Requirements Analysis and Performance Evaluation of SDN Controllers for Automotive Use Cases

Randolf Rotermund, Timo Häckel, Philipp Meyer, Franz Korf, and Thomas C. Schmidt
Dept. Computer Science, *Hamburg University of Applied Sciences*, Germany
{randolf.rotermund, timo.haeckel, philipp.meyer, franz.korf, t.schmidt}@haw-hamburg.de

***Abstract*—Future vehicles will be more connected than ever leading to increased dynamics in vehicle on-board networks. Software-Defined Networking (SDN) is a promising technology to meet the emerging needs for flexibility and security in future automotive use cases. Although SDN controllers have been evaluated in data center networks, to the best of our knowledge there is a lack of an analysis and performance evaluation of SDN controllers for automotive use cases. In this work we provide a detailed requirements analysis for the use of SDN controllers in cars. Based on this requirements analysis we choose existing controller implementations for a performance analysis. Finally, we analyze automotive specific use cases for SDN controllers with controller application examples and show how these can fulfill additional requirements. Our evaluation provides a helpful basis for the design and development of SDN controllers that can be used in vehicles.**

***Index Terms*—SDN, Software-Defined Networking, In-Vehicle Networks, Performance Analysis, Requirements Analysis**

I. Introduction

Modern cars implement functions using sensors, actuators, and Electronic Control Units (ECUs). They are linked via a combination of traditional bus systems such as Controller Area Network (CAN) and Ethernet technologies for newly selected links. In-vehicle networks will soon form flat, switched Ethernet topologies [1] forwarding real-time and cross-traffic concurrently. The Time-Sensitive Networking (TSN) collection of standards defined in IEEE 802.1Q-2018 [2] extends Ethernet with the ability to provide Quality-of-Service (QoS) guarantees and is the leading candidate for in-car networks.

Future cars will be connected with their environment (Vehicle-to-X) and thus will have increased network dynamics. This network traffic poses many challenges for safety critical real-time systems. Software-Defined Networking (SDN) is an impactful approach for dynamic traffic steering which originated in campus and data center networks [3]. Traditional network devices contain both the control plane, which is used for network routing, and the data plane, which is used for forwarding. In SDN the control plane, and the data plane are separated. A logically centralized component called the SDN controller implements the control plane while network devices in the data plane are only used for forwarding. The controller functionality can be enhanced with applications for specific use cases. In recent years, use cases for SDN extend to the vehicular domain. For example, the central control entity can help to manage the newly introduced dynamic traffic and protect the safety, security, and real-time constraints of a vehicle, which we analyzed in former work [4]. To utilize SDN in cars, the controller must fulfill various requirements. This work performs the necessary functional and performance analysis of suitable SDN controllers.

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

This paper evaluates SDN controllers with the focus on the use in a future automotive environments. One contribution and the base of this paper is a detailed requirements analysis for SDN controllers in cars. These requirements are used to pick the most relevant existing SDN controller implementations and examine the best fitting candidates in a performance analysis. Based on controller application examples we show the impact of SDN in a realistic vehicle on-board network and how controller applications can fulfill specific automotive requirements. Analysis of SDN controllers provide a detailed view on the current state of SDN controller capabilities and gives information about the fields in which SDN controllers must be adapted for the use in future vehicles.

The remainder of this paper is structured as follows. Section II gives an overview on related work. Section III presents all chosen requirements for SDN controllers in a vehicle. In section IV all controller candidates are listed. Section V shows the process and the findings of the performance analysis. The controller application examples are described in Section VI. This paper ends with a conclusion and outlook in Section VII.

II. Background & Related Work

Current vehicle on-board networks use multiple bus systems. Future in-vehicle networks will transform to flat Ethernet topologies [1]. Vehicle on-board communication consists of concurrent best effort and safety critical traffic. Real-time Ethernet extensions like IEEEs Time-Sensitive Networking (TSN) (802.1Q-2018 [2]) provide robust QoS guarantees for a safe and real-time proof internal communication.

In Software-Defined Networking, forwarding devices only implement the data plane. A central network controller takes over the tasks of the control plane. Most of the available SDN controllers use OpenFlow [5] as the Southbound API to communicate with forwarding devices. The forwarding devices use programmable flow tables to decide on the forwarding of frames. The controller installs rules into these flow tables to control the forwarding. Another way of managing heterogeneous devices in SDN is provided through NETCONF protocol (RFC 6241 [6]) with the data modeling language YANG. A

YANG model contains a data model of the network device and possible interactions. NETCONF uses the YANG model to define remote procedure calls to read state data and update the configuration of a network device. The SDN controller exposes information gathered from forwarding devices to controller applications. A variety of applications can be added to the controller which allow users to customize the behavior of the network. The controller applications use the controller's Northbound API to lookup the network state and to program the network. Further information about SDN can be found in the survey of Kreutz et al. [7].

In previous work, we examined the use of Time-Sensitive Software-Defined Networking (TSSDN) for in-vehicle networks [4], which enables the use of SDN in the automotive network while preserving the QoS guarantees provided by TSN. In automotive networks, SDN can add much value regarding safety, robustness, security, cost efficiency, and flexibility with easily updatable network configurations.

A lot of research has already been done on the performance of SDN controllers. Gonzales et al. [8] provide insights into the design of fault tolerant controllers. Mathebula et al. [9] and Cui et al. [10] have carried out detailed research into security deficits in controllers and corresponding solutions. Recent work in the field of performance analysis of SDN controllers has been done by Mostafavi et al. [11] evaluating controller performance and scalability in a comparative study. Mamushiane et al. [12] and Tello et al. [13] examine a collection of controllers using similar methodologies to the ones defined in RFC-8456 [14].

For use in the automotive domain, SDN controllers must fulfill automotive requirements. However, all of these investigations, SDN controllers are examined from the perspective of data center or campus networks. To the best of our knowledge, a controller evaluation according to the requirements of a car is still missing. This work focuses on the use of SDN controllers in vehicles, so only automotive-relevant metrics are considered.

III. Automotive Requirements on SDN Controllers

In vehicles, SDN controllers can be used for the dynamic control of network paths. We assume that the SDN controller is installed on a dedicated computer in the car, which is connected to the vehicle's on-board network. The collected requirements for the use of SDN controllers in cars are depicted in Table I. We differentiate between requirements that must be available as features of the controller implementation (evaluated in Section IV), depend on the performance of the controller (evaluated in Section V), or can be implemented in a controller application (evaluated in Section VI). The requirements are divided into groups and are explained below.

TABLE I
REQUIREMENTS FOR SDN CONTROLLERS IN IN-VEHICLE NETWORKS

ID	Requirement	Evaluation Type (Section)
A. Real-time requirements		
1)	Quality-of-Service	Performance (V-A), Application (VI-1)
2)	Scheduled configurations	Application (VI-2)
3)	Short start-up time	Performance (V-B)
B. Safety requirements		
1)	Transactions	Application (VI-3)
2)	Redundant paths	Feature (IV)
3)	Link failure detection	Performance (V-C)
4)	Controller redundancy	Feature (IV, V-D)
C. Security requirements		
1)	Secure Southbound API	Feature (IV)
2)	Secure Northbound API	Feature (IV)
3)	Application flow ownership	Feature (IV)
4)	Access control	Application (VI-4)
5)	Network statistics	Application (VI-5)
D. Remaining functional requirements		
1)	Scalability	–
2)	NETCONF/YANG support	Feature (IV)
E. Remaining non-functional requirements		
1)	Software quality	Feature (IV)
2)	Cost-efficiency	–
3)	Embedded system compatibility	Feature (IV)

A. Real-Time Requirements

1) Quality-of-Service: Future vehicle on-board networks will transport safety critical real-time traffic and best effort traffic concurrently. Thus, the network must provide Quality-of-Service (QoS) guarantees for critical flows. Real-time Ethernet extensions such as IEEE 802.1Q Time-Sensitive Networking (TSN) [2] can provide robust QoS guarantees. The SDN controller must enforce these guarantees by mapping each flow to certain QoS classes modelled in TSN. TSN specifies YANG data models (see 802.1Qcp), which can be used to configure the TSN components of network devices via the NETCONF protocol (RFC 6241 [6]), e.g. reprogram network schedules.

2) Scheduled configurations: Vehicle on-board networks use offline configured synchronous traffic (Time Division Multiple Access (TDMA)) for selected safety critical real-time communication. Therefore, devices contain configured periods with reserved timeslots that need to be synchronized in the network. This will not change with the introduction of SDN. If a controller makes changes to the configuration of synchronous traffic on multiple forwarding devices, these changes also need to be synchronized to a specific period so that all network devices update their state at the same time. Otherwise, inconsistency in the network traffic may occur. To avoid such a risk, an SDN controller must support scheduled configurations. Mizhrah et al. already presented a method for implementing accurate time-based updates [15]. The TSN YANG models also provide a mechanism to apply configuration changes at a specific time.

3) Short start up time: Almost all ECUs of a parked vehicle are inactive. As soon as the driver contacts the car, e.g., via keyless entry, ECUs and services are activated step by step. In current cars many ECUs, e.g. the door ECUs must be working after approximately 150ms to 200ms. The network must be functional to allow communication between ECUs. Network devices can have an offline configuration, which activates on start up and is functional without controller interaction. The network controller needs to be started before any additional flows can be added. This means that the requirements are not as high as without an offline configuration but are still in the range of a few seconds, e.g. for infotainment services.

B. Safety Requirements

1) Transactions: To provide functions of single network devices like firewalls and load balancing several flows have to be established by the controller. Each individual flow is important for the correct execution of a function. Sometimes multiple network paths must be changed at once to implement a complete function. According to Cui et al. an SDN controller should provide atomic transactions [16]. This means that a function can only be executed if all intended flows of a set of changes have been successfully established. If one or more flows could not be set up for certain reasons, all changes must be reset. This approach makes it possible to prevent inconsistency in the network, making it essential for the safe use of an SDN controller in the vehicle.

2) Redundant paths: A vehicle on-board network needs redundant network paths in case a path gets interrupted or is not available. A controller should be capable of managing a network with redundant paths. Safety critical network paths in cars will be configured redundantly in an offline configuration. Still, dynamically configured traffic needs to be rerouted by the SDN controller if a link fails. Management of redundant paths is a feature that most SDN controllers possess, especially all controllers supporting OpenFlow.

3) Link failure detection: Management of redundant paths is only possible when an SDN controller is able to detect changes in the network especially errors such as a broken link on a forwarding device. In case of a defect link the SDN controller must notify the driver e.g. by switching the malfunction indicator light on. This requirement is only fulfilled if the detection time is acceptable for a vehicle on-board network.

4) Controller redundancy: An SDN controller can be a single point of failure. Through redundancy, an SDN controller is logically centralized but physically distributed in the network. Redundant SDN controllers are often referred to as a controller cluster with 2 or more controller instances. The instances of a cluster must implement inter-controller communication with a master selection algorithm [17], and a shared network information base [8] to enable take-over on failure. Forwarding devices with statically configured flows can operate safety-critical traffic without the SDN controller and therefore there is no need for redundancy. On the other hand, if an SDN controller is used to steer safety-critical traffic, it is indeed a safety-critical component needing redundancy with at least a second controller instance in hot standby. If all controllers crashed, a fallback configuration on the forwarding devices is required to keep the system functional. As mentioned earlier most driving related traffic will be configured and verified offline. This fallback configuration preserves all safety critical network flows and makes the forwarding devices independent of the controller [18]. Although hot redundancy might not be needed in in-vehicle networks today, it might be needed for future scenarios, such as autonomous driving. The quality of the redundancy depends on the time it takes to handle a failure, which is evaluated in the performance analysis.

C. Security Requirements

Security requirements address the information security of the vehicle. Manipulation or interruption of safety-critical communication due to an attack can have fatal consequences. Hence, new standards such as the ISO/SAE 21434 [19] regulate the implementation of security methods in road vehicles.

1) Secure Southbound API: In SDN the connection between forwarding devices and the SDN controller via the Southbound API must be trusted at all time. It must be assured that attackers cannot imitate the forwarding devices or the SDN controller and thus change the network behavior. In data-center environments malfunctioning forwarding device, or the SDN controller can be quarantined [20] and flows rerouted. This procedure is not applicable in the automotive environment, because of the small numbers of network participants in comparison to data centers. To prevent the controller and forwarding devices from being attacked by each other, there must be a secure communication between the controller and switches ensured [9]. The trust between the controller and switches can only be ensured when there is an additional feature to secure the Southbound API.

2) Secure Northbound API: Multiple controller applications can run on an SDN controller using the Northbound API to implement network control mechanisms. To protect the control logic from malicious applications, there must be controller application trust establishment [9]. Applications must not be able to alter a network configuration without correct access rights. This requirement is fulfilled when a controller provides this function as an additional feature.

3) Application flow ownership: As multiple controller applications can exist on one controller, flows of forwarding devices can be altered by more than one application. If several applications can influence the network, two applications may attempt to manipulate the same network path and conflicts may occur. In addition, an application could modify the flows installed by other applications. A compromised application would then be able to remove flows created by other applications. To prevent this, a flow-application mapping must be provided. This ensures that applications can only remove or change flows that they installed themselves. The fulfillment of this requirement can only be provided as an additional feature the controller possess.

4) Access control: In automotive networks, some control messages should only be sent from a selected ECUs, e.g. the brake signal. It would be a security threat if any other device e.g. the infotainment system would be able to send those messages. Controller applications should therefore enforce this behavior by controlling which ECU can access which flows and send certain information.

5) Network statistics: Networks statistics are a collection of data obtained from the APIs of the controller e.g. packets

TABLE II
MATRIX EVALUATING WHICH FEATURE REQUIREMENTS FROM SECTION III ARE MET BY THE EXAMINED SDN CONTROLLER IMPLEMENTATIONS. (THE CANDIDATES SELECTED FOR OUR EVALUATION ARE MARKED IN GREEN)

Controller	**Redundant paths**	**Controller redundancy**	**Secure Northbound API**	**Secure Southbound API**	**Application-flow ownership**	**NETCONF/ YANG support**	**Software quality**	**Embedded system compatibility**
ONOS	✓	✓	✓	✓	X	✓	-	X
ODL	✓	✓	✓	✓	X	✓	-	X
Lumina	✓	✓	✓	✓	X	✓	-	X
Huawei Agile	✓	✓	✓	✓	X	✓	-	X
NOX	✓	X	X	✓	X	X	X	✓
POX	✓	X	X	✓	X	X	X	X
RYU	✓	X	X	✓	X	✓	-	X
Beacon	✓	X	X	✓	X	X	X	X
OpenMUL	✓	✓	X	✓	X	✓	X	✓
RunOS	✓	✓	X	✓	X	X	-	✓
Trema	✓	X	X	✓	X	X	-	X
Faucet	✓	✓	X	✓	X	X	-	X
OpenContrail	✓	✓	✓	✓	X	✓	X	X
Floodlight	✓	✓	✓	✓	X	X	X	X

processed by the controller or forwarding devices. This data is mandatory to reveal conspicuous behavior or even errors and is important for maintenance and network analysis.

D. Remaining functional requirements

1) Scalability: Most SDN controller implementations are designed for large scale deployments, such as data center environments, which frequently change in size. Hence SDN controllers usually must be scalable in relation to their environment. Regarding the rather small and static structure of an in-vehicle network, scalability is not an important requirement for these networks. It is listed to underline the unimportance of an usually relevant requirement for SDN controllers and is therefore not evaluated.

2) NETCONF/YANG support: Automotive Ethernet networks will most likely use the IEEE Time-Sensitive Networking (TSN) standard collection. The IEEE is working on the standardization (see IEEE 802.1Qcc & 802.1Qcp) of a Central Network Controller (CNC) configuring TSN devices via the NETCONF protocol (RFC 6241 [6]). SDN controller deployments in cars must support the configuration of the TSN components and therefore be able to manage YANG data models via NETCONF.

E. Additional non-functional requirements

1) Software quality: Software quality refers to the certification of the development process, software documentation and implementation according to the state of the art. A controller must fulfill all these important aspects of software quality required by automotive standards.

2) Cost-efficiency: Components in automotive networks must be cost effective in terms of hardware, licensing, and maintenance. Excessive system requirements of an SDN controller could mean that the installation of an SDN controller would not be worthwhile. This effect is even more important if there are several redundant SDN controllers running on different computers. Worth is a factor which depends on the financial resources available and is therefore not evaluated.

3) Embedded system compatibility: Considering cost efficiency and real-time requirements, an SDN controller should be compatible with hardware of embedded systems. In this case the SDN controller could run on cost-efficient and automotive proof ECU.

IV. CONTROLLER CANDIDATE SELECTION

We collected a list of over thirty SDN controllers containing both open source and commercial representatives. A portion of this list can be found in Table II. The most promising controller candidates (marked in green) were chosen for a more detailed evaluation. Most SDN controller implementations are built for data centers. To the best of our knowledge, none has been created for the automotive use case. Hence, no SDN controller on the market is able to fulfill all criteria of our requirements analysis. Table II contains only the requirements from Table I, which must be supported by the SDN controller implementation itself and cannot be implemented by SDN controller applications, for example.

Through the OpenFlow protocol all controllers can manage redundant paths. Controllers made for campus networks and prototyping like Ryu, NOX, POX, Beacon and Trema do not have support for redundancy. Only a few controllers have a secured Northbound API, but all controllers have a secured Southbound API due to the Transport Layer Security (TLS) support of OpenFlow. None of the examined SDN controllers can set ownership of flows for applications. ONOS, ODL, Lumina, Huawei Agile, Ryu, OpenMUL and OpenContrail all support the NETCONF protocol. It is not clear to what extent the SDN controllers meet the automotive requirements for software quality. Furthermore, the software quality requirements themselves are still partially open, as they depend on the use of the SDN controllers. Therefore, only active development is considered here. SDN controllers whose last deployment was more than a year ago are marked with an X. Only eight of over thirty controllers had a version not older than one year. Besides, RunOS the embedded system compatible controllers OpenMUL and NOX are not up to date.

The main selection considered all controllers which fulfilled as many of the controller features as possible. In the end the only SDN controllers fitting most of the requirements were the JAVA-based controllers ONOS, OpenDayLight and Lumina. The Huawei Agile controller and OpenContrail have very high system requirements and could not be installed on our test environment. Therefore, both controllers were not further considered. To have a more heterogeneous collection of controllers we added the C-based OpenMUL controller and the Python-based Ryu controller, although both controllers fulfill fewer requirements than the main selection.

V. Performance Evaluation

This section evaluates the performance of the chosen SDN controllers ONOS, ODL, Lumina, RYU and OpenMUL for all measurable requirements from Section III. Because of differences in the controller implementations not every measurement could be evaluated for every controller. The study was carried out on a Laptop with an Intel Core i7-4800MQ CPU @ 2.70GHz x 8 cores with 16 GB RAM running Ubuntu 16.04. Figure 1 shows the testbed used in the performance evaluation with the tools Mininet, Wireshark, and Cbench [1].

Cbench is an established benchmarking tool for SDN controllers. It is used to measure the controller response time (latency) and packet-in message rate (throughput) by emulating multiple OpenFlow switches and sending packet-in messages to a connected SDN controller. The throughput mode generates as many packet-in messages as possible and sends them to the connected controller, while the latency mode waits for a controller response before it sends the next message.

We use Mininet to simulate networks of different sizes and structures and measure network traffic between the emulated switches and the SDN controller using Wireshark. All SDN controllers have an OpenFlow and forwarding application enabled that creates flows for packet-in messages. All metrics have been tested with 10 or less forwarding devices, due to the relatively small and static in-vehicle network environment.

A. *Quality-of-Service*

QoS is provided by the forwarding devices in the network for all known flows. For dynamically created network flows the controller must process incoming packets in an appropriate time that does not violate the deadlines. The reference values for the time to be respected are the maximum end-to-end delays of vehicle services. Lim et al. of the BMW Research Group specified 10 ms maximum end-to-end delay for control data and 150 ms for multimedia data [21]. We measure the asynchronous message processing time which is defined as the delay between the arrival of an asynchronous message

[1] Cbench: github.com/mininet/oflops/tree/master/cbench

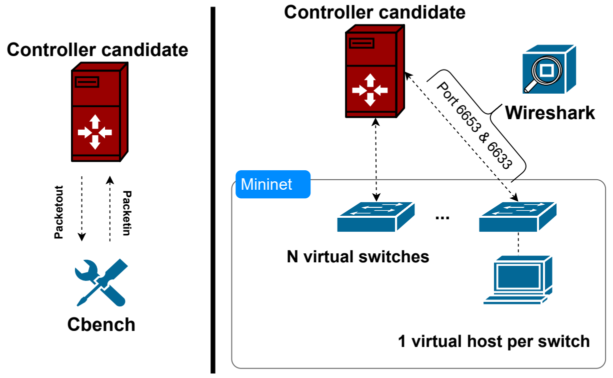

Fig. 1. Performance analysis testbed using Cbench, Mininet, and Wireshark

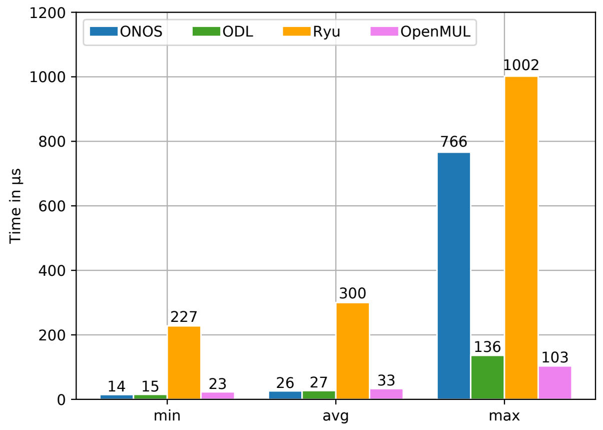

Fig. 2. Maximum processing time of packet-in messages from 10 measurements using latency mode in Cbench

at the controller port, and the subsequent outgoing message from the controller. The processing time is only relevant for the first packet of a new flow as the controller will implement forwarding rules that handle the flow in a forwarding device.

The asynchronous message processing time is tested via Cbench using the latency mode. It is repeated 10 times with a duration of 10 seconds. Cbench generates packet-ins for 100.000 MAC-addresses to ensure uniqueness of the generated flows. The results are depicted in Figure 2. Lumina was not tested because it does not provide a forwarding application that installs flows within a forwarding device.

The maximum asynchronous message processing time of all tested SDN controllers is about 1 ms. ODL has a maximum processing time of 136 µs, OpenMUL is at 103 µs, ONOS is at 766 µs and for RYU it is the highest with 1002 µs. This data indicates that all tested SDN controllers have a low enough processing time. In terms of the maximum end-to-end delays given above and due to the particularly low processing time of the OpenMUL SDN controller, which is implemented in C, only a negligible portion of time would be added when a controller would have to establish a new flow. To manage QoS not only the processing time is important but there also must be a guarantee that the processing time won't exceed a certain limit. The values of ONOS and Ryu have high jitter and none of the controllers gives a guarantee that these times are

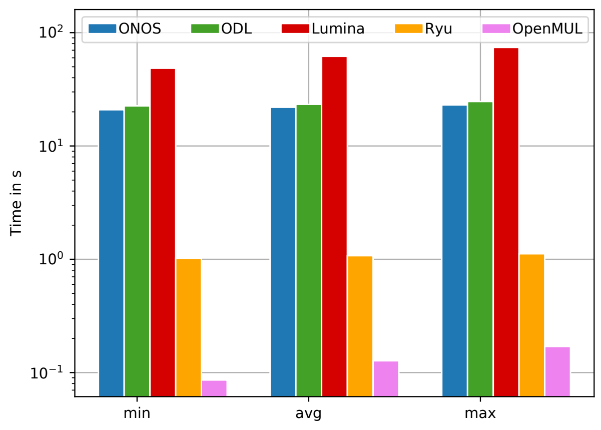

Fig. 3. Start-up time of SDN controllers until the OpenFlow port can be reached (10 measurements on a logarithmic scale)

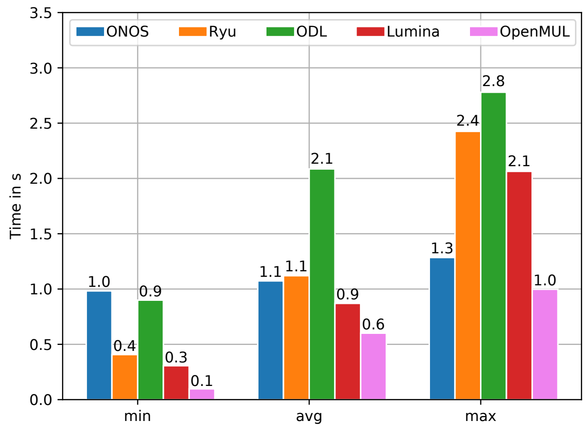

Fig. 4. Time between changes in the network topology, and their detection by the SDN controllers emulated with Mininet (10 measurements)

respected in every case, which is unacceptable for a real-time system. If these times are always guaranteed, all examined controllers could manage every automotive specific traffic.

B. Short Start-Up Time

In today's cars the start-up time of ECUs must not exceed $200\,\mathrm{ms}$, while multimedia- and other non-critical applications have a couple of seconds to start (see Section III). With a start-up configuration the switches can handle the static in-vehicle traffic without a controller. Still, the SDN controller should be available for other applications.

We measure the start-up time of a controller as the time between the start of the controller and all applications being completely operational. The start-up time was obtained with by measuring how long it takes from the controller start until the OpenFlow ports 6653 or 6633 are up. Figure 3 shows start-up time on a logarithmic scale. Although the test machine is more capable than a conventional ECU, the controllers needed several seconds to start up. The biggest part of the waiting time is caused by starting the system environment of the controller. ONOS and ODL have a relatively constant start-up time around $20\,\mathrm{s}$ while Lumina even reaches over $70\,\mathrm{s}$. RYU has a constant boot up time of $1\,\mathrm{s}$. Only the C representative OpenMUL is below $200\,\mathrm{ms}$ which clearly shows the advantage of embedded system capable controllers for in-vehicle networks. The results of the OpenMUL SDN controller meet the requirements of automotive applications.

C. Link Failure Detection

In a network design with redundant paths, the SDN controller has the possibility to dynamically change network paths to prevent disconnection. This can only be done if the controller detects link failures as soon as possible.

Link failure detection is measured using the topology change detection time of SDN controllers. When an OpenFlow switch detects a change on its ports it sends a port status message to the controller. Then the controller sends a Link Layer Discovery Protocol (LLDP) message to refresh the network topology. The topology change detection time is measured as the time between the port status message, and the corresponding LLDP message. For this analysis a Mininet network with one OpenFlow switch and one host was emulated. The connection between the switch and its host was interrupted to generate a port status message. Wireshark captured the relevant packets.

As shown in Figure 4, the OpenDayLight SDN Controller has the highest change detection time of nearly $3\,\mathrm{s}$. Apart from ONOS, all controllers have a maximum value of over $2\,\mathrm{s}$ and a minimum value of less than $1\,\mathrm{s}$. ONOS has a relatively constant change detection time of $1\,\mathrm{s}$, while the others have a relatively large difference between the minimum and the maximum value.

The link failure detection time of all controllers is too high in relation to the $150\,\mathrm{ms}$ maximum end-to-end delay of all automotive traffic types. With these values the maximum end-to-end delay of any automotive traffic would be exceeded multiple times before a corresponding action could be executed. The minimum values of the OpenMUL controller, indicate that it is possible for controllers to comply with the requirements.

D. Controller Redundancy

For redundant controllers in a controller cluster the failover time must be minimal. This time indicates how long it takes a controller cluster to fully replace the function of an unavailable controller. The failover time indicates how long the controller cluster is not available to manage new flows. A new controller instance must maintain all flows to prevent interruption.

The controller failover time was only tested on ONOS and ODL. RYU does not support redundancy, and the system requirements for multiple Lumina controller instances are too high for the test environment. In our test OpenMUL only executed a failover only when the controller was shutdown gracefully, for this reason the results of OpenMUL were also not considered although it supports redundancy. When a controller of a cluster has a fault, the cluster has a mech-

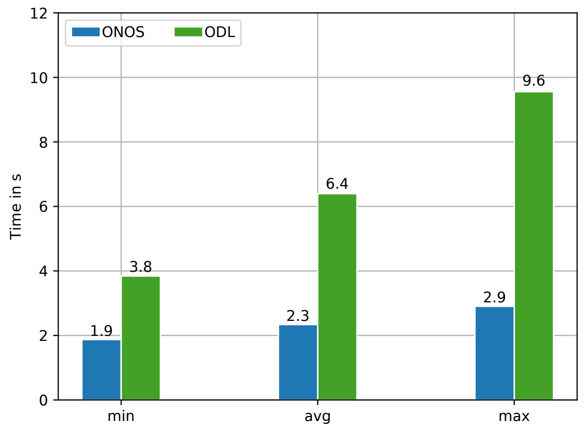

Fig. 5. Failover time until a failed controller is replaced by a hot standby instance of the cluster (10 measurements)

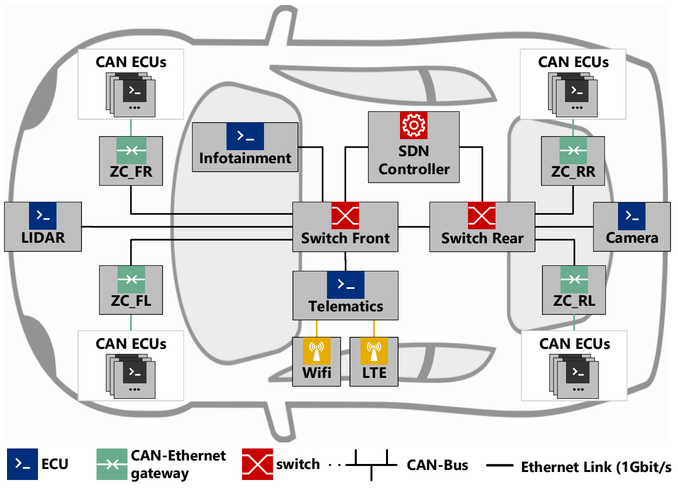

Fig. 6. Network topology of our automotive case study

anism to report the fault to the other members of the cluster and replace the controller. Forwarding devices, which were managed by the failed controller must be assigned to another active controller. This process is done with the OpenFlow role request and role reply messages. Controller failover time is measured as the time between a controller fault, and the role reply of the last unassigned switch. A Cluster with 3 controllers was created for the tested SDN controllers. A linear topology with 2 switches and 2 hosts was emulated. Flows are created between two hosts and UDP-traffic is generated for 60 seconds between the hosts to see if any packets are lost during the failover. The primary controller of the cluster gets turned off after 30 seconds. The network was captured via Wireshark.

Both controllers had no problem with maintaining the preconfigured flows in the switches after the failure and no packets were lost. Figure 5 shows how ODL and ONOS performed. With a maximum failover time of 3 s ONOS is considerably faster than ODL with 10 s. The results of over a second are too high for the in-vehicle environment if the controller manages and installs safety-critical flows. If the controller manages only non-safety-critical communication the results would be acceptable as this way functions such as multimedia applications could be maintained with only a small interruption. Still, it is questionable if controller redundancy is needed for the in-vehicle network.

VI. Controller Application Examples

The controller application examples demonstrate advantages of an SDN controller for vehicle on-board networks. We show how controller applications can fulfill the requirements QoS, scheduled configurations, transactions, access control, and network statistics from Section III. We use ONOS as our SDN controller because it is one of the better performing candidates, is well documented, and easy to setup. The applications are tested in a realistic vehicle on-board network.

Our vehicle on-board network is based on a real production car. The original network contains domain CAN busses and uses a central gateway architecture for transmission over domain boundaries. We converted the original CAN bus architecture into an Ethernet topology which is depicted in Figure 6. In this topology all ECUs are assigned to a zone according to their physical location in the vehicle (Front Left (FL), Front Right (FR), Rear Left (RL), Rear Right (RR)). Each zone has one zonal controller (ZC) connected to all CAN devices in its vicinity and to the Ethernet backbone. A ZC acts as a gateway between the CAN busses and the Ethernet backbone and executes other computation intensive tasks. The Ethernet backbone consists of two OpenFlow switches. This switch is capable of traffic prioritization with 802.1Q priorities, but it does not provide any other TSN features. The ONOS SDN controller runs on a dedicated ECU and is connected to the management port of the switches. Future networks will contain more Ethernet ECUs. Hence, we have introduced some additional ECUs that add native Ethernet traffic such as video streams and LIDAR data. The following describes how requirements can be fulfilled by controller applications.

1) Quality of Service: This controller application assigns each flow to a specific QoS queue. TSN is not supported by the switches so there is no scheduling in our prototype. Still, QoS is guaranteed by strict priority queues. The switch prioritizes packets when they are forwarded according to the VLAN priority of the queue, which the controller application has assigned to the flow. If no queue is specified in the flow rule the switch prioritizes via the VLAN Priority Code Point.

2) Scheduled configurations: This application adds an execution timestamp to the configuration changes in the forwarding devices. The implementation uses corresponding TSN YANG models and the NETCONF protocol. Together with a system-wide synchronized clock the changes can be executed on multiple devices simultaneously. Due to the lack of time synchronization in the used forwarding devices this application could not be deployed in the presented prototype.

3) Transactions: A controller application installs multiple flows in one bulk operation from an offline configured flow list. This allows an expert to configure and verify very precise flow rules for a specific situation. For example, changing the vehicle on-board network from the parking- to the driving state, which will prohibit some flows while prioritizing others. Such measures restrict the allowed network traffic to the smallest number possible for each situation. Still, it does not implement a complete transaction and there is no automatic rollback implemented when some flows are not installed.

4) Access control: An access control list application allows the dynamic forwarding of incoming data packets based on a whitelist and blacklist containing protocol types, IP addresses or full network flow descriptions. This increases the flexibility of the network by allowing the installation of dynamic flows while also restricting the access to on specific flows.

5) Network statistics: The SDN controller can gather network statistics. An application collects logs of reported, unknown or blacklisted behavior as well as statistics of forwarding devices. Those can then be evaluated, and flows can be changed accordingly.

VII. Conclusion & Outlook

SDN can help to manage the dynamic traffic and protect the safety, security, and QoS constraints of the vehicle. To utilize SDN in cars, the SDN controller must fulfill various requirements. We collected automotive specific requirements on SDN controllers and categorized them as controller features, performance measurements and controller applications.

Based on compliance to the controller feature requirements we chose controller candidates for our performance evaluation. No controller implementation was able to fulfill all feature requirements. All of them lack security features, most of them are not compatible to embedded systems because of their programming language and resource consumption, and many of them are out of date. We selected the most promising candidates to evaluate their performance. All tested controllers have an acceptable processing time, but no controller gives a guarantee that these values are always respected. The start-up time of the majority of the tested SDN controllers is too high for a vehicle, only the C representative OpenMUL has an acceptable start-up time of 200 ms. The failover time of ONOS and ODL is too high for safety critical automotive communication. However, it is questionable if controller redundancy is needed for non-safety-critical flows in the vehicle on-board network. Our controller application examples deployed in a realistic automotive network demonstrate how controller applications can be used to adapt a controller to fulfill additional requirements.

It was confirmed that SDN controllers must undergo a redesign for a safe installation in future vehicles. Future work should develop an automotive specific SDN controller implementation meeting the requirements discussed in this paper. In particular, the QoS requirements and the compatibility to embedded systems should be respected regarding resource consumption and the choice of programming language. In addition, an analysis of specific use cases for SDN controllers in vehicles could be performed. Various controller applications may be developed that fulfill additional requirements.

References

[1] S. Brunner, J. Roder, M. Kucera, and T. Waas, "Automotive E/E-Architecture Enhancements by Usage of Ethernet TSN," in *2017 13th Workshop on Intelligent Solutions in Embedded Systems (WISES)*. Piscataway, NJ, USA: IEEE Press, Jun. 2017, pp. 9–13.

[2] IEEE 802.1 Working Group, "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," IEEE, Standard Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014), Jul. 2018.

[3] 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.

[4] T. Häckel, P. Meyer, F. Korf, and T. C. Schmidt, "Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*. Piscataway, NJ, USA: IEEE Press, Apr. 2019, pp. 1–5.

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

[6] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," IETF, RFC 6241, June 2011.

[7] 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.

[8] A. J. Gonzalez, G. Nencioni, B. E. Helvik, and A. Kamisinski, "A Fault-Tolerant and Consistent SDN Controller," in *2016 IEEE Global Communications Conference (GLOBECOM)*. Piscataway, NJ, USA: IEEE Press, Dec 2016, pp. 1–6.

[9] I. Mathebula, B. Isong, N. Gasela, and A. M. Abu-Mahfouz, "Analysis of SDN-Based Security Challenges and Solution Approaches for SDWSN Usage," in *2019 IEEE 28th Int. Symposium on Industrial Electronics (ISIE)*. Piscataway, NJ, USA: IEEE Press, June 2019, pp. 1288–1293.

[10] H. Cui, Z. Chen, L. Yu, K. Xie, and Z. Xia, "Authentication Mechanism for Network Applications in SDN Environments," in *2017 20th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. Piscataway, NJ, USA: IEEE Press, Dec 2017, pp. 1–5.

[11] S. Mostafavi, V. Hakami, and F. Paydar, "Performance Evaluation of Software-Defined Networking Controllers: A Comparative Study," *Journal of Computer and Knowledge Engineering, 2020*, Mar. 2020.

[12] L. Mamushiane, A. Lysko, and S. Dlamini, "A Comparative Evaluation of the Performance of Popular SDN Controllers," in *2018 Wireless Days (WD)*. Piscataway, NJ, USA: IEEE Press, Apr. 2018, pp. 54–59.

[13] A. Tello and M. Abolhasan, "SDN Controllers Scalability and Performance Study," in *2019 13th Int. Conf. on Signal Processing and Com. Systems (ICSPCS)*. Piscataway, NJ, USA: IEEE Press, 2019, pp. 1–10.

[14] V. Bhuvaneswaran, A. Basil, M. Tassinari, V. Manral, and S. Banks, "Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance," IETF, RFC 8456, October 2018.

[15] T. Mizrahi, O. Rottenstreich, and Y. Moses, "TimeFlip: Using Timestamp-Based TCAM Ranges to Accurately Schedule Network Updates," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 849–863, April 2017.

[16] J. Cui, S. Zhou, H. Zhong, Y. Xu, and K. Sha, "Transaction-Based Flow Rule Conflict Detection and Resolution in SDN," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. Piscataway, NJ, USA: IEEE Press, Jul. 2018, pp. 1–9.

[17] O. Blial, M. Ben Mamoun, and R. Benaini, "An Overview on SDN Architectures with Multiple Controllers," *Journal of Computer Networks and Communications 2016*, vol. 2016, Apr. 2016.

[18] N. L. M. v. Adrichem, B. J. v. Asten, and F. A. Kuipers, "Fast Recovery in Software-Defined Networks," in *2014 Third European Workshop on Software Defined Networks*. Piscataway, NJ, USA: IEEE Press, Sep. 2014, pp. 61–66.

[19] Int. Org. for Standardization, "Road vehicles — Cybersecurity engineering," ISO, Geneva, CH, Standard ISO/SAE DIS 21434, 2020.

[20] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards Secure and Dependable Software-Defined Networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 55–60.

[21] H.-T. Lim, K. Weckemann, and D. Herrscher, "Performance Study of an In-Car Switched Ethernet Network without Prioritization," in *Communication Technologies for Vehicles*, T. Strang, A. Festag, A. Vinel, R. Mehmood, C. Rico Garcia, and M. Röckl, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 165–175.