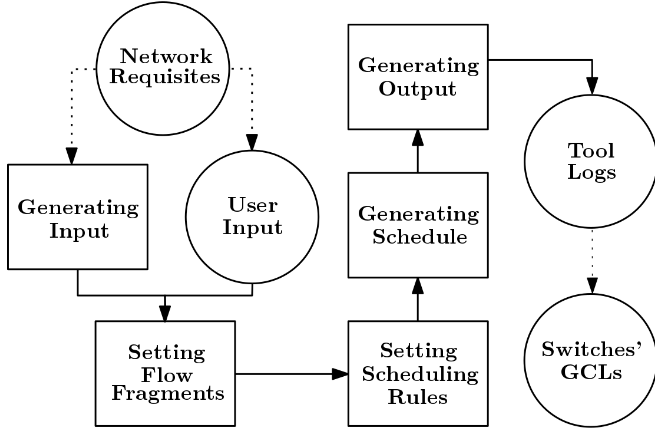


Fig. 3. TSNSCHED execution process overview.

performance criteria, namely, maximum latency and jitter per packet per flow, and returns the scheduling configurations for all ports of the TSN switches in the network topology.

Figure 3 depicts the steps taken by TSNSCHED for generating a TSN schedule. It starts by the user specifying the **Network Requisites**, extracting the necessary network information, *e.g.*, flows, maximum latency and jitter, etc.

User Input: If the user is importing TSNSCHED as a project library, a JAVA file containing the specification of the topology made with the classes provided by the TSNSCHED package can be used to model a network. The user specifies the scope of the network, providing properties of switches, devices and flows involved in the network. These properties include:

- Interval between packets sent by a device;
- Time to travel between switches;
- Time taken to process a packet in an egress port (dependant on link speed and packet size);
- Maximum cycle duration per switch;
- Minimum cycle duration per switch;
- Maximum size of priority transmission window;
- Maximum guard band size per switch;
- Maximum tolerated latency per packet per flow;
- Maximum tolerated jitter per packet per flow.

The tool can also be used as a standalone software by providing the class with the network specification as input.

Generating Input: TSNSCHED also includes a topology generator, which generates topologies according to user-controlled parameters, such as number and size of flows. These parameters will be used to define the values mentioned in the list above, and also to define certain properties of the flows. This is a useful feature to carry out experiments.

Setting Flow Fragments: With the scenario completely modeled as JAVA objects, the first task performed by TSNSCHED is the division of a flow into flow fragments, which include the conversion of the user-defined data to Z3 variables. Section IV describes Flow Fragments and their encoding in Z3.

```
@ Switch6: Cycle duration: 400
...
@ Fragment name: flow1Fragment1
@ Fragment node: switch6 Port name
(Virtual Index): switch6Port0
@ Fragment next hop: switch2 Fragment
priority: 1
@ Fragment slot start: 2 Fragment slot
duration: 40
@ (1) Fragment departure time: 0
@ (1) Fragment arrival time: 1
@ (1) Fragment scheduled time: 57/4
```

Fig. 4. Illustration of TSNSCHED output.

Setting Scheduling Rules: Once all the Z3 variables are created, the rules that shape the schedule are given to Z3. Flow fragments allows to apply the rules over each link of the flow, as every port object can iterate over the flow fragment objects applying the constraints to the variables used by the solver. As described in Section IV, additional constraints can be added in order to model variants of the TSN scheduling problem, such as those specified in [9] and [11].

Generating Schedule: With the specified rules, Z3 attempts to solve the specified scheduling problem. Given the complexity of such problem to an SMT solver, most of the execution time of the software is spent here. If the problem cannot be solved using the rules that were previously specified, a warning is issued; otherwise TSNSCHED proceeds to the next stage.

Generating Output: After successfully solving the scheduling problem, the tool extracts the values for the variables from the solutions, such as:

- Start of the cycle of each switch;
- Duration of the cycle of each switch;
- Start of time window for transferring priority traffic;
- Duration of time window for transferring priority traffic;
- Priority of the packet in each hop;
- First sending time of a packet of a flow.

The task now is to store these values into the objects used to model the network, so the user can operate them as prefered.

Tool Logs: A set of logs containing the information of the network is also created. This way, if used as a standalone tool, TSNSCHED still can provide a way to the user to analyze the generated schedule.

An example of the output is shown in Figure 4. It shows that the inferred cycle duration of the `switch6` is 400μs. It also specifies the Flow Fragment `flow1Fragment1` connecting `switch6` to the switch `switch2` using port `switch6Port0`. This flow is assigned by TSNSCHED to priority 1, and uses a time slot with duration of 40μs at a starting time of 2μs after the start of the switch's cycle. It also specifies the departure, arrival and scheduled time of the flow fragment. A packet departs at time 0μs, arrives at the switch at time 1μs and it leaves the switch at time 57/4 = 14.25μs.

Switches' Gate Control Lists (GCLs): With the informa-

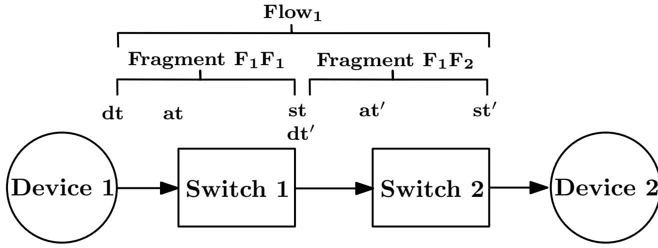


Fig. 5. Illustration of a flow represented by Flow Fragments.

tion contained in the log, the user can give the information given by TSNSCHED as an input to the network configuration, creating the schedules in the switches' GCLs. This step is currently done manually. We intend to generate these configuration files automatically, once the exact format is decided.

IV. Z3 ENCODING AND EXPRESSIVENESS

We describe part of the encoding in Z3 carried out by TSNSCHED and illustrate the expressiveness of TSNSCHED later in this Section. We start by introducing some data structures which contain the domain-specific data required by TSNSCHED to generate a TSN schedule.

For generating a schedule, we will be interested in the times when a packet departs, arrives and is scheduled to exit the devices and switches, as specified in the following definition.

Definition 1. *Timing Variables:* The timing variables t of a packet[3] are defined by the triple $\langle \mathsf{dt}, \mathsf{at}, \mathsf{st} \rangle$.

- dt is the departure time of the packet;
- at is the arrival time of the packet;
- st is the scheduled time of the packet.

Figure 5 illustrates the timing variables t_i of two flow fragments, represented in the image by the variables $\langle \mathsf{dt}, \mathsf{at}, \mathsf{st} \rangle$ and $\langle \mathsf{dt}', \mathsf{at}', \mathsf{st}' \rangle$. They intuitively express the times when packets depart, arrive and are scheduled in network nodes. We can chain the events by simply adding suitable constraints. For example, in Figure 5, the scheduled time of a flow should have the same value of the departure time in the fragment stored in the next hop. This is accomplished by simply adding the constraint: $\mathsf{st} = \mathsf{dt}'$.

Therefore, instead of expressing the properties of the whole flow as a single data-structure, we split flows into Flow Fragments. Recall that the applications assumed here are cyclic and deterministic. This means that in the application period, a bounded number of packets is sent through a flow, and in its corresponding flow fragments. Therefore, flow fragments come with a sequence of time variables, as defined below.

Definition 2. *Flow Fragment:* A flow fragment ff is a tuple $\langle \mathsf{prt}, \mathsf{T} \rangle$, where:

- prt is the priority of the packets of the flow in the port where the fragment is;
- T is a sequence of time variables $\mathsf{T} = [\mathsf{t}_1, \ldots, \mathsf{t}_n]$. Each timing variables t_i represents the departure, arrival and scheduled time of the i-th packet traversing the flow fragment. We use $\mathsf{T}.size$ to denote the size of T.

[3] We abuse slightly of terminology and use packet to also express frames.

The cycle of a port contains the necessary properties to encapsulate all the variables regarding the start, duration and transmission windows on a cycle, as defined below.

Definition 3. *Cycle:* A cycle c is the 4-tuple $\langle \mathsf{SS}, \mathsf{SD}, \mathsf{s}, \mathsf{d} \rangle$, where:

- SS is a mapping from a priority number to a real value, ss, specifying the start point in the cycle of the priority;
- SD is a mapping from a priority number to a real value, sd, specifying the duration of the priority slot;
- s is the start of the first cycle;
- d is the duration of a cycle.

Notice that in the current implementation, each priority can have a single priority slot per cycle.

A switch port contains a cycle, the flow-fragments using the port, as well as specifications on the time required to transmit a packet and to travel to the node connected to the port.

Definition 4. *Port:* A port p of a switch is a 4-tuple $\langle \mathsf{c}, \mathsf{PFF}, \mathsf{travelT}, \mathsf{transT} \rangle$, being the logical representation of a port in the perspective of the scheduling problem.

- c is the cycle of the port;
- PFF is a set of flow fragments that go through that port;
- $\mathsf{travelT}$ is the time taken to travel from this port to its destination and vice-versa;
- transT is the time taken by the port to transfer the packet into the network.

Definition 5. *Switch:* A switch S is a set of ports.

We denote the set of all switches in the network as SW.

A network is a set of flows F composing all the flows contained in the network. A flow is defined as follows.

Definition 6. *Flow:* A flow f is as a 4-tuple $\langle \mathsf{FF}, \mathsf{P}, \rho, \phi \rangle$, where:

- FF is a sequence containing the decomposition of a flow into flow fragments ff in the path order;
- P is a sequence containing the ordered path of egress ports p of switches from the source to destination node of the flow;
- ρ is the periodicity of a flow, that is, the periodicity in which packets are generated by device source of the flow;
- ϕ is the offset of sending, or the sending time of the first packet of the flow.

Notice that the last hop of the packets is not included in the conception of a flow for TSNSCHED (as visualized in Figure 5), since the travel time from the last switch in the path to the end-device if fixed and be added when needed.

With the model defined above, we can establish a set of constraints to shape schedules according to the given rules of the network. TSNSCHED uses *25 types of constraints to specify the conditions of TAS scheduling*, with a few additional optional constraints regarding avoidance of egress interleaving

[9] and reservation of bandwidth for best-effort traffic. Here, we illustrate the core constraints for the transmission of the packets.

For example, we define that the sending offset ϕ of a flow is the moment in time where the first packet is sent. In other words, the departure time of the first fragment of a flow $\mathsf{f.FF}(1).\mathsf{T}(1).\mathsf{dt}$ is equivalent to the flow's sending offset. Also, periodically, a packet is sent according to the periodicity ρ of a flow, meaning that every ρ units of time, another packet will be sent. This is enforced using the *Packet First Departure* constraint (Equation 1).

$$\begin{aligned} &\forall \mathsf{f} \in \mathsf{F};\ i \in Z;\ 0 < i \leq \mathsf{f.FF}(1).\mathsf{T}.size. \\ &\mathsf{f.FF}(1).\mathsf{T}(i).\mathsf{dt} = \mathsf{f}.\phi + \mathsf{f}.\rho \times (i-1) \end{aligned} \tag{1}$$

For a more involved constraint, we shall enforce that the transmission order of packets is kept, as TSN switches do not transmit two packets that are in the same queue in an order different from the arrival order. To do so, we use the *FIFO Priority Queue* constraint (Equation 2).

$$\begin{aligned} &\forall \mathsf{S} \in \mathsf{SW}.\ \forall \mathsf{p} \in \mathsf{S}.\ \forall \mathsf{ff}_1, \mathsf{ff}_2 \in \mathsf{p.PFF};\ i, j \in Z; \\ &0 < i \leq \mathsf{ff}_1.\mathsf{T}.size; 0 < j \leq \mathsf{ff}_2.\mathsf{T}.size; \mathsf{ff}_1 \neq \mathsf{ff}_2 \wedge i \neq j. \\ &\mathsf{ff}_1.\mathsf{T}(i).\mathsf{at} \leq \mathsf{ff}_2.\mathsf{T}(j).\mathsf{at} \wedge \mathsf{ff}_1.\mathsf{prt} = \mathsf{ff}_2.\mathsf{prt} \\ &\implies \mathsf{ff}_1.\mathsf{T}(i).\mathsf{st} \leq \mathsf{ff}_2.\mathsf{T}(j).\mathsf{st} + \mathsf{p.transT} \end{aligned} \tag{2}$$

This constraint ensures that, for every packet, on any flow, its transmission will only happen after the ones that arrived before are transmitted. In a temporal perspective, following the terminology used by TSNSCHED, this order can be kept by making sure that a packet's scheduled time is less than another packet's scheduled time only if its arrival time is also smaller, given that both belong to the same priority.

To ensure that the transmission of a packet will not happen outside of the boundaries of the time window for its priority queue, it must be explicit in the constraints of the scheduler that the transmission of a frame can only happen between the start and end of a slot. To do so, a packet can be scheduled to leave any time between the start of its priority slot plus the transmission time up until the end of a slot according to the *Transmit Inside a Time slot* constraint (Equation 3). The constant $numCycles$ is the maximum number of cycles in an application period. It is computed taking into account the minimum allowed cycle time.

$$\begin{aligned} &\forall \mathsf{S} \in \mathsf{SW}.\ \forall \mathsf{p} \in \mathsf{S}.\ \forall \mathsf{ff} \in \mathsf{p.PFF}, i, j \in Z; \\ &0 < i \leq \mathsf{ff.T}.size; 0 \leq j < numCycles. \\ &\mathsf{ff.T}(i).\mathsf{st} \geq \mathsf{p.c.s} + \mathsf{p.c.d} \times j + \mathsf{p.c.SS}(\mathsf{ff.prt}) + \mathsf{p.transT}\ \wedge \\ &\mathsf{ff.T}(i).\mathsf{st} \leq \mathsf{p.c.s} + \mathsf{p.c.d} \times j + \mathsf{p.c.SS}(\mathsf{ff.prt}) + \mathsf{p.c.SD}(\mathsf{ff.prt}) \end{aligned} \tag{3}$$

For example, lets assume a packet is transmitted in a time window that begins at the 20th microsecond of a cycle and ends at the 30th microsecond of the same cycle and that the transmission time of that packet in that port takes 5 microseconds. Then the scheduled time of that packet must fall between 25 and 30 microseconds of that cycle.

Other much more involved constraints ensure, for example, that there will be no gaps between the gate opening and the transmission process of a packet.

Due to the size of this constraint, it has been broken into 4 predicates *Send Aft Another Packet*, *Arr Bef Slot Start*, *Arr In Slot* and *Arr Aft Slot End*, depicted, respectively, by Equations 5, 6, 7, and 8. These sub-formulas form the *Send As Soon As Possible* constraint shown in Equation 4.

$$\begin{aligned} &\forall \mathsf{S} \in \mathsf{SW}.\ \forall \mathsf{p} \in \mathsf{S}.\ \forall \mathsf{ff}_1, \mathsf{ff}_2 \in \mathsf{p.PFF}. \\ &SendAftAnotherPacket\ \vee \\ &(ArrBefSlotStart \wedge ArrInSlot \wedge ArrAftSlotEnd) \end{aligned} \tag{4}$$

The $SendAftAnotherPacket$ predicate (Equation 5) specifies that the scheduled time of every packet must be equal to the scheduled time of another packet plus the transmission time. This predicate covers cases where a packet *i* arrives and there is another packet *j* in its priority queue, which is verified by comparing the arrival time of *i* and the scheduled time of *j*. The *FIFO Priority Queue* constraint (Equation 2) guarantees that *i* will be properly transmitted in its turn in case there are multiple packets in the queue.

$$\begin{aligned} &\exists i, j \in Z;\ \mathsf{ff}_1 \neq \mathsf{ff}_2 \wedge i \neq j; \\ &0 < i \leq \mathsf{ff}_1.\mathsf{T}.size;\ 0 < j \leq \mathsf{ff}_2.\mathsf{T}.size. \\ &(\mathsf{ff}_1.\mathsf{prt} = \mathsf{ff}_2.\mathsf{prt} \wedge \mathsf{ff}_1.\mathsf{T}(i).\mathsf{at} > \mathsf{ff}_2.\mathsf{T}(j).\mathsf{at})\ \wedge \\ &\mathsf{ff}_1.\mathsf{T}(i).\mathsf{st} = \mathsf{ff}_2.\mathsf{T}(j).\mathsf{at} + \mathsf{p.transT} \end{aligned} \tag{5}$$

The $ArrBefSlotStart$ predicate (Equation 6) specifies that if a packet arrives before the beginning of a time slot, it will be transmitted as soon as the slot starts.

$$\begin{aligned} &\forall i, j \in Z;\ 0 < i \leq \mathsf{ff}_1.\mathsf{T}.size;\ 0 \leq j < numCycles. \\ &((\mathsf{ff}_1.\mathsf{T}(i).\mathsf{at} < \mathsf{p.c.SS}(\mathsf{ff}_1.\mathsf{prt}) + \mathsf{p.c.s} + \\ &\mathsf{p.c.d} \times j) \wedge (\mathsf{ff}_1.\mathsf{T}(i).\mathsf{at} \geq \mathsf{p.c.s} + \mathsf{p.c.d} \times j)) \implies \\ &\quad \mathsf{ff}_1.\mathsf{T}(i).\mathsf{st} = \mathsf{p.c.SS}(\mathsf{ff}_1.\mathsf{prt}) + \mathsf{p.c.s}\ + \\ &\quad\quad \mathsf{p.c.d} \times j + \mathsf{p.transT} \end{aligned} \tag{6}$$

The $ArrInSlot$ predicate (Equation 7) specifies that if a packet arrives during a time slot and there is enough time to transmit, it must be transmitted immediately.

$$\begin{aligned} &\forall i, j \in Z;\ 0 < i \leq \mathsf{ff}_1.\mathsf{T}.size;\ 0 \leq j < numCycles. \\ &((\mathsf{ff}_1.\mathsf{T}(i).\mathsf{at} \leq \mathsf{p.c.SS}(\mathsf{ff}_1.\mathsf{prt}) + \mathsf{p.c.SD}(\mathsf{ff}_1.\mathsf{prt})\ + \\ &\mathsf{p.c.s} + \mathsf{p.c.d} \times j - \mathsf{p.transT}) \wedge (\mathsf{ff}_1.\mathsf{T}(i).\mathsf{at} \geq \\ &\mathsf{p.c.SS}(\mathsf{ff}_1.\mathsf{prt}) + \mathsf{p.c.s} + \mathsf{p.c.d} \times j)) \implies \\ &\mathsf{ff}_1.\mathsf{T}(i).\mathsf{st} = \mathsf{ff}_1.\mathsf{T}(i).\mathsf{at} + \mathsf{p.transT} \end{aligned} \tag{7}$$

The $ArrAftSlotEnd$ predicate (Equation 8) specifies that if a packet arrives during a slot and there is no time to transmit or after the slot, it will be transmitted at the beginning of the next slot in the next cycle.

$$\begin{aligned} &\forall i, j \in Z;\ 0 < i \leq \mathsf{ff}_1.\mathsf{T}.size;\ 0 \leq j < numCycles. \\ &((\mathsf{ff}_1.\mathsf{T}(i).\mathsf{at} > \mathsf{p.c.SS}(\mathsf{ff}_1.\mathsf{prt}) + \\ &\mathsf{p.c.SD}(\mathsf{ff}_1.\mathsf{prt}) + \mathsf{p.c.s} + \mathsf{p.c.d} \times j - \mathsf{p.transT}) \wedge \\ &(\mathsf{ff}_1.\mathsf{T}(i).\mathsf{at} < \mathsf{p.c.s} + \mathsf{p.c.d} \times (j+1))) \\ &\quad \implies \mathsf{ff}_1.\mathsf{T}(i).\mathsf{st} = \mathsf{p.c.SS}(\mathsf{ff}_1.\mathsf{prt}) + \\ &\quad\quad \mathsf{p.c.s} + \mathsf{p.c.d} \times (j+1) + \mathsf{p.transT} \end{aligned} \tag{8}$$

By defining a constraint which guarantees the order of transmission of the packets according to its arrival time, one of the implications in $ArrBefSlotStart$, $ArrInSlot$ or $ArrAftSlotEnd$ will have its left side evaluates to true. Therefore, one of the predicates will be true.

Equation 4 then specifies that a packet is either sent after another which is in the queue, or immediately after arriving if there is enough time or at the beginning of a time slot if it arrived too late or too early.

A. Network Requirements Specification

To enforce the user-defined maximum latency, a constraint can be created specifying that the difference between the last scheduled time and first departure time must be smaller than a maximum latency *maxLatency*. Since it is considered that the time to reach a device is fixed and defined in the port by the user, there is no need to add the travel time to the last scheduled time in this equation. Given that each flow is broken into *n* fragments, this is enforced by the *Maximum Allowed Latency* constraint (Equation 9).

$$\begin{aligned} &\forall \mathsf{f} \in \mathsf{F}; i \in Z.\ 0 < i \leq \mathsf{f.FF}(1).T.size. \\ &maxLatency \geq \mathsf{f.FF}(n).\mathsf{T}(i).\mathsf{st} - \mathsf{f.FF}(1).\mathsf{T}(i).\mathsf{dt} \end{aligned} \tag{9}$$

With the latencies of the packets of a flow, the constraint for maximum jitter per packet of a flow can be created. To do so, a constraint must specify that the difference between the latency of a packet and the average latency of a flow f ($AvgLatency(\mathsf{f})$) must not be greater than the maximum jitter *maxJitter*, as shown in the *Maximum Allowed Jitter* constraint.

$$\begin{aligned} &\forall \mathsf{f} \in \mathsf{F}; i \in Z.\ 0 < i \leq \mathsf{f.FF}(1).T.size. \\ &maxJitter \geq \\ &|(\mathsf{f.FF}(n).\mathsf{T}(i).\mathsf{st} - \mathsf{f.FF}(1).\mathsf{T}(i).\mathsf{dt}) - AvgLatency(\mathsf{f})| \end{aligned} \tag{10}$$

These two constraints can be exemplified using a scenario with a flow that sends three packets. If the maximum allowed latency for this flow is 500 μs and the maximum allowed jitter is 25 μs, a solution where the latencies of the packets are equals to, respectively 495, 500 and 505 microseconds wouldn't be valid, as the latency of the third packet would disrespect the *Maximum Allowed Latency* constraint, even if the jitter has a value of 5. Similarly, if the latencies of the packets are equal to, respectively 400, 450, and 500, even though these values respect the *Maximum Allowed Latency* constraint, the average latency variation of the packets 1 and 3 are equals to 50 microseconds, disrespecting the *Maximum Allowed Jitter* constraint.

Regarding the usage of Z3 and its offered theories, TSNsched only uses Linear Arithmetic theory. Although we use quantifiers in the formal model, they are flattened in the implementation to improve efficiency.

B. TSNSCHED *Expressiveness*

We demonstrate the expressiveness of our encoding. As illustrated above, the encoding of the TSN scheduling problem is complicated, as there is a large number of aspects to consider. Therefore, finding suitable abstractions for both the formal model and implementation, such as, Flow Fragments, helps not only for the understandability of the constraints, but also allows to express more conditions and properties, when compared to other encodings such as [7], [11], [10]. This is also reflected in the tool implementation, leading to more maintainable code. We illustrate TSNSCHED's expressiveness below.

1) Avoid starvation of best-effort traffic: While most of the work on scheduling TSN traffic focuses on aspects regarding priority traffic, it is of great importance to consider the best-effort traffic transmitted through the TSN switches. This is because the focus on the transmission of priority packets might damage the QoS of applications using non-priority traffic [14].

For instance, consider a vehicle using TSN compliant equipment is to transfer data between sensors and cameras using best-effort traffic to transfer data from cameras to a screen. If not enough bandwidth is reserved for such traffic, a jittery and delayed video might be presented at the vehicle's output screen.

With TSNSCHED's modular design, users can configure how much of a cycle must remain free for non-priority traffic without affecting the timing variables of the flows' packets. One can easily add a constraint for this purpose. In particular, we simply constraint the sum of the size of the slots as a proportion of the cycle duration (see Definition 3). The remaining proportion will be available for best-effort traffic.

2) Implementation of unicast and multicast flows: As stated in [9], regarding the formal modeling of unicast flows, it is a trivial step to generalize such model to work with multicast flows. Still, this simple generalization of the model can entail in implementation problems as the flows will have to be individually specified. Convoluted solutions may lead to the creation of additional formulas that the solver must handle.

By definition, the fragment of a flow contains the necessary information regarding the timing of packets and the priority of a flow in one switch. Hence, this low level encapsulation of the flows allows for the simplification of a multicast flow. If the same packet from a flow is sent from a switch to multiple switches, the fragments created in these switches only need to be linked to the previous fragment without transforming multicast flow into multiple bound unicast flows.

3) TSN Scheduling Problem Variants: With respect to the work of [9] and of [10], [11], there are some important differences in the encoding. On the one hand, [9] does not explicitly encode TSN time windows, thus not allowing to reason about its properties, but focus on the frame transmission constraints. On the other hand, [10], [11] encodes TSN time windows, but not the frame transmission time constraints.

By having control of the individual fragments of the flows and the cycles of each switch, TSNSCHED is able to specify conditions on both TSN time windows (SS and SD in Definition 3) and frame transmission time (ϕ in Definition 6), even mixing the two types.

One difference, however, is that in the current implementation only one gate of a queue can be opened per priority in a cycle, whereas [10], [11] allows for multiple slots per priority per cycle. As described above, we can easily accomodate this change, but given our experiments, we did not need to do so.

Whilst not currently available, the iterative approach to solving the problem presented in [9], in order to improve the scheduler's execution time, can be implemented by continuously adding flows to the solver. Since our experimental results are adequate, we did not further investigate this option.

Finally, it should be possible to extend the class Flow Fragment, specifying the translation to Z3, in order to generate other types of TSN mechanisms, such as, the Asynchronous Traffic Shaper [4] or Cyclic Queuing and Forwarding [3]. It is also possible to combine the different schedulers in a convergent network. An example of standard outside of the TAS's boundaries which can be used by TSNSCHED is the Per-Stream Filtering and Policing standard (IEEE 802.1Qci/802.1CB), allowing packets from a flow to have different priorities in different hops [8]. For the scenarios considered in the tool evaluation presented in Section V, the schedules implement Per-Stream Filtering and Policing.

V. Experimental Evaluation

For all the scenarios considered here, we assume a network with 10 TSN switches and 50 physical devices. We also assume that each flow generates 5 packets in an application period. We generate a number of Publish Subscribe flows using the topology generator and classify these flows according to their size and their packet periodicity. We used the following three different size flows, each with one publisher:

- **Small Flows**, containing 3 switches and at most 5 subscribers;
- **Medium Flows**, containing 5 switches and at most 10 subscribers;
- **Large Flows**, containing 7 switches and at most 15 subscribers.

The larger the flow, the harder is the scheduling as the depth of the flow and the number of subscribers increase.

High network performance applications have flows sending packets with periodicity between 100μs and 2000μs [5]. We consider here the following types of flows:

- **Normal Performance Flows**, which send packets with periodicity of 2000μs;
- **High Performance Flows**, which send packets with periodicity of 1000μs.

For each flow, we considered a maximum latency of 1000μs and maximum jitter of 25μs, which are adequate for the type of network flow performance described above.

Unfortunately, there are no available benchmarks for the analysis of schedules for TSN networks. A realistic example for a distributed industrial application is the VDMA R+A demonstrator described in [6]. While the exact topology is not provided, this demonstrator contains 26 nodes and 28 unicast flows, including best-effort traffic. The size of this application is in the same order as the size of our scenarios. Therefore, TSNSCHED would, in principle, be capable of generating schedules for the VDMA R+A Demonstrator.

a) Evaluation: Table I summarizes our experimental results, which were carried out on 8 virtual cores of a Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz processor with 16 Gb DDR4 memory. For each scenario (i.e., small, medium, or large flows), we ran TSNSCHED assuming each flow is of Normal and High performance. We evaluated the average jitter, average latency and TSNSCHED's execution time. We set a timeout of 80 hours for our experiments (marked with TO in the table). Since schedules typically need to be generated only once, such long execution times are acceptable in practice.

Overall TSNSCHED performed well. It was able to construct schedules that satisfy the network performance criteria (maximum latency of 1000μs per packet and maximum jitter of 25μs per packet per flow) for networks with up to 73 subscribers and up to 10 publishers. We were also positively surprised by the overall average jitter which in most cases was well below the maximal admissible jitter. Even in the larger experiments, the generated schedule could guarantee an average jitter of 4.85μs (high performance, 3 medium flows). For a comparison, the average jitter in experiments with unicast flows described in [16] was between 10μs and 50μs.

Furthermore, for the normal performance scenarios, the latency was much lower than the maximum latency per flow (1000μs). For the scenarios with high performance flows, the average latency was still acceptable, but closer to the maximum admissible latency.

Less surprising was the fact that TSNSCHED had more difficulties in generating schedules for scenarios with larger flows. This was expected as the complexity of the scheduling problem increases with the size of flows, *e.g.*, number of switches to be configured.

The execution time of the experiments varied also as expected, following an exponential pattern, as depicted by Figures 6.(a), 6.(b), and 6.(c). The generation of schedules for large flows also required much more time than for smaller flows, which was also expected. Interestingly, TSNSCHED required more time to generate schedules for normal than for high performance flows. It is hard to understand exactly why this happened, but we believe it is related to the range of possible results available for normal performance flows, leading to an increased search tree.

VI. Related Work and Conclusions

Up to the best of our knowledge, TSNSCHED is the first openly available tool for generating TSN schedules. The tool described in [11] is proprietary. We have been inspired, however, by the foundational work carried out in [9] and [10] in that they reduce the TSN scheduling problem to Z3. [9] does not explicitly encode TSN time windows, thus not allowing to reason about its properties. [10] and [16] encodes TSN time windows, but not the frame transmission time constraints (see section IV-B). TSNSCHED encodes both the transmission of individual packets and the time windows in which the packets

Normal Performance Flows (Periodicity = 2000μs)

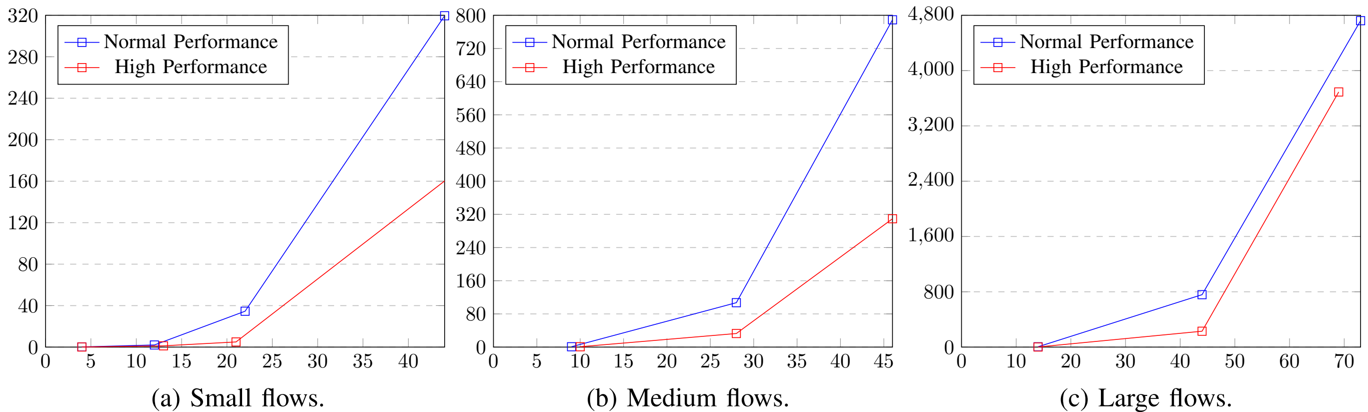
No of Flows	1	3	5	10
Small Only	0.68μs/509.55μs (4)	4.13μs/689.69μs (12)	4.54μs/631.54μs (22)	3.01μs/840.77μs (44)
Medium Only	1.38μs/887.18μs (9)	3.18μs/891.93μs (28)	4μs/653.78μs (46)	TO (93)
Large Only	2.75μs/972.25μs (14)	4.85μs/621.16μs (44)	3.64μs/755.73μs (73)	TO (147)

High Performance Flows (Periodicity = 1000μs)

No of Flows	1	3	5	10
Small Only	2.51μs/997.07μs (4)	1.79μs/903.69μs (13)	2.83μs/751.49μs (21)	1.95μs/946.43μs (45)
Medium Only	3.55μs/957.18μs (10)	2.81μs/797.70μs (28)	4.75μs/807.82μs (46)	TO (91)
Large Only	1.07μs/993.55μs (14)	2.64μs/860.32μs (44)	3.02μs/936.40μs (69)	NA

TABLE I

SUMMARY OF EXPERIMENTAL RESULTS. THE RESULTS ARE EXPRESSED AS jit / lat (sub), WHERE jit IS THE AVERAGE JITTER, lat IS THE AVERAGE LATENCY AND sub IS THE NUMBER OF SUBSCRIBERS IN THE SCENARIO. WE MARK WITH TO THE EXPERIMENTS FOR WHICH TSNSCHED DID NOT RETURN A RESULT WITHIN 80 HOURS AND NA IF THE EXPERIMENT WAS NOT PERFORMED.



(a) Small flows. (b) Medium flows. (c) Large flows.

Fig. 6. Execution time in minutes (y-axis) per number of subscribers (x-axis).

are placed, giving the user a wide range of configuration parameters to comply with the desired results. Also, they consider unicast flows, while we also extend TSNSCHED by multicast flows. This has a great impact on implementation and experimental results. In particular, to encode multicast flows in Z3 in a modular fashion, we use flow fragments as basic components for flows. This leads to a more readable, modular and modifiable implementation. [16] mentions that the adaptation of their model to work with multicast flows is trivial, but is important to notice that the translation of this model to code can be more challenging than initially thought, as details in the implementation can lead to greater execution times and wastage of resources. Moreover, in contrast to [16], [10], we evaluate our tool on Publish Subscribe flows.

Finally, [13] proposes a graphical approach for modeling the network as input and methods for generating schedules from unsatisfiable cores in Z3. It has a very similar structure of solution as the previous mentioned work. This work seem complementary as it seems possible to use TSNSCHED with the proposed graphical approach.

It is also important to notice that the previously mentioned work does not take in consideration any form of waiting for transmission of packets (except if a packet arrives and there is another in the transmission queue). The solver always attempt to wrap the transmission of packets inside the arrival of a packet and its transmission. This may lead to a more limited set of possible answers for the scheduling problem. TSNSCHED considers the scheduling of packets that may arrive outside of its transmission windows or during the transmission of other packets, creating more flexible schedules.

Aside from the Time-Aware Shaper to which TSNSCHED was design for, other schedulers can be used to handle the transmission of packets aiming to comply with performance requirements on TSN networks, but, if used in their out-of-the-box state, they might fail to meet the requirements in time-critical applications [17].

This paper describes the tool TSNSCHED, the first openly accessible tool that generates schedules for high performance deterministic cyclic Time Sensitive Networks. We present the first experimental results for schedule generation for multicast flows, evaluation our tool on relatively large networks (up to 73 subscribers) with varying flow characteristics (number, size and performance). TSNSCHED generated schedules with acceptable network performance (jitter and latency).

As future work, we are currently integrating TSNSCHED into the 4diac framework for modelling industrial automation systems. We are also currently investigating techniques for further improving TSNSCHED's performance by exploiting the shape of the network, *e.g.*, whether flows share switches or not leading to divide and conquer methods.

REFERENCES

[1] IEEE 802.1ASRev - Timing and Synchronization for Time-Sensitive

Applications. Available online at https://1.ieee802.org/tsn/802-1as-rev/ last accessed on October 30th 2018.
[2] IEEE 802.1Qbv - Enhancements for Scheduled Traffic. Available online at http://www.ieee802.org/1/pages/802.1bv.html last accessed on October 30th 2018.
[3] IEEE 802.1Qch Cyclic Queuing and Forwarding . Available online at https://1.ieee802.org/tsn/802-1qch/ last accessed on October 30th 2018.
[4] IEEE 802.1Qcr - Bridges and Bridged Networks Amendment: Asynchronous Traffic Shaping. Available online at https://1.ieee802.org/tsn/802-1qcr/ last accessed on October 30th 2018.
[5] A. Ademaj, D. Puffer, D. Bruckner, G. Ditzel, L. Leurs, M. Stanica, P. Didier, R. Hummen, R. Blair, and T. Enzinger. Iic results white paper: Time sensitive networks for flexible manufacturing testbed - description of converged traffic types, April 2018.
[6] B. Brandenbourger and F. Durand. Design pattern for decomposition or aggregation of automation systems into hierarchy levels. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 895–901. IEEE, 2018.
[7] S. S. Craciunas and R. S. Oliver. Combined task- and network-level scheduling for distributed time-triggered systems. *Real-Time Systems*, 52(2):161–200, Mar 2016.
[8] S. S. Craciunas, R. S. Oliver, and T. C. AG. An overview of scheduling mechanisms for time-sensitive networks. *Proceedings of the Real-time summer school LÉcole dÉté Temps Réel (ETR)*, 2017.
[9] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner. Scheduling real-time communication in ieee 802.1 qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 183–192. ACM, 2016.
[10] S. S. Craciunas, R. S. Oliver, and W. Steiner. Formal scheduling constraints for time-sensitive networks. *CoRR*, abs/1712.02246, 2017.
[11] S. S. Craciunas, R. S. Oliver, and W. Steiner. Demo abstract: Slate xns–an online management tool for deterministic tsn networks. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 103–104, April 2018.
[12] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *TACAS*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.
[13] M. H. Farzaneh, S. Kugele, and A. Knoll. A graphical modeling tool supporting automated schedule synthesis for time-sensitive networking. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2017.
[14] V. Gavriluţ and P. Pop. Scheduling in time sensitive networks (tsn) for mixed-criticality industrial applications. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–4. IEEE, 2018.
[15] R. Hummen, S. Kehrer, and O. Kleineberg. White paper: Tsn-time sensitive networking. *Belden, St. Louis, MI, USA, Tech. Rep*, 2017.
[16] W. Steiner, S. S. Craciunas, and R. S. Oliver. Traffic planning for time-sensitive communication. *IEEE Communications Standards Magazine*, 2(2):42–47, JUNE 2018.
[17] S. Thangamuthu, N. Concer, P. J. Cuijpers, and J. J. Lukkien. Analysis of ethernet-switch traffic shapers for in-vehicle networking applications. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 55–60. EDA Consortium, 2015.