2019 International Conference on Networked Systems (NetSys)

NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++

Jonathan Falk[*], David Hellmanns[*], Ben Carabelli[*], Naresh Nayak[*], Frank Dürr[*], Stephan Kehrer[‡], Kurt Rothermel[*]

[*] University of Stuttgart
Institute of Parallel and Distributed Systems
70569 Stuttgart, Germany
{*firstname.lastname*}@ipvs.uni-stuttgart.de

[‡] Hirschmann Automation and Control GmbH
CTO Office
72654 Neckartenzlingen, Germany
stephan.kehrer@belden.com

***Abstract*—IEEE 802.1 Time-sensitive Networking (TSN) enables real-time communication with deterministically bounded network delay and jitter over standard IEEE 802.3 networks ("Ethernet"). In particular, TSN specifies a time-triggered scheduling mechanism in IEEE Std 802.1Qbv implemented by switches to control when outgoing queues get access to switch ports. Besides this time-triggered scheduling mechanism, other scheduling mechanisms can be active in the network at the same time including priority queuing and a credit-based shaper. Moreover, further supporting mechanisms such as the possibility to interrupt frames already in transmission (frame preemption) are specified by the TSN standards. Overall, this leads to a complex network infrastructure transporting both, real-time and non-real-time traffic in one converged network, making it hard to analyze the behavior of converged networks.**

To facilitate the analysis of TSN networks, we present TSN-specific extensions to the popular OMNeT++/INET framework for network simulations in this paper including, in particular, the time-triggered scheduling mechanism of IEEE Std 802.1Qbv. Besides the design of the TSN simulator, we present a proof-of-concept implementation and exemplary evaluation of TSN networks.

***Index Terms*—Time-sensitive Networking (TSN), real-time communication, network simulator, scheduling, quality of service**

I. Introduction

Cyber-physical Systems (CPS), where computers control processes of the physical world, have become ubiquitous. Many of these CPS are real-time systems requiring the ability to react within given deadlines to events from the physical world. One prominent application domain of such *time-sensitive CPS* is the Industrial Internet of Things (Industry 4.0), for instance, an emergency stop function triggered by sensors detecting personnel within the working area of a machine, or motion control of robots equipped with networked sensors. Another application domain are automotive systems utilizing various sensors such as cameras and radar to implement, for instance, automated breaking assistants and ultimately enable autonomous self-driving cars.

J. Falk, D. Hellmanns, and B. Carabelli contributed equally to this work.

Typically CPS are distributed systems connecting sensors, controllers, and actuators via a communication network. Consequently, to meet the real-time requirements of time-sensitive CPS, the communication network must be able to deliver messages (frames/packets) with bounded delay and delay variation (jitter). Considering the safety-critical applications mentioned above, network delay and jitter often must be bounded *deterministically*, i.e., bounds must be strictly guaranteed ("hard" real-time communication).

Due to their high relevance, network technologies with deterministic real-time guarantees have been available for quite some time. In particular, so-called field buses such as SERCOS III, EtherCAT, or PROFINET, all based on the popular Ethernet technology, provide deterministic delay guarantees. However, one major problem of these field bus technologies is their mutual incompatibility. To solve this problem, the *IEEE 802.1 Time-sensitive Networking (TSN) Task Group* of the Institute of Electrical and Electronics Engineers (IEEE) has specified standards for IEEE 802.3 networks enabling deterministic real-time communication over standard Ethernet. These standards will enable converged networks transporting the whole range of traffic classes—from best-effort to deterministic real-time traffic—with a single network technology. Together with the high network speeds supported by IEEE 802.3 networks, these networks have the potential to largely simplify network infrastructures and at the same time boost the performance of networked systems by transporting large bulk data and real-time traffic over the same network infrastructure.

At the heart of the IEEE 802.1 TSN standards are various scheduling algorithms. In particular, the IEEE Std 802.1Qbv specifies a time-triggered gating mechanism controlling when traffic belonging to different traffic classes will be transmitted by network elements (switches). Packets buffered in a queue associated with an (egress) port can only be transmitted by a switch if the gate of this queue is open. The opening and closing of gates are controlled by a time schedule, which can be configured either centrally or through a distributed mechanism. Different algorithms for calculating gate schedules have been proposed in the literature [1]–[4].

Besides the timed gating mechanism, further scheduling mechanisms can be active *at the same time*. For instance, priority scheduling can be used to select packets if the gates of multiple queues belonging to the same egress port are open at the same time according to queue priority. Moreover, a credit-based shaper has been specified, which performs per-hop, per-traffic class/queue traffic shaping. In addition, supporting mechanisms such as frame preemption are specified to interrupt lower priority frames that are already in transmission.

Overlaying the effects of all these mechanisms leads to a complex network behavior making it hard to predict the quality of service to be expected for each flow. For instance, one might ask the following questions about the converged network: How does the time-triggered gate schedule of deterministic real-time traffic impact the performance of a lower priority best-effort TCP flow or a multimedia flow running in addition to the timed gating mechanism through the credit-based shaper? How do inevitable adverse effects such as inaccurate time synchronization of switches and hosts impact the delay and jitter, e.g., during failures of the master time server or in different configurations of the time synchronization protocol (Precision Time Protocol (PTP, IEEE Std 1588), IEEE Std 802.1AS)? Three methods can be used to answer such questions: experiments in a real network, formal network analysis, and network simulation. Experiments are often limited to small topologies due to cost. Formally analyzing the network performance using, for instance, Network Calculus or queuing theory is very hard since it requires detailed and accurate models of all mechanisms (timed gating mechanism, credit-based shaping, frame-preemption, TCP, etc.), traffic (load), etc. Moreover, frameworks such as the Network Calculus focus on a worst case analysis, which is well-suited for analyzing deterministic real-time traffic, but not specifically targeting other important traffic classes such as best-effort TCP traffic.

Therefore, we focus on network simulation in this paper. We present a simulation framework to analyze the behavior of converged IEEE 802.1 TSN networks called NeSTiNg (Network Simulator for Time-Sensitive Networking). NeSTiNg is based on the popular OMNeT++/INET discrete event simulation framework. Therefore, it directly benefits from the features already available in OMNeT++/INET. OMNeT++ [5] provides a core simulation framework written in C++, and utilities such as an Eclipse-based IDE, a tool for analyzing the recorded simulation traces, an interactive graphical visualization framework to interact with the simulation, and its own domain-specific language (NED) for composing simulation models. OMNeT++ is free-to-use in noncommercial settings under a GPL-like license. Besides the tooling provided by OMNeT++ itself, there exists an ecosystem of projects (including this one), which provides additional simulation model components. Similar to CoRE4INET [6], our simulation models leverage components of the INET [7] model suite, such as simulation models of the TCP protocol and basic switch models, and is developed with the aim to be interoperable with INET components. Our original contribution is to extend the existing framework with simulation models of the most essential TSN features, namely, timed gating mechanism (IEEE Std 802.1Qbv), frame preemption (IEEE Std 802.1Qbu and IEEE Std 802.3br), and credit-based shaper (IEEE Std 802.1Qav). We will show how we implemented and integrated these mechanisms into the simulation framework. Moreover, we will show an evaluation of the performance characteristics and scaling behavior of NeSTiNg on a number of proof-of-concept scenarios that illustrate the overhead of simulating the TSN features covered by our simulation model. The implementation of NeSTiNg is open source and available at [8].

The rest of this paper is structured as follows. In Section II, we give an overview of related work, before we explain the background of our work including a short introduction to the TSN standards in Section III. In Section IV, we present the design and implementation of our TSN-extensions to the OMNeT++/INET network simulation framework. Finally, we present a proof-of-concept implementation and performance evaluation in Section V, before we finally conclude the paper in Section VI.

II. Related Work

In the following section, we introduce research regarding the simulation of TSN, its predecessor AVB, and further Ethernet-based real-time extensions. Steinbach et al. [9] extended the well-established INET framework by adding a model for Time-Triggered Ethernet (TTEthernet). This contribution is available as open-source in the CoRE4INET framework [6]. The authors show that the overhead of their model is reasonable and that the model's behavior conforms to real hardware. However, TTEthernet is a proprietary extension of Ethernet and non-conformant with the IEEE 802.1 TSN standards.

Jiang et al. [10] presented a TSN simulation model for OMNeT++ based on the CoRE4INET framework that adds functionality of IEEE Std 802.1Qbv for traffic scheduling and IEEE Std 802.1AS for time synchronization. To show the validity of their model, the authors evaluated an exemplary scenario. The simulation results of the evaluated scenario show that scheduled traffic is unaffected by best effort transmissions. However, the authors did not make their code freely available, which rules out a straight-forward comparison with NeSTiNg.

Meyer et al. [11] built a module for AVB traffic shaping augmented with time-triggered transmission to examine the mutual influence of these mechanisms. At the time of publication, the IEEE Std 802.1Qbv was not available. Therefore, the authors developed their own scheme for time-triggered transmissions, which is non-conformant to the final version of IEEE Std 802.1Qbv. Nevertheless, Meyer et al. present valuable insights regarding the interaction of time-triggered transmission and the Credit-based Shaper (CBS).

Heise et al. [12] proposed TSimNet, which is an OMNeT++ simulation model. TSimNet focuses on the non-time-based components of the TSN mechanisms and, thus, does not cover the important time-based parts of TSN.

Pahlevan et al. [13] present a TSN module for OPNET that implements IEEE Std 802.1Qbv and IEEE Std 802.1Qci. However, the implementation is not publicly available impeding the usage and extension of the framework by the networking

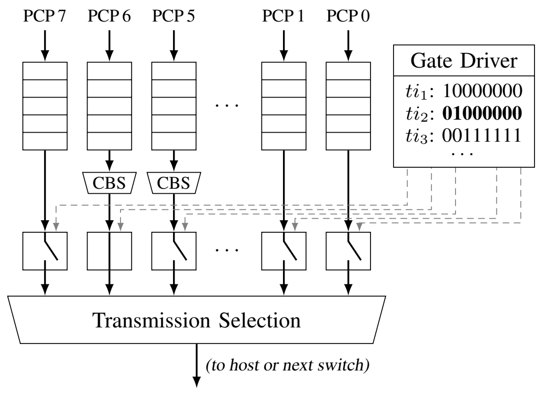

Fig. 1. Gating and transmission selection architecture for one port of a switch complying with IEEE Std 802.1Qbv. The figure shows a situation at some time within the time interval with duration ti_2, where the second gate schedule entry is applied, i.e., only the gate for queue 6 is open. Additionally, the traffic of queues 5 and 6 is shaped using the Credit-based Shaper.

community as well as comparison to our TSN simulator with respect to performance and accuracy.

III. TSN Background

Before we present the technical details of our TSN simulator, we first give an overview of the various mechanisms specified as part of the TSN standards. Besides providing the technical background required to understand the simulator presented in the next section, we would like to highlight the complexity of a TSN network making it non-trivial to predict the effects on forwarded traffic and thereby motivating the need for a simulator tool.

First of all, TSN does not only include a single but multiple scheduling mechanisms (cf. Figure 1). The most important one to ensure low and deterministic network delay bounds and jitter is the so called *Time-aware Shaper* specified in IEEE Std 802.1Qbv "Enhancements for Scheduled Traffic" [14]. This scheduler uses so-called gates to control when (FIFO) queues can transmit buffered packets over an egress switch port. A switch might implement up to eight queues for egress traffic per port. Each packet is tagged with a three-bit Priority Code Point (PCP) that is part of the VLAN tag, and the PCP is mapped to a corresponding queue in which the forwarded frame is enqueued during forwarding. Buffered frames from a queue can only be transmitted over a port, if the gate associated with this queue is open. The gate schedule, called Gate Control List, defines when gates open and close. To this end, the clocks of all switches are synchronized using the Precision Time Protocol (PTP, IEEE Std 1588) [15] or the (very similar) synchronization protocol defined in IEEE Std 802.1AS [16]. The gate schedule contains the time duration for how long the associated set of states for the gates is applied (cf. Figure 1, second entry: all gates are closed except gate for queue 6). Every time the gate driver progresses to the next entry in the gate schedule, the gates where the state differs are opened or closed accordingly (cf. Figure 1 from second to third entry: the gate in front of queue 6 is closed, gate in front of queue 5 to queue 0 are opened). This operation is cyclic, i.e., the schedule is reset to the first entry in an endless loop after the cycle time. Similar to the paradigm of logically centralized control known from Software-defined Networks (SDN), the gate schedules of switches can be configured either from a Centralized Network Controller (CNC) talking to the switch via the SNMP (Simple Network Management Protocol), NETCONF, or RESTCONF protocol. Algorithms to calculate the gate schedules are out of the scope of the standard. Different algorithms for calculating the schedules have been proposed in the literature [1]–[4], [17], typically resulting in complex constraint satisfaction and optimization problems to meet delay and jitter bounds, optimize network utilization or similar. It is important to realize that multiple gates might be open at the same time! In that case, another scheduling algorithm decides, which of the queues with open gates is eligible for transmitting frames. In the simplest case, priority queuing as specified in IEEE Std 802.1Q can be used to this end.

With the Time-aware Shaper the question comes up what happens if the gate closes during the transmission of a frame? This would result in frames reaching into the gate open intervals of other queues, possibly delaying frames beyond the desired bounds. One solution to avoid this problem altogether is to use a length-aware scheduler, which checks *before* starting the frame transmission, whether the transmission will finish before the gate closes. However, this solution has its limitations since the length of the frame must be known before transmission, which is trivial for store-and-forward operation but hard for "cut-through" switching, which starts forwarding already before the complete frame has been received to reduce forwarding latency. Therefore, as an alternative to length-aware scheduling, guard bands between closing one gate and opening another gate can be introduced.

Another scheduling mechanism that can be active in addition to the Time-aware Shaper is the *Credit-based Shaper*, which was specified already before the Time-aware Shaper for audio/video bridging (AVB) in IEEE Std 802.1Qav "Forwarding and Queuing Enhancements for Time-Sensitive Streams" [14]. The Credit-based Shaper introduces two traffic classes called Class A and Class B, each mapped to a different egress queue (cf. Figure 1). The Credit-based Shaper performs traffic shaping per hop/per class, i.e., on each AVB queue, to smooth-out bursts, which could otherwise increase the delay beyond desired bounds. While packets wait in a queue for transmission, the queue can build up credit with at a certain rate called the *idle slope*. While frames are transmitted from a queue, the queue's credit is decreased with a rate called *send slope*. A frame is eligible for transmission, if the credit of its queue is non-negative regardless of the frame size. Positive queue credit is reset to zero if no frames are waiting for transmission in this queue. Calculating delay bounds for traffic running through the Credit-based Shaper requires complex formal analysis [18]–[20] and often will be higher than what is possible with the Time-aware Shaper. Thus, for deterministic real-time traffic, the Time-aware Shaper is better suited.

Another mechanism in TSN reducing latency for high priority frames is *Frame preemption* which has been specified in IEEE Std 802.1Qbu [14] (with corresponding extensions to

Ethernet MAC specified in IEEE Std 802.3br [21]) allowing for interrupting and later resuming lower-priority frames that are already in transmission by higher priority frames. Frame preemption can interrupt frames of a minimum size of 127 B since every frame fragment must be at least 64 B long. Therefore, frame preemption reduces the worst case delay for high priority frames caused by links blocked by lower priority frames in transmission.

As an alternative to the Time-aware Shaper, the Asynchronous Traffic Shaper (previously called "Urgency-based Shaper") has been proposed [22] to provide deterministic bounds on delay. The main advantage of this scheduling method is that it does not require synchronized clocks for each switch, hence the name *Asynchronous* Traffic Shaper. The Asynchronous Traffic Shaper performs per-hop/per-flow shaping of traffic, in contrast to the per-hop/per-class shaping performed by the Credit-based Shaper. Although a per-flow shaper is required at each hop—which is cheap to implement using basically a counter—, only a fixed number of queues, *independent* of the number of flows is required. Rather than on the number of flows, the number of queues depends on the (fixed) number of input ports. Thus, in contrast to other methods requiring one (expensive) queue per flow, such as the Integrated Services (IntServ) in the Internet, the Asynchronous Traffic Shaper can be considered scalable with respect to the number of flows. Since this scheduling method has not been standardized and worked out fully yet, we do not consider it further in this paper and focus on the mature Time-aware Shaper for deterministic guarantees in combination with other scheduling methods, in particular, priority scheduling.

Without going into detail, we would like to point out that further standards are relevant for TSN dealing with path calculation (bridging along shortest and non-shortest paths), multi-pathing for providing redundancy in case of link and switch failures, flow filtering for dealing with senders not conforming to their traffic specification and thus potentially impacting the quality of service of other conformant flows, time-synchronization, etc. Overall, this description shows that TSN is actually a combination of various non-trivial mechanisms leading to complex network behavior. This observation motivates the need for a simulation tool assisting in the analysis of TSN networks.

IV. TSN Simulator NeSTiNg

Next, we present the technical details of our TSN simulator NeSTiNg. Due to space restrictions, we provide an overview of the NeSTiNg simulator and highlight some of the technical details. For a comprehensive description of the software components and instructions how to use NeSTiNg, we refer to the openly accessible repository [8]. The NeSTiNg version referred to throughout this paper is based on OMNeT++ version 5.4.1 and INET version 3.6.4.

A. Frame Tagging

To provide frames from different traffic classes with different Quality of Service (QoS) requirements, frames are tagged as described by the IEEE Std 802.1Q. In NeSTiNg, frames carry a VLAN tag including a 3-bit Priority Code Point (PCP) to map frames to one of up to eight queues per outgoing port.

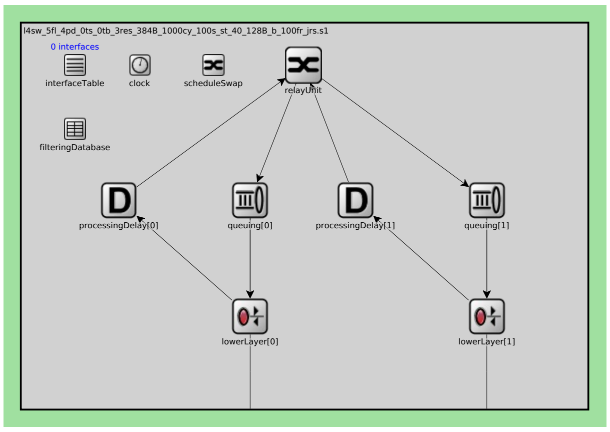

Fig. 2. Components of a NeSTiNg TSN switch with 2 ports.

B. TSN Switch Components and Frame Processing

Next, we describe the components of a TSN switch as implemented by NeSTiNg. Figure 2 provides an overview of the components of a TSN-enabled switch with two ports (in fact, the number of ports is variable). The *lowerLayer* components are the boundary between the switch and the network where the link components are connected to the switch. The *lowerLayer* component is a so-called compound module, which aggregates several subcomponents in one module. In NeSTiNg, the *lowerLayer* module contains the MAC component and components for handling the VLAN tags. The MAC component models the transmission delay and hands over the frames to the link component. The NeSTiNg TSN switch supports link components provided by INET.

To explain the process of forwarding a frame by a simulated TSN switch, we trace the flow of a frame through the switch components from ingress to egress (cf. Figure 2). First, the incoming frame is received by the lower layer of an ingress port. Then, the frame is delayed by the *processingDelay* component for the simulated processing delay of the switch, before the *relayUnit* routes the frame to the *queuing* component of the respective egress port. The *queueing* component encapsulates the output port queues and the components responsible for scheduling outgoing frames as discussed next.

The *queuingFrames* component (cf. Figure 3) evaluates the PCP field of a frame, to enqueue it in one of the queues according to the PCP value. Frames then can be retrieved by the *transmissionSelection* component (cf. Figure 3 and Figure 1), which passes frames to the previously mentioned lower layer. The *transmissionSelection* component is responsible for selecting the next frame eligible for transmission. There are two paths in the control flow of the simulation model of the switch that can trigger the transmission selection process. In the first, downward control flow path, information about frames ready for transmission is propagated from the ingress components down towards the lower layers (egress) of an output port. The downward control flow path triggers the processing of newly arrived frames after an idle period of an output port, thereby avoiding inefficient active polling during those idle periods. On this control flow path, the *transmissionSelection* component continues to notify the lower layer about the arrival of a frame ready for transmission after being activated, which in turn starts transmitting the frame immediately after an idle period. The second control flow path, called upward control flow path, is activated by the lower layer of an output port during—or more exactly, directly after—a busy period. When the lower layer completes the transmission of a frame, it requests the next frame from the *transmissionSelection* component, which in turn polls the shapers (described below) and queues for the next eligible frame. This upward control flow path ensures that as long as there are frames eligible for transmission, they will be processed.

The task of selecting the appropriate next frame is complicated by the fact that two events can be scheduled for exactly the same simulation time. It has to be ensured that these concurrent events are resolved in the correct sequential order to achieve the desired functional behaviour. As shown next, even a simple strict-priority forwarding scenario is non-trivial: Assume an initially idle output port. At time t_x, there are two events e_l, e_h. Event e_l indicates the arrival of a frame for a low-priority queue, and e_h indicates the arrival for high-priority queue. In the global event schedule, e_l can be scheduled before e_h, e.g., by putting e_l in the schedule before e_h. If the transmission selection is executed immediately after receiving a notification about some frame being ready for transmission (in this case triggered by e_l), the lower layer erroneously forwards the frame from the lower priority queue since the transmission selection did not "see" the frame in the higher priority queue. Therefore, the transmission selection has to be synchronized with respect to all changes of the output queues and shapers before performing the selection of the next frame, i.e., transmission selection has to be performed only after all other events in the queue and shapers have been processed. In NeSTiNg this is implemented in the *transmissionSelection* component by scheduling a self-message at the current simulation time. The event of this self-message is placed after all events scheduled before the current simulation time by OMNeT++ itself. By exploiting the rules for the event execution order of events in the future event list in OMNeT++, we keep the implementation simple (especially when considering combinations of multiple TSN mechanisms), and avoid any (error-prone) manual manipulation of the future event list from OMNeT++. Thus, when the *transmissionSelection* component receives its self-message, it can finally select the next frame for transmission by checking the queues and shaper states.

C. Shapers

Next, we discuss the components implementing the different TSN scheduling mechanisms, called—consistent with the wording of the IEEE standards—*shaper* components. Shapers control whether a frame in an egress port queue has to be con-

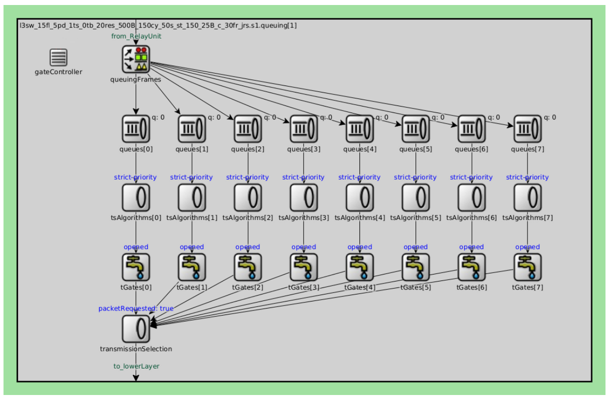

Fig. 3. Queueing network of egress port.

sidered in the transmission selection. The shaper components are logically located between the output port queue holding the frames and the *transmissionSelection* component (cf. Figure 3). From a software architectural perspective, the shapers provide a queue interface for the *transmissionSelection* component. If according to their current state the frames in the associated queue component are eligible for transmission, the shaper component transparently forwards all frame requests from "below" and all notifications about frame arrivals from "above". If, however, the frames from the queue component associated with the shaper component are not eligible for transmission, the shaper component pretends to be an empty queue, effectively blocking the selection (and subsequent transmission) of the frames from the associated queue component. A state change of the shaper from the blocking state triggers the same notifications as if a frame were enqueued at the actual queue component. Thus, information or requests about frames either propagate through the chain of shaper components from or to the actual queue, or can be blocked by a shaper component. This allows for cascading multiple shaper components.

D. Time-aware Shaper

Next, we discuss the shaping mechanisms available in NeSTiNg in more detail starting with the *Time-aware shaper* (TAS). The functionality of the TAS is distributed among multiple components, namely, one gate component in the data path (symbolized by a faucet icon in Figure 3) for each queue, one gate controller component, and one clock component per switch. The gate controller contains the entries of the gate schedule, which govern the state changes of the gate (open, close) according to the simulated *local* switch time provided by the clock component. To decouple the global simulation time and the local clock time without generating several million clock tick events per second of simulated time, we utilize the observer pattern. The gate controller (or any switch component) can subscribe itself as a listener at the clock component of the switch and request to be notified at a certain time. The clock component schedules a clock event for each request and notifies the subscriber when the clock event occurs. The clock itself can decide when with *respect to the global simulation time* the clock event for the requested time is scheduled. This allows for

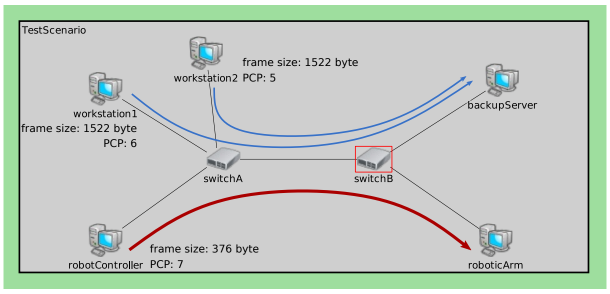

Fig. 4. Exemplary test scenario of converged network.

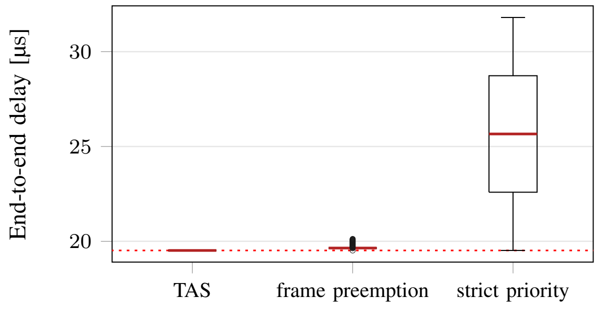

Fig. 5. End-to-end delays for time-triggered flow for scenario in Figure 4 with different switch configurations.

implementing a clock with user-defined accuracy to evaluate the impact of synchronization errors onto a TSN system.

E. Credit-based Shaper

The second TSN shaping mechanism implemented by NeSTiNg is the *Credit-based shaper* (CBS). The CBS is modeled as a finite state machine with three states, namely, credit stays unchanged, increases, or is spent. For an efficient implementation by the discrete event simulator, the CBS only creates events and updates its credit if state changes are necessary. For example, if a frame from a queue is being transmitted, the CBS changes to the "spending" state and precomputes the time of the next state change considering the size of the currently transmitted frame and the gate schedule. The CBS then schedules an event for the time when this state change will occur. From an outside perspective, this makes the credit "jump" at each state transition.

V. Evaluation

NeSTiNg targets simulations of converged networks, where time-triggered and non-time-triggered traffic share one physical network infrastructure and their interaction is non-trivial to predict. Next, we will show exemplary results of simulating such converged networks, once as a proof-of-concept, and second to quantify the performance (runtime) of simulating TAS in NeSTiNg.

A. Exemplary Simulation

In Figure 4, we show a very simple "converged" network scenario with only three traffic flows, that share the bottleneck link between *switchA* and *switchB*. All links have a bandwidth of $1\,\mathrm{Gbit\,s^{-1}}$ and a propagation delay of $100\,\mathrm{ns}$. Each tail-dropping queue in the switches can store at most 30 MTU-sized frames (MTU of $1522\,\mathrm{B}$), and there is one queue per PCP.

The "best-effort" hosts (*workstation1* and *workstation2*) send MTU-sized frames with line rate to the backupServer[1]. Messages from the robotController to the roboticArm form the time-triggered flow with highest priority (PCP=7), frame size $376\,\mathrm{B}$ and cycle time of $1000\,\mathrm{\mu s}$.

We simulated this scenario and evaluated the end-to-end delay for the time-triggered flow with three different switch configurations, namely: TAS, frame preemption, and strict priority (cf. Figure 5). In the TAS setting, the time-triggered traffic is assigned to a exclusive time slot. While in the remaining scenarios all gates are open during the whole cycle. For the time triggered traffic, the theoretical end-to-end delay d_{tot} (taking the inter-frame gap into account) is represented by the dotted line in Figure 5, and $d_{\text{tot}} = 3 \cdot d_{\text{prop}} + 2 \cdot d_{\text{proc}} + 3 \cdot d_{\text{trans}} = 19.516\,\mathrm{\mu s}$ with propagation delay d_{prop}, transmission delay d_{trans} and processing delay d_{proc}. Without the exclusive time slot but frame preemption enabled, the time-triggered flow suffers from jitter and a slightly increased end-to-end-delay (up to $\approx 20.02\,\mathrm{\mu s}$) caused by blocked links due to best-effort fragments in transmission. Finally, without TAS and frame preemption, the time-triggered traffic is heavily impacted by the best-effort traffic. In the worst case, a time-triggered frame is being queued for the time an MTU-sized frame requires for transmission. This results in much larger jitter and a worst-case end-to-end-delay of almost $32\,\mathrm{\mu s}$.

The .ned-files and .ini-files for these scenarios can be found in the NeSTiNg repository to reproduce these simulations.

B. Performance Evaluation of IEEE 802.1Q TAS

After presenting the basic functionality of NeSTiNg, we continue with the performance evaluations of the TAS. To auto-generate simulation scenarios with frame-based schedule configurations for the performance evaluations, we use an internal toolchain based on scheduling algorithms described in [1], [2].

1) Setup: We executed the performance evaluation on a machine with four Intel Xeon E7-4850 processors at a clock speed of $2.1\,\mathrm{GHz}$, and total RAM of $1.057\,\mathrm{TB}$, running Linux with kernel version 4.19.4. Moreover, we used Docker to containerize and parallelize the experiments and to guarantee a reproducible evaluation environment.

2) Parameters: In the following, we present the parameters for our performance evaluations. In general, the scalability of a network simulation model depends strongly on the *scenario size* since each frame induces one or multiple events on each hop. To achieve a predictable load pattern, we focus on time-triggered traffic and use a *line topology* with all TSN hosts attached at the ends of the line. For these topologies, we define the scenario size as the product of the number of switches and the number of flows. For each time-triggered flow, there

[1]This serves as a sanity check for a) strict-priority (all frames of workstation2 have to be dropped at switchA, since there are always frames from workstation1 available for transmission), and b) for CBS (ratio of sum of received packets at backupServer from workstations $\propto$ ratio idle slopes).

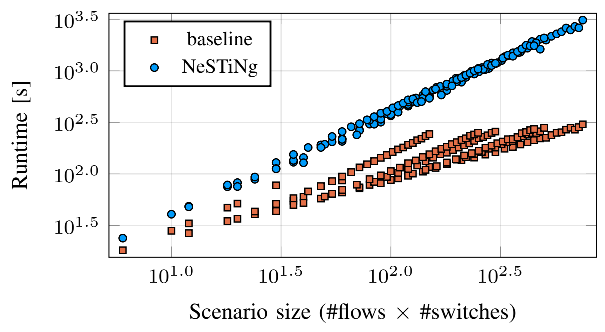

Fig. 6. Comparison of the average runtime for baseline simulations and NeSTiNg simulations over scenario size.

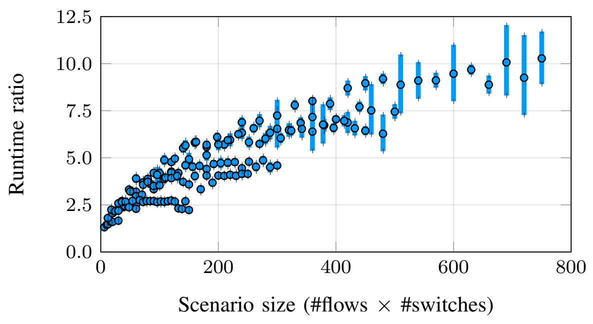

Fig. 7. Mean runtime ratio, i.e., the runtime of a NeSTiNg simulation divided by the runtime of the corresponding baseline simulation over scenario size. Vertical bars indicate 95% confidence intervals.

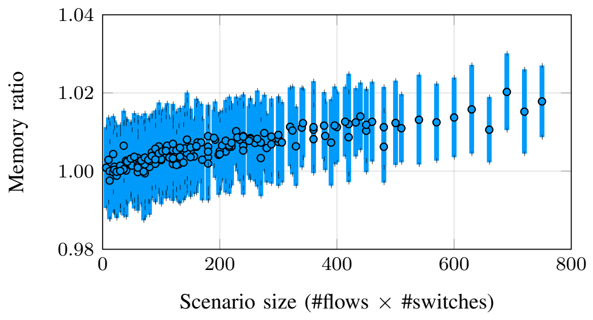

Fig. 8. Mean ratio of memory consumption, i.e., the total memory required by a NeSTiNg simulation divided by the total memory required for the corresponding baseline simulation over scenario size. Vertical bars indicate 95% confidence intervals.

is one dedicated "source" host sending Ethernet frames 376 B with a cycle time of 1000 µs, to one dedicated host acting as sink receiving frames on the opposite side of the line. Even though our toolchain supports arbitrary, meshed networks with arbitrarily placed hosts, all frames cross all switches in the chosen (restricted) network topology thus inducing the maximum amount of events. To simplify the mapping of the schedule configurations to simulation scenarios in our tool chain, we chose the frame size for the time-triggered flows such that the total delay per switch is close to an integer value in micro-seconds. Switch and link parameters are the same as in the exemplary scenario in Section V-A.

To show the scalability of our approach and validate the soundness of our scenario size definition, we measure the simulation runtime of NeSTiNg with a varying number of switches and flows. In detail, we simulate 75 scenarios with increasing number of switches ranging from 2 to 50 with steps size 2 for three fixed numbers of flows (5, 10, 15), and 75 scenarios with increasing number of flows ranging from 2 to 50 with step size 2 for three fixed numbers of switches (3, 6, 9). For each scenario, we performed ten *experiments* (i.e. simulation runs to simulate 100 s of the scenario) with TAS enabled and ten experiments with all TSN features disabled (*baseline experiment*). The baseline experiments will be used to quantify the overhead of simulating the TAS mechanism using NeSTiNg.

3) Results: Next, we describe the findings of the TAS performance evaluation of NeSTiNg.

Figure 6 depicts the runtime to simulate a scenario for 100 s. The x-axis denotes the *scenario size*, which we introduced in Section V-B2, and the y-axis denotes the runtime. Both axes are scaled logarithmically. The diagram shows that both the baseline and the IEEE 802.1Qbv implementation of NeSTiNg scale linearly with the scenario size. Since each frame of every flow requires a certain number of events and processing on each hop of its path, we anticipated this behaviour and the results verify our hypothesis.

However, due to the non-negligible complexity of the TSN mechanisms in terms of required processing of events, the runtime of the NeSTiNg simulation shows a significant overhead. Figure 7 presents the runtime of NeSTiNg simulations relative to the corresponding baseline simulations. The x-axis shows the scenario size and the y-axis shows the average runtime ratio over ten runs of each parameter setting, where error bars indicate 95% confidence intervals. We observe that the relative overhead of NeSTiNg increases with the scenario size, due to additional events per frame and hop. However, we note that the NeSTiNg simulations achieved an average number of 1001 (±44 std. dev.) simulated events per millisecond runtime, while the baseline simulations reached only 292 (±49 std. dev.) events per millisecond.

Figure 8 shows the maximal memory consumption of NeSTiNg simulations relative to that of the corresponding baseline simulations. The x-axis shows the scenario size and the y-axis shows the average memory consumption ratio over ten runs of each parameter setting, where error bars indicate 95% confidence intervals. We observe that the memory overhead of NeSTiNg is insignificant, and increases only sightly with the scenario size. On average, NeSTiNg simulations required 75.1 MB (±2.1 MB std. dev.) total memory, while the baseline simulations required 74.6 MB (±1.8 MB std. dev.). Memory consumption of the NeSTiNg process was sampled with 1 Hz for each experiment.

In conclusion, the presented performance evaluation shows that simulating TAS with NeSTiNg comes with a non-negligible runtime overhead due to processing the additional events required for the simulation of the TSN features, but only minimal increment in memory footprint. The results of the performance evaluation unveil a linear complexity with regard

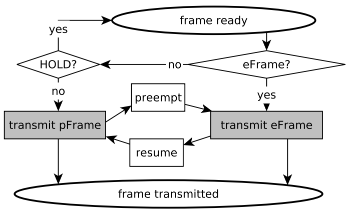

Fig. 9. Simplified control flow with frame preemption.

to the scenario size and thereby demonstrate that the simulation of larger networks is possible within reasonable time.

VI. Summary and Future Work

In this paper, we presented NeSTiNg, a network simulator for converged IEEE TSN networks. Our contributions include simulation model components for time-aware and credit-based shaping. NeSTiNg also provides a framework for switch-local clocks that can be used to simulate switches without perfectly synchronized clocks, as well as VLAN tagging, thus, supporting—as a by-product—strict-priority scheduling.

Due to space restrictions, we focused on the time-triggered TSN scheduling mechanism in this paper. However, NeSTiNg also provides simulation models for further TSN features, in particular, frame preemption. Frame preemption significantly complicates the lower layers, which now have to be able to handle *two* frames, an express frame and a preemptable frame, resulting in a control flow with possibly several branches and loops (cf. Figure 9). Any scheduled event regarding the future transmission of a preemptable frame can become meaningless when frame preemption occurs, but at the time of scheduling these events, we do not know whether a frame in trasmission will be preempted. However, the original design of the MAC in INET strongly exploits the assumption that once a frame has been forwarded to the lower layers, it will be transmitted completely. In NeSTiNg, we work around this problem by introducing a secondary "meta-data" message type that is sent over the link component between the two MAC components of the connected switch ports, to retain compatibility with NeSTiNg links. We leave a detailed description of how to work around this problem to future work.

Moreover, we also plan to integrate further TSN mechanisms in the future such as the asynchronous traffic shaper and clock synchronization protocols, as well as interfaces to application-layer functions, for instance, to simulate cyber-physical systems, in particular, networked control systems.

Acknowledgments

We thank all contributors to the NeSTiNg project: Robin Finkbeiner, Alexander Glavackij, Urim Limani, Adriaan Nieß, Patrick Schneefuss, Nicklas Segedi and Michel Weitbrecht.

This work was supported by the German Research Foundation (DFG) under the research grant RO 1086/20-1 "Integrated Controller Design Methods and Communication Services for Networked Control Systems (NCS)".

References

[1] F. Dürr and N. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Networks and Systems (RTNS)*, Brest, France, Oct. 2016, best presentation award.

[2] J. Falk, F. Dürr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for TSN with ILP," in *Proc. 24th Int. Conf. Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Hakodate, Japan, 2018, pp. 136–146.

[3] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based joint routing and scheduling for time-triggered networks," in *Proc. 25th Int. Conf. Real-Time Networks and Systems (RTNS)*, Grenoble, France, Oct. 2017.

[4] V. Gavriluţ and P. Pop, "Scheduling in time sensitive networks (TSN) for mixed-criticality industrial applications," in *Proc. 14th IEEE Int. Workshop on Factory Communication Systems (WFCS)*, Imperia, Italy, Jun. 2018.

[5] "OMNeT++ discrete event simulator," https://www.omnetpp.org/.

[6] "CoRE4INET_Background – CoRE simulation models for real-time networks," http://core4inet.core-rg.de/trac/wiki/CoRE4INET_Background.

[7] "INET framework," https://inet.omnetpp.org/.

[8] Institute of Parallel and Distributed Systems, University of Stuttgart, "NeSTiNg project repository," https://gitlab.com/ipvs/nesting.

[9] T. Steinbach, H. D. Kenfack, F. Korf, and T. C. Schmidt, "An extension of the OMNeT++ INET framework for simulating real-time Ethernet with high accuracy," in *Proc. 4th Int. ICST Conf. Simulation Tools and Techniques (SIMUTools)*, Barcelona, Spain, 2011, pp. 375–382.

[10] J. Jiang, Y. Li, S. H. Hong, A. Xu, and K. Wang, "A time-sensitive networking (TSN) simulation model based on OMNeT++," in *IEEE Int. Conf. Mechatronics and Automation (ICMA)*, Aug. 2018, pp. 643–648.

[11] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt, "Extending IEEE 802.1AVB with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic," in *IEEE Vehicular Networking Conf. (VNC)*, Dec. 2013, pp. 47–54.

[12] P. Heise, F. Geyer, and R. Obermaisser, "TSimNet: An industrial time sensitive networking simulation framework based on OMNeT++," in *8th IFIP Int. Conf. New Technologies, Mobility and Security (NTMS)*, Nov. 2016, pp. 1–5.

[13] M. Pahlevan and R. Obermaisser, "Evaluation of time-triggered traffic in time-sensitive networks using the OPNET simulation framework," in *26th Euromicro Int. Conf. Parallel, Distributed and Network-based Processing (PDP)*, Mar. 2018, pp. 283–287.

[14] Institute of Electrical and Electronics Engineers, "IEEE standard for local and metropolitan area network – bridges and bridged networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, Jul. 2018.

[15] IEEE Instrumentation and Measurement Society, "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, Jul. 2008.

[16] I. C. Society, "IEEE standard for local and metropolitan area networks – timing and synchronization for time-sensitive applications in bridged local area networks," *IEEE Std 802.1AS-2011*, Mar. 2011.

[17] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Networks and Systems (RTNS)*, Brest, France, Oct. 2016.

[18] L. Zhao, P. Pop, Z. Zheng, and Q. Li, "Timing analysis of AVB traffic in TSN networks using network calculus," in *Proc. 24th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS)*, Porto, Portugal, Apr. 2018.

[19] J. Diemer, J. Rox, and R. Ernst, "Modeling of Ethernet AVB networks for worst-case timing analysis," *IFAC Proceedings Volumes*, vol. 45, no. 2, pp. 848–853, 2012, 7th Vienna Int. Conf. Mathematical Modelling.

[20] F. Reimann, S. Graf, F. Streit, M. Glaß, and J. Teich, "Timing analysis of Ethernet AVB-based automotive E/E architectures," in *Proc. 18th IEEE Conf. Emerging Technologies and Factory Automation (ETFA)*, Cagliari, Italy, Sep. 2013.

[21] Institute of Electrical and Electronics Engineers, "IEEE standard for Ethernet," *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, Aug. 2018.

[22] J. Specht and S. Samii, "Urgency-based scheduler for time-sensitive switched Ethernet networks," in *Proc. 28th Euromicro Conf. Real-time Systems (ECRTS)*, Toulouse, France, Jul. 2016.