Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication

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*—Future in-vehicular networks will be based on Ethernet. The IEEE Time-Sensitive Networking (TSN) is a promising candidate to satisfy real-time requirements in future car communication. Software-Defined Networking (SDN) extends the Ethernet control plane with a programming option that can add much value to the resilience, security, and adaptivity of the automotive environment. In this work, we derive a first concept for combining Software-Defined Networking with Time-Sensitive Networking along with an initial evaluation. Our measurements are performed via a simulation that investigates whether an SDN architecture is suitable for time-critical applications in the car. Our findings indicate that the control overhead of SDN can be added without a delay penalty for the TSN traffic when protocols are mapped properly.**

***Index Terms*—In-Car Networks, TSN, SDN, Real-Time Ethernet, Performance Evaluation**

I. Introduction

Over the past years, Ethernet has emerged as the next high bandwidth communication technology for in-car networks [1]. There were several attempts to introduce support for real-time requirements of which the IEEE collection of standards for Time-Sensitive Networking (TSN) [2] is the most promising. The use of Ethernet in automotive networks enables the adoption of multiple standards from the internet domain, such as the internet protocols and transport protocols.
An emerging trend in Ethernet networks is Software-Defined Networking (SDN). SDN has proven especially useful in well-known local area networks such as data centers, as it decreases the complexity and management effort [3]. The basic approach of SDN is to replace the switches with simple forwarding devices and connect them to a logically centralised intelligent network controller [4]. This allows the execution of complex flow-based forwarding rules in simple and inexpensive devices. In the automotive environment SDN could provide benefits regarding safety, robustness, security, cost efficiency, and future-readiness with easily updatable network devices.

In this paper, we analyse the integration of TSN and SDN to Time-Sensitive Software-Defined Networking (TSSDN) for in-vehicular networks and explore its potentials and expectations. We contribute a mapping for deploying time-sensitive traffic in TSSDN switches and describe our methodology of registering time-sensitive flows with OpenFlow. We implement and evaluate this architecture in OMNeT++, a common event-based network simulation environment and conduct first evaluations on the real-time capabilities. Overall, we show that time-sensitive traffic performance remains unaltered when adding SDN control logic.

This paper is structured as follows. Section II provides background knowledge and gives an overview on related work. The potentials and expectations of TSSDN are analysed in Section III. Section IV describes the evolution of the switch architecture to TSSDN switches and how time-sensitive flows can be implemented with OpenFlow. In Section V a case study is conducted showing the real-time capability of the presented TSSDN concept. Finally, Section VI concludes this work and gives an outlook on future work.

II. Background & Related Work

TSN is a set of standards which are defined by the TSN task group [5] of the IEEE. IEEE 802.1Q-2018 [2] extends Ethernet with the ability to forward concurrent real-time- and cross-traffic. For supporting a wide range of Quality-of-Service (QoS) requirements, TSN supports several real-time traffic classes. These can be synchronous (Time Division Multiple Access (TDMA)) or asynchronous such as TSNs predecessor Audio Video Bridging (AVB), which we analysed in former work [6]. The draft IEEE P802.1Qcc is an amendment to the TSN standards and provides enhancements to the Stream Reservation Protocol (SRP). Besides performance improvements, the draft introduces a controller for central network management. However, this controller is only a centralised configuration unit. It neither specifies a vendor neutral standardised interface between the controller and the switches, nor extracts the control plane functionality of the network devices.

SDN enables a standardised configuration of forwarding devices by an OpenFlow controller, by separating the control and data plane of the network devices [4]. Kreutz et al. describe the paradigms and concepts of SDN in their comprehensive survey [3]. The network logic is split into three layers: (1) The data plane on which each switch forwards packets according to flow rules, (2) the control plane on which each switch is connected to a logically (not necessarily physically) centralised 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 [7].

Thiele & Ernst [8] show by formal analysis that the concept of SDN is generally suitable for real-time environments, especially if the flows are implemented in all switches prior to the

978-1-7281-1217-6/19/$31.00 ©2019 IEEE

data exchange. They derive possible worst case boundaries for network configuration latency and deem SDN applicable for admission control and fault recovery in automotive networks. As the network requirements of the Industrial Automation (IA) are similar to in-car networks, research in this area can often be transferred directly to vehicles. First efforts to create a real-time capable SDN have been taken in the IA. Herlich et al. present the idea of a real-time Ethernet SDN [9] and show typical use cases for IA networks as a proof of concept. Some of these use cases are directly related to in-vehicular networks, for instance supporting arbitrary network topologies, central and dynamic network (re-)configurations, and fast fail-over mechanisms at network level.

Nayak et al. [10] contribute research regarding robustness and reconfiguration in IA networks by using exclusive links for real-time traffic. They go one step further in developing a Time-Sensitive Software-Defined Network with a scheduling process for time-triggered traffic [11]. However, the schedule is not programmed into the switches, but instead the hosts in the network know the full schedule and send data in their time slots. The switches are not real-time capable and unscheduled cross traffic could change the network behaviour.

Kobzan et al. share their concepts of software-defined networks for the production plants of the future in the FlexSi-Pro research project [12]. They evaluate the combination of TSN and SDN for IA and declare the central configuration of real-time traffic with SDN as an open research task.

Fussey and Parisis summarised the advantages and disadvantages of using SDN in vehicles focusing on enabled features [13]. We expand this list focusing on real-time capabilities and benefits that SDN can have on TSN.

Halba et al. use SDN for data interoperability and robustness in the in-vehicular network and thus, substantiate the benefit of SDN for in-car networks. They show SDN controller applications implementing robustness and safety with fast fail-over mechanisms [14]. However, they do not consider real-time requirements in their network design.

As shown above, there were already some efforts in applying SDN to automotive networks and making them real-time capable. However, the combination of the TSN standards with SDN remains an open research task.

III. The Case for In-Vehicular TSSDN

The OpenFlow standard defines a configuration interface between the controller and the switches in a network. This enables the vendor neutral selection of controller logic and forwarding devices. Additionally, the network logic in SDN is mostly centralised at the controller. According to Du et al. [15] these two properties pave the way for simple, exchangeable, inexpensive, and future proof forwarding devices.

The SDN controller has global knowledge of the network and all active flows. This is reinforced by the static nature of the in-vehicular networks. The network knowledge enables efficient route determination which can prevent link overloading. On the other hand, it enables the calculation and verification of timings over multiple links during run time.

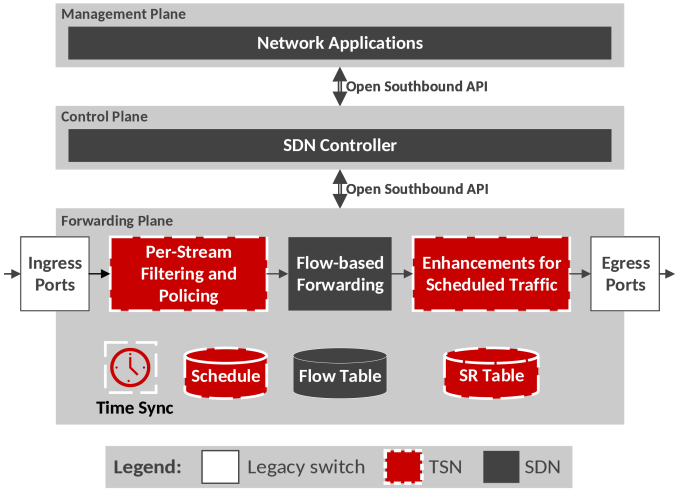

Fig. 1: TSSDN Switch Architecture

With the central network knowledge, it is possible to re-route traffic, add new devices (plug & play) and update the network logic of all switches from a central point. This simplifies the management of the various configurations of a car model. For real-time networks this enables the central (re-) calculation and distribution of time-sensitive schedules [11].

One of the largest potentials of SDN in vehicles is adding robustness. SDN supports arbitrary network topologies with redundant paths [15]. In combination with the global network knowledge, this is an enabler for robust safety methods, such as fail-operational, fail-over or fast re-route through reconfiguration [14]. One of the weak points of SDN regarding robustness and safety is the single point of failure introduced by a central network controller. This problem has been solved by using multiple connected controllers [16], a second controller in hot standby, or a fallback configuration in the switches.

There are already many security suites for SDN controllers enabling flow-based firewalls [17] or advanced security applications such as anomaly or intrusion detection [18]. To secure in-vehicular networks, the knowledge about the network can be used to only permit statically whitelisted flows in critical sections of the network.

While gaining these advantages and potentials of SDN for in-car networks, it must still be possible to maintain the deterministic QoS provided by TSN.

IV. TSSDN Switching Methodology

This section presents the evolution of the switch architecture from standard Ethernet to TSSDN switches and describes how OpenFlow can be used to implement time-sensitive flows.

A. Switch Architecture

In TSN 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 for TSN is the “Per-Stream Filtering and Policing” module. It is used to filter incoming Ethernet frames and control the arrival times of TDMA-based traffic. In TSN

the egress is controlled by the "Enhancements for Scheduled Traffic" module. It implements priority queuing, real-time scheduling and traffic shaping. The schedule for TDMA-based traffic is defined in the "Schedule" table and needs a precise and synchronised time in all network devices. This is managed by the "Time Sync" module. The SRP is used to dynamically reserve bandwidth along a path of an asynchronous stream. The "SR Table" module contains all registered talkers and listeners for time-sensitive streams.

In SDN switches, the forwarding module of a standard Ethernet switch is replaced by a flow-based forwarding module that performs flow table lookups. If 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. Tasks such as topology discovery and route determination are performed by the SDN controller. The switch is connected to the controller via the Open Southbound API which implements the OpenFlow standard. Additional network applications can be executed on top of the controller.

In Time-Sensitive Software-Defined Networking (TSSDN), flow based operations require merging the components of the TSN and SDN switches. To ensure that the real-time capabilities are not altered in any way, the TSN ingress and egress control must remain unchanged. When data packets arrive, the TSN ingress control manages the timing and applies stream-based filters. Then the packet is matched against the flow table and the discovered actions in the matching entry are executed. The packet is then forwarded to the correct egress ports, where the TSN egress control manages the timing and shaping of the outgoing traffic.

TSN devices exchange additional control information, e.g. to reserve bandwidth with the SRP. To implement correct flow rules in the forwarding logic, this information needs to traverse to the SDN controller. The SDN controller updates its network state and informs the switch about changes, if needed. In this work we implemented a controller application managing the Stream Reservation (SR) table for the switches. In the future, further parts of the TSN control plane could potentially be extracted from the forwarding devices and embedded into the controller. Nevertheless, the scheduling and transmission selection needs to remain in the switches to guarantee timing.

B. Implementing Time-Sensitive Flows with OpenFlow

An extension for TSSDN match rules to OpenFlow is needed to control the forwarding of TSN flows. Still, the additional control information of the SRP needs to be communicated between the TSSDN switches and the controller.

In a normal TSN switch, forwarding is performed based on the destination MAC address (multicast) that identifies the stream's listener group. By using flow-based forwarding rules in SDN, the forwarding can be performed with multiple additional match fields, which enable the realization of non-functional requirements. We used the match fields and OpenFlow identifiers shown in Table I to forward time-sensitive flows. By matching the talker source MAC address, we ensure that a stream always originates from the same talker and prevent misuse of the multicast group. If the TSN path redundancy feature is not in use, a certain TSN stream should always arrive at the same ingress port of the switch. If it arrives at another port, the SDN controller needs to be informed about changes in the network. Other match fields are the VLAN ID and VLAN Stream priority, as a stream multicast group might be used in multiple VLANs. This could be useful for future security applications. With these match fields, a TSN stream can be identified as a flow and can now be forwarded correctly.

TABLE I: OpenFlow match fields used for TSN streams

	Match field	**OpenFlow Identifier**
Listener Group	Eth. Dst. Addr.	OFPXMT_OFB_ETH_DST
Talker Address	Eth. Src. Addr.	OFPXMT_OFB_ETH_SRC
Ingress Port	Switch Ingress Port	OFPXMT_OFB_IN_PORT
VLAN ID	802.1Q VLAN ID	OFPXMT_OFB_VLAN_VID
Stream Priority	802.1Q Priority	OFPXMT_OFB_VLAN_PCP

In TSSDN all SRP messages need to be forwarded to the controller instead of being processed in the switch to enable the controller to configure the forwarding of streams. As the OpenFlow protocol does not specify a way to exchange additional control information, we added a new OpenFlow control message type *ForwardSRP*, implementing the standard message signature of OpenFlow and containing the SRP Message as a payload. In future work this could be mapped to the OpenFlow standard messages.

To set up a new stream the talker sends a 'talker advertise'. The TSSDN switch forwards the SRP message to the controller. The controller then registers the talker in its SR table and sends the SRP message back to the switch which then updates its own SR table and broadcasts the 'talker advertise'. This way, the talker advertisement is propagated through the network and each switch goes through the same process until the talker advertisement reaches the clients. When a client subscribes to a stream by sending a 'listener ready' message, the switch forwards it to the controller as well. The controller updates the listener in the SR table. It then pushes a forwarding rule for the TSN stream to the switch before it sends the 'listener ready' back to the switch. When the switch receives the *ForwardSRP* message it delivers it on the direct path to the talker and each switch along the path goes through the same process. The talker starts streaming when the 'listener ready' arrives. Using this approach, we can guarantee the timing since the bandwidth is already reserved and the forwarding rules are already implemented in all forwarding devices along the path.

V. Case Study

Our case study analyses the real-time capabilities of the proposed TSSDN concept in a simulation environment. Our simulation is based on the open-source OMNeT++ IDE and the INET framework [19]. It provides simulation models of standard Internet technologies. On top, we use the CoRE4INET framework [20] [21] which implements real-time Ethernet protocols and the OpenFlowOMNeTSuite [22] [23] which implements the OpenFlow protocol. For the proposed TSSDN concept, we merged the CoRE4INET framework and the OpenFlowOMNeTSuite in parts.

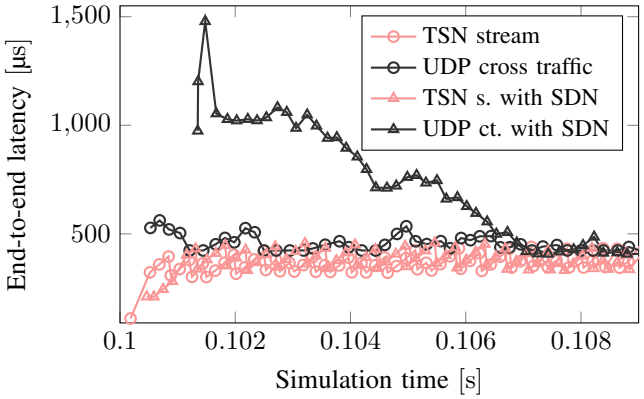

Fig. 2: TSN stream and UDP end-to-end latency of the first Ethernet frames

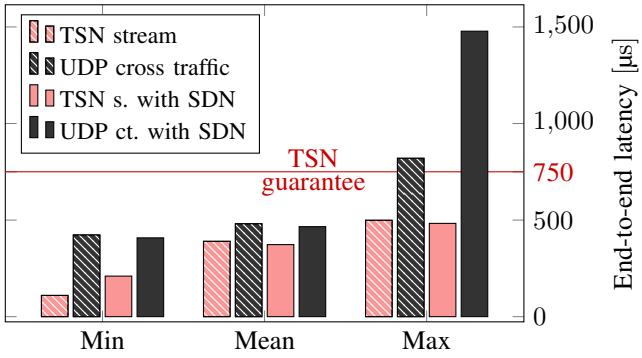

Fig. 3: TSN stream and UDP min/max end-to-end latency of Ethernet frames

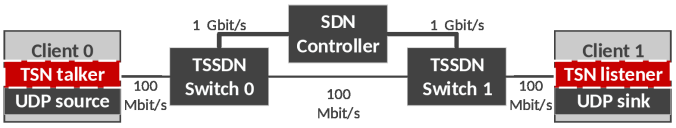

Fig. 4: Network topology of the case study

The network topology of our simulation-based case study consists of two clients, two TSSDN switches, and one SDN controller as shown in Figure 4. Client 0 is the source for the TSN stream and the UDP cross traffic. Client 1 is the receiver of all traffic. With this setup, the timing requirements of TSN across multiple links under load of cross traffic can be analysed. For comparison, the identical scenario without SDN using TSN-only switches was considered. All flows remain the same, so that the result can be directly compared.

At the beginning of the simulation, all the flow tables in the TSSDN switches are empty. After the controller and switches are connected to each other as specified in the OpenFlow standard, the controller transmits an updated default action for table misses, to be forwarded to the controller for further decision. For this reason, an idle time of 100 ms for setup operations is introduced in both versions of the network. After that, the clients can start to work.

Figure 2 shows the latency during the connection setup phase for all frames transferred in 9 ms after the start of transmission. It also compares the latency of the networks with and without SDN.

To set up the TSN stream, the talker advertises the stream and its listeners subscribe with the help of the SRP. In the TSSDN version of the network, the controller receives the SRP messages from the forwarding devices, updates its SR tables and implements the correct flow entries in the forwarding devices. In our simulation this introduces a delay of 0.3 ms to the SR process compared to the network without SDN. When the talker receives the listener ready, it begins to stream. As the stream is already registered in the TSSDN switches during the SR, no further inspection by the SDN controller is needed. Thus, no additional latency is introduced by SDN after the SR. Before the UDP source starts sending frames, it resolves the cross traffic destination MAC address by ARP. In the TSSDN version of the network those ARP frames are forwarded by the SDN controller, which again introduces a slight delay. Thereafter, the UDP source starts sending datagrams. There is no matching flow entry for the first UDP frame in the TSSDN switches. UDP frames are sent to the controller and the flow tables are updated accordingly. This leads to a much higher latency for the first UDP frames than for the TSN stream.

In the TSSDN version of the network, the latency of the TSN stream increases slowly as the cross traffic traverses the links. At first, only the link between client 0 and switch 0 is under load of cross traffic. When the controller implements the flow rule in switch 0, the UDP frames put load on the link between switch 0 and 1. The latency for the TSN stream increases step by step, until all network links are under load of cross traffic. In the version without SDN, the UDP cross traffic effects the TSN stream immediately. We examined the impact of cross traffic on time-sensitive communication in previous work [24]. Due to the delay of the first UDP frames in the TSSDN version of the network, the UDP cross traffic builds up at the TSSDN switches which increases the latency. The latency normalises after 7 ms. Afterwards both networks behave identical.

Figure 3 shows the minimum, maximum and average latency for the transmission of Ethernet frames from client 0 to client 1 and compares the simulation results of the UDP cross traffic and the TSN stream for both versions of the network. The guarantee of 750 µs shows the analytic maximum latency for the TSN stream. As the TSN stream in this example has the highest possible priority, the maximum latency introduced per port scheduling process is 250 µs (see max. latency AVB Stream Class A [25]). The TSN stream is scheduled at three output ports along the path: Client 0, Switch 0 and Switch 1. The minimum latency of the TSN stream is lower than that of the UDP cross-traffic in both versions of the network, because the transmission of cross traffic starts with some delay. After the initialization of the UDP path, the minimum latency is 320 µs for the TSN stream and 100 µs higher for UDP in both networks. The average latency is very similar for all transmissions. This is expected as TSN only guarantees a maximum latency for the TSN stream. The maximum latency of the UDP frames in the TSSDN version of the network is a result of the delay introduced by the SDN controller implementing the flow

rules. After the setup phase, the maximum Latency for UDP is 910 µs which is still above the TSN guarantee. In the version of the network without SDN the maximum latency for UDP is about 820 µs. Figure 3 shows that the time-sensitive streams are not delayed by SDN control traffic and all deadlines are met. The minimum, maximum and average latency of the TSN streams is about the same for both versions of the network.

Our results show that the combination of the SDN paradigm and TSN in the proposed architecture works as expected. The real-time traffic deadlines for the TSN streams are met and not affected by the introduction of SDN. At the same time, the forwarding is controlled by an SDN controller which enables the potentials described in Section III.

VI. Conclusion & Outlook

This paper made the case for in-vehicular Time-Sensitive Software-Defined Networking and explored its potentials and expectations. We presented our proposal of an integrated TSSDN switching that combines TSN and SDN capabilities. The core of this approach is an implementation of time-sensitive flows via OpenFlow, and we specified in detail how TSN streams can be mapped and registered at SDN controllers. In a case study of TSSDN in an event-based simulation environment, we could demonstrate that while gaining all the potentials of SDN in the automotive network its real-time capabilities remain unaltered.

In future work, we plan to investigate possibilities of transferring further parts of the TSN control logic into the SDN controller. One important part will be the implementation of static TDMA flows in a TSSDN and the analysis of possible side effects of SDN on such time triggered flows. Another open research challenge will be to investigate real-world potentials for vehicles including desired improvements of robustness and security. In addition, it will be of interest whether those security mechanisms could also be applied to time-sensitive flows without sacrificing the timing constraints.

Acknowledgements

This work is funded by the Federal Ministry of Education and Research of Germany (BMBF) within the SecVI project.

References

[1] K. Matheus and T. Königseder, *Automotive Ethernet*. Cambridge, United Kingdom: Cambridge University Press, Jan. 2015.

[2] IEEE, "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pp. 1–1993, July 2018.

[3] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[5] IEEE 802.1 TSN Task Group, "IEEE 802.1 Time-Sensitive Networking Task Group." [Online]. Available: https://1.ieee802.org/tsn/

[6] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)," in *2012 IEEE Vehicular Technology Conference (VTC Fall)*. Piscataway, NJ, USA: IEEE Press, Sep. 2012.

[7] Open Networking Foundation, "OpenFlow Switch Specification," ONF, Standard ONF TS-025, 2015. [Online]. Available: https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[8] D. Thiele and R. Ernst, "Formal Analysis Based Evaluation of Software Defined Networking for Time-Sensitive Ethernet," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 31–36.

[9] M. Herlich, J. L. Du, F. Schörghofer, and P. Dorfinger, "Proof-of-concept for a Software-defined Real-time Ethernet," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2016, pp. 1–4.

[10] N. G. Nayak, F. Dürr, and K. Rothermel, "Software-defined Environment for Reconfigurable Manufacturing Systems," in *2015 5th International Conference on the Internet of Things (IOT)*, Oct 2015, pp. 122–129.

[11] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive Software-defined Network (TSSDN) for Real-time Applications," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. New York, NY, USA: ACM, 2016, pp. 193–202.

[12] T. Kobzan, S. Schriegel, S. Althoff, A. Boschmann, J. Otto, and J. Jasperneite, "Secure and Time-sensitive Communication for Remote Process Control and Monitoring," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, Sep. 2018, pp. 1105–1108.

[13] P. Fussey and G. Parisis, "Poster: An In-Vehicle Software Defined Network Architecture for Connected and Automated Vehicles," in *Proceedings of the 2Nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services*, ser. CarSys '17. New York, NY, USA: ACM, 2017, pp. 73–74.

[14] K. Halba, C. Mahmoudi, and E. Griffor, "Robust safety for autonomous vehicles through reconfigurable networking," in Proceedings 2nd International Workshop on *Safe Control of Autonomous Vehicles,* Porto, Portugal, 10th April 2018, ser. Electronic Proceedings in Theoretical Computer Science, M. Gleirscher, S. Kugele, and S. Linker, Eds., vol. 269. Open Publishing Association, 2018, pp. 48–58.

[15] J. L. Du and M. Herlich, "Software-defined Networking for Real-time Ethernet." in *ICINCO (2)*, 2016, pp. 584–589.

[16] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in *Proc. of the 2010 Internet Network Management Conf. on Research on Enterprise Networking*, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3.

[17] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "FLOWGUARD: Building Robust Firewalls for Software-defined Networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 97–102.

[18] P. Varga, G. Kathareios, Á. Máté, R. Clauberg, A. Anghel, P. Orosz, B. Nagy, T. Tóthfalusi, L. Kovács, and M. Gusat, "Real-Time Security Services for SDN-based Datacenters," in *Network and Service Management (CNSM), 2017 13th Int. Conf. on*. IEEE, 2017, pp. 1–9.

[19] OpenSim Ltd., "OMNeT++ Discrete Event Simulator & INET Framework." [Online]. Available: https://omnetpp.org/

[20] T. Steinbach, H. Dieumo Kenfack, F. Korf, and T. C. Schmidt, "An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. New York: ACM-DL, Mar. 2011, pp. 375–382.

[21] CoRE Research Group, "CoRE Simulation Models for Real-time Networks." [Online]. Available: http://sim.core-rg.de

[22] N. Gray, "OpenFlowOMNeTSuite," 2016. [Online]. Available: https://github.com/lsinfo3/OpenFlowOMNeTSuite

[23] D. Klein and M. Jarschel, "An OpenFlow Extension for the OMNeT++ INET Framework," in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, pp. 322–329.

[24] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Beware of the Hidden! How Cross-traffic Affects Quality Assurances of Competing Real-time Ethernet Standards for In-Car Communication," in *2015 IEEE Conference on Local Computer Networks (LCN)*, Oct. 2015, pp. 1–9, lCN Best Paper Award.

[25] Institute of Electrical and Electronics Engineers, "IEEE 802.1AB - IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery," IEEE, Standard IEEE 802.1AB-2009, Sep. 2009.