SDN4CoRE: A Simulation Model for Software-Defined Networking for Communication over Real-Time Ethernet

Timo Häckel, Philipp Meyer, Franz Korf, and Thomas C. Schmidt

Dept. Computer Science, *Hamburg University of Applied Sciences*, Germany
{timo.haeckel, philipp.meyer, franz.korf, t.schmidt}@haw-hamburg.de

Abstract

Ethernet has become the next standard for automotive and industrial automation networks. Standard extensions such as IEEE 802.1Q Time-Sensitive Networking (TSN) have been proven to meet the real-time and robustness requirements of these environments. Augmenting the TSN switching by Software-Defined Networking functions promises additional benefits: A programming option for TSN devices can add much value to the resilience, security, and adaptivity of the environment. Network simulation allows to model highly complex networks before assembly and is an essential process for the design and validation of future networks. Still, a simulation environment that supports programmable real-time networks is missing.

This paper fills the gap by sharing our simulation model for Software-Defined Networking for Communication over Real-Time Ethernet (SDN4CoRE) and present initial results in modeling programmable real-time networks. In a case study, we show that SDN4CoRE can simulate complex programmable real-time networks and allows for testing and verifying the programming of real-time devices.

1 Introduction

In recent years, Ethernet has emerged to become the next standard for automotive networks. Complementary protocols such as IEEE 802.1Q Time-Sensitive Networking (TSN) have proven to meet the real-time and robustness requirements of these environments. On the other hand, the Software-Defined Networking (SDN) paradigm has revolutionized campus and data center networks. Separating the control from the data plane of network devices at a central control unit with global network knowledge enables simple and fast-forwarding at devices, while high-level control applications can steer the entire network. SDN, though, cannot grant Quality-of-Service (QoS) guarantees to the forwarding plane. First results from integrating the SDN paradigm with TSN standards are promising [2]. A programming option for TSN devices can add much value to the resilience, security, and adaptivity of the environment. Although the application of the combination of TSN and SDN is expected to expand with the introduction of 5G and the Internet of Things, we focus on the use case in automotive networks in this work.

Network simulation allows highly complex networks to be modeled before the assembly. This is an important technique for the design and validation of future networks and therefore

remains an active research topic. The discrete event simulation platform OMNeT++ suits well as a simulation toolchain for automotive communication, as shown in prior work [10].

In this paper, we present our simulation model for Software-Defined Networking for Communication over Real-Time Ethernet (SDN4CoRE) and describe the first results in modeling programmable real-time networks. SDN4CoRE is built on top of the INET framework and uses the CoRE4INET simulation models developed in previous work [7]. To make the network devices programmable, we implement client and server modules for the NetConf protocol and integrate the OpenFlowOMNeTSuite [4] for OpenFlow protocol modules. In our case study, we show how our model can be used to test and evaluate configuration mechanisms in real-time networks.

The remainder of this paper is structured as follows. Section 2 provides background knowledge and related work. The concept of programming options for the different real-time Ethernet components is discussed in Section 3. In Section 4, we describe the components of the SDN4CoRE simulation models, followed by a case study in Section 5. Finally, Section 6 concludes this work with an outlook on future work.

2   Background and Related Work

Today, more and more software components get deployed in automotive networks, demanding a steady increase in communication bandwidth and timing guarantees. Ethernet has emerged as the next high-bandwidth communication technology for in-car networks. There have been several attempts to introduce support for real-time requirements in Ethernet networks. Time-Triggered Ethernet (AS6802) provides a synchronous Time Division Multiple Access (TDMA) implementation, as well as rate-constrained traffic classes for Ethernet. Audio Video Bridging (AVB) defines dynamic bandwidth reservation mechanisms for streams and traffic shapers to guarantee maximum latency. AVBs successor Time-Sensitive Networking (IEEE 802.1Q-2018 [3]) is a set of standards which are defined by the TSN task group of the IEEE. These standards extend Ethernet to concurrently forward real-time and Cross-Traffic (CT). It supports both synchronous TDMA traffic and asynchronous bandwidth reservation for streams. CoRE4INET [7] (Communication over Real-time Ethernet for INET) is a suite of real-time Ethernet simulation models developed over the past decade. Currently, CoRE4INET supports the AS6802 protocol suite, traffic shapers of Ethernet AVB, and implementations of IEEE 802.1Q, as well as models to map IP traffic to real-time traffic classes.

As communication requirements increase, so does the complexity and network adaptability requirements. The SDN paradigm promises to solve this problem by moving the control logic out of the network devices into a central control unit [6]. Kreutz et al. discuss the paradigms and concepts of SDN in their comprehensive survey [5]. The network logic is split into three layers: (1) The data plane on which each switch forwards packets according to the flow rules, (2) the control plane on which each switch is connected to a logically (not necessarily physically) centralized controller that manages the forwarding logic, and (3) the management plane on which network administrators manage the controller applications. The communication between the SDN controller and the switches is specified in the OpenFlow standard of the ONF [9]. The OpenFlowOmnetSuite – originally developed at the University of Würzburg – is an OMNeT++ simulation model that implements a concept of SDN with OpenFlow [4]. It provides the OpenFlow standard message types, implementations of forwarding devices, a SDN controller implementation, and interfaces for controller applications.

In TSN, configuration of various components is enabled through the NetConf protocol (RFC 6241 [1]). It specifies a management architecture in which the managed unit contains a

NetConf server, and the administrative unit connects through a NetConf client. On the server-side, configurations are stored in configuration data stores. NetConf provides Remote Procedure Call (RPC) operations such as *get-config* and *edit-config*, which allow the client to request or edit the configuration data stores at the server. SDN4CoRE provides a base implementation of the NetConf protocol, including client, server, data stores, and operations.

Rather little work has been done to combine the concepts of TSN and SDN. Nayak et al. mention the term "Time-Sensitive Software-Defined Network" in 2016 for the first time and show a concept of a programmable scheduler [8]. In previous work, we introduced a concept of software-defined networks supporting time-sensitive in-vehicle communication [2] along with a switching methodology for TSN streams, the programming of the bandwidth reservation via the Stream Reservation Protocol (SRP), and a detailed analysis.

To the best of our knowledge, no simulation environment supports programmable real-time networks. Making the modules of CoRE4INET programmable allows using other available simulation models as well, such as CAN bus signals and gateways to model realistic in-vehicular communication [7].

3   Programmable Real-Time Ethernet Communication

Introducing a programming option for real-time Ethernet devices that improves resilience, security, and adaptability of the environment requires changes on all three layers of the SDN concept: Forwarding devices must provide a programming interface using open standards, control functionality for real-time must be extracted from the switches and integrated into the SDN controller, and controller applications must be able to program and manage real-time devices.

In real-time Ethernet and SDN, switches contain additional modules to extend the functionality of regular switching hardware, which is depicted in Figure 1. One of the additional modules introduced in some real-time communication is an ingress control module. In TSN this is the "Per-Stream Filtering and Policing" module. It is used to filter incoming Ethernet frames for controlling bandwidth and arrival times. A similar module is used at the egress of a switch and implements in particular priority queuing, real-time scheduling and traffic shaping. In TSN this module is called "Enhancements for Scheduled Traffic" and specified in IEEE 802.1Qbv. It uses a Gate Control List (GCL) to indicate, which 802.1Q priorities are allowed to pass through a particular port at a specific time. The scheduling information is stored in the "Schedule" table and needs a precisely synchronized time at all network devices, which is managed by the "Time Sync" module. For stream-based bandwidth reservation, the "SR Table" module contains all registered talkers and listeners for time-sensitive streams.

In a SDN switch, the forwarding module of a standard Ethernet switch is replaced by a flow-based forwarding module that performs flow table lookups based on packet match rules. The SDN controller performs tasks such as topology discovery, MAC-Address learning, and route determination. The programming interface between the switch and the SDN controller is an open southbound API that implements standard protocols such as NetConf or OpenFlow. Additional network applications can be executed on top of the controller.

In programmable real-time networks, flow-based operations require merging the modules of the real-time and the SDN switching components. To ensure that the real-time capabilities are not altered in any way, the ingress and egress control modules must remain unchanged by the additional programming option. When data packets arrive, the ingress control manages the timing and applies stream-based filters. After that, the packet is matched against the flow table, and the discovered actions are executed. The packet then gets forwarded to the correct egress ports, where the egress control manages the timing and shaping of the outgoing traffic. If
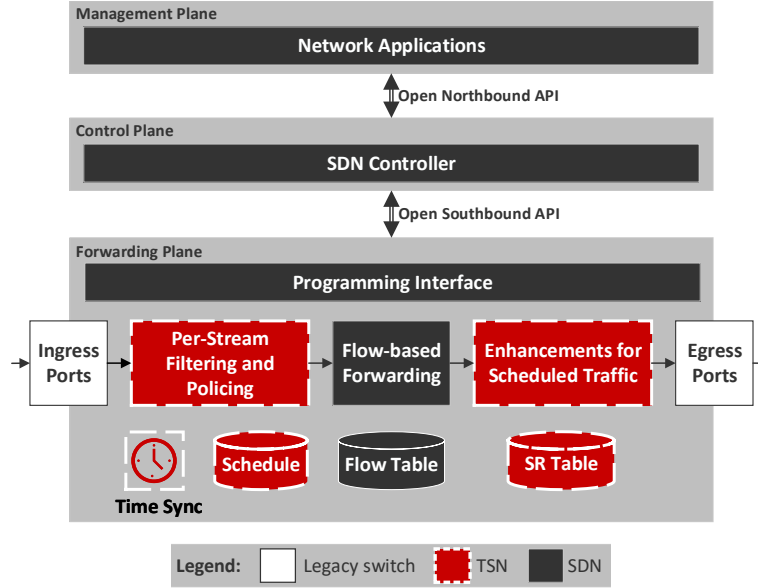
Figure 1: Components of a programmable real-time switch.

no corresponding forwarding rule exists, the packet is dropped by default, while most controller applications insert the default rule to forward the packet to the controller.

SDN controller applications manage the forwarding and store information for the control functionality of the SDN devices. This usually includes features such as neighbor discovery or source MAC address learning. Because many real-time Ethernet protocols use multicast MAC addresses to transfer data to a group of receivers, the controller applications must be able to learn and control multicast MAC addresses. AVB and TSN use the SRP to announce the sender (talker) and receivers (listener) of the multicast groups. Therefore, the controller application must receive and understand the SRP packets. When a forwarding device receives a SRP message, it is forwarded directly to the SDN controller via the OpenFlow protocol. The controller application then registers the talker or listener in its SR table and sends the SRP message back to the switch via the OpenFlow protocol. The switch updates its SR table and forwards the message to the next hop. In this way, the 'talker advertise' and 'listener ready' messages are spread across the network, and each switch goes through the same process until the messages reach the clients. When a new client subscribes to a stream with a 'listener ready' message, a forwarding rule for the TSN stream is inserted in the switches flow table before the 'listener ready' command is sent back to the switch and forwarded along the path. In a future release, the modules handling the SRP protocol could be removed from the switches to simplify them. The controller could insert a match rule for the SRP messages to receive them directly. Subsequently, the SRP tables could then be updated directly via the NetConf protocol.

Besides the dynamic stream reservation, the controller must be able to control the scheduled traffic. Therefore, the switch must provide a specific NetConf data store to control the scheduling tables. In this work, we implemented such a data store for the gate control of 802.1Qbv. To change the GCL, the controller sends an 'edit config' message containing updates or a full configuration for the active GCL. The NetConf server module forwards those messages to the data store which then updates the GCL in the corresponding port. In a future release, additional modules may be added to the data store. For example, the controller could program the 802.1Qci filters, gates, and meters. On the other hand, algorithms to calculate a new schedule
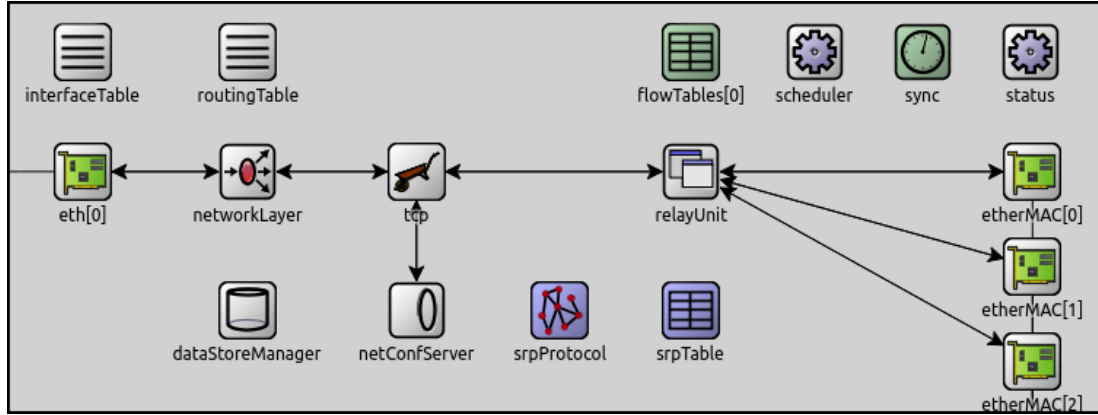
4

Figure 2: OMNeT++ Module of an OpenFlow and NetConf capable switch.

for the entire network could be implemented to adapt the schedules dynamically.

4   SDN4CoRE Simulation Model

The SDN4CoRE simulation module is based on the discrete event simulator OMNeT++ (https://omnetpp.org/) and the INET framework (https://inet.omnetpp.org/). It uses the CoRE4INET [11] simulation model for real-time Ethernet communication and extends some modules to be programmable. On the other hand, SDN4CoRE uses the OpenFlow protocol implemented in the OpenFlowOMNeTSuite [4]. We forked and extended this suite to harmonize with our models. For example, we introduced interface modules to enhance flexibility and modularity and created a new flow table structure.

Figure 2 shows the general module structure of a forwarding device that is capable of OpenFlow and NetConf. The eth0 port connects the switch to the controller. The NetConf server module is connected to this port via TCP, but can also be connected to other protocols, e.g. for security reasons. It handles the NetConf protocol and extracts RPCs to forward them to the data store manager, which in turn applies the command on the proper data store and creates the response. The OpenFlow protocol client is implemented in the relay unit and also connected via TCP. Besides, the relay unit performs the flow table lookup in the flow tables and forwards incoming data packets to the correct Ethernet port. For the etherMAC interfaces, we provide several real-time Ethernet port implementations that control the ingress and egress timings. Some of them require the Time Synchronization and Scheduler modules, which also contribute a realistic device clock. The stream reservation table and protocol modules are needed for AVB/TSN streams. The switch implementation also provides the option to import or export a launch configuration for all modules and tables.

The SDN controller module is based on a standard host provided by the INET framework so that other applications can run smoothly. The OpenFlow controller module from the OpenFlowOMNeTSuite and the NetConf client module connect to the forwarding devices via TCP. The NetConf client module handles connectivity to the forwarding devices and forwards RPC requests and replies to the corresponding NetConf application or server. We include a base implementation for NetConf and OpenFlow applications as well as specific implementation for real-time controllers. With this, it is possible to create controller apps that use NetConf and OpenFlow simultaneously. The controller also provides the ability to import or export a launch configuration.
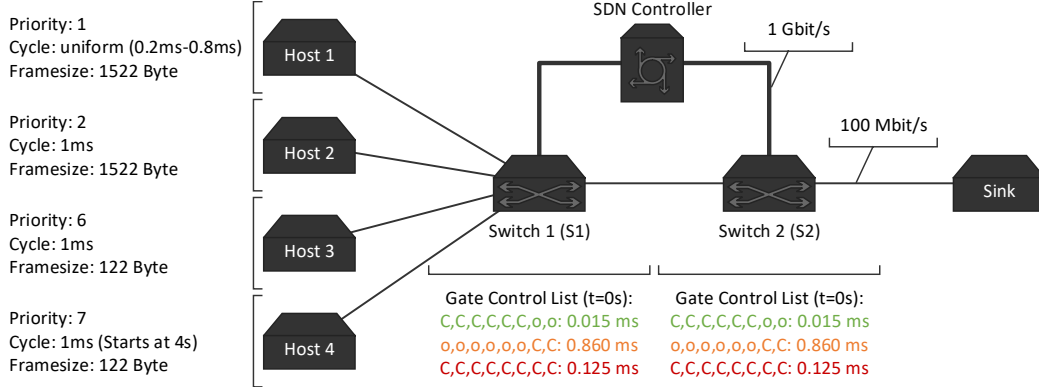
Figure 3: Network topology and device configurations at startup.

5 Case Study

Our results in previous work show that SDN control overhead can be added to TSN streams without a delay penalty, provided protocols are mapped properly [2]. In this work, we analyze the programmability of the GCL in switches supporting IEEE 802.1Qbv. Since this article focuses on programming rather than calculating correct schedules, we have disabled clock jitter and time synchronization for all devices to simplify the simulation and calculation of schedules.

The network topology of our case study consists of four hosts sending messages to one sink via two switches in the presence of one SDN controller as shown in Figure 3. Each host sends one periodic IEEE 802.1Q message, so the traffic shaping is controlled by the switches in the network. These messages have a fixed priority, frame size, and cycle. As an exception, host 1 sends low priority messages with high frequency that varies according to gaussian normal distribution. To introduce a change in high priority traffic, host 4 starts transmitting frames with the highest priority after 4 seconds. The two switches are NetConf and OpenFlow capable. The output on each switch port is controlled by an 802.1Qbv module, which contains gates for each 802.1Q priority that can either be closed (C) or opened (O) at a certain point in time. This information is stored in the GCL that can be programmed at run time using the NetConf protocol. In this evaluation we use three phases for this list: A time slot T_{red}, closing all gates to ensure that the transmission of previous frames has ended before switching to the green phase; A time window T_{green}, for high priority traffic with priorities 6-7; And a window T_{yellow}, for all low priority traffic of priorities 0-5.

The cycle time of the schedule is equivalent to the largest cycle time of the high priority messages, which is 1 ms in our scenario. We calculated the phases and added a safety margin:

$$T_{red} = T_{tx}^{MaxFrame} + T_{margin} = \frac{1,522\,Byte}{100\,Mbit/s} + 5\,\mu s = 121,76\,\mu s + 5\,\mu s \approx 125\,\mu s \qquad (1)$$

$$T_{green} = T_{tx}^{Host3} + T_{margin} = \frac{122\,Byte}{100\,Mbit/s} + 5\,\mu s = 9.76\,\mu s + 5\,\mu s \approx 15\,\mu s \qquad (2)$$

$$T_{yellow} = T_{cycle} - T_{green} - T_{red} = 1\,ms - 15\,\mu s - 125\,\mu s = 860\,\mu s \qquad (3)$$

Figure 4 is a graph showing the changes in end-to-end latency during $10s$ simulation time. These changes are introduced by updates to the configuration of the GCL depicted in Table 1 and changes in the traffic pattern.

At the beginning of the simulation all devices are configured as shown in Figure 3. The high priority packets from host 3 to the sink have a constant delay of 1.03 ms (see Figure 4). In the
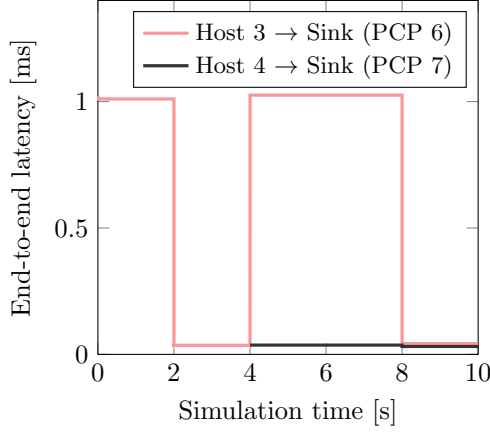
t(s)	Dev	Gate Control List (µs)
0-2	S1	G: 15 ; Y: 860 ; R: 125
	S2	G: 15 ; Y: 860 ; R: 125
2-4	S1	R: 10 ; G: 15 ; Y: 860 ; R: 115
	S2	R: 20 ; G: 15 ; Y: 860 ; R: 105
4-6	S1	R: 10 ; G: 15 ; Y: 860 ; R: 115
	S2	R: 20 ; G: 15 ; Y: 860 ; R: 105
6-8	S1	R: 10 ; G: 30 ; Y: 845 ; R: 115
	S2	R: 20 ; G: 15 ; Y: 860 ; R: 105
8-10	S1	R: 10 ; G: 30 ; Y: 845 ; R: 115
	S2	R: 20 ; G: 30 ; Y: 845 ; R: 105

Figure 4: The end-to-end latency of high priority data flows from the host to the sink. Host 4 starts sending after 4 s simulation time.

Table 1: Changes to the 802.1Qbv Gate Control List of the switches (S1 and S2) with the three phases green (G), yellow (Y) and red (R).

simulation, we can see that this delay is introduced at the second switch, where the packets miss their time slot because of the transmission time of $20\,\mu s$ over two links. To eliminate this delay, the controller updates the GCL of both forwarding devices at $2s$ simulation time. The red phase is split into two parts, one at the beginning of the cycle and one at the end. The red phase at the beginning is exactly the transmission time of the high priority packet to the device. For switch 1 this is the transmission delay of $10\,\mu s$ from host 3 to switch 1. For switch 2 this phase has to be shifted another $10\,\mu s$ for the transmission from switch 1 to switch 2. This update reduces the end-to-end latency to the expected value of $30\,\mu s$.

At $4s$ simulation time, host 4 starts sending frames with the same size and cycle but a higher priority than those of host 3. Therefore, the transmission time of high priority frames at the switches doubles. The packets of host 4 have the expected end-to-end latency of $30\,\mu s$. Those of host 3 stay in time for their slot at switch 1 because of the safety margin of $5\,\mu s$. At switch 2, however, they miss their time slot and need to wait for one cycle. To solve this time conflict, the controller updates the GCLs at $6\,s$ simulation time, but only for switch 1 to simulate a failure in the configuration of switch 2. Since the update was unsuccessful on switch 2, the end-to-end latency remains unchanged. Although the configuration of switch 2 was unsuccessful, the update of the configuration of switch 1 was still performed. In very sensitive environments this can lead to dangerous behavior of the network. Additional mechanisms are required to ensure that changes to the schedule are executed at the same time and are consistent across the network. At $8s$ simulation time, the controller performs the update of switch 2. This reduces the end-to-end latency of host 3 as expected.

This initial evaluation shows how the programming of schedules in the network can increase the flexibility and performance of the system. On the other hand, it indicates that re-configuration of time slots should be used with care during runtime. Configuring multiple devices might lead to an inconsistent state of the network and loss or delay of critical messages.

6   Conclusions and Future Work

In this paper, we shared our simulation model SDN4CoRE and described in detail how to make real-time communication programmable with NetConf and OpenFlow. In a case study, we demonstrated the use of our model for testing and evaluating configuration mechanisms in

real-time networks. An initial assessment indicates that re-configuration of time slots should be used with care during runtime, and additional mechanisms are required to ensure a consistent state across the network.

In future work, we plan to further extend the programmability of the CoRE simulation models, for example, with the recently published implementation of IEEE 802.1Qci modules. The evaluation of control plane concepts for automotive networks is one of our next tasks.

The SDN4CoRE simulation model, including its original fork of the OpenFlowOMNeTSuite, as well as all other simulation models and analyses tools, are published as open-source on our GitHub page at https://github.com/CoRE-RG/.

Acknowledgments

References

[1] Rob Enns, Martin Björklund, Andy Bierman, and Jürgen Schönwälder. Network Configuration Protocol (NETCONF). RFC 6241, IETF, June 2011.

[2] Timo Häckel, Philipp Meyer, Franz Korf, and Thomas C. Schmidt. Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication. In *Proc. of the IEEE 89th Vehicular Technology Conference: VTC2019-Spring*, Piscataway, NJ, USA, April 2019. IEEE Press.

[3] Institute of Electrical and Electronics Engineers. IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks. Standard, IEEE, July 2018.

[4] Dominik Klein and Michael Jarschel. An OpenFlow extension for the OMNeT++ INET framework. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pages 322–329. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013.

[5] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

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

[7] Philipp Meyer, Franz Korf, Till Steinbach, and Thomas C Schmidt. Simulation of mixed critical in-vehicular networks. In *Recent Advances in Network Simulation*, pages 317–345. Springer, 2019.

[8] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. Time-sensitive Software-defined Network (TSSDN) for Real-time Applications. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 193–202, New York, NY, USA, 2016. ACM.

[9] Open Networking Foundation. OpenFlow Switch Specification. Standard ONF TS-025, ONF, 2015.

[10] Till Steinbach, Hermand Dieumo Kenfack, Franz Korf, and Thomas C. Schmidt. An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy. In *SIMUTools 2011 – 4th International OMNeT++ Workshop*, pages 375–382, New York, USA, March 21-25 2011. ACM DL.

[11] Till Steinbach, Philipp Meyer, Stefan Buschmann, and Franz Korf. Extending omnet++ towards a platform for the design of future in-vehicle network architectures. In Anna Foerster, Vladimír Vesely, Antonio Virdis, and Michael Kirsche, editors, *Proceedings of the 3rd OMNeT++ Community Summit, Brno, Czech Republic, September 15, 2016*. ArXiv e-prints, September 2016.