HAW HAMBURG

BACHELORTHESIS
Randolf Rotermund

Performanzanalyse von SDN-Controllern für Ethernet-basierte Kommunikationsarchitekturen im Fahrzeug

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Randolf Rotermund

Performanzanalyse von SDN-Controllern für Ethernet-basierte Kommunikationsarchitekturen im Fahrzeug

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Thomas C. Schmidt

Eingereicht am: 9.Juli.2020

Randolf Rotermund

Thema der Arbeit

Performanzanalyse von SDN-Controllern für Ethernet-basierte Kommunikationsarchitekturen im Fahrzeug

Stichworte

Automotive, SDN, Netzwerk, Sicherheit, Auto, Fahrzeug, TSN, OpenFlow, NETCONF, YANG, Ethernet

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der Analyse und Evaluation von SDN-Controllern, die voraussichtlich in modernen Fahrzeugen mit Ethernet als Kommunikationsarchitektur eingebaut werden. Die Analyse erfolgt in drei Evaluationsschritten. Anfangs werden vier SDN-Controller mittels einer Anforderungsanalyse ausgewählt. Der zweite Schritt beinhaltet, die Performanzanalyse, die mittels einer Zusammenstellung von Metriken durchgeführt wird. Abschließend wird der SDN-Controller mit den besten Ergebnissen bei der Performanzanalyse in einem realen Fallbeispiel genutzt. In diesem Fallbeispiel werden mögliche Anwendungsgebiete und Schwächen von SDN-Controllern in einem Fahrzeugbordnetzwerk deutlich gemacht. Die Anforderungsanalyse hat gezeigt, dass die SDN Controller aus der Auswahl nicht alle kritischen Anforderungen erfüllen konnten. Die Ergebnisse der Performanzanalyse machen deutlich, dass SDN-Controller in der Lage sind innerhalb der zeitlichen Einschränkungen von Fahrzeugen zu agieren. Bei einem Ausfall würden diese für die zeitlichen Verhältnisse in einem Fahrzeug trotzdem zu lange brauchen. Das Fallbeispiel hat gezeigt, dass in einem realen Umfeld einige fehlende Anforderungen, durch Applikationen erfüllt werden können. Das Zeitverhalten bei einem Neustart oder Ausfall des Controllers ist dementsprechend der einzige ausschlaggebende Grund, der den Einbau in ein Fahrzeug verhindert.

Randolf Rotermund

Title of Thesis

Performance analysis of SDN-controllers for Ethernet-based communication architectures in vehicles

Keywords

automotive, SDN, network, security, safety, car, vehicle, Software Defined Networking, OpenFlow, NETCONF, YANG, Ethernet

Abstract

This thesis deals with the analysis and evaluation of SDN controllers, which are expected to be installed in modern vehicles with Ethernet as the communication architecture. The analysis is divided into three evaluation steps. Initially, four SDN controllers are selected by means of a requirement analysis. The second step includes the performance analysis, which is carried out by using a set of metrics. Finally, the SDN controller with the best results of the performance analysis is used in a realistic case study. In this case study, possible use cases and weaknesses of SDN controllers in a vehicle on-board network are pointed out. The requirements analysis has shown that the SDN controllers from the selection could not fulfil all critical requirements. The results of the performance analysis have made it clear that SDN controllers are able to operate within the time constraints of vehicles. In case of a failure, they would still take too long for the time constraints of a vehicle. The last evaluation step has shown, that in a real world environment some missing requirements, can be fulfilled by applications. Accordingly, the time response in the event of failure is the only decisive reason, which prevents the installation in a vehicle.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

BMW Bayerische Motoren Werke AG.

CAN Control Area Network.

DoIP Diagnostics over Internet Protocol.

DoS Denial of Service.

ECU Electronic Control Units.

ESP elektronische Stabilitätsprogramm.

IETF Internet Engineering Task Force.

LIN Local Interconnect Network.

LLDP Link-Layer-Discovery-Protocol.

MAC Media-Access-Control.

MD-SAL Model Driven Service Abstraction Layer.

MOST Media Oriented Systems Transport.

NADS Network-Anomaly-Detection-System.

NETCONF Network Configuration Protocol.

ONF Open Network Foundation.

ONOS Open Network Operating System.

PCP Priority Code Point.

QoS Quality of Service.

RPC Remote Prodecure Call.

SDN Software Defined Networking.

SecVI Security for Vehicular Information.

SOME/IP Scalable Service-Oriented Middleware over IP.

TLS Transport Layer Security.

TSN Time Sensitive Networking.

TSSDN Time Sensitive Software Defined Networking.

XML Extenisible Markup Language.

YANG Yet Another Next Generation.

1 Einleitung

Abgesehen von den mechanischen Einzelteilen, bestehen moderne Fahrzeuge aus einer großen Anzahl an elektronischen Steuereinheiten. Diese werden Electronic Control Units (ECU) genannt. Die ECUs steuern und überwachen das Fahrzeug, vereinfachen die Diagnose von Fehlern oder Schäden im System und ermöglichen komplexe Dienste, wie zum Beispiel Fahrerassistenzsysteme oder das Elektronische Stabilitätsprogramm (ESP). ECUs übernehmen unterschiedliche Funktionen im Fahrzeug und kommunizieren heutzutage über Control Area Network (CAN) und andere Bussysteme, wie beispielsweise von der OPEN Alliance SIG spezifiziertes Ethernet [60]. Ohne die Kommunikation zwischen ECUs kann ein Großteil der Funktionalitäten eines modernen Fahrzeuges nicht sichergestellt werden. Es besteht die weiterhin steigende Tendenz, Anbindungen zur Außenwelt einzubauen, beispielsweise für Over-the-Air-Updates oder Navigation. Dies erhöht die Komplexität und steigert die Dynamik des Netzwerkverkehrs innerhalb des Fahrzeuges [61]. Security im Fahrzeugbordnetzwerk wird durch Schnittstellen zur Außenwelt auch zu einem relevanten Aspekt, da das moderne Fahrzeug nicht mehr als geschlossenes System gilt [64]. Infolgedessen existieren Spezifikationen, die sich speziell mit der Security von Fahrzeugbordnetzwerken auseinandersetzen [29].
In zukünftigen Fahrzeugbordnetzwerken wird mit hoher Wahrscheinlichkeit ein Großteil des Kommunikationsflusses auf Ethernet basieren [61]. Ethernet als lang etablierte Kommunikationstechnologie erlaubt es Fahrzeugen von der hohen Anzahl an Protokollen, aus der Internetdomäne zu profitieren. Software Defined Networking (SDN) ist ein Netzwerkkonzept, das unter anderem die Verwaltung von dynamischen Netzwerken vereinfacht [40]. SDN wird üblicherweise für die Verwaltung von Datenzentren oder lokalen Netzwerken genutzt. Das grundlegende Ziel von SDN ist ein Controller, der alle steuernden und verwaltenden Aufgaben im Netzwerk übernimmt. Dadurch können Switches und andere Netzwerkgeräte durch simple Weiterleitungsgeräte ersetzt werden. SDN bringt die Vorteile mit sich, die Verwaltung eines Netzwerkes zu vereinfachen und durch Erweiterbarkeit und Programmierbarkeit des SDN-Controllers das Verhalten des Netzwerkes unmittelbar zu bestimmen. Durch ethernetbasierte Kommunikationsarchitektu-

ren kann SDN genutzt werden, um den Umständen die mit der steigenden Dynamik in Fahrzeugbordnetzwerken entstehen, etwas entgegenzusetzen. Zudem können auch Securitymaßnahmen durch SDN im Fahrzeug durchgesetzt werden. Es gibt momentan eine Vielzahl an SDN-Controller, durch ihre Eigenschaften und Ansätze bei der Entwicklung können diese sich stark voneinander unterscheiden. Es muss ein Vergleich zwischen verschiedenen SDN-Controller-Lösungen stattfinden. So kann entschieden werden, welche Controller für ein Fahrzeug in Betracht gezogen werden können.

Das Ziel dieser Arbeit ist es, eine Auswahl an SDN-Controllern, auf deren Tauglichkeit für den Gebrauch im Fahrzeug in einem dreistufigen Evaluationsprozess auszuwerten. Anhand einer Anforderungsanalyse werden die wichtigsten Eigenschaften von Software in einem Fahrzeug zusammengestellt, um Anforderungen speziell für SDN-Controller in einem Fahrzeugbordnetzwerk zu erstellen. Diese Anforderungen werden genutzt, um ungeeignete Controller auszusortieren. Damit wird ein Kollektiv an SDN-Controllern ausgewählt wird, die für diese Arbeit relevant sind. Anhand einiger ausschlaggebender Anforderungen konnte die Anzahl von mehr als dreißig Controllern auf vier relevante Controller reduziert werden. Mithilfe von Metriken wird ein Vergleich der SDN-Controller in einer Performanzanalyse ausgeführt. Die Ergebnisse geben im besten Fall einen oder mehr Controller wieder, die für den Einbau in ein Fahrzeugbordnetzwerk in Betracht gezogen werden können.

Die Arbeit ist wie folgt gegliedert: Kapitel 2 vermittelt grundlegende Informationen zum Verständnis der Arbeit. Zu den Grundlagen gehören ein detaillierter Einblick in SDN und eine Erklärung von Southbound-APIs (OpenFlow und NETCONF), Echtzeitsystemen und Kommunikationsarchitekturen im Fahrzeug. In Kapitel 3 wird eine Liste von Anforderungen zusammengestellt, die SDN-Controller erfüllen müssen. Mit den Anforderungen wird eine Auswahl von Controllern erstellt, die in Kapitel 4 einer Performanzanalyse anhand von zusammengestellten Metriken unterzogen werden. Der SDN-Controller mit der besten Leistung in der Performanzanalyse wird in Kapitel 5 in einem realistischen Tischaufbau als Fallbeispiel für SDN-Controller genutzt. Als Abschluss folgt Kapitel 6 mit einem Ausblick und einem Fazit der Arbeit.

2 Grundlagen

Dieses Kapitel soll die Grundlagen zum allgemeinen Verständnis dieser Arbeit nahe bringen. Es wird die Architektur und die Funktion vom Software Defined Networking und von den Schnittstellen des SDN-Controllers erklärt.

2.1 Einführung in Software Defined Networking

Übliche Netzwerkgeräte in einem traditionellen Netzwerk sind nicht nur für das Weiterleiten von Paketen zuständig, sondern steuern auch den Netzwerkverkehr. Datenpakete kommen bei Netzwerkgeräten wie Switches, Routern oder Firewalls an und werden nach den vorhandenen Richtlinien im Netzwerk verarbeitet und weitergeleitet. Die verteilte Logik einzelner Netzwerkgeräte erschwert besonders in größeren Netzwerken die Anpassung an definierte Richtlinien und Rekonfiguration für Ausfälle und Änderungen [33].
In konventionellen Netzwerken gibt es drei Schichten, diese bilden die grundlegenden Komponenten in Telekommunikationsarchitekturen: *Control Plane*, *Data Plane* und die *Management Plane*. Die Control Plane ist die Kontrollschicht und steuert den Netzwerkverkehr sowie das Weiterleitungsverhalten von Geräten in der Data Plane. Die Data Plane oder auch Datenschicht ist für den Transport von Paketen zuständig und bildet den Netzwerkverkehr. Die Management Plane ist die Schicht, die für die Überwachung und Verwaltung des Netzwerkes zuständig ist. Diese wird üblicherweise von einem Systemadministrator realisiert, der über Software das Netzwerk und das Netzwerkverhalten verändern kann. Aufgrund des relativ ähnlichen Aufgabenbereichs mit der Control Plane wird die Management Plane oft als Teil dieser angesehen. In Abbildung 2.1 wird einem traditionellen Netzwerk (oben) ein SDN-Netzwerk gegenübergestellt (unten). SDN entkoppelt die Control Plane von der Data Plane, indem SDN die vollständige Kontrolllogik des Netzwerkes in eine zentrale Softwarekomponente extrahiert, den SDN-Controller.

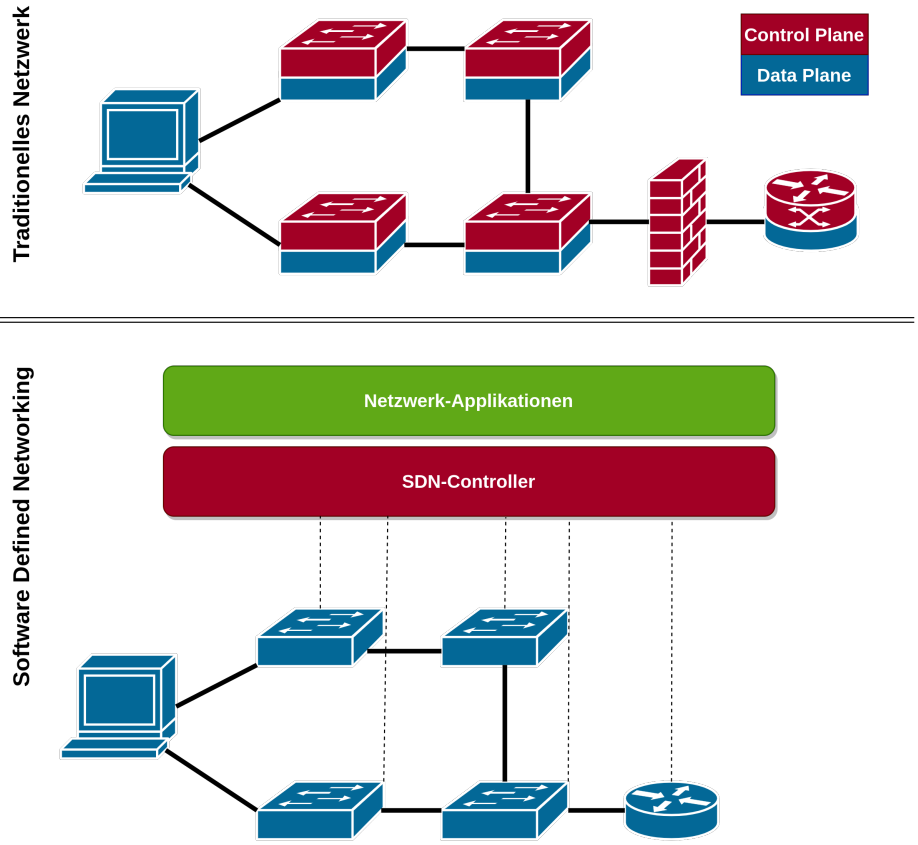

Abbildung 2.1: Traditionelle Netzwerke und SDN im Vergleich [33]

Netzwerkgeräte werden zu einfachen Weiterleitungsgeräten. Weiterleitungsgeräte leiten ankommende Datenpakete nur in eine von der Control Plane angegebene Richtung an und einzelne Netzwerkfunktionen werden in Netzwerkapplikationen des SDN-Controllers abstrahiert. In SDN ist jedes Weiterleitungsgerät mit dem SDN-Controller verbunden. Der SDN-Controller besitzt somit eine globale Sicht auf die Netztopologie. SDN macht durch die Entkopplung von Control Plane und Data Plane Netzwerke flexibel, vereinfacht die Skalierung und Verwaltung und ermöglicht eine anwendungsgesteuerte Programmierung von Netzwerken. In einem Fahrzeugbordnetzwerk ist vor allem die Möglichkeit zur dynamischen Verwaltung von Netzwerkflüssen gewünscht, um durch den SDN-Controller das Netzwerk von unerlaubten Datenströmen abzuschirmen. Abgesehen von der Control Plane und der Data Plane unterscheidet sich auch die Management Plane von der Implementierung in traditionellen Netzwerken.

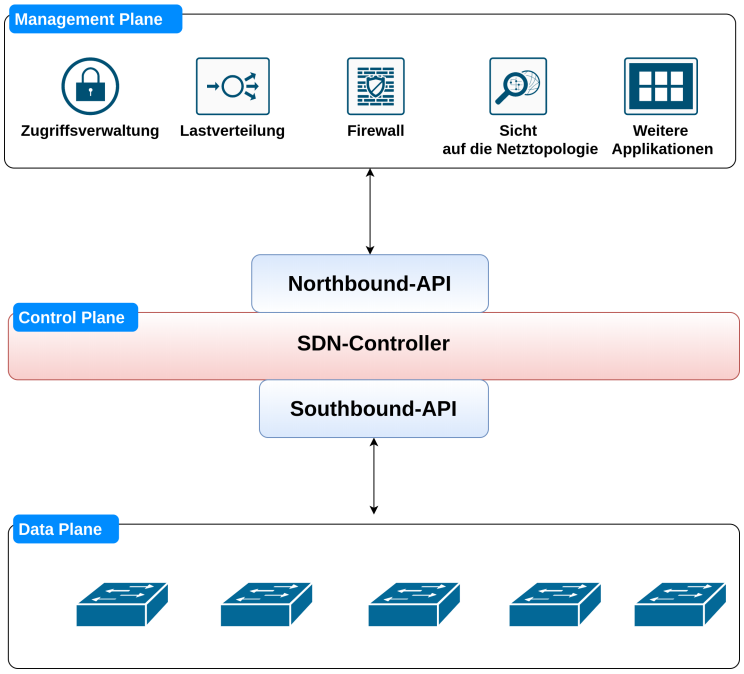

Abbildung 2.2: SDN Schichtenarchitektur

Die Management Plane wird durch Applikationen realisiert, die das Verhalten des SDN-Controllers bestimmen. Applikationen eines SDN-Controllers realisieren nicht nur Netzwerkfunktionen. Diese sind auch in der Lage, Richtlinien im Netzwerk durch den SDN-Controller durchzusetzen oder die Netztopologie zu visualisieren.
Abbildung 2.2 zeigt, wie die Schichten in SDN aufgeteilt sind. Auffallend ist die Verbindung der Schichten durch zwei Schnittstellen: das Northbound-API und das Southbound-API. Das Northbound-API verbindet Applikationen mit dem SDN-Controller und ermöglicht den Applikationen, Netzwerkinformationen für deren Funktionen am Controller abzurufen. Üblicherweise greifen Applikationen mit Anforderungen von REST [17] oder RESTCONF [8] auf den SDN-Controller zu. Das Southbound-API verbindet den SDN-Controller mit den Weiterleitungsgeräten. Die genaue Funktion sowie die Relevanz vom Southbound-API für SDN werden im folgenden Abschnitt 2.2 ausführlich erklärt.

2.2 Southbound-APIs

Ein Southbound-API in SDN definiert das Kommunikationsverhalten zwischen SDN-Controller und den Weiterleitungsgeräten im Netzwerk. Das Southbound-API erlaubt

es dem Controller zum Beispiel Veränderungen an den Weiterleitungsgeräten und somit direkt am Netzwerkverkehr selbst vorzunehmen. In den letzten Jahren wurden viele SDN-Controller entwickelt, die hauptsächlich auf das Southbound-API OpenFlow gesetzt haben. Durch die Weiterentwicklung und weitläufige Nutzung von SDN-Controllern ist aufgefallen, dass OpenFlow als alleiniges Southbound-API nicht flexibel genug ist. Medved et al. haben dieses Problem aufgegriffen und mit der modellbasierten Architektur des OpenDayLight SDN-Controllers einen Controller vorgestellt, der nicht nur auf OpenFlow basiert [41]. Das Network Configuration Protocol (NETCONF) ist ein Southbound-API, das sich abgesehen von OpenFlow auch in SDN etabliert hat und von mehreren Controllern unterstützt wird (siehe Abschnitt 3.7).

2.2.1 OpenFlow

OpenFlow ist ein Protokollstandard, der von der Open Network Foundation (ONF) [51] veröffentlicht wurde. OpenFlow dient als Southbound-API und ermöglicht dem SDN-Controller direkten Zugriff auf die Datenschicht im Netzwerk. Somit kann der Controller mit dem OpenFlow-Protokoll das Weiterleitungsverhalten von Weiterleitungsgeräten steuern und überwachen. Der Netzwerkfluss von Datenpaketen zwischen Netzwerkteilnehmern wird als Fluss bezeichnet. OpenFlow unterstützende Switches können sicher über Transport Layer Security (TLS) mit dem Controller kommunizieren und enthalten mindestens eine oder mehrere Fluss-Tabellen, die im Gegensatz zu den MAC-Address-Tabellen in herkömmlichen Switches vom SDN-Controller modifiziert werden können und nicht aus Port und MAC-Adresse bestehen. Da OpenFlow-Switches nicht Port und MAC basiert weiterleiten wird spezielle Hardware gebraucht. Diese ermöglicht eine auf Fluss basierte Paketweiterleitung.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Tabelle 2.1: Fluss-Tabelleneintrag eines OpenFlow-Switches [54]

Einträge einer Fluss-Tabelle sind wie in Tabelle 2.1 nach der OpenFlow-Spezifikation der ONF aufgebaut [54]. Wichtig für die Einzigartigkeit eines Fluss-Tabelleneintrages sind die *Match Fields* und die *Priority*. Die Match Fields geben die Paketheader und den Eingangsport an. Diese werden genutzt, um eingehende Datenpakete dem jeweiligen

Fluss-Eintrag zuzuordnen. Die Priority gibt die Priorität der Zuordnung an. *Counters* sind eine Reihe von Zählern, die Statistiken wiedergeben, wie zum Beispiel die empfangene Paketanzahl über einen bestimmten Port. *Instructions* sind Anweisungen, die bei einem Paketeingang über einen Fluss ausgeführt werden. Veränderung am Paketinhalt oder der Verarbeitung des Paketes im Switch wären Beispiele für Instructions. Um den Speicherverbrauch von nicht mehr genutzten Fluss-Einträgen zu verhindern, gibt es das *Timeout*-Feld. Es gibt an, wie lange ein Fluss-Eintrag ungenutzt bzw. im Leerlauf sein darf, bis dieser gelöscht wird. Das Cookie-Feld wird vom Controller erzeugt, um nach Fluss-Informationen wie Statistiken oder Modifizierungen zu filtern. Die Paketverarbeitung folgt einem Algorithmus, der alle Felder abgesehen vom Cookie-Feld nutzt.

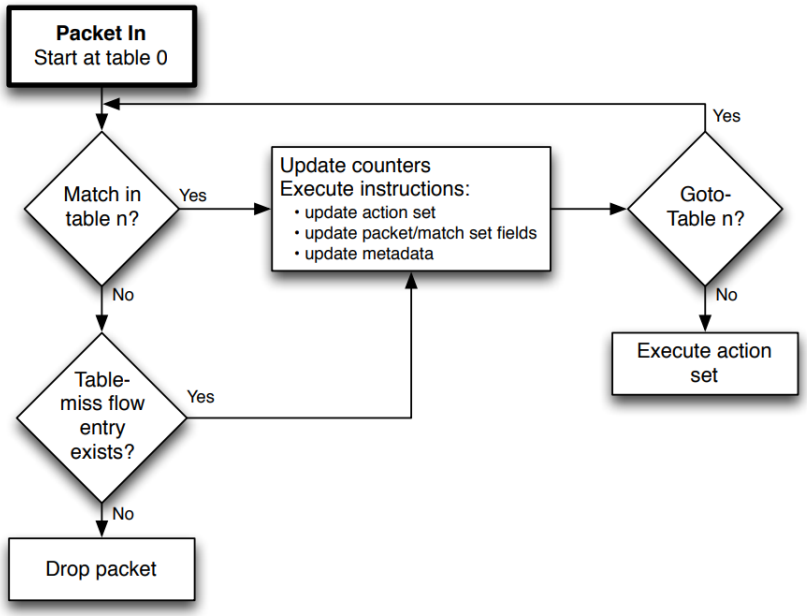

Abbildung 2.3: Paketverarbeitung in einem OpenFlow-Switch [54]

Aus Abbildung 2.3 kann entnommen werden, wie Pakete in einem OpenFlow-Switch verarbeitet werden. Bei einem Paketeingang an einem OpenFlow-Switch-Port werden die Match Fields des Paketes nach einem passenden Eintrag in der Fluss-Tabelle durchsucht. Gibt es mehrere passende Einträge, wird der Eintrag mit der höchsten Priorität genutzt. Die entsprechenden Zähler werden bei einem Paket erhöht, das einem Fluss zugeordnet werden kann und es werden Anweisungen für den bestimmten Fluss ausgeführt. Abschließend wird überprüft, ob es nötig ist in der nächsten Fluss-Tabelle nach einer Zuordnung zu suchen. Falls nicht, wird ein Satz von flowspezifischen Aktionen ausgeführt und das

Paket wird weitergeleitet. Falls ein Fluss keiner Tabelle zugeordnet werden kann, wird von einem Table-miss gesprochen. Wenn ein Table-miss durch einen unbekannten Fluss verursacht wird, werden vorgegebene Aktionen ausgeführt. Beispielsweise können Paketen unbekannter Flüsse an einen bestimmten Netzwerkteilnehmer weitergeleitet werden. Wurde keine Aktion für einen Table-miss vorgegeben werden unbekannte Pakete üblicherweise verworfen.

Die OpenFlow-Spezifikation definiert eine Vielzahl an Nachrichten. Für diese Arbeit sind die Folgenden von Relevanz:

- *Hello:* Diese Nachricht wird zum Verbindungsaufbau zwischen Controller und Switch genutzt.
- *PacketIn:* Diese Nachricht wird von Switches an den Controller gesendet. Dies kann geschehen, entweder wenn ein Table-miss verursacht wurde oder wenn eine Aktion dies explizit anfordert.
- *PacketOut:* Eine ohne spezifischen Inhalt gefüllte Nachricht die vom Controller ausgeht.
- *FlowMod:* Diese Nachricht ermöglicht es dem Controller die Fluss-Tabellen von Switches zu modifizieren.
- *PortStatus:* Eine Benachrichtigung der Switches an den Controller, die Veränderungen der Switch-Ports wiedergibt.
- *RoleReq:* Controller senden diese Nachricht an Switches, wenn diese eine neue Rolle anfordern.
  Es existieren zwei Rollen: Master und Slave. Diese beschreiben die Rolle des SDN-Controllers in einem Cluster von mehreren Controllern. Die Master-Rolle bedeutet, dass der SDN-Controller die Erlaubnis hat den Switch zu modifizieren. Die Slave-Rolle gibt an, dass ein anderer SDN-Controller im Cluster die Master-Rolle besitzt.
- *RoleReply:* Switches bestätigen mit dieser Nachricht die angeforderte Rolle durch einen *RoleReq*.

2.2.2 NETCONF

NETCONF ist ein von der Internet Engineering Task Force (IETF) [3] entwickeltes Protokoll für die Konfiguration und Verwaltung von Netzwerkgeräten. In SDN wird NETCONF genau wie OpenFlow als Southbound-API genutzt. Remote Prodecure Call (RPC) wird als Übertragungstechnik über eine gesicherte verbindungsorientierte Verbindung genutzt, um Datenmodelle von Geräten abzurufen und zu senden. Alle Daten werden bei der Übertragung in Extenisible Markup Language (XML) codiert. Anders als in OpenFlow ist die verwaltende und überwachende Instanz in NETCONF nicht der Server, sondern der Client. In Bezug auf SDN bedeutet dies, dass der SDN-Controller als NETCONF-Client agiert und die Weiterleitungsgeräte einzelne NETCONF-Server sind. Das RFC 6241 [16] spezifiziert NETCONF und unterscheidet zwischen Konfigurationsdaten und Zustandsdaten auf NETCONF-Servern. Yet Another Next Generation (YANG) [9] ist eine Sprache für die Datenmodellierung. Diese wird genutzt, um Zustandsdaten und Konfigurationsdaten für das NETCONF-Protokoll zu modellieren. Zustandsdaten sind Informationen, die vom Controller abgerufen, aber nicht von diesem manipuliert werden können. Die Informationen dienen als Sammlung von Statistiken des Gerätes. Konfigurationsdaten dagegen werden genutzt, um das Gerät zu manipulieren. Diese werden in verschiedene Datenspeicher unterschieden. Wie in Abbildung 2.4 ersichtlich hat jedes Gerät immer mindestens einen laufenden Datenspeicher. Dieser spiegelt die aktuelle Konfiguration des Gerätes wider und kann weitere optionale Datenspeicher besitzen. Abgesehen vom laufenden Datenspeicher gibt es noch Konfigurationsdatenspeicher-Kandidaten. Ein Konfigurationsdatenspeicher-Kandidat beinhaltet wie der laufende Datenspeicher Konfigurationen am Gerät. Im Gegensatz zum laufenden Datenspeicher ist ein Kandidat nicht aktiv und kann bei Bedarf zum neuen laufenden Datenspeicher eingestuft werden, indem der Kandidat durch den aktuell laufenden Datenspeicher ausgewechselt wird. Kandidaten machen es möglich, vollständige Konfigurationssätze an ein Gerät zu übergeben und vermeiden somit Inkonsistenzen bei der Manipulierung des laufenden Datenspeichers. Weitere Möglichkeiten zur Vermeidung von Inkonsistenzen sind das vorherige Sperren eines Datenspeichers zur Verhinderung des Zugriffs von anderen Clients und das Speichern des aktuellen Datenspeichers in einer Checkpoint-Datei.

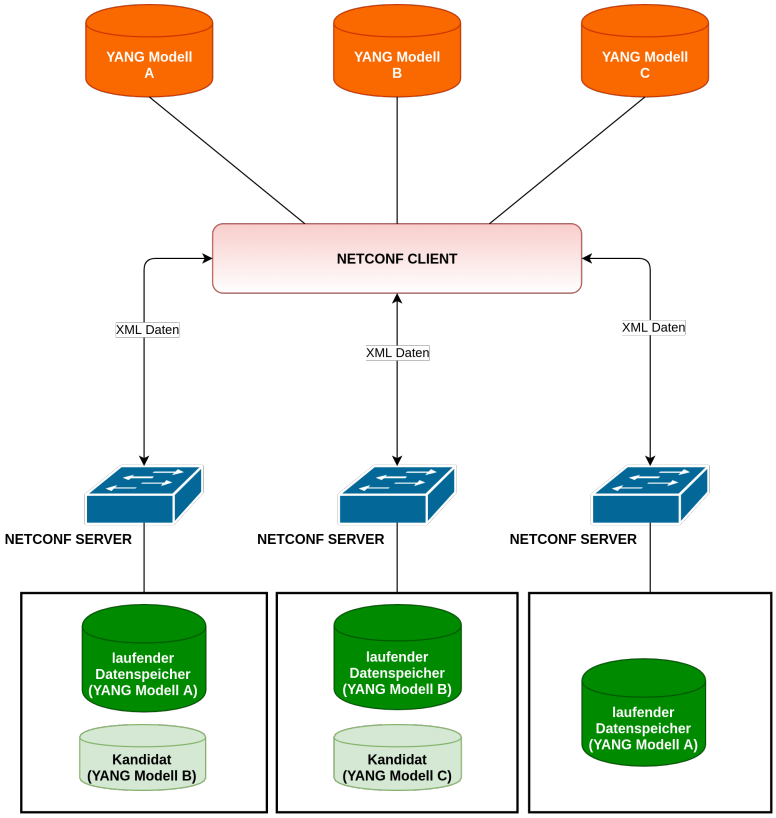

Abbildung 2.4: Server und Client in NETCONF

Diese kann bei Fehlern einer NETCONF-Operation genutzt werden, um den Datenspeicher zur vorherigen Konfiguration wiederherzustellen. Einige Geräte besitzen einen Konfigurationsdatenspeicher zum Neustart, der den laufenden Datenspeicher bei Inbetriebnahme des Gerätes ersetzt. Bei Verbindungsaufbau werden grundlegende Eigenschaften zwischen Client und Server ausgetauscht, so kann beispielsweise das Vorhandensein von Datenspeicher-Kandidaten festgestellt werden.
Um die Datenspeicher der Server zu verändern und abzurufen, definiert das NETCONF-Protokoll eine Reihe an Basisoperationen. Jedes NETCONF-fähige Gerät muss diese unterstützen:

- *get* ruft den laufenden Datenspeicher und die Zustandsdaten eines Gerätes ab.
- *get-config* ruft alle Informationen eines bestimmten Datenspeichers ab.
- *edit-config* lädt einem Konfigurationssatz in einen vorgegebenen Datenspeicher.

- *copy-config* ersetzt den Inhalt eines spezifizierten Datenspeicher mit dem eines anderen.
- *delete-config* löscht einen angegeben Datenspeicher, außer den laufenden.
- *lock* sperrt einen Datenspeicher, zum Beispiel gegen andere Clients.
- *unlock* entsperrt einen Datenspeicher.
- *close-session* schließt eine NETCONF-Sitzung und alle verknüpften Sitzungen, entsperrt außerdem alle gesperrten Datenspeicher und löst genutzte Ressourcen. Eine Sitzung schließt erst, wenn alle Operationen vollständig durchgeführt wurden.
- *kill-session* schließt eine NETCONF-Sitzung und alle verknüpften Sitzungen, entsperrt außerdem alle gesperrten Datenspeicher und löst genutzte Ressourcen. Alle Operationen werden sofort abgebrochen.

Eine übliche Vorgehensweise zur Veränderung des Datenspeichers eines Servers wäre als erstes den Datenspeicher mit *lock* zu sperren, danach eine Checkpoint-Datei des Datenspeichers mit *copy-config* zu erstellen und mit *copy-config* oder *edit-config* den Datenspeicher zu überschreiben. Der Inhalt des Datenspeichers sollte nach der Operation mit *get-config* validiert werden. Abschließend kann dieser wieder mit *unlock* entsperrt werden. Bei gleichzeitiger Veränderung mehrerer NETCONF-Server müssen die gleichen Schritte parallel vollzogen werden. Es kann entschieden werden, ob die Manipulierung mehrerer Server als eine Gesamtkonfiguration vom gesamten Netzwerk oder als ein Teil des Netzwerkes gesehen wird. Ein Fehler einer Operation auf einem einzelnen Server müsste in solch einem Fall dazu führen, dass alle von der Konfiguration betroffenen Datenspeicher durch Checkpoints auf den Stand vor der Operation gebracht werden. Bei einer individuellen Sicht der Konfigurationen würden nur die Datenspeicher zurückgesetzt werden, bei denen Fehler entstanden sind. In vielen Fällen kann diese Art der Sicht zu Inkonsistenzen im Netzwerk führen. Beispielsweise könnten Netzwerkgeräte einer neuen VLAN-Gruppe zugeordnet werden. Hierbei würde die individuelle Sicht auf die Konfiguration dazu führen, dass alle Geräte mit eingetretenen Fehlern während der NETCONF-Konfiguration nicht mit der entsprechenden VLAN-Gruppe kommunizieren können.

2.3 Echtzeitsysteme

Echtzeitsysteme sind Computersysteme, die in einem präzisen Zeitfenster auf Ereignisse in der Umgebung reagieren müssen [13]. Ein typisches Echtzeitsystem besteht aus

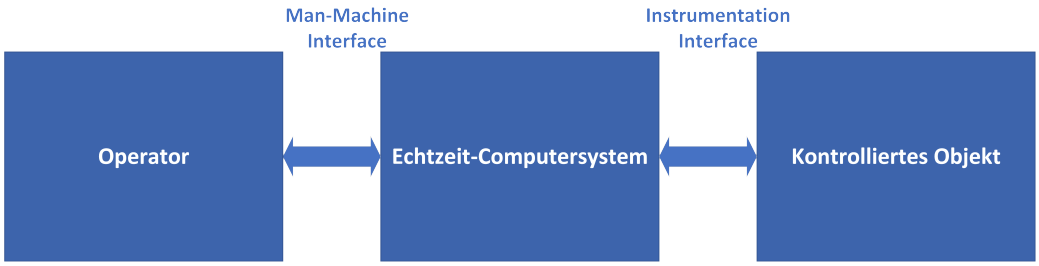

Abbildung 2.5: Echtzeitsystem [31]

drei Komponenten: einem *Operator*, einem *Echtzeit-Computersystem* und einem *kontrollierten Objekt*. Der Operator ist über das Man-Machine Interface mit dem Echtzeit-Computersystem verbunden und das Echtzeit-Computersystem über das Instrumentation Interface mit dem kontrollierten Objekt. Sieht man das Fahrzeug als ein kontrolliertes Objekt, ist der Fahrer der Operator und das Kollektiv aus ECUs das Echtzeit-Computersystem. Wenn zeitabhängige Ereignisse durch das kontrollierte Objekt oder den Operator entstehen, muss ein Echtzeit-Computersystem abhängig vom Ereignis ein Ergebnis zu einem bestimmten Zeitpunkt zurückgeben. Dieser Zeitpunkt wird Deadline genannt [31]. Deadlines können in unterschiedliche Klassen eingeordnet werden, die abhängig von der Verwertbarkeit des Ergebnisses nach Verpassen der Deadline sind. Wenn ein Ergebnis nach einer Deadline eintrifft und immer noch verwertbar ist, wird diese Deadline als weiche Deadline bezeichnet. Falls das Ergebnis nach der Deadline nicht mehr verwertbar ist, handelt es sich um eine feste Deadline. Entstehen nach dem Verpassen einer festen Deadline fatale Folgen für System und Umgebung, wird diese harte Deadline genannt. Echtzeitsysteme mit mindestens einer harten Deadline werden als sicherheitskritisches Echtzeitsysteme bezeichnet. Autos sind übliche Repräsentanten für sicherheitskritische Echtzeitsysteme. Ein Beispiel für eine harte Deadline in einem Fahrzeug wäre das Bremssignal im Auto, das über die Bussysteme im Auto weitergeleitet wird. Sollte das Bremssignal nicht rechtzeitig ankommen, würden das Fahrzeug und die Insassen mit hoher Wahrscheinlichkeit physischen Schaden erleiden. Um dies zu vermeiden, muss ein Echtzeitsystem Mechanismen bzw. Zustände zur Eindämmung von Schäden besitzen. Diese treten ein, wenn eine Deadline nicht eingehalten werden konnte. Als Safe-State wird ein Zustand bezeichnet, der den Schaden auf das System minimiert und in dem

keine weiteren Schäden auftreten können. Besitzt ein Echtzeitsystem für jeden Fehlerfall einen Safe State, wird dieses Fail-Safe genannt. Falls für einen bestimmten Zustand kein Safe-State existiert, müssen Grundfunktionen des Systems gegeben sein. So ein Zustand mit minimalem Funktionsumfang wird Fail-Operational genannt.

2.4 Kommunikationsarchitekturen im Fahrzeug

Vor den 1990ern wurde ein Großteil der Fahrzeugsignale über Punkt-zu-Punkt-Verbindungen von ECUs transportiert [46]. In modernen Fahrzeugen dagegen werden eine Reihe von unterschiedlichen Technologien genutzt, um die interne Kommunikation eines Fahrzeuges zu ermöglichen. Zu den in heutigen Fahrzeugen am meisten genutzten Kommunikationstechnologien gehören unter anderem CAN [25], Local Interconnect Network (LIN) [27], Media Oriented Systems Transport (MOST) [14], FlexRay [26] und Automotive Ethernet. Die Nutzung dieser Kommunikationstechnologien unterscheidet sich je nach Anwendung. Der Tabelle 2.2 können die einzelnen Anwendungsgebiete der unterschiedlichen Kommunikationstechnologien und Übertragungsraten entnommen werden. LIN wird für einfache Schaltsignale in der Karosserieelektronik genutzt, beispielsweise für das Ansteuern der Sitzheizung. Low Speed CAN hat ein ähnliches Anwendungsgebiet wie LIN, dafür wird High Speed CAN aber zum Beispiel für den Antriebsstrang im Auto genutzt. FlexRay wird ausschließlich für Signale mit harten Echtzeitanforderungen verwendet wie bei Airbag-Systemen oder bestimmten Fahrerassistenzsystemen. MOST ist für Multimedia-Anwendungen konzipiert, dies schließt das Navigationssystem, Autoradios oder integrierte Telefone ein. Für diese Anwendungen wird im Vergleich zu den anderen Bussystemen eine höhere Übertragungsrate gebraucht. Dementsprechend ist MOST, die Kommunikationstechnologie mit der höchsten Übertragungsrate, abgesehen von Automotive Ethernet.

Kommunikationstechnologie	Anwendung	Datenübertragungsrate
LIN	Karosserieelektronik	25 Kbit/s
CAN (Low Speed)	Karosserieelektronik	25-125 Kbit/s
CAN (High Speed)	Antriebsstrang Fahrwerk Diagnose	125-1000 Kbit/s
FlexRay	Fahrerassistenzsysteme Airbag-Systeme X-by-Wire-Kontrollsysteme	bis zu 1 Mbit/s
MOST	Multimedia und Infotainment	bis zu 150 Mbit/s
Automotive Ethernet	X-by-Wire-Kontrollsysteme Backbone Netz Multimedia	bis zu 10 Gbit/s

Tabelle 2.2: Kommunikationstechnologien des Autos im Vergleich [67, 23]

2.4.1 Automotive Ethernet

Die Automobilindustrie hat eine Reihe an Lösungen erstellt, die nahezu ausschließlich für den Gebrauch in Fahrzeugen angedacht sind und in deren Anwendung grundsätzlich Signale übertragen. Die unterschiedlichen Architekturen führen mit jeder neuen Kommunikationstechnologie zu Kompatibilitätsproblemen. Eine Vielzahl verschiedener Komponenten erzeugt hohe Kosten, die mit einer einheitlichen Kommunikationsarchitektur verhindert werden könnten. Hinsichtlich dieser Probleme und dem ansteigenden Bandbreitenbedarfes wird Ethernet in Zukunft als Kommunikationstechnologie in modernen Fahrzeugen genutzt werden. Ethernet als weitverbreitete Kommunikationstechnologie besitzt die Vorteile, immer weiterentwickelt zu werden und die Möglichkeit von den Internetprotokollen IP, TCP, UDP etc. Gebrauch zu machen. Das vereinfacht die Einbindung üblicher Alltagsgeräte mit Internetverbindung. Darüber hinaus ist Ethernet mit bis zu 10 Gbit/s das Bussystem mit der höchsten Übertragungsrate in einem Fahrzeug. Die Bayerische Motoren Werke AG (BMW) hat bereits 2004 nach einer Lösung für die Übertragung von Systemsoftware auf ECUs geforscht [61]. Mit dem üblichen CAN-BUS-Interface dauerte eine Übertragung von über 1 Gigabyte Systemsoftware zu der Zeit ca. 16 Stunden. Ethernet wurde deshalb schon im Jahr 2008 als 100BASE-TX-Ethernet [22] mit einer Übertragungsrate von 100 Mbit/s in Serienfahrzeugen von BMW genutzt. Ethernet hat trotzdem erst später an weitläufiger Relevanz gewonnen. Das lag hauptsächlich daran, dass Ethernet ohne weitere Konfiguration keine Echtzeitanforderungen einhält und dass

Kosten in der Verkabelung befürchtet wurden. Eine Sammlung von Standards und Lösungsansätzen hat sich über die Jahre entwickelt, um Ethernet an die Echtzeitanforderungen im Automobilbereich anzupassen. Automotive Ethernet hat sich als Bezeichnung für diese Sammlung durchgesetzt. Beispiele dieser Echtzeit-Ethernet-Lösungen wären: Time Sensitive Networking (TSN) [19] (siehe Abschnitt 3.5) und Time Sensitive Software Defined Networking (TSSDN) [47]. TSSDN ist ein Konzept, das versucht die globale Sicht des SDN-Controllers zu nutzen. So soll der durch die TSN-Substandards ermöglichte Echtzeit-Datenverkehr gesteuert werden. Der SDN-Controller ist ein zentraler Bestandteil von Lösungsansätzen, wie durch TSSDN bewiesen wird. Eine wie in dieser Arbeit durchgeführte Evaluation gewinnt dadurch noch mehr an Relevanz.

3 Anforderungsanalyse

In diesem Kapitel werden anhand einer Anforderungsanalyse Kriterien zur Auswahl von SDN-Controllern gefunden. Es wird davon ausgegangen, dass der SDN-Controller auf einem dedizierten Rechner im Auto installiert wird, der ans Fahrzeugbordnetzwerk angeschlossen ist. Die erarbeiteten Anforderungen sind in einer Liste zusammengefasst. Jede Anforderung besitzt eine Priorität. Diese gibt an, wie ausschlaggebend die Anforderung für die Auswahl eines Controllers ist. Eine Anforderung kann niedrig, mittel oder hoch priorisiert sein. Eine niedrig priorisierte Anforderung bringt Vorteile mit sich, die für die Hauptfunktionalität eines SDN-Controllers im Auto weniger relevant sind und daher vernachlässigt werden können. Mittel priorisierte Anforderungen sollten nur unter bestimmten Umständen vernachlässigt werden, wenn zum Beispiel die Anforderung indirekt durch eine andere Eigenschaft gedeckt wird. Anforderungen mit hoher Priorität sind ausschlaggebend für die Auswahl eines Controllers und dürfen unter keinen Umständen fehlen, wenn ein SDN-Controller als Komponente in einem Fahrzeug eingebaut werden soll.

3.1 Echtzeitanforderungen

Echtzeitanforderung beschreiben alle Anforderungen, die für ein Echtzeitsystem gelten. In dieser Arbeit liegt der Fokus auf sicherheitskritischen Echtzeitsystemen, da Autos als sicherheitskritische Echtzeitsysteme gelten. In der Tabelle 3.1 sind die erarbeiteten Echtzeitanforderungen an SDN-Controller ersichtlich.

ID	Anforderung	Priorität	Beschreibung
A1	Zeitbasierte Updates	hoch	Konfiguration an Weiterleitungsgeräten mit Zeitstempel
A2	Quality of Service Applikationen	hoch	Applikationen die Garantien für Netzwerkdienste sicherstellen

Tabelle 3.1: Echtzeitanforderungen an SDN-Controller

Es kann zu Inkonsistenzen im Netzverkehr kommen, sollten mehrere Weiterleitungsgeräte durch den Controller gleichzeitig manipuliert werden. Dies geschieht beispielsweise durch eine vollständige oder teilweise Änderung am Netzwerkverkehr. Erhält ein Weiterleitungsgerät eine Änderungsnachricht früher als andere Weiterleitungsgeräte, wird dies in einigen Fällen zu einem temporären Verbindungsverlust einiger Netzwerkteilnehmer führen. Ein temporärer Verbindungsverlust kann im schlimmsten Fall dazu führen, dass eine harte Deadline nicht eingehalten wird. Um ein solches Risiko zu vermeiden, muss ein SDN-Controller die Funktion zum Senden eines *zeitbasierten Updates* besitzen. Mizhrah et al. haben bezüglich SDN und *zeitbasierter Updates* mehrere Publikationen veröffentlicht, was die Priorität dieser Anforderung bestärkt [45, 44, 42, 43]. Eine solche Funktion soll dem SDN-Controller erlauben, bei Änderungen einen Zeitstempel oder etwas Ähnliches an die Weiterleitungsgeräte zu übergeben. Die Weiterleitungsgeräte führen die gewollte Änderung dann zum gegebenen Zeitpunkt durch. Die vom SDN-Controller gegebenen Zeitpunkte sind somit wiederum neue harte Deadlines, an die sich Weiterleitungsgeräte halten müssen. Diese Funktion verhindert nicht nur Inkonsistenz, sondern ermöglicht auch bei Bedarf zeitliche Ablaufpläne für Änderungen im Netzwerk zu erstellen und durchzusetzen. Um diese Funktionen gewährleisten zu können, müssen mindestens die Weiterleitungsgeräte und der SDN-Controller über eine synchrone Zeitbasis verfügen.

Da Echtzeitsysteme im Grunde genommen auf zeitliche Garantien einzelner Komponenten angewiesen sind, sollte es einen Mechanismus zur Durchsetzung dieser Garantien im Netzwerk geben. Der SDN-Controller kann diese Garantien durchsetzen, indem er eine *Quality of Service Applikation* besitzt [12, 1, 18]. Quality of Service (QoS) ist die Sammlung von Aktivitäten und Maßnahmen, die zur Verbesserung und Kontrolle der Netzwerkressourcen genutzt wird. Damit soll die Qualität von Netzwerkdiensten gewährleistet werden. Wenn ein Fahrzeug ohne QoS nach dem Best-Effort-Prinzip alle Netzwerkflüsse gleichbehandeln würde, könnte eine Videoübertragung wie bei einer Rückfahrkamera zur Beeinträchtigung bestimmter Funktionen sorgen. Laufzeitverzögerungen sind beispielsweise bei sicherheitskritischen Systemen mit harten Deadlines nicht zu vernachlässigen. Eine Vielzahl an älteren und aktuellen Fahrzeugen besitzen mehrere separate Bussysteme, keine Internetanbindung und einen statischen Netzwerkverkehr. Mit diesen Eigenschaften konnte QoS im Fahrzeug auf andere Weise sichergestellt werden. In Zukunft soll vollständig auf Ethernet und nicht mehr auf separate Bussysteme gesetzt werden. Der Netzwerkverkehr der einzelnen Bussysteme wird somit zusammengelegt. Bei entsprechender Verkabelung könnten vorher voneinander unabhängige Netzwerkflüsse, sich mit

Ethernet in Zukunft gegenseitig beeinträchtigen. Es müssen dementsprechend alle Anforderungen für alle Funktionen im Fahrzeug zu jeder Zeit durch QoS garantiert werden. Eine QoS-Applikation ist aus diesem Grund als hoch priorisiert.

3.2 Safety-Anforderungen

Safety-Anforderung beschreiben alle Anforderungen, die für die physikalische Sicherheit der Passagiere des Fahrzeuges, des Systems und der unmittelbaren Umgebung des Fahrzeuges zuständig sind, siehe Abschnitt 2.3. Diese Art von Anforderungen sind ausschlaggebend für die Zulassung eines Autos. Die erarbeiteten Safety-Anforderungen für einen SDN-Controller in einem Fahrzeug sind in Tabelle 3.2 zu sehen.

ID	Anforderung	Priorität	Beschreibung
B1	Transaktionen	hoch	Controller Aktionen sind atomar
B2	Rückfallstrategie	hoch	Sicherer Zustand im Falle eines Ausfalls
B3	Redundanz	mittel	Ausfallsicherheit durch mehrere Controller
B3.1	Intercontroller Kommunikation	hoch	Kommunikation zwischen den Controller Replica
B3.2	Geteilte Netzwerkinformationsbasis	hoch	Controller teilen die gleichen Netzwerkinformationen
B3.3	Ausfallsicherungsmechanismus	hoch	Algorithmus um einem Controller Replica zu ersetzten

Tabelle 3.2: Safety-Anforderungen für SDN-Controller

Cui et al. beschreiben [15], dass in einem von einem SDN-Controller kontrollierten Netzwerk meistens mehrere Netzwerkflüsse auf einmal geändert werden. Eine Netzwerkfunktion besteht aus einem Änderungssatz. Dieser besteht aus mehreren Flüssen, die im Kollektiv die Netzwerkfunktion realisieren. Jeder einzelne Fluss dieses aus Änderungen bestehenden Satzes trägt dazu bei, dass die Netzwerkfunktion korrekt ausgeführt wird. Das bedeutet auch, dass fehlende Flüsse die Korrektheit der Netzwerkfunktion beeinträchtigen können. Falsch gesetzte Flüsse hingegen können wichtigen Datenverkehr unterbrechen und eine Gefahr für die Insassen des Fahrzeuges bedeuten. Nach Cui et al. sollte ein SDN-Controller atomare *Transaktionen* nutzen. Eine Netzwerkfunktion kann

dann nur ausgeführt werden, wenn alle vorgesehenen Flüsse eines Änderungssatzes erfolgreich etabliert wurden. Falls ein Fluss oder mehrere Flüsse aus bestimmten Gründen nicht gesetzt werden konnten, müssen alle Änderungen zurückgesetzt werden. Cui et al. geben auch ein Beispiel für die genaue Implementierung von Transaktionen in SDN. Ein anderer Ansatz ist NETCONF mit Checkpoints zu nutzen, wie in Abschnitt 2.2.2 beschrieben. Diese Vorgehensweise ermöglicht Inkonsistenz im Netzwerk zu verhindern. Damit ist diese unentbehrlich für die Safety eines SDN-Controllers im Fahrzeug.

Das Ausfallen einzelner oder mehrerer Komponenten führt in vielen Fällen zwangsläufig zu Fehlverhalten im ganzen System. Ohne Schutz vor Ausfällen besteht immer die Möglichkeit von fatalen Folgen für Nutzer, System und unmittelbare Umgebung. *Ausfallsicherheit* ist damit die wichtigste Anforderung für ein sicherheitskritisches Echtzeitsystem. Bei einem Fahrzeugbordnetzwerk könnte es passieren, dass Signale nicht rechtzeitig oder gar nicht mehr an die Endpunkte gelangen. Dies geschieht beispielsweise durch einen Ausfall des Rechners, auf dem der Controller installiert ist. Wenn der Netzwerkfluss für die Bremse unterbrochen wird, würden Bremssignale nicht mehr ankommen. Insbesondere für sicherheitskritische Prozesse wie das Bremsen muss eine Ausfallsicherheit gewährleistet sein.

Eine *Rückfallstrategie*, wie der in Abschnitt 2.3 beschriebene Fail-Operational ist deshalb höchst notwendig für Weiterleitungsgeräte im Fahrzeug, da diese möglichst unabhängig vom Controller agieren sollten [2, 18]. Ein Ausfall des Controllers darf nicht dazu führen, dass der Netzwerkverkehr unterbrochen wird. Es können hierfür Rückfallstrategien auf den Weiterleitungsgeräten installiert werden, die alle kritischen Netzwerkflüsse erhalten.

Der Ausfall eines SDN-Controllers kann im schlimmsten Fall die Unterbrechung von wichtigen Netzwerkflüssen und infolgedessen den Ausfall des ganzen Systems herbeiführen. Ein SDN-Controller ist ein Single Point of Failure, sofern nur ein physikalischer Rechner mit einem SDN-Controller im Netzwerk existiert. Durch *Redundanz* soll der SDN-Controller im Netzwerk logisch zentral dafür aber mehrfach physikalisch verteilt existieren [63, 48]. Das hat den Vorteil, dass beim Ausfall eines physikalischen Rechners es noch mindestens einen Ersatz-Controller gibt. Redundanz hat unterschiedliche Typen: kalte, warme und heiße Redundanz. Die unterschiedlichen Typen beschreiben unter anderem die Zeit, die zwischen einem Ausfall und dem Ersatz einer Instanz vergehen darf. Kalte Redundanz ist für nicht kritische Prozesse ausgelegt und lässt es zu, dass für eine ausgefallene Instanz für unbestimmte Zeit kein Ersatz existiert. Bei war-

mer Redundanz muss innerhalb eines bestimmten Zeitabschnittes eine neue Instanz, die ausgefallene ersetzen. Der Ersatz muss nicht sofort erfolgen, dafür kann aber ein nicht Einhalten des Zeitfensters zu Schäden führen. Für sicherheitskritische Systeme wird die heiße Redundanz genutzt, die bei Fehlererkennung eine direkte Ersetzung der ausgefallenen Instanz erfordert. Obwohl ein Fahrzeug ein sicherheitskritisches System darstellt, ist ein direkter Ersatz des SDN-Controllers nicht zwangsläufig nötig. Der Controller steuert hauptsächlich das Verhalten der Weiterleitungsgeräte. Sofern die Weiterleitungsgeräte ohne den SDN-Controller agieren können, muss es keinen sofortigen Ersatz geben. Der SDN-Controller muss spätestens kurz vor der Aktivierung der Rückfallstrategie ersetzt werden. Warme Redundanz mit der Zeit bis zur Aktivierung der Rückfallstrategie als Ablaufzeit reicht für SDN-Controller im Fahrzeug aus. Um *Redundanz* von SDN-Controllern im Fahrzeugbordnetzwerk zu nutzen, muss der Controller erst eine entsprechende Schnittstelle zur Verfügung stellen. Diese unterliegt bestimmten Anforderungen. Neben vielen weiteren Aspekten gehören Anforderungen wie *Intercontroller Kommunikation* [10] und eine *geteilte Netzwerkinformationsbasis* [11] zu jenen, die *Redundanz* realisierbar machen.

Intercontroller Kommunikation ist bei *Redundanz* unentbehrlich, da die Controller-Replica von einem Ausfall mitbekommen müssen. Würden die Controller-Replica nicht miteinander kommunizieren, könnte auf den Ausfall eines Controllers nicht reagiert werden und es müsste eine Rückfallstrategie eingeleitet werden. Mit einer Kommunikation zwischen Controller-Replicas erschließen sich aber auch weitere Optionen, wie beispielsweise die Verteilung von Verantwortung zwischen den Controllern.

Die *Intercontroller Kommunikation* zieht eine Konsequenz mit sich. Alle Controller-Replica sollten die gleichen Informationen über das Netzwerk haben. Wenn eine Controller-Replica ausfällt, muss ein anderer alle Informationen des ausgefallen Controllers besitzen. Nur so kann gewährleistet werden, dass der Netzwerkfluss im Fahrzeug nicht beeinträchtigt wird [63].

Für einen Cluster aus SDN-Controller gibt es für den Aufbau mehrere Strategien. Beispiele für Cluster-Strategien wären Controller-Cluster, die nur einen aktiven Controller haben und dafür ein oder mehrere passive Controller-Replica. Es kann aber auch mehrere aktive Controller geben, die einzelne unabhängige Netzwerkregionen steuern. Bei einer Cluster-Strategie mit nur einem aktiven Controller, muss es bei Ausfall eines Controllers einen *Ausfallsicherungsmechanismus* für die Auswahl eines neuen primären Controllers geben [11]. Bei mehreren müsste es dementsprechend eine Strategie geben, wie

eine ausgefallene Controller-Replica ersetzt wird. Da SDN im Fahrzeug nicht ohne eine Rückfallstrategie für die Weiterleitungsgeräte genutzt werden sollte, könnte theoretisch zur Kostensenkung auf *Redundanz* verzichtet. Es sollte hierbei beachtet werden, dass sich bei einem Ausfall des SDN-Controllers die Rückfallstrategie der Weiterleitungsgeräte einschalten muss und das Auto in einen Ausnahmezustand gebracht wird. Mit *Redundanz* könnte der normale Betriebszustand erhalten bleiben, sofern der Controller rechtzeitig ersetzt wird. Aus diesem Grund ist *Redundanz* mittel priorisiert. Die Unteranforderungen *Intercontroller Kommunikation*, *geteilte Netzwerkinformationsbasis* und der *Ausfallsicherungsmechanismus* sind hoch priorisiert, falls *Redundanz* für den SDN-Controller genutzt wird. Diese Unteranforderungen würden ohne *Redundanz* wegfallen und keine Priorität bekommen.

3.3 Security-Anforderungen

Security-Anforderungen sind Anforderungen, die die Informationssicherheit eines Computersystems behandeln. Dazu gehören Mechanismen, um die Vertraulichkeit und den Zugriff auf Informationen auf einem Computersystem sicherzustellen. Die Security-Anforderungen sind in der Tabelle 3.3 aufgelistet.

ID	Anforderung	Priorität	Beschreibung
C1	Controller-Applikation-Verifikation	hoch	Befugnisverifikation von Controller-Applikationen
C2	Controller-Switch-Verifikation	hoch	Zuordnungsverifikation zwischen SDN-Controller und Weiterleitungsgeräten
C3	Fluss-Applikation-Zuordnung	mittel	Applikationen haben von ihnen angelegte Flüsse als Eigentum
C4	Telemetrie	hoch	Speichern von Statistiken

Tabelle 3.3: Security-Anforderungen für SDN-Controller

Der Schutz der Insassen und des Fahrzeuges kann nur gewährleistet werden, wenn alle Komponenten gegen das Eindringen von Unbefugten abgesichert sind. Falls Unbefugte die Möglichkeit zur Übernahme von Komponenten eines Fahrzeuges oder zur Einschränkung der Funktionalität bekommen sollten, kann das fatale Folgen für Fahrzeug, Insassen und Umgebung nach sich ziehen. Die Security für ein modernes Fahrzeug ist stark mit der

Safety verknüpft, was sich in der Priorisierung der Security Anforderungen entsprechend widerspiegelt [32, 29].

Eine Angriffsfläche bieten die über das Northbound-API verbundenen SDN-Controller-Applikationen. Applikationen können bei Nutzung von SDN im Auto den Weiterleitungsgeräten Flüsse in die Fluss-Tabellen einfügen, die Unbefugten Zugriff von außen geben. Es könnten aber auch Fluss-Tabellen-Einträge entfernt werden, womit kritischer Datenverkehr im Fahrzeug unterbrochen oder alle Netzwerkflüsse im Fahrzeugbordnetzwerk unterbunden werden können. Um die Kontrolllogik des Controllers vor schädlichen Applikationen zu beschützen, muss es *Controller-Applikation-Verifikation* geben. Diese wird auf dem Controller installiert und schützt das Northbound-API vor schädlichen Applikationen [48, 15]. Applikationen dürfen nicht ohne Verifikation des Controllers Flüsse setzen oder entfernen. Eine mögliche Umsetzung wäre ein Authentifizierungsserver, auf dem Applikationen durch eine befugte Person registriert werden, wie von Oktian et al. beschrieben [49]. Eine erfolgreiche Registrierung erzeugt einen API-Schlüssel und einen API-Sicherheitsschlüssel, die bei jeder API-Anfrage mitgegeben werden müssen.

Auf einem Controller können mehrere Applikationen existieren. Dies bedeutet, dass Fluss-Tabellen von Weiterleitungsgeräten von mehr als einer Applikation entfernt werden können. Wenn mehrere Applikationen den Netzwerkfluss beeinflussen können, könnten zwei Applikationen den gleichen Netzwerkpfad manipulieren wollen. Deshalb können Konflikte entstehen. Wichtiger ist aber die Tatsache, dass eine Applikation die Möglichkeit zur Modifizierung von Flüssen anderer Applikationen besitzt. Um dies zu verhindern, muss eine *Fluss-Applikations-Zuordnung* vorhanden sein. Applikationen haben somit nur die Möglichkeit Flüsse zu entfernen, die diese selbst hinzugefügt haben. Die entsprechende Applikation sollte nicht selbst kompromittiert sein. Sollte die Applikation dennoch kompromittiert werden, wäre mit einer Fluss-Applikations-Zuordnung nur das Hinzufügen von Flüssen möglich. Eine schadhafte Applikation verliert damit die Möglichkeit, kritische Netzwerkflüsse zu unterbrechen. Mit einer *Controller-Applikation-Verifikation* sollte es aber nicht möglich sein eine schädliche Applikation in den SDN-Controller einzubinden. Aus diesem Grund ist *Fluss-Applikations-Zuordnung* nur als mittel priorisiert.

Falls Angreifer einen fremden Controller in das Fahrzeugbordnetzwerk einbinden und sich dieser mit den Switches verbindet, kann keine Verifikation von Applikationen das Hinzufügen und Entfernen von falschen Flüsse verhindern. Beim Einbinden von fremden Weiterleitungsgeräten sind Angreifer zum Beispiel in der Lage Denial of Service (DoS) Attacken auf das Netzwerk zu fahren. Ein fremdes Weiterleitungsgerät könnte über eine

Großzahl von Media-Access-Control (MAC)-Adressen agieren und somit Weiterleitungsgeräte und/oder den Controller beeinträchtigen. Um die Angriffsfläche zwischen Controller und Switches zu schließen, muss es eine Authentifizierung zwischen Controller und Switches geben [48].
In einem Fahrzeug ist der Netzwerkverkehr vorhersehbar und normales Verhalten eines Weiterleitungsgerätes oder eines Controllers kann bestimmt werden. Sollten ungewöhnlich viele Flüsse durch den Controller hinzugefügt werden oder leitet ein Weiterleitungsgerät nicht mehr über den geplanten Pfad weiter, kann wie in [34] beschrieben, ein Weiterleitungsgeräte oder der SDN-Controller in Quarantäne gebracht werden und somit vom weiteren Netzwerkfluss ausgeschlossen werden.

Zwingend erforderlich ist *Telemetrie* bzw. die Aufnahme von Statistiken [48]. Statistiken sollen das Verhalten von Komponenten wiedergeben. Bei Einbindung in das Fahrzeugbordnetzwerk kann ein durchschnittliches Verhalten von allen Komponenten und vor allen Dingen vom SDN-Controller gesetzt werden. Die Statistiken geben damit auffälliges Verhalten oder sogar Fehlverhalten wieder. Sollte ein Controller durch einen Angreifer kompromittiert werden, ist die automatische Reaktion auf Statistiken möglich. Statistiken sind ein wichtiger Bestandteil bei der Wartung, Analyse und Weiterentwicklung. Mit einem Einblick in die Statistiken können Softwareaktualisierungen für die Sicherheit und Robustheit eines SDN-Controllers entwickelt werden. Die Anzahl von über den Controller-Port verschickten eingehenden und ausgehenden Paketen, sind Statistiken, die viel über das Verhalten vom Controller und von mit dem Controller interagierenden Switches aussagen.

3.4 Anforderungen an die Softwarequalität

Die Anforderungen an die Softwarequalität beschreiben nichtfunktionale Anforderungen, die sich mit der allgemeinen Qualität von Software beschäftigen. Die Anforderungen an die Softwarequalität sind in der Tabelle 3.4 ersichtlich.

ID	Anforderung	Priorität	Beschreibung
D1	Zertifizierung	hoch	eine Zertifizierung nach dem ISO 26262 Standard
D2	Aktuelle Software	hoch	Controller wird aktiv weiterentwickelt und arbeitet auf aktuellen Softwarekomponenten
D3	Gute Systemdokumentation	niedrig	umfangreiche und einfach geschriebene Beschreibung von Architektur und Nutzung des Controllers

Tabelle 3.4: Anforderungen an die Softwarequalität für SDN-Controller

Automobilhersteller können in den meisten Fällen zugelieferte Software nur annehmen, wenn diese eine bestimmte *Zertifizierung* besitzen. Ein SDN-Controller, der abhängig von der Implementierung das ganze Fahrzeugbordnetzwerk steuert, muss den Standard für funktionale Sicherheit ISO 26262 [28] einhalten und auch unter Einhaltung dieses Standards entwickelt werden. ISO 26262 gibt nicht nur Vorgaben für die Sicherheit von Software an, sondern auch für den Entwicklungsprozess und die Implementierung von Software.

Nach ISO 26262 sollte Software nur nach dem Stand der heutigen Technik entwickelt werden. Somit ist die Anforderung unter keinen Umständen zu vernachlässigen, dass der SDN-Controller auf aktuelle Software setzt und aktiv weiterentwickelt wird. Ein Großteil der relevanten SDN-Controller sind Open-Source-Projekte. Diese stehen im Internet für jeden zur freien Verfügung, der an einem Projekt mitwirken möchte. Wenn keine Entwickler mehr aktiv am Controller arbeiten, verliert der SDN-Controller an Relevanz und die Software wird auch nicht mehr instand gehalten und veraltet. Veraltete Software zu aktualisieren kann sich als umständlich erweisen, besonders wenn die Zeit der letzten Wartung mehrere Jahre her ist. Gerade in Anbetracht des Aufwandes der Softwareaktualisierung ist ein aktueller SDN-Controller die sinnvollste Wahl.

Eine *gute Systemdokumentation* kann die Nutzung des SDN-Controllers erleichtern und somit den Einbindungsprozess in das Fahrzeug beschleunigen. Automobilhersteller und Zulieferer können sich SDN-Controller zu eigen machen, sofern der Quellcode Open-Source ist. Die Systemdokumentation sollte mit jeder neuen Controller-Version aktualisiert werden. In dieser sollte es ausführliche Einführungen zu den einzelnen Funktionen und Basis-Applikationen des Controllers geben. Wenn Hersteller und Zulieferer den SDN-Controller zur Anpassung an deren Anforderungen verändern wollen, hilft die System-

dokumentation bei der Handhabung des SDN-Controllers. Die Systemdokumentation ist aber für den eigentlichen Gebrauch und für die Hauptfunktionalität des SDN-Controllers unerheblich, damit kann diese für den SDN-Controller vernachlässigt werden.

3.5 Sonstige funktionale Anforderungen

Die sonstigen funktionalen Anforderungen beschreiben funktionale Anforderungen, die keiner bestehenden Kategorie zugeordnet werden können. Diese sind in der Tabelle 3.5 aufgelistet.

ID	Anforderung	Priorität	Beschreibung
E1	Kompatibilität mit bestehenden Fahrzeug-kommunikationsprotokollen	hoch	Empfang und Verarbeitung von Datenpaketen bestehender Kommunikationsprotokolle in Fahrzeugen
E2	Skalierbarkeit	niedrig	Anpassung an die Größe des Netzwerkes ohne Leistungsverlust
E3	Unterstützung von NETCONF/YANG	hoch	Umfangreiche und einfach geschriebene Beschreibung von Architektur und Nutzung des Controllers

Tabelle 3.5: Sonstige funktionale Anforderungen für SDN-Controller

Um einen Einsatz von SDN-Controllern in Fahrzeugen zu erlauben, müssen diese mit den heutigen Fahrzeugkommunikationsprotokollen kompatibel sein, wie Scalable Service-Oriented Middleware over IP (SOME/IP) [5] oder Diagnostics over Internet Protocol (DoIP) [6]. Die Kommunikationsprotokolle der anderen Bussysteme im Fahrzeug dürfen für den Übergang auf Ethernet nicht vernachlässigt werden. Diese könnten für den Übergang in einer getunnelten Form auftreten, indem aus anderen Bussystemen ankommende Pakete in Ethernet-Pakete umgewandelt werden. Der Netzwerkverkehr der Fahrzeugkommunikationsprotokolle wird einen Bestandteil der zu verarbeitenden Pakete für einen SDN-Controller ausmachen.

Hinsichtlich des statischen Aufbaus eines Fahrzeugbordnetzwerkes ist *Skalierbarkeit*, wie diese üblicherweise von SDN-Controllern in Rechenzentren erwartet wird [65] eine nicht so relevante Funktion und es ist deshalb niedrig priorisiert. Die Größe eines Fahrzeugbordnetzwerkes ändert sich normalerweise eher selten. Ein SDN-Controller muss nicht unbedingt ein Wachstum oder eine Abnahme von mehreren Weiterleitungsgeräten im

Fahrzeugbordnetzwerk unterstützen. Selbst bei einer Aufrüstung oder Umrüstung eines Fahrzeuges wird es üblicherweise nicht zu einem Wachstum von mehreren Netzwerkteilnehmern im Fahrzeugbordnetzwerk kommen.

Es gibt einige Echtzeitlösungen für Ethernet (siehe Abschnitt 2.4). Eine der relevantesten Echtzeitlösungen für Ethernet ist TSN. Die IEEE TSN Gruppe definiert eine Gruppe von Substandards für echtzeitfähiges Ethernet und bietet damit für Automobilhersteller eine Möglichkeit, die fehlende Echtzeitfähigkeit von Ethernet zu lösen. So kann eine vollständige ethernetbasierte Kommunikationsarchitektur in Autos erstellt werden. Unter bestehenden Substandards von TSN existieren unter anderem IEEE 802.1Qcw [21] und IEEE 802.1Qcp [20]. Diese definieren alle YANG-Modelle die in Kombination mit NETCONF genutzt werden. Ein SDN-Controller muss unbedingt *NETCONF/YANG* unterstützen bzw. NETCONF als Southbound-API besitzen, um die von der TSN-Gruppe definierten YANG Modelle zu verstehen und mittels NETCONF umzusetzen.

3.6 Sonstige nichtfunktionale Anforderungen

Die sonstigen nichtfunktionalen Anforderungen beschreiben nichtfunktionale Anforderungen, die keiner bestehenden Kategorie zugeordnet werden können. Diese sind in der Tabelle 3.5 aufgelistet.

ID	Anforderung	Priorität	Beschreibung
F1	Ressourcenschonend bzgl. der Netzwerklast	hoch	Netzlast des Controllers ist so gering wie möglich
F2	Kosteneffizient	niedrig	Kosten für den Einbau des Controllers sind gering

Tabelle 3.6: Sonstige nichtfunktionale Anforderungen für SDN-Controller

SDN-Controller sollten die Netzwerklast nicht unnötig erhöhen [12]. Je mehr Datenpakete durch das Fahrzeugbordnetzwerk befördert werden müssen, desto ausgelasteter ist das Netzwerk. Die Netzwerkauslastung kann Auswirkung auf die allgemeine Leistung im Fahrzeug ausüben. Verschickt ein SDN-Controller übermäßig viele Datenpakete, kann es unter anderem zu Einbußen in der Latenz und Bandbreitenverteilung kommen. Für Fahrzeuge wäre so eine Leistungseinbuße im Netzwerk, aufgrund der harten Deadlines im Fahrzeugbordnetzwerk nicht akzeptabel.

Der Einbau eines SDN-Controllers sollte kosteneffizient sein. *Kosteneffizienz* bedeutet, dass der Einbau nicht zu hohe Hardwarekosten mit sich bringt oder im Falle eines proprietären SDN-Controllers die Lizenzkosten nicht zu hoch für den jeweiligen Automobilhersteller oder Zulieferer sind. Bei den Kosten für die genutzte Hardware muss der Einbau in Serienfahrzeuge in Betracht gezogen werden. Wenn der SDN-Controller in Serienfahrzeugen in Betrieb genommen werden soll, muss eine hohe Zahl an physischen Rechnern eingekauft werden. Eine zu hohe Systemanforderung kann dazu führen, dass aus rein wirtschaftlicher Sicht der Einbau eines SDN-Controllers nicht lohnenswert ist. Die Redundanz eines SDN-Controllers macht die Systemanforderung zu einem noch wichtigeren Aspekt, da in diesem Fall mehrere physische Rechner eingebaut werden müssen und der Preis sich vervielfacht. Die Kosten eines SDN-Controllers können ein Hindernis für Automobilhersteller sein, für das Konzept ist diese Eigenschaft aber unerheblich.

3.7 Auswahl der SDN-Controller

SDN-Controller werden im Normalfall in Rechenzentren genutzt, die im Vergleich zu Fahrzeugen viel größere Netzwerke mit einer Vielzahl von Weiterleitungsgeräten besitzen. Typische Rechenzentren besitzen keine harten Deadlines. Daher gibt es bei einzelnen Ausfällen von einem SDN-Controller, keine fatalen Folgen, die in einem Fahrzeug normalerweise berücksichtigt werden müssen. Durch Verbindung mit dem Internet betreffen die Security-Anforderungen auch SDN-Controller in Rechenzentren. Während im Fahrzeug die Security indirekt dem Schutz von Insassen, System und Umwelt gewidmet ist, geht es in Rechenzentren primär um Datenintegrität. Der Netzwerkfluss ist in Rechenzentren dynamischer. Abhängig vom Rechenzentrum muss ein SDN-Controller in einem Rechenzentrum mit einem Zuwachs von einer höheren Anzahl von Weiterleitungsgeräten mitskalieren können. Mehr Weiterleitungsgeräte bedeuten für den SDN-Controller auch höhere Systemanforderungen, um die hohe Anzahl an Geräten im Netzwerk zu verwalten. Höhere Systemanforderungen führen dazu, dass die Kosten für einen Rechner mit dem installierten SDN-Controller entsprechend hoch sind. Im Gegensatz zu Fahrzeugen werden Rechenzentren aber nicht Massen produziert, weswegen die *Kosteneffizienz* in diesen keine zentrale Rolle spielt.

SDN-Controller sind im Konzept nicht für ein Fahrzeugbordnetzwerk ausgelegt. Ein wichtiger Teil der in diesem Kapitel ermittelten Anforderungen trifft aus diesem Grund auf eine Vielzahl an SDN-Controller die auf dem Markt oder im Internet erhältlich sind nicht

zu. Es gibt eine große Zahl an SDN-Controllern mit unterschiedlichen Zielen und Konzepten. Daher wurden Anforderungen gewählt, die eine hohe Anzahl an SDN-Controllern ausfiltern und die in Abhängigkeit zu der genutzten Hardware testbar sind. Es wurden über dreißig SDN-Controller in Betracht gezogen. Mittels der Anforderung *aktuelle Software* konnte die Anzahl der Controller auf etwa die Hälfte reduziert werden. Die zuletzt veröffentlichte Version des SDN-Controller darf nicht älter als ein Jahr sein (Stand April 2020). Von diesen SDN-Controllern besitzen sechs Controller Systemanforderungen, die die Testumgebung erfüllen können. Die SDN-Controller sind Open Network Operating System (ONOS) [52], OpenDayLight [55], Lumina SDN-Controller [38], Ryu [56], Runos [4] und Faucet [62]. Um den Fokus auf die Anwendung im Automotive-Bereich zu bringen, wurde zu Unterstützung die Anforderung *NETCONF/YANG* herangezogen. Runos und Faucet wurden mit dieser Anforderungen aus der Auswahl der zu testenden SDN-Controller entfernt. Die abschließende Auswahl der Controller von Lumina SDN-Controller, ONOS, OpenDayLight und Ryu wurden einer Performanzanalyse unterzogen.

Für die Auswahl der SDN-Controller wurde in Abbildung 3.7 eine Matrix erstellt. Diese Matrix soll einen Überblick über die Anforderungen geben, die die SDN-Controller im Werkszustand erfüllen oder nicht erfüllen. Zeitbasierte Updates, Transaktionen, Controller-Applikation-Verifikation und Zertifizierung sind Anforderungen, die kein SDN-Controller in dieser Auswahl erfüllt. Jede dieser Anforderungen ist hoch priorisiert und somit kritisch, um den Einbau in ein Fahrzeug zu gewährleisten. Nur auf Basis der Anforderungsanalyse kann der sichere Einbau von SDN-Controllern in ein Fahrzeug im heutigen Zustand nicht ohne weitere Konfigurationen gewährleistet werden.

ID	Anforderung	ONOS	ODL	Ryu	Lumina
A1	Zeitbasierte Updates	x	x	x	x
A2	QoS Applikation	x	x	x	x
B1	Transaktionen	x	x	x	x
B2	Rückfallstrategie	x	x	x	x
B3	Redundanz	✓	✓	x	✓
B3.1	Intercontroller Kommunikation	✓	✓	x	✓
B3.2	Geteilte Netzwerkinformationsbasis	✓	✓	x	✓
B3.3	Ausfallsicherungsmechanismus	✓	✓	x	✓
C1	Controller-Applikation-Verifikation	x	x	x	x
C2	Controller-Switch-Verifikation	✓	✓	✓	✓
C3	Fluss-Applikation-Zuordnung	x	x	x	x
C4	Telemetrie	✓	✓	✓	✓
D1	Zertifizierung	x	x	x	x
D2	Aktuelle Software	✓	✓	✓	✓
D3	Gute Systemdokumentation	✓	x	✓	✓
E1	Kompatibilität mit bestehenden Fahrzeugkommunikationsprotokollen	✓	✓	✓	✓
E2	Skalierbarkeit	✓	✓	✓	✓
E3	Unterstützung von NETCONF/YANG	✓	✓	✓	✓
F1	Ressourcenschonend bzgl. der Netzwerklast	✓	✓	✓	✓
F2	Kosteneffizient	✓	✓	✓	k.A.

Tabelle 3.7: Anforderungsmatrix für die ausgewählten SDN-Controller

4 Performanzanalyse

Dieses Kapitel befasst sich mit der zweiten Stufe der Evaluation von SDN-Controllern. Die SDN-Controller werden einer Performanzanalyse unterzogen. Diese gibt anhand von Metriken zu erkennen, ob die heutigen SDN-Controller die nötige Leistung für den Einbau in ein Fahrzeug erfüllen. In diesem Kapitel werden die Metriken für die Analyse der Leistung der Controller vorgestellt. Daraufhin wird die Testumgebung vorgestellt und erklärt, wie die Tests mittels der Testumgebung aufgebaut wurden. Für jede Metrik wurde ein entsprechender Test erstellt, aus dem die relevanten Werte ausgelesen werden können. Abschließend werden die Werte der Tests analysiert und bewertet.

4.1 Metriken

In diesem Abschnitt werden die einzelnen Metriken vorgestellt, auf denen die Performanzanalyse aufbaut.

Die in dieser Arbeit genutzten Metriken stammen hauptsächlich von der IETF, die für SDN-Controller eine Liste von empfohlenen Methodiken und Metriken im RFC 8456 [7] vorgegeben hat. Die Auswahl der Metriken kommt durch eine Zusammensetzung des RFC 8456, den erarbeiteten Anforderungen für Fahrzeuge sowie mehreren Arbeiten, die bereits ähnliche Untersuchungen vollzogen haben [57, 30, 59, 39]. In allen betrachteten Arbeiten wurden die Metriken *asynchrone Nachrichten-Verarbeitungszeit* und *asynchrone Nachrichten-Verarbeitungsrate* untersucht. Dazu kommt die von Jawaharan et al. untersuchte *Topologie-Entdeckungszeit* [30], die von Tello et al. untersuchte *Erkennungszeit von Topologie-Änderungen* sowie die für diese Arbeit hinzugefügte vom Controller erzeugte *Grundlast* im Netzwerk, die *Controller-Ausfallsicherungszeit* und die *Zeit für einen Systemstart*. Diese Metriken wurden ausgewählt, weil sie besonders für den Automotive-Bereich von Relevanz sind, siehe Abschnitt 4.5.

- *Asynchrone Nachrichten-Verarbeitungszeit* beschreibt die Verzögerung zwischen der Ankunft einer asynchronen Nachricht am Controller-Port und die darauffolgende ausgehende Nachricht des Controllers.
- *Asynchrone Nachrichten-Verarbeitungsrate* beschreibt die maximale Anzahl an asynchronen Nachrichten, die ein Controller in einer bestimmten Zeit verarbeitet.
- *Topologie-Entdeckungszeit* ist die Zeit, die ein Controller braucht zur Erkennung aller verbundenen Netzwerkteilnehmer in der Topologie.
- Die *Grundlast* beschreibt die durch den SDN-Controller erzeugte Netzwerklast im Werkszustand.
- Die *Erkennungszeit von Topologie-Änderungen* ist die Zeit, die ein SDN-Controller zur Erkennung einer Veränderung in der Netzwerktopologie braucht. Dazu gehört zum Beispiel der Ausfall von Ports an Weiterleitungsgeräten.
- Die *Controller Ausfallsicherungszeit* ist die Zeit, die ein Controller-Cluster zur Ersetzung eines ausgefallenen Controllers braucht.
- Die *Zeit für einen Systemstart* ist die Zeit, in der alle für die Nutzung des Controllers erforderlichen Komponenten vollständig hochgefahren sind.

4.2 Testumgebung

In diesem Abschnitt werden die einzelnen Testkomponenten und die Testumgebung vorgestellt.

4.2.1 Wireshark

Wireshark [66] ist eine Open-Source-Anwendung zur Aufnahme und Analyse von Datenverkehr. Wireshark unterstützt die meisten Internetprotokolle, aber auch Protokolle aus dem Automotive- und Industriebereich. Wireshark besitzt eine Nutzeroberfläche, die eine Vielzahl an Funktionen für den Nutzer offenlegt. In Wireshark ist es möglich, die Header und den Inhalt von einzelnen Datenpaketen zu untersuchen. Wireshark ermöglicht es unter anderem nach Protokoll, Zeit, Paketgröße, Quelle und Ziel zu sortieren und

zu filtern. Diese Funktionen erlauben genaue und vereinfachte Analysen von Datenverkehr. Wireshark existiert auch als terminalbasierte Anwendung Tshark, die Parameter zur Aufnahme von Datenpaketen über die Kommandozeile übernimmt.
Wireshark bzw. Tshark werden in dieser Arbeit genutzt, um den Netzwerkverkehr zwischen dem Controller und den Weiterleitungsgeräten zu analysieren.

4.2.2 Mininet

Mininet ist eine Anwendung zur Emulation von Netzwerken auf einem Rechner. Mininet ist in Python geschrieben und baut im Werkszustand auf virtuelle OpenFlow-Switches und SDN-Controller auf. Der Anwender besitzt die Möglichkeit, Netzwerke zu erstellen und mit einem SDN-Controller zu verbinden. Die Anwendung ermöglicht eigene Netzwerktopologien mit Python-Code zu programmieren und aufzubauen. Eine Netzwerktopologie kann in Mininet durch die Anzahl der virtuellen Hosts, Switches und Ports bestimmt werden. Mininet besitzt vorgefertigte Topologien. Zu den Topologien gehören lineare Topologien, Baumtopologien sowie eine Topologie basierend aus einem Switch und ein Host. Mininet erlaubt außerdem die Verbindung mit externen SDN-Controller. Abgesehen vom Netzwerkaufbau macht Mininet auch Gebrauch von Netzwerkapplikationen, wie Ping, Iperf oder Wireshark. Diese Applikationen und eigene Anwendungen können alle auf den virtuellen Hosts genutzt werden. Alle Hosts haben die Möglichkeit separate Kommandozeilen zu öffnen und Anwendungen zu starten. In Mininet können zu der erstellten Netzwerktopologie entsprechende Tests ausgeführt werden.
In dieser Arbeit werden Mininet-Topologien genutzt, um flexibel beliebig große Netzwerke für die entsprechenden SDN-Controller-Tests zu erstellen.

4.2.3 Cbench

Cbench [58] ist eine Benchmarking-Applikation um die Leistung von SDN-Controller zu prüfen. Cbench emuliert eine gegebene Anzahl von Weiterleitungsgeräten und verbindet diese mit dem SDN-Controller. Ein emuliertes Weiterleitungsgerät besitzt eine Vielzahl von virtuellen Hosts, die Netzwerklast durch OpenFlow-Pakete erzeugen. Durch die verschiedenen Hosts am Weiterleitungsgerät werden am SDN-Controller *PacketIn*-Nachrichten erzeugt. Cbench besitzt zwei Modi: Latenz und Durchsatz. Im Latenzmodus werden Datenpakete verschickt und die Hosts warten auf eine entsprechende *FlowMod*-Nachricht des SDN-Controllers. Der Durchsatzmodus verschickt Datenpakete so schnell

wie möglich, ohne auf eine Antwort zu warten und zählt die ankommenden *FlowMod*-Nachrichten.
Diese Benchmarking-Applikation wird genutzt, da diese in mehreren verwandten Arbeiten genutzt wurde [36, 30]. Cbench zeichnet sich speziell durch die Wiederholbarkeit der Tests und durch die Parameter aus, die angegeben werden können.
Mittels der beiden Modi von Cbench werden die asynchrone Nachrichten-Verarbeitungszeit und -Verarbeitungsrate ausgelesen.

4.3 SDN-Controller

Die SDN-Controller bestehen aus der Auswahl von SDN-Controller aus dem Abschnitt 3.7.

4.3.1 Ryu

Der Ryu SDN-Controller basiert auf Python. Ryu ist leichtgewichtiger und simpler aufgebaut als die anderen Controller in der Auswahl, bietet aber weniger Funktionen und ist für kleinere Campus-Netzwerke und Prototyp-Erstellung gedacht. Der Grundaufbau von Ryu hält sich an die übliche Architektur von SDN-Controllern, wie in Abbildung 4.1 zu sehen. Neben dem OpenFlow-Protokoll besitzt dieser eine Reihe von Southbound-Protokollen. Die einzelnen Southbound-APIs sind in Python-Bibliotheken für Ryu ausgelagert. Die Bibliotheken werden für Ryu-Applikationen genutzt, die durch ein Southbound-API Informationen des Netzwerkes abrufen oder das Netzwerk verändern. Der Fokus in der Ryu-Architektur liegt bei OpenFlow. Der OpenFlow-Controller ist eine Hauptkomponente von Ryu und ist zuständig für die Verwaltung von OpenFlow-Switches. Controller-Applikationen werden entweder einzeln oder im Kollektiv durch den Ryu-Manager ausgeführt. Der App-Manager ist für die Verwaltung der Applikationen zuständig und wird von jeder Controller-Applikation genutzt. Zustände und einkommende Ereignisse werden von den Kern-Prozessen verarbeitet, sodass Applikationen auf diese reagieren können. Zusätzlich besitzt Ryu noch ein REST-API, das externe Applikationen nutzen können.

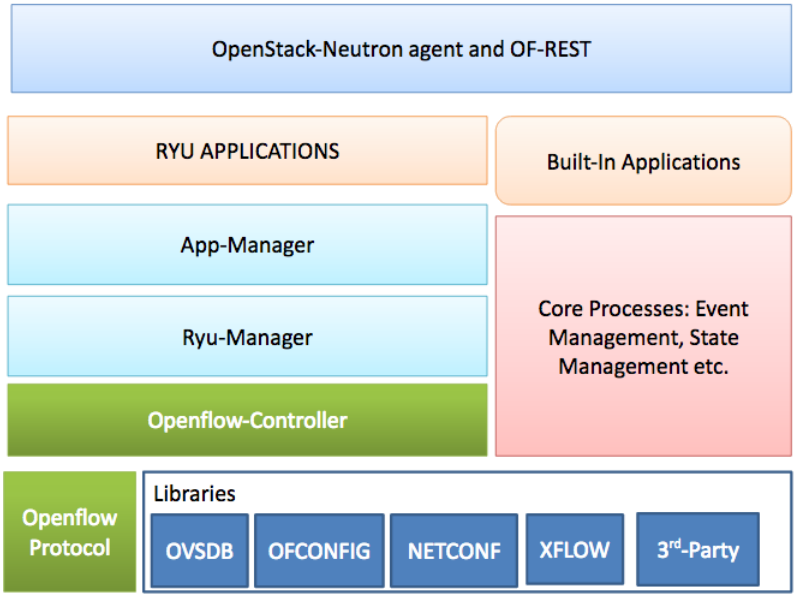

Abbildung 4.1: Architektur des Ryu SDN-Controllers

4.3.2 ONOS

ONOS ist ein Open-Source SDN-Controller der von der ONF veröffentlicht und verwaltet wird. ONOS hat eine Vielzahl renommierter Unterstützer wie Google, AT&T oder Samsung und wird auch in Netzen der Deutschen Telekom genutzt. ONOS sowie die auf ONOS aufbauenden Applikationen sind in Java programmiert. Die Verwaltung der Applikationen und Controller-Module wird über eine Nutzeroberfläche oder eine modulare Laufzeitumgebung namens Apache Karaf gemacht. Für ONOS gibt es eine große Zahl an Applikationen und Funktionen, die von Schnittstellen und Subsystemen in ONOS abhängig sind. In Abbildung 4.2 sind die auf Apache Karaf basierenden Funktionen ersichtlich, die sich um die interne Verwaltung (grau) des SDN-Controllers kümmern. Die einzelnen Netzwerkschnittstellen (rot) nutzen die Southbound-APIs, die ONOS zur Verfügung stellt. ONOS-Applikationen (blau), wie L2 Forwarding nutzen sowohl die Netzwerkschnittstellen als auch die internen Verwaltungsfunktionen von ONOS, um ihre Dienste leisten zu können. Externe Applikationen können sich über das REST-API die Funktionen der ONOS-Applikationen zunutze machen.

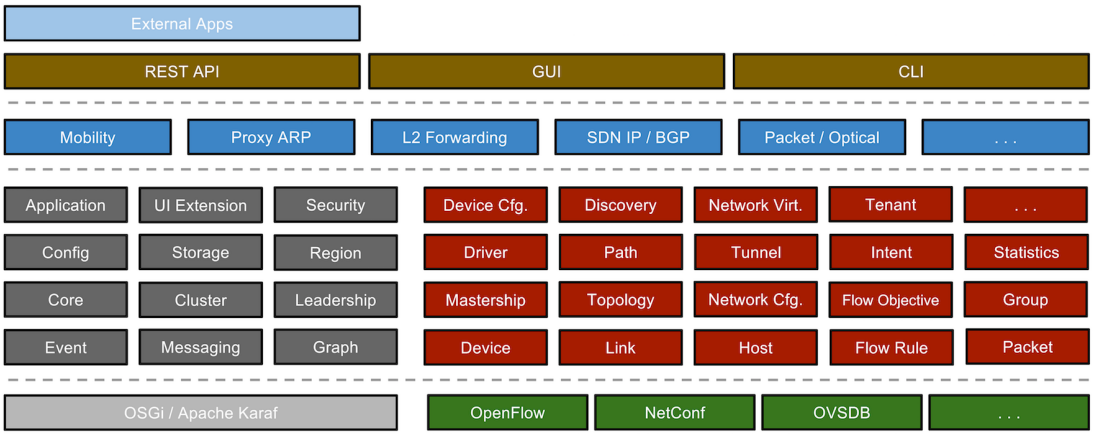

Abbildung 4.2: ONOS-Dienste und Subsysteme

4.3.3 OpenDayLight

OpenDayLight ist ein Open-Source SDN-Controller im Besitz der Linux Foundation. Genauso wie ONOS ist OpenDayLight ein SDN-Controller der von größeren Unternehmen unterstützt und genutzt wird. Mehr als 35 SDN Lösungen wie beispielsweise der Lumina SDN-Controller basieren auf dem OpenDayLight SDN-Controller und nutzen ihn als Kern um unternehmensspezifische Lösungen anzubieten. Auch der OpenDayLight Controller ist in Java programmiert und nutzt Apache Karaf als Laufzeitumgebung, um Applikation und Module zu verwalten. Wie in Abbildung 4.3 ersichtlich hält sich auch der OpenDayLight SDN-Controller im Grundaufbau an die SDN-Architektur. Ein signifikantes Merkmal der OpenDayLight-Architektur ist die OpenDayLight-Platform. Der Kern der OpenDayLight-Platform ist der Model Driven Service Abstraction Layer (MD-SAL). Durch den MD-SAL werden alle Netzwerkapplikationen und Netzwerkgeräte als Objekte oder YANG-Modelle repräsentiert. Von den YANG-Modellen sind verallgemeinerte Beschreibungen von Netzwerkgeräten oder Funktionen einer Applikation zu entnehmen. Der MD-SAL unterscheidet im Wesentlichen zwischen Produzenten und Konsumenten. Produzenten definieren Schnittstellen, produzieren Information und senden Nachrichten. Der MD-SAL speichert diese Informationen in Datenspeichern ab. MD-SAL ermöglicht Konsumenten, von Produzenten definierte Schnittstellen zu nutzen. Konsumenten können von Produzenten erzeugte Informationen aus den Datenspeichern mittels RPC auszulesen und sich auf Nachrichten von Produzenten anzumelden. Durch die vom MD-SAL erzeugte Modularität und Generalisierung von Diensten, Applikationen und Geräten hat die

Entwicklung einer OpenDayLight-Applikation eine höhere Lernkurve als die der anderen SDN-Controller.

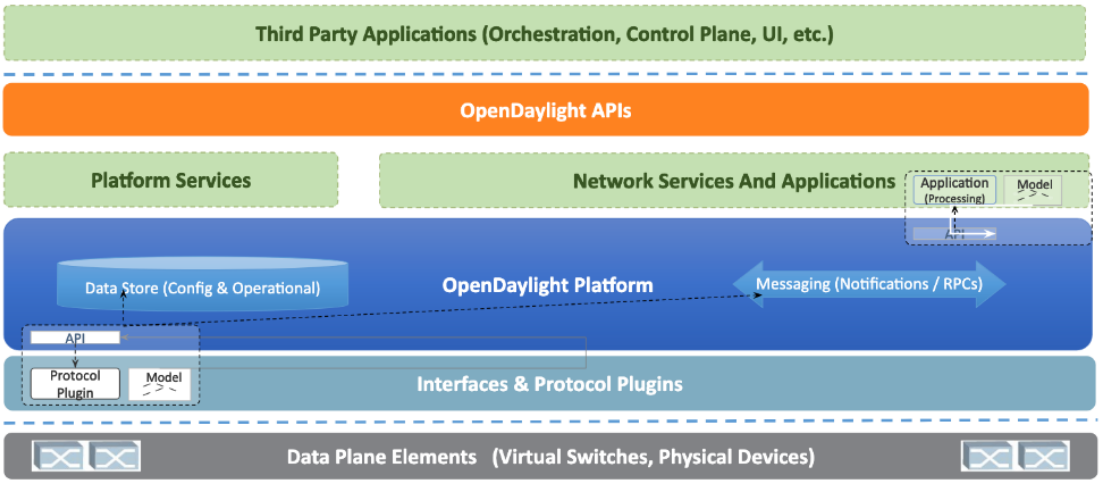

Abbildung 4.3: OpenDayLight-Architektur

4.3.4 Lumina SDN-Controller

Der Lumina SDN-Controller ist der einzige proprietäre SDN-Controller in der Auswahl und ist Eigentum der Firma Lumina Networks. Der Lumina SDN-Controller baut auf OpenDayLight als Basis auf. OpenDayLight als Basis bedeutet, dass Lumina in Java programmiert ist, Apache Karaf als Umgebung nutzt und die gleiche MD-SAL-Architektur wie OpenDayLight teilt. Dafür wurden jedoch Änderungen und eigene Anwendungen hinzugefügt hat. Die OpenFlow-Applikation von Lumina ist beispielsweise eine andere als die von OpenDayLight, auch wenn diese im Grunde genommen die gleichen Schnittstellen nutzt. Wie OpenDayLight in der Architektur des Lumina SDN-Controllers eingeordnet ist, kann aus Abbildung 4.4 entnommen werden. Lumina ist proprietär. Dadurch, können Nutzer von einer Einführung, einem technischen Support und anderen Diensten Gebrauch machen.

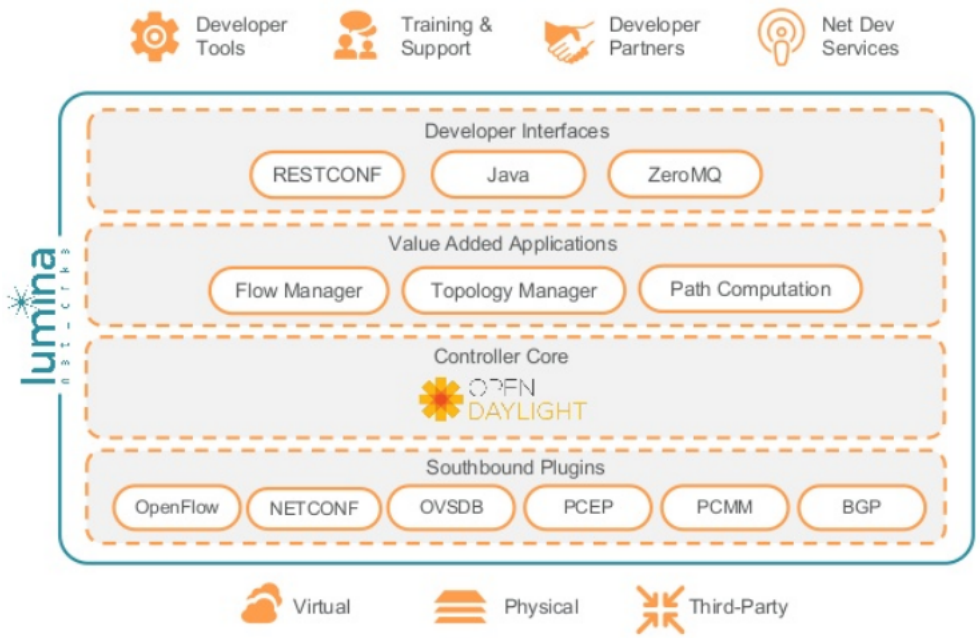

Abbildung 4.4: Architektur des Lumina SDN-Controllers

4.4 Testaufbau

Alle Tests der Performanzanalyse wurden auf einem Rechner mit einem Intel Core i7-4800MQ CPU @ 2.70GHz x 8 Kerne mit 16 GB RAM durchgeführt. Es besteht die Annahme, dass Rechner in zukünftigen Fahrzeugen aufgrund neu hinzukommender Rechenlast und durch komplexere Funktionen im Fahrzeug eine ähnlich hohe Rechenkraft haben werden. Die SDN-Controller wurden in möglichst aktuellen Versionen getestet:

- ONOS: 2.2.1 "Sparrow"
- OpenDayLight: 0.11.2 "Sodium"
- Ryu: 4.32
- Lumina: 9.1.0 basierend auf OpenDayLight ("0.9.1" Flourine)

Die Controller wurden mit dem jeweiligen OpenFlow-Modul getestet. Jede Metrik wurde mit bis zu zehn emulierten OpenFlow-Switches getestet. Die niedrige Anzahl an Weiterleitungsgeräten im Vergleich zu den Tests in verwandten Arbeiten, liegt an den Unterschieden zwischen Fahrzeugen und Rechenzentren (siehe Abschnitt 3.6).

Die Metriken asynchrone Nachrichten-Verarbeitungszeit und asynchrone Nachrichten-Verarbeitungsrate wurden mithilfe von Cbench getestet. Eine Controller-Antwort auf eine asynchrone Nachricht darf üblicherweise nur durch eine *PacketIn*-Nachricht herbeigeführt werden. Cbench sendet eine Vielzahl solcher Nachrichten in einem kurzen Zeitraum. Um SDN-Controller mit Cbench testen zu können, müssen die SDN-Controller eine aktive Applikation zur Erstellung von Flüssen haben. Diese Applikation muss bei Ankunft von *PacketIn*-Nachrichten Flüsse auf den Weiterleitungsgeräten etablieren. Die Aufrufe für die Cbench Tests sehen so aus:
cbench -c localhost -p 6653 -m 10000 -s N -l 11 -t.
Die Aufrufparameter werden wie folgt beschrieben:

- c ist die IP-Adresse oder der Hostname des Controllers
- p ist die Port-Nummer des Controllers
- m ist die Zeit pro Testwiederholung in ms
- s ist die Anzahl der erstellten Switches
- l ist die Anzahl der Wiederholungen pro Test
- t ist der Durchsatzmodus, ohne diesen Parameter läuft Cbench im Latenzmodus

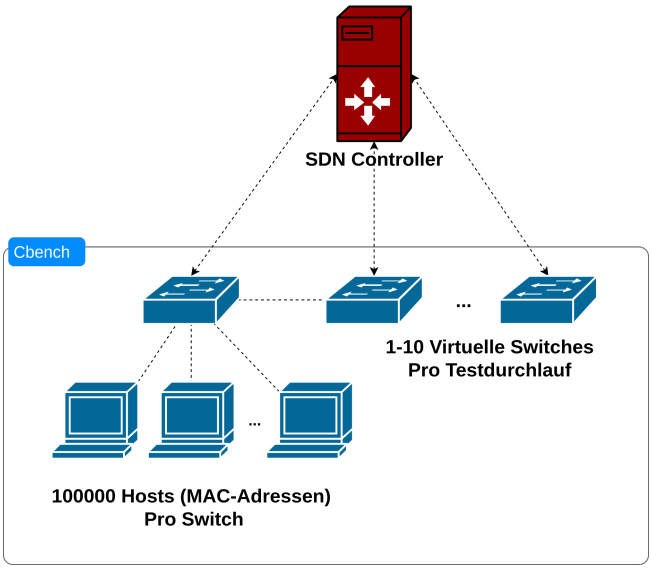

Abbildung 4.5: Testaufbau mit Cbench[39]

Cbench wird auf der lokalen Maschine ausgeführt und hört auf OpenFlow-Port 6653. Die Tests werden mit ein bis zehn Switches ausgeführt. Ein Cbench-Test wird zehnmal 10 Sekunden mit einem Aufwärmdurchlauf ausgeführt. Standardmäßig startet Cbench den Test mit 100.000 verschiedenen Host MAC-Adressen per Switch. Für die asynchrone Nachrichten-Verarbeitungszeit wird Cbench im Latenzmodus gestartet und für die asynchrone Nachrichten-Verarbeitungsrate im Durchsatzmodus. Die Werte können direkt aus Cbench abgelesen werden. Wie ein solcher Aufbau mit Cbench aussieht, kann Abbildung 4.5 entnommen werden.

Die Grundlast, die Controller-Ausfallsicherungszeit, die Erkennungszeit von Topologie-Änderungen und die Topologie-Entdeckungszeit sind mithilfe von Mininet und Wireshark erfasst worden. Diese Tests wurden mittels einer linearen Topologie getestet. Die lineare Topologie vereinfacht das Analysieren vom Netzwerkverkehr mittels Wireshark aufgrund der simplen Zuordnung von Hosts und Switches. Das Aufdecken komplexerer Netzwerkstrukturen oder die Erkennung und Beseitigung von Verbindungsschleifen im Netzwerk, kann nicht mittels der linearen Topologie getestet werden. Bei der voraussichtlich kleinen Anzahl in Fahrzeugen eingebauten an Weiterleitungsgeräten ist kein komplexer Aufbau von Weiterleitungsgeräten nötig. Wie in Abbildung 4.6 zu sehen ist, besteht die verwendete Topologie aus N Switches mit einem Host pro Switch. N stellt die für einen Test verwendete Anzahl an Switches dar.

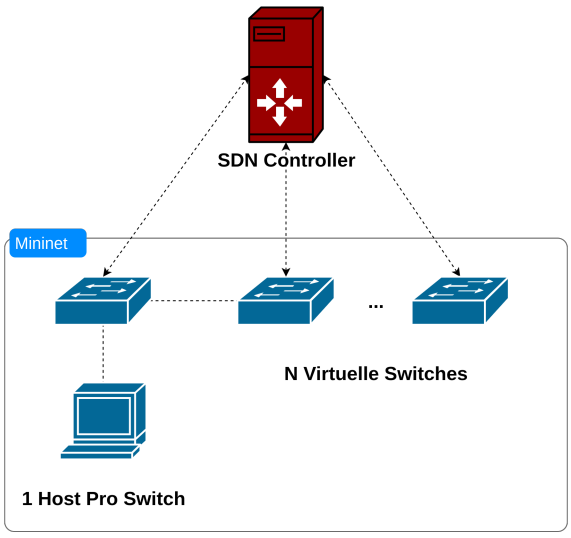

Abbildung 4.6: Testaufbau mit Mininet

Um die Grundlast der SDN-Controller zu bestimmen, wurde eine Mininet-Topologie mit einem emulierten OpenFlow-Switch und einem Host, mit dem jeweiligen SDN-Controller verbunden. Die Grundlast in Bytes/s wurde nach einer Minute Testlaufzeit mittels Wireshark ausgelesen. Um den genauen Wert zu erfahren, wurden in Wireshark nur vom Controller-Port stammende Pakete gezählt. Mittels Filter-Optionen in Wireshark konnte das erreicht werden. Dieser minimale Aufbau reicht, da in diesem Fall nur die vom Controller ausgehenden Pakete relevant sind.

Für die Topologie-Entdeckungszeit wurde ein Skript erstellt, das durch Mininet eine lineare Topologie mit zehn Switches erstellt und mit Tshark die ein- und ausgehenden Pakete aufzeichnet. OpenFlow-fähige Geräte melden sich mit einer *Hello*-Nachricht bei einem Controller an. Diese Metrik hängt von der Anzahl der Switches ab. Je mehr Switches sich bei dem Controller anmelden, desto größer ist der Aufwand zur Entdeckung der Topologie. Das liegt daran, dass mehr Verbindungen aufgedeckt werden müssen. Aus diesem Grund wurde für diesen Test eine möglichst hohe Anzahl an Switches genutzt. Nachdem eine Verbindung zum SDN-Controller hergestellt wird, senden Controller mittels OpenFlow in einem bestimmten Intervall, Link-Layer-Discovery-Protocol (LLDP) Pakete. Diese LLDP-Pakete werden genutzt, um die Verbindungen einzelner Weiterleitungsgeräte aufzudecken. Die Intervallslänge beim Versenden der LLDP-Pakete von den Controllern zur Aufdeckung der Topologie wurde auf 60 Sekunden gestellt, damit die Pakete in Wireshark besser einzuordnen sind. Die Topologie-Entdeckungszeit wird gemessen als die Zeit zwischen der letzten eingehenden *Hello*-Nachricht zum Controller-Port und der letzten eingehenden *PacketIn*-Nachricht mit LLDP-Paket als Antwort.

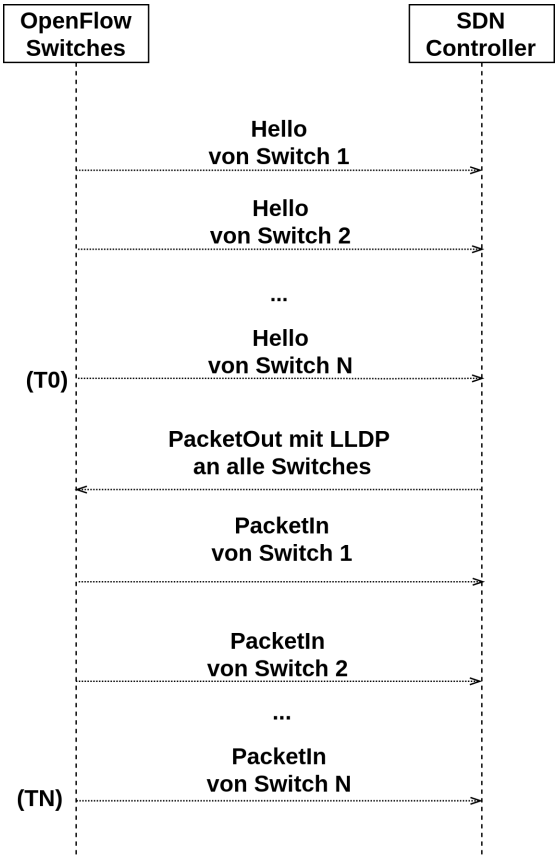

Abbildung 4.7: Ablauf der Topologie-Entdeckung

Abbildung 4.7 zeigt die Sequenz auf, die im Test gemessen wurde. T0 ist der Zeitstempel, ab dem die Zeit gemessen wird und TN ist der Zeitstempel, an dem die Zeit gestoppt wird. Die Differenz der beiden Werte (TN - T0) ergibt die Topologie-Entdeckungszeit.

Für die Erkennungszeit von Topologie-Änderungen wird ein ähnliches Skript wie für die Topologie-Entdeckungszeit genutzt. Für diesen Testfall wurde allerdings nur ein Switch emuliert. Zusätzlich wurde für jeden Testdurchlauf die Verbindung zwischen dem OpenFlow-Switch und dessen Host getrennt, in dem der Port des Switches deaktiviert wurde. Da in diesem Test nur der Verbindungsabbruch an einem Switch ausgelöst wurde, wurden keine weiteren Switches benötigt. OpenFlow-Switches besitzen für alle Ports einen *PortState*. Dieser ist eine Struktur aus Statusbits, die den aktuellen Zustand für einen Port beschreibt. Die Statusbits des *PortState* sind wie folgt definiert:

- $OFFPS\_LINK\_DOWN = 1$: Es existiert keine Verbindung an diesem Port.

- *OFFPS_BLOCKED = 1*: Der Port ist blockiert.
- *OFFPS_LIVE = 1*: Es existiert eine Verbindung an diesem Port.

Wenn die *PortState*-Struktur, wie in diesem Testfall verändert wird, senden OpenFlow-Switches eine *PortStatus*-Nachricht an den SDN-Controller. Die *PortStatus*-Nachricht soll den SDN-Controller über die Änderung am jeweiligen Port informieren. Der SDN-Controller sendet daraufhin LLDP-Pakete an alle Switches, die von der Änderung betroffen sind. Die Erkennungszeit von Topologie-Änderungen ist dementsprechend die Zeit zwischen der ersten *PortStatus*-Nachricht und dem ersten *PacketOut* als Reaktion auf die Veränderung der Topologie. Abbildung 4.8 zeigt den beschriebenen Ablauf. Der Zeitstem-

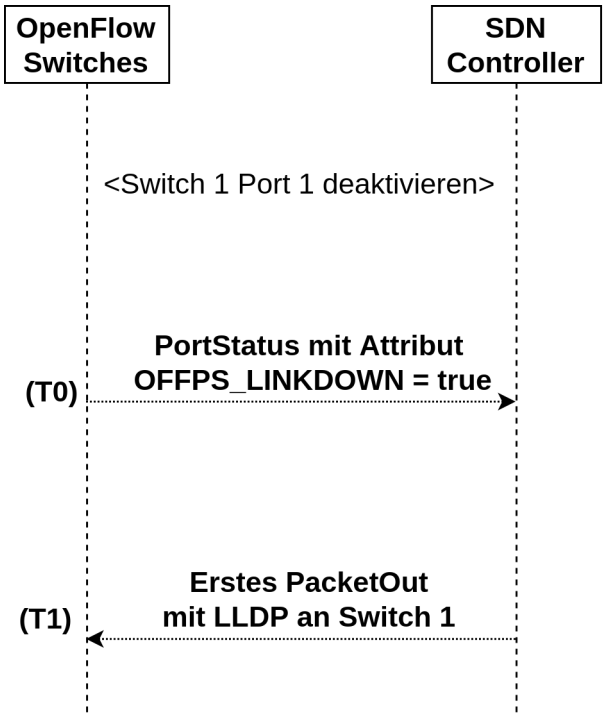

Abbildung 4.8: Ablauf der Erkennung von Topologie-Änderungen

pel T0 ist die Zeit, in der die *PortStatus*-Nachricht ankommt und der Zeitstempel TN beschreibt die Zeit, in der der Controller das erste LLDP-Paket als Reaktion versendet. Die Erkennungszeit von Topologie-Änderungen ist definiert als die Differenz zwischen den beiden Werten (T1 - T0).

Anders als für die anderen Tests wurde bei der Zeit für den Systemstart, weder Wireshark noch Mininet genutzt. Der Controller wurde mit den Basis-Applikationen und dem

OpenFlow-Modul hochgefahren. Der vollständige Start des SDN-Controllers wurde per Log-Analyse überprüft. Abgesehen von Ryu erstellten alle Controller eine Log-Datei mit Datum und Uhrzeit. Diese gibt wieder, wann alle Komponenten hochgefahren sind. Für Ryu wurde die Aufnahme von Log-Informationen im Nachhinein hinzugefügt. Die Zeit für den Systemstart ist die Differenz vom ersten Log-Eintrag beim Start des Controllers T0 und dem letzten Log-Eintrag TN, der die Betriebsbereitschaft einer Komponente mitteilt. Der Ablauf ist in Abbildung 4.9 ersichtlich.

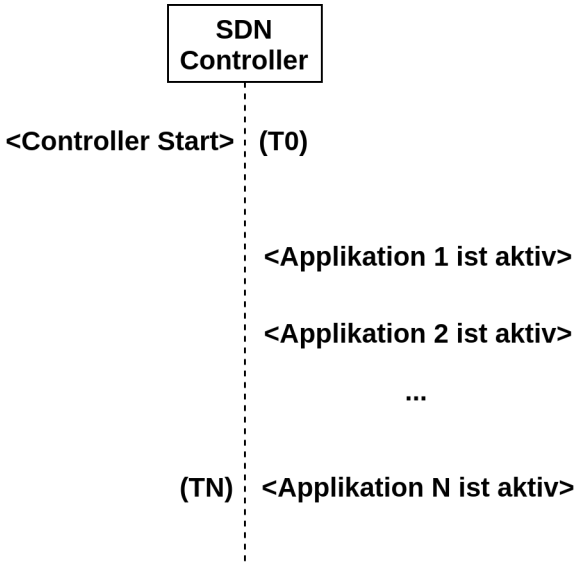

Abbildung 4.9: Ablauf des Systemstartes

Für die Controller-Ausfallsicherungszeit wurde ein Controller-Cluster mittels den gegebenen Funktionen des SDN-Controllers erstellt. Diese Metrik wurde nur auf dem ONOS SDN-Controller und dem OpenDayLight SDN-Controller getestet.
Der OpenDayLight SDN-Controller nutzt das RAFT-Protokoll [50] als Konsensusalgorithmus, um einen primären Controller auszuwählen. Der RAFT-Algorithmus ist ein Abstimmverfahren, um einen Anführer auszuwählen. Damit mittels RAFT ein primärer Controller ausgewählt werden kann bedarf es einer Mehrzahl von Stimmen. Ein einzelner Controller ist nicht in der Lage, eine Mehrzahl zu generieren. Ein OpenDayLight-Cluster ist nur funktionstüchtig, wenn mindestens zwei Controller aktiv sind. Aus diesem Grund wurden die Ausfallsicherungstests mit drei Controllern getestet. Wenn ein Controller ausfällt, ist so immer noch für eine Mehrzahl gesorgt.
ONOS hat in früheren Versionen einen Cluster ähnlich wie OpenDayLight aufgestellt. In Versionen nach 1.14 wurde die Erstellung eines Clusters geändert. ONOS nutzt im Hinter-

grund Atomix [53], ein von der ONF unterstütztes Framework zum Erstellen von fehlertoleranten verteilten Systemen. Für einen ONOS-Cluster muss ein zusätzlicher Atomix-Cluster aufgestellt werden. Der Atomix-Cluster stellt Speicherknoten dar, die für die Verwaltung und für eine geteilte Netzwerkinformationsbasis sorgen. In diesen Speicherknoten schreiben die einzelnen ONOS-Knoten ihre Informationen. Diese Informationen werden an die anderen Atomix-Speicherknoten weitergeben. Die ONOS-Knoten werden durch die Speicherknoten über Ereignisse und Änderungen im Netzwerk informiert. Der RAFT-Algorithmus wurde nach Atomix ausgelagert, dementsprechend kann ein ONOS-Cluster im Gegensatz zu OpenDayLight mit einem Controller-Knoten funktionstüchtig bleiben.
Die Mindestanforderung für einen Lumina SDN-Controller sind ein Rechner, der über 4 Kerne und 6 GB RAM verfügt. Ein Lumina-Cluster wird genau wie ein OpenDayLight-Cluster aufgebaut und muss deswegen auch mit mindestens drei Controllern aufgestellt werden. Für einen Ausfallsicherungstest mit drei Lumina SDN-Controller fehlen dementsprechend die Speicher- und Rechenkapazitäten. Der Ryu SDN-Controller besitzt nicht die Funktion, einen Cluster aufzustellen. Aus diesen Gründen wurden der Ryu SDN-Controller und der Lumina SDN-Controller nicht dem Ausfallsicherungstest unterzogen. Die beiden Controller-Cluster wurden gemäß deren Anleitungen aufgestellt. Der ONOS-Cluster wurde mittels drei Docker-Container aufgestellt, der OpenDayLight SDN-Controller mittels drei virtuellen Maschinen mit jeweils 2 Kernen und 4 GB RAM. Es wurde eine lineare Topologie mit zwei OpenFlow-Switches erstellt. Jeder Switch wurde mit allen Controllern im Cluster verbunden und jeder wurde einem Controller zugewiesen. Vor dem Test wird sichergestellt, dass die SDN-Controller erfolgreich mit dem Netzwerk verbunden sind und die Topologie bereits entdeckt wurde. Der Netzwerkverkehr wurde 2 Minuten mit Tshark aufgenommen. Während der Aufnahme wurde 60 Sekunden durch Iperf UDP-Netzwerklast zwischen dem ersten und zweiten Host erzeugt. Für diesen Fall wurden bei beiden Switches entsprechende Flüsse in die Tabellen eingetragen. Während der Erzeugung von Netzlast wurde der Rechner vom primären Controller des Clusters nach 30 Sekunden ausgeschaltet. Für ONOS wurden ein ONOS-Knoten und ein Atomix-Knoten ausgeschaltet. In Abbildung 4.10 wird beschrieben, wie die Zeit der Ausfallsicherung gemessen wurde. Der Zeitstempel T0 ist der Zeitpunkt, an dem der Rechner vom primären Controller des Clusters ausgeschaltet wird. Nach dem ein neuer primärer Controller ausgewählt wurde, sendet der Cluster *RoleReq*-Nachrichten. Die Switches senden als Antwort *RoleReply*-Nachrichten an die Controller. Die letzte von den Switches kommende *RoleReply*-Nachricht, ist der Zeitpunkt TN. Die Zeit zwischen der letzten *Ro-*

leReply-Nachricht und dem Ausschalten des Anführers ist die Ausfallsicherungszeit (TN - T0).

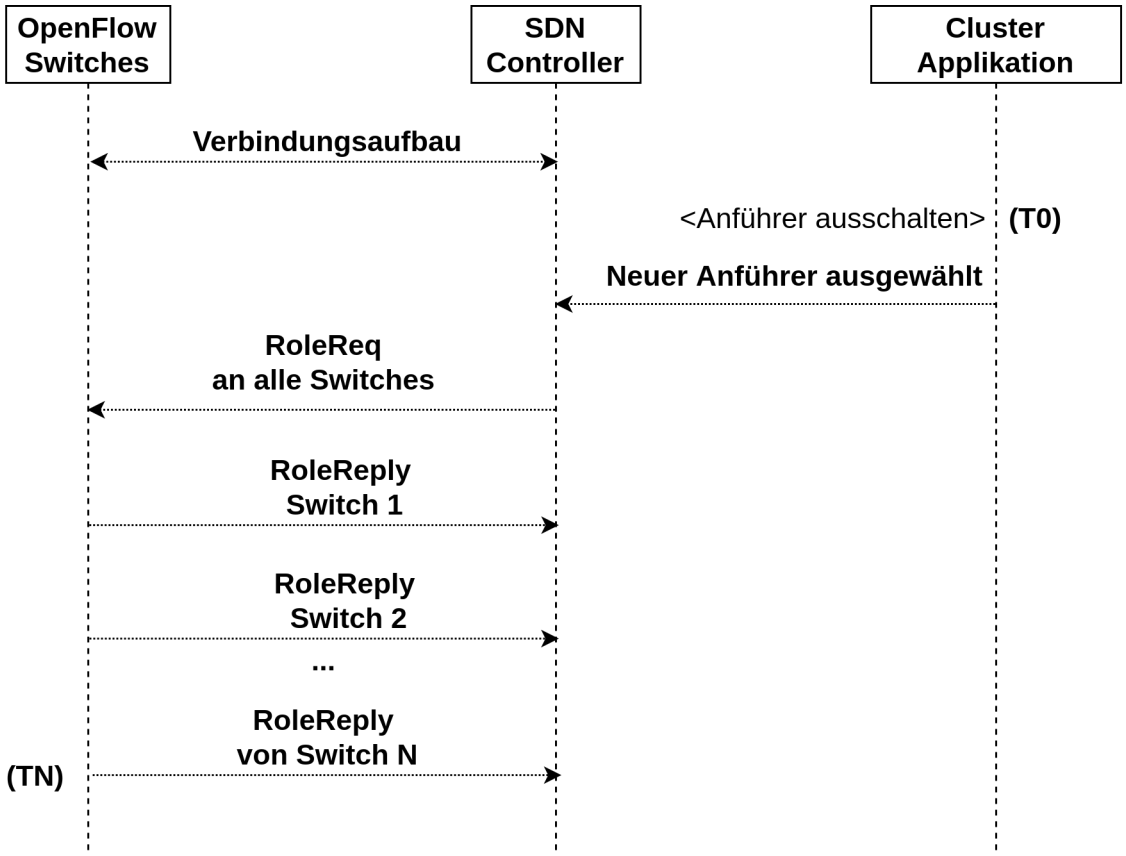

Abbildung 4.10: Ablauf der Ausfallsicherung

Für ONOS wurde ein weiterer Test aufgestellt. Es wurde eine Applikation genutzt, die direkt über den Controller weiterleitet und nicht in die Fluss-Tabellen schreibt. Dieser zusätzliche Test zeigte den Paketverlust, den ONOS bei Ausfall eines Controllers bewirkt. Hierzu wurde ein Skript erstellt, das dasselbe Netzwerk, wie im Ausfallsicherungstest nutzt. In diesem Test wurden genau 10.000 UDP-Pakete von einem Host zu einem anderen geschickt. Während die Pakete gesendet wurden, wurde einer der Controller ausgeschaltet. Die Anzahl der nicht angekommenen Pakete ist der Paketverlust, der durch den Ausfall erzeugt wurde. Das Weiterleiten von Paketen liegt nicht im eigentlichen Aufgabenbereich des Controllers, dementsprechend wird dieser Wert nicht als Metrik aufgeführt. Eine Applikation für ONOS war schon vorhanden, für OpenDayLight

hätte eine solche Applikation erst erstellt werden müssen. Für OpenDayLight wurde aus diesem Grund dieser Wert nicht getestet.

4.5 Evaluation

In diesem Abschnitt werden die Ergebnisse der Tests ausgewertet. Die einzelnen SDN-Controller werden direkt durch die erzielten Werte in den Tests verglichen und bewertet. Der Schwerpunkt der Auswertung liegt auf den Auswirkungen im Fahrzeugbordnetzwerk.

4.5.1 Asynchrone Nachrichten-Verarbeitungszeit

Für den Einbau in ein Fahrzeug ist die asynchrone Nachrichten-Verarbeitungszeit kritisch. Diese gibt die durchschnittliche Dauer an, bis eine Anfrage eines Weiterleitungsgerätes vollständig bearbeitet wurde. Wenn ein SDN-Controller für die zeitlichen Verhältnisse eines Fahrzeuges zu lange für die Verarbeitung eines Paketes braucht, kann dieser nicht für den Einbau in ein Fahrzeug in Erwägung gezogen werden. Die Werte werden den Referenzwerten von Lim et. al. der BMW Research Group gegenübergestellt [37], siehe Tabelle 4.1.

Datentyp	Max. End-zu-End Verzögerung
Kontrolldaten	$\leq$ 10 ms
Fahrerassistenzsysteme Kameradaten	$\leq$ 45 ms
Multimedia Audiodaten	$\leq$ 150 ms
Multimedia Videodaten	$\leq$ 150 ms

Tabelle 4.1: Maximale End-zu-End Verzögerung von Fahrzeugdiensten [37]

Diese Referenzwerte beschreiben die End-zu-End Verzögerung die einzelne Dienste in Fahrzeugen erlauben. Die End-zu-End Verzögerung für Kontrolldaten, Fahrerassistenzsysteme und Kameradaten stellen harte Deadlines dar. Die Audio- und Videodaten haben End-zu-End Verzögerungen, die feste Deadlines darstellen. In der Tabelle ist zu sehen,

dass die Multimediadaten die höchste End-zu-End Verzögerungen akzeptieren. Ein SDN-Controller sollte mindestens in der Lage sein, Multimediadaten in dem angegeben Zeitfenster von 150 ms zu verarbeiten zu können. Dieser könnte sonst keine der vorgegebenen Deadlines einhalten und wäre für keinen Anwendungsfall im Fahrzeug nutzbar. Die Ergebnisse der asynchronen Nachrichten-Verarbeitungszeit der einzelnen SDN-Controller können von Abbildung 4.11 abgelesen werden. Es fällt auf, dass Ryu die höchste Verarbeitungszeit hat. OpenDayLight hat eine ähnlich kurze Verarbeitungszeit wie ONOS, dafür ist ONOS in der Verarbeitungszeit konstanter. Wichtig ist in diesem Fall aber die Tatsache, dass jeder einzelne SDN-Controller in der Auswahl weit unter der maximalen End-zu-End Verzögerung aller Datentypen der Tabelle 4.1 liegt. Diese geringe Verarbeitungszeit kann mit der geringen Anzahl an Weiterleitungsgeräten zusammenhängen. SDN-Controller müssen üblicherweise viel mehr Anfragen von Weiterleitungsgeräten auf einmal verarbeiten als in diesem Test. Wenn ein SDN-Controller an der Weiterleitung von Paketen beteiligt ist, muss in Bezug auf die Werte in Tabelle 4.1 nur mit einer geringen Erhöhung der End-zu-End Verzögerung gerechnet werden.

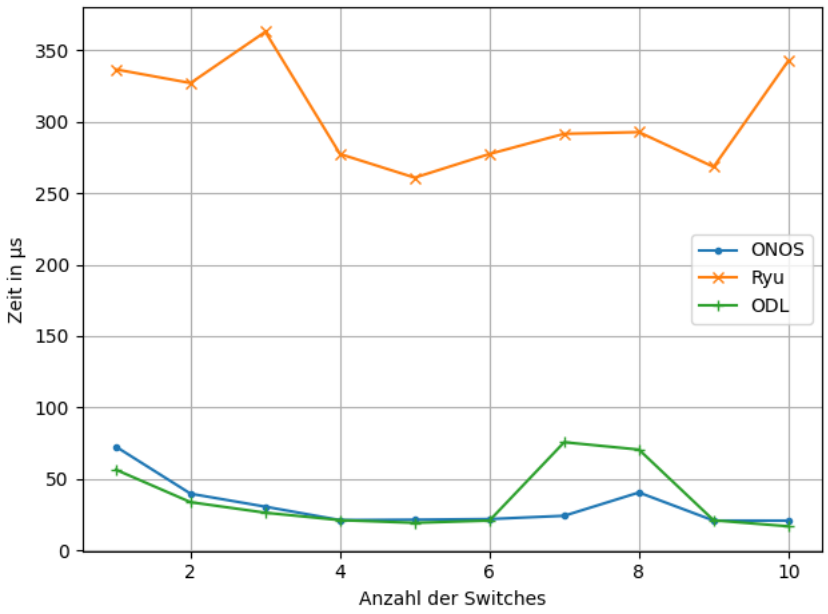

Abbildung 4.11: Asynchrone Nachrichten-Verarbeitungszeit der SDN-Controller

4.5.2 Asynchrone Nachrichten-Verarbeitungsrate

Asynchrone Nachrichten kommen beim SDN-Controller nur an, wenn neue Netzwerkflüsse auftreten. In modernen Fahrzeugen könnte dies beispielsweise durch eine Verbindung mit einem Server für ein Update zustandekommen. Üblicherweise werden in so einem Fall nur einige asynchrone Nachrichten an den SDN-Controller geschickt, damit dieser entsprechend reagieren kann. Der Wert der asynchronen Nachrichten-Verarbeitungsrate liegt in der Information, von der Robustheit der SDN-Controller gegenüber Attacken, die eine Vielzahl asynchroner Nachrichten erzeugen. In Abbildung 4.12 sind die Werte der asynchronen Nachrichten-Verarbeitungsrate aufgezeichnet. Für diese gilt: je höher, desto besser. Der Abbildung kann entnommen werden, dass ONOS eine höhere Verarbeitungsrate hat als Ryu und OpenDayLight. Mit zehn Switches schafft ONOS eine Verarbeitungsrate von über 600 eingehenden Nachrichten pro Millisekunde. Ein Angreifer müsste also einen Netzwerkverkehr mit über 600.000 unterschiedlichen Nachrichten pro Sekunde erzeugen, um einen ONOS SDN-Controller in dessen Funktionen einzuschränken. Dabei müssen die Nachrichten immer wieder ein Table-miss erzeugen. OpenDayLight und Ryu schaffen im Vergleich eine Verarbeitung von weniger als 100 eingehenden Nachrichten pro Millisekunde.

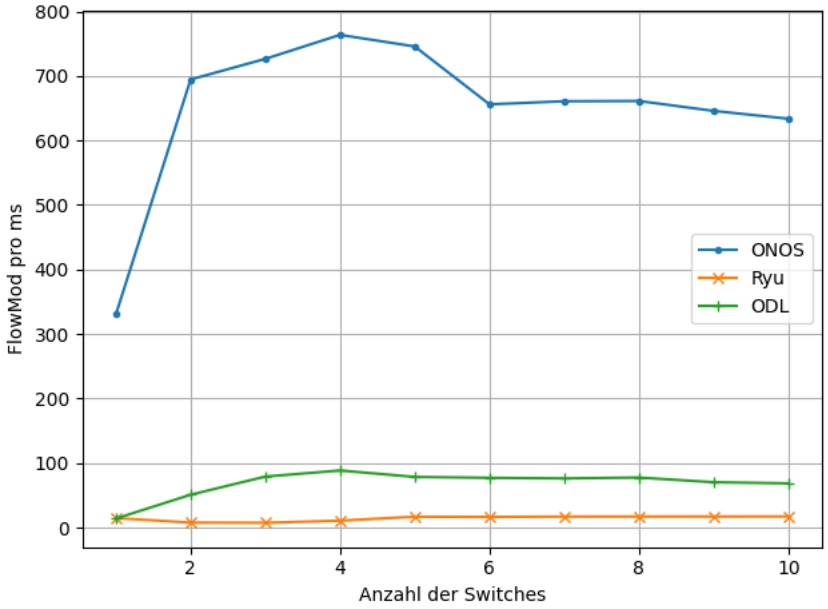

Abbildung 4.12: Asynchrone Nachrichten-Verarbeitungsrate der SDN-Controller

4.5.3 Grundlast

Die Grundlast bestimmt die zusätzliche Last, die ein Controller auf ein Fahrzeugbordnetzwerk legt. Der SDN-Controller sollte mit einer möglichst geringen Grundlast, alle wichtigen Dienste leisten können. Eine hohe Grundlast kann zu Verzögerungen im Fahrzeugbordnetzwerk führen und würde die Netzwerkstabilität negativ beeinflussen. Grundlast wird in allen Controllern durch die zyklischen Nachrichten, zur Überprüfung der Topologie erzeugt. In Abbildung 4.13 ist deutlich zu sehen, dass kein Controller nennenswerte Zusatzlast erzeugt. Keiner der Controller erzeugt mehr als 1,5 Kbyte/s Netzlast. Es gibt auch keine starken Schwankungen bei diesen Werten, da diese alle in konstanten Intervallen Pakete versenden. Während ONOS periodisch mit OpenFlow-Paketen den Status der einzelnen Ports der Switches abfragt, fragen OpenDayLight und Lumina periodisch Statistiken der einzelnen Switches ab. Dazu gehören beispielsweise die Anzahl an eingegangenden Paketen an einem Port. Ryu fragt diese Werte in den Standardapplikationen nicht automatisch ab und erzeugt im Vergleich zu den anderen SDN-Controllern eine geringe Netzlast.

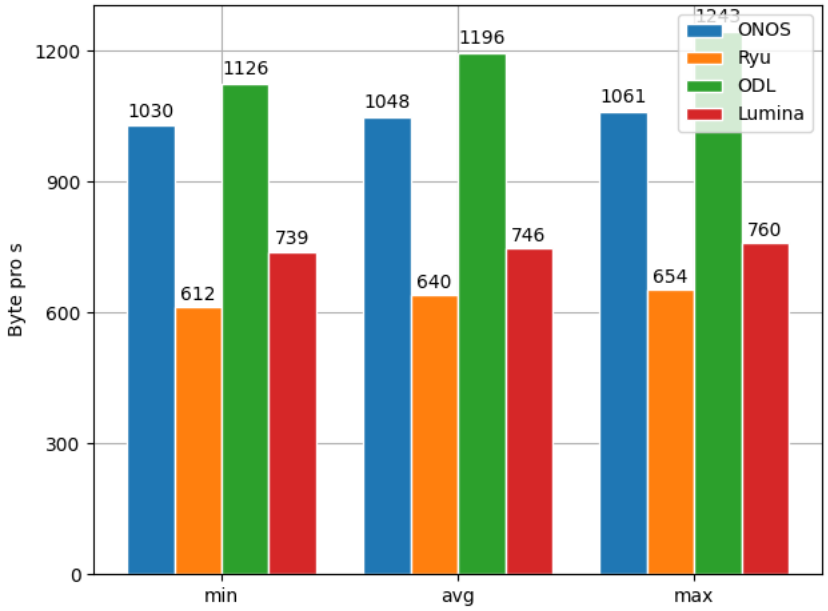

Abbildung 4.13: Grundlast der SDN-Controller

4.5.4 Topologie-Entdeckungszeit

Die Weiterleitungsgeräte des Fahrzeugbordnetzwerkes müssen zu jedem Start des Systems einmalig erkannt werden. Bei jedem Start wird damit ersichtlich, ob Weiterleitungsgeräte nicht funktionstüchtig sind und der Fahrer benachrichtigt werden muss. Der SDN-Controller kann nur Flüsse auf Geräten ändern, die ihm bekannt sind. Die Topologie-Entdeckungszeit ist also besonders beim Start oder Neustarten des SDN-Controllers im Netzwerk wichtig. Für die Topologie-Entdeckungszeit gilt: je niedriger, umso besser. In Abbildung 4.14 können die Werte für die Topologie-Entdeckungszeit der einzelnen Controller entnommen werden. Der ONOS SDN-Controller hat die niedrigste durchschnittliche Topologie-Entdeckungszeit. OpenDayLight dagegen hat die niedrigste Differenz zwischen Minimalwert und Maximalwert und ist somit stabiler im Wert.

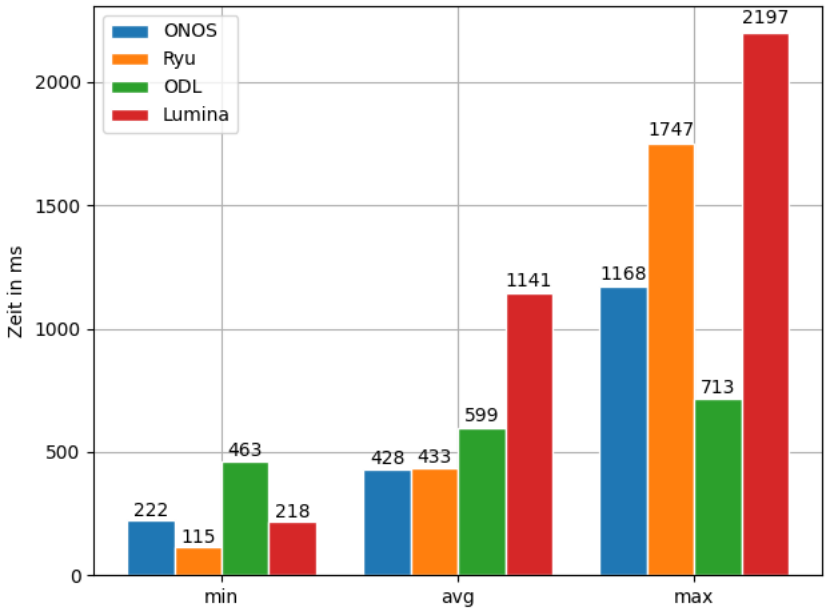

Abbildung 4.14: Topologie-Entdeckungszeit der SDN-Controller

4.5.5 Zeit für den Systemstart

Zusätzlich zu der Topologie-Entdeckungszeit spielt die Zeit zum Starten des Systems eine große Rolle. Weder die Topologie-Entdeckung noch andere Funktionen des SDN-Controllers können genutzt werden, solange dieser hochfährt. Aus diesem Grund ist

ein zügiger Systemstart des SDN-Controllers bei einem Ausfall umso wichtiger. Abbildung 4.15 zeigt, wie lange die einzelnen Controller zum Hochfahren brauchen. Abgesehen vom Ryu SDN-Controller in Abbildung 4.16 brauchen alle Controller mehrere Sekunden zum Hochfahren, der Lumina SDN-Controller sogar über 30 Sekunden. Die hauptsächliche Wartezeit beim Start der SDN-Controller verursachen die zu hochfahrenden Systemkomponenten. Die Applikationen müssen auf Systemkomponenten und Schnittstellen des Controllers warten, damit diese genutzt werden können. Nachdem dieser Prozess beendet ist, werden die einzelnen Applikationen hochgefahren. Es ist immer abhängig von der Implementierung einer Applikation, wie lange diese zum Hochfahren braucht. Der Ryu SDN-Controller hat in diesem Fall einen Vorteil, da die Programmiersprache Python und die simple Architektur von Ryu relativ leichtgewichtig sind. Aus diesem Grund ist die von Ryu zum Hochfahren benötigte Zeit deutlich geringer als die der anderen SDN-Controller.

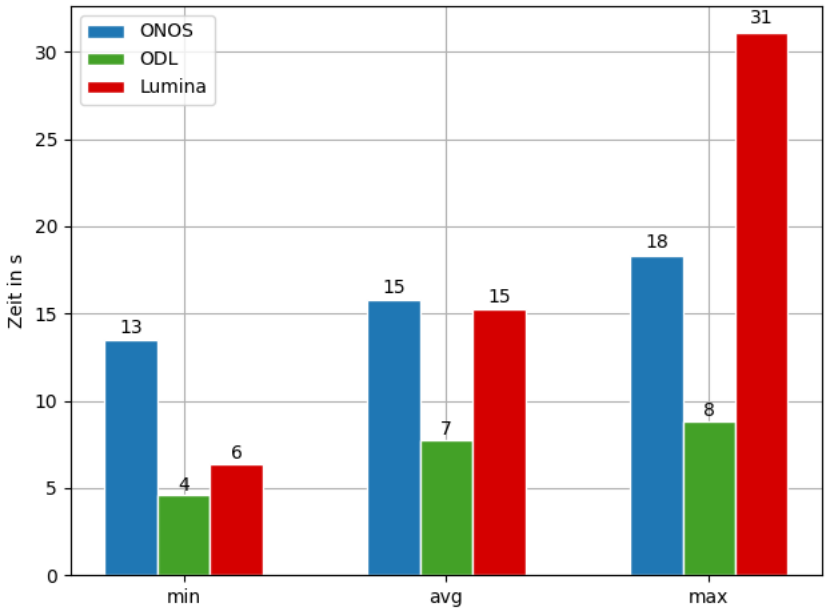

Abbildung 4.15: Zeit für den Systemstart der einzelnen SDN-Controller

Im Falle eines Ausfalls wären die Werte in Abbildung 4.15 in Abhängigkeit zu den zeitlichen Einschränkungen eines Fahrzeuges, nicht zu tolerieren und die Rückfallstrategie müsste eingeschaltet werden. Mit solchen Zeiten könnte ein Fahrzeug ohne Redundanz niemals in den normalen Betriebszustand zurückkehren, nachdem ein Controller ausgefallen ist. Eine Rückfallstrategie würde mindestens die maximale End-zu-End Verzöge-

rung berücksichtigen. Ebenso wichtig ist auch der Start des Fahrzeuges. Passagiere des Fahrzeuges müssten mehrere Sekunden warten, bevor diese losfahren könnten. Heutige Fahrzeuge sind unmittelbar nach der Zündung des Motors betriebsbereit. Das Warten auf den Controller würde die durchschnittliche Startzeit von Fahrzeugen stark erhöhen. Der Ryu SDN-Controller schneidet in diesem Test deutlich besser als der Rest der Controller ab.

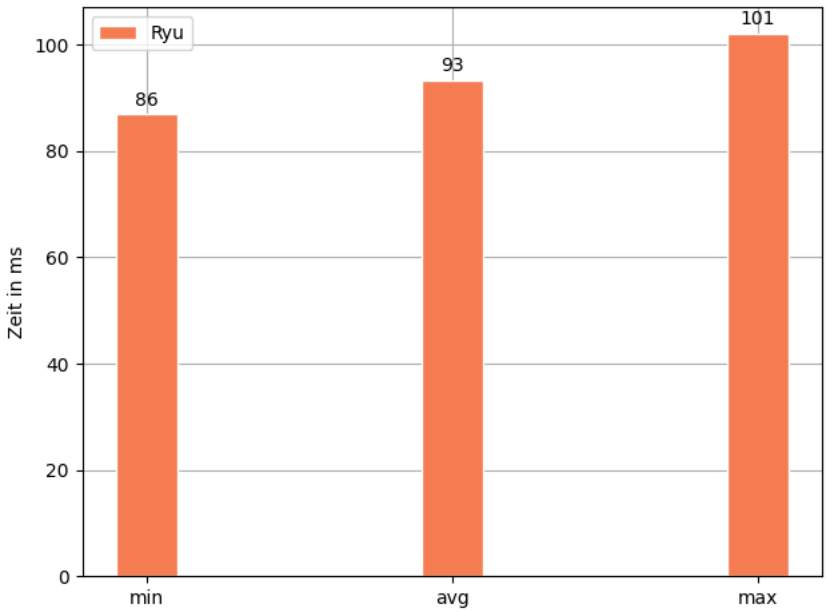

Abbildung 4.16: Zeit für den Systemstart des Ryu SDN-Controllers

4.5.6 Erkennungszeit von Topologie-Änderungen

Die Erkennungszeit von Topologie-Änderungen ist ein kritischer Wert für SDN-Controller im Fahrzeug. Die Erkennung von Fehlern muss so schnell wie möglich geschehen, damit der Fahrer des Fahrzeuges rechtzeitig über diese benachrichtigt werden kann. Da beispielsweise OpenFlow-Switches Auskunft über Änderungen an den Ports geben, können so auch Ausfälle von angeschlossen Komponenten gemeldet werden. Je schneller ein SDN-Controller eine Änderung erkennt, desto schneller erfolgt die Fehlererkennung. Wie auf Abbildung 4.17 zu sehen, hat der Lumina SDN-Controller die geringste durchschnittliche Zeit, zur Erkennung von Änderungen in der Topologie. Eine vergleichsweise hohe Zeit hat der OpenDayLight-Controller, der als Basis für den Lumina SDN-Controller dient.

Anzumerken ist der relativ hohe Unterschied zwischen dem Minimalwert und dem Maximalwert, den alle Controller außer ONOS vorweisen. Abgesehen von ONOS besitzen alle Controller einen Maximalwert von über zwei Sekunden und einen Minimalwert von weniger als einer Sekunde. Die ONOS-Werte sind im Vergleich zu den anderen Controllern stabiler und tendieren weniger dazu 'Ausreißer' zu bekommen. Besonders in sicherheitskritischen Fällen wie der Fehlererkennung ist nicht nur eine schnelle Reaktion wichtig, sondern auch ein stabiler Wert.

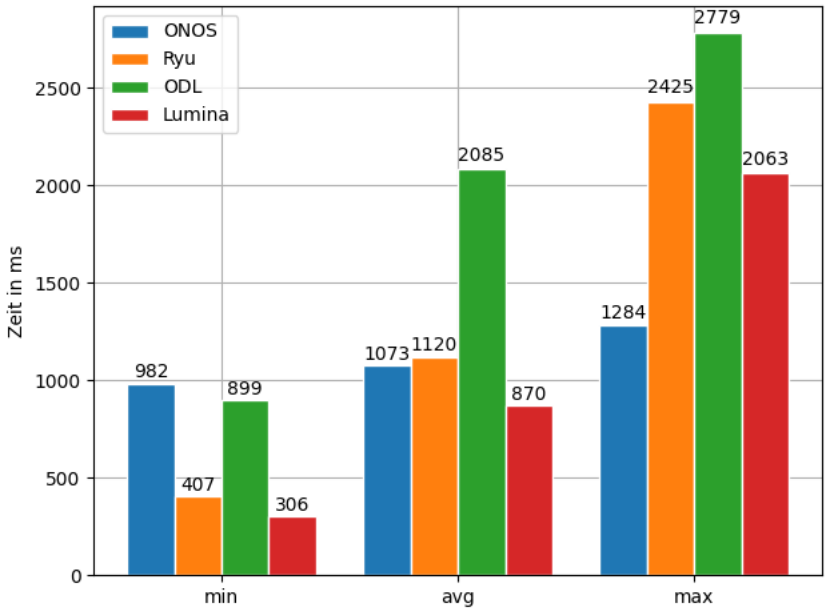

Abbildung 4.17: Erkennungszeit von Topologie-Änderungen der SDN-Controller

4.5.7 Ausfallsicherungszeit

Die Ausfallsicherungszeit ist ein ebenso kritischer Wert für ein Fahrzeug. Diese Zeit gibt an, wie lange ein Controller zur Verhinderung der Rückfallstrategie der Weiterleitungsgeräte im Fahrzeug braucht. Ein Controller, sollte spätestens bis zur nächsten Anfrage eines Weiterleitungsgerätes erreichbar sein oder bis zum nächsten Ereignis, das eine Flussänderung erfordert. Des Weiteren darf es nicht zu Veränderungen der Fluss-Tabellen durch den Ausfall kommen. Eine Veränderung bzw. die Löschung von kritischen Flüssen durch den Ausfall, könnte dazu führen, dass die maximalen End-zu-End Verzögerungen nicht eingehalten werden.

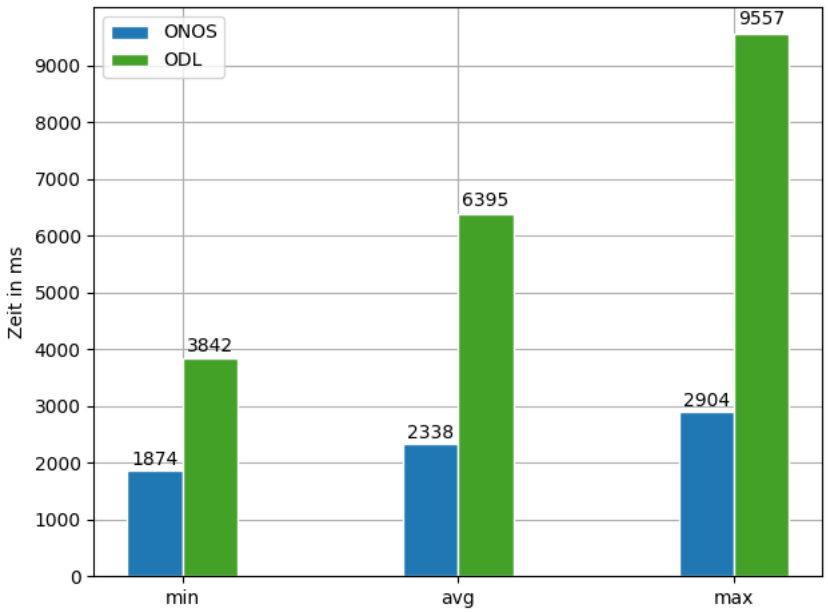

Abbildung 4.18: Ausfallsicherungszeit von OpenDayLight und ONOS

Bei keinem der Controller gab es mit vorkonfigurierten Flüssen einen Abbruch des UDP-Datenverkehrs. In Abbildung 4.18 kann gesehen werden, wie OpenDayLight und ONOS im Test abgeschnitten haben. OpenDayLight brauchte nach der Ausfallsicherung durchschnittlich 6 Sekunden, bevor alle Weiterleitungsgeräte neu zugewiesen wurden. Das Verhindern der Rückfallstrategie würde mit OpenDayLight also durchschnittlich insgesamt 6 Sekunden dauern. Im schlimmsten Fall kann dies fast 10 Sekunden mit OpenDayLight dauern. 10 Sekunden ist bezüglich der zeitlichen Einschränkung eines Fahrzeuges ein sehr hoher Wert. ONOS braucht durchschnittlich etwa 2,5 Sekunden und hat im schlimmsten Fall etwas unter 3 Sekunden gebraucht. ONOS braucht somit deutlich weniger Zeit als OpenDayLight. Soll eine Rückfallstrategie im Fehlerfall Insassen, Umwelt und Fahrzeug sichern, muss diese spätestens direkt nach Überschreitung der maximalen End-zu-End Verzögerungen aktiviert werden. Die Werte der beiden Controller sind dementsprechend inakzeptabel, da sie sich in einem viel höheren Bereich befinden.

In Abbildung 4.19 ist der durch den Ausfall erzeugte Paketverlust von ONOS ersichtlich, dieser liegt durchschnittlich bei etwa bei 2.300 Paketen. In kritischen Netzwerkpfaden wie zum Beispiel dem Motor oder der Bremse führt auch der Verlust weniger Pakete zu fatalen Folgen. Kann ein Signal bzw. eine Nachricht aufgrund des Paketverlustes nicht versendet werden, führt dies zwangsläufig zum Verfehlen von Deadlines.

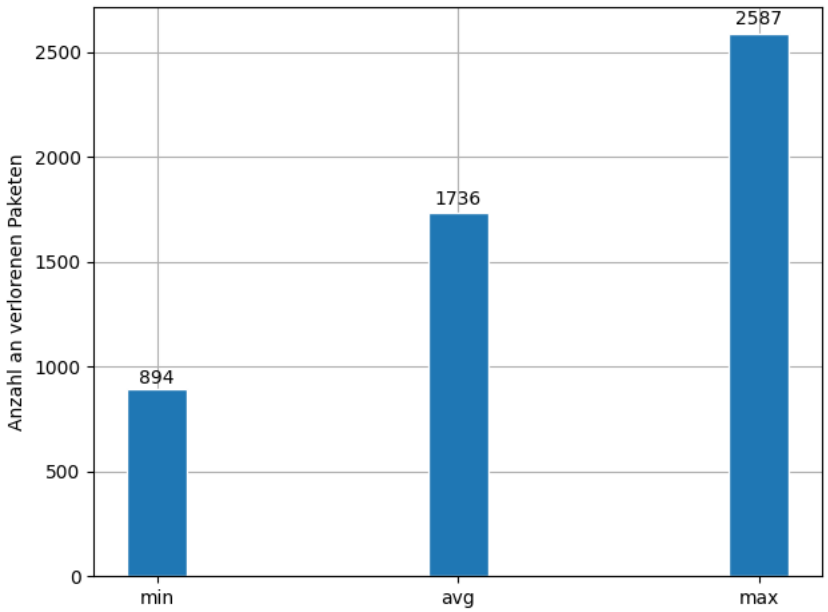

Abbildung 4.19: Paketverlust beim Ausfall von ONOS

4.5.8 Ergebnis

Der ONOS SDN-Controller hat in den Tests der Performanzanalyse am besten abgeschnitten. ONOS hat die kürzeste asynchrone Nachrichten-Verarbeitungszeit, die höchste asynchrone Nachrichten-Verarbeitungsrate, die niedrigste durchschnittliche Topologie-Entdeckungszeit und eine kürzere Ausfallsicherungszeit als OpenDayLight. Nur beim Grundlasttest und beim Systemstart liegt ONOS auf dem vorletzten Platz.
Die einzelnen Testergebnisse sollten abhängig von der Verwendung des SDN-Controllers im Fahrzeug betrachtet werden. Einzelne Metriken können bei bestimmten Anwendungsfällen vernachlässigt werden, während andere umso wichtiger werden. Der Paketverlust bei einem Ausfall wäre nur von Relevanz, wenn keine Fluss-Tabellen-Einträge vorhanden wären und der SDN-Controller für das Weiterleiten von Paketen zuständig wäre. Eine niedrige Erkennungszeit von Topologie-Änderungen ist nur wichtig, wenn der SDN-Controller für kritische Flüsse zuständig ist.
Die einzige Metrik die den Einbau von SDN-Controllern im Fahrzeug momentan verhindert, ist die Zeit für den Systemstart. Wenn SDN-Controller ohne Redundanz eingebaut werden, sind die gemessenen Zeiten für den Systemstart bzw. eines Neustarts der SDN-Controller und die darauffolgende Topologie-Entdeckungszeit, ungeeignet für ein

Fahrzeugbordnetzwerk. Mit Redundanz sind diese Zeiten für einen Ausfall weniger relevant, da ein Ersatz-Controller übernehmen würde. In diesem Fall würde die Ausfallsicherungszeit eine wichtige Rolle spielen. SDN-Controller können den zeitlichen Umständen entsprechend Pakete weiterleiten und Flüsse erstellen. Trotzdem ist deren Zeitverhalten bei einem Ausfall nicht akzeptabel, egal ob mit oder ohne Redundanz. Ein Ausfall durch einen Angriff oder einen technischen Defekt würde ein Fahrzeug mit den getesteten SDN-Controllern, jedes Mal in einen Ausnahmezustand versetzen.

5 Fallbeispiel: SDN-Controller im Fahrzeug

Für den Einbau in ein Fahrzeug hat sich der ONOS SDN-Controller als am geeignetsten gezeigt. Der ONOS SDN-Controller erfüllt die meisten Anforderungen und ist der leistungsfähigste Controller. Aus diesem Grund wird dieser in diesem Kapitel als Fallbeispiel für die Nutzung von SDN-Controllern genutzt. Anders als in den vorherigen Evaluationsschritten ist der Fokus nicht auf dem Controller, sondern auf den Controller-Applikationen. Ziel ist es die vorhandene Applikationslandschaft des Controllers vorzustellen und die gegebenen Anwendungsgebiete aufzuzeigen. Anfangs wird das genutzte Fahrzeugbordnetzwerk mit allen Komponenten beschrieben. Daraufhin wird die Nutzung vom SDN-Controller im Netzwerk erklärt, indem die einzelnen ONOS-Applikationen für dieses Fahrzeugbordnetzwerk vorgestellt werden. Anschließend wird eine der Controller-Applikationen ausgetestet und mit einer bestehenden Automotive-Lösung verglichen.

5.1 Tischaufbau

Das in diesem Kapitel genutzte Fahrzeugbordnetzwerk ist ein von der CoRE-Gruppe erstellter Tischaufbau des Projektes Security for Vehicular Information (SecVI) [24]. Das SecVI-Projekt fokussiert sich auf die Konzipierung und Entwicklung einer simplen und robusten Netzwerk-Architektur, die Security für den Kommunikationsfluss im Fahrzeug sicherstellt. Der Fokus des Tischaufbaus liegt dementsprechend weniger auf der Untersuchung und Durchsetzung von Echtzeit-Anforderungen, sondern auf der Untersuchung von Security-Konzepten. Der Tischaufbau besteht aus mehreren Komponenten, wie in Abbildung 5.1 zu sehen ist. Die meisten heutigen Fahrzeuge nutzen ein zentrales Gateway für die Kommunikation zwischen ECUs. Im Tischaufbau dagegen wird eine Zonenarchitektur genutzt. Im Allgemeinen gruppiert eine Zonenarchitektur die einzelnen Komponenten, nach deren physikalischem Ort. Für die Kommunikation zwischen den Zonen werden Zonen-Controller, Gateways oder ähnliche Komponenten genutzt, die für

die Weiterleitung von eingehenden und ausgehenden Paketen zuständig sind. Die Zonen im Tischaufbau sind wie folgt beschrieben:

- ZC_FR: beschreibt den vorderen rechten Teil des Fahrzeuges.
- ZC_FL: beschreibt den vorderen linken Teil des Fahrzeuges.
- ZC_RR: beschreibt den hinteren rechten Teil des Fahrzeuges.
- ZC_RL: beschreibt den hinteren linken Teil des Fahrzeuges.

Jede dieser Zonen wird durch ein CAN-Ethernet-Gateway realisiert, das auf einem Raspberry Pi 4 läuft. Die Aufgabe der CAN-Ethernet-Gateways ist das Empfangen von CAN-Nachrichten an deren Netzwerkschnittstelle. Empfangene Nachrichten werden nach vorgegebener Zuordnung in Ethernet-Pakete transformiert und an deren Ethernet-Schnittstelle weitergeleitet. Die CAN-Ethernet-Gateways sind auch in der Lage, Ethernet-Pakete zu CAN-Paketen umzuwandeln. Der Vorgang erfolgt dabei von der Ethernet-Schnittstelle zur CAN-Schnittstelle.

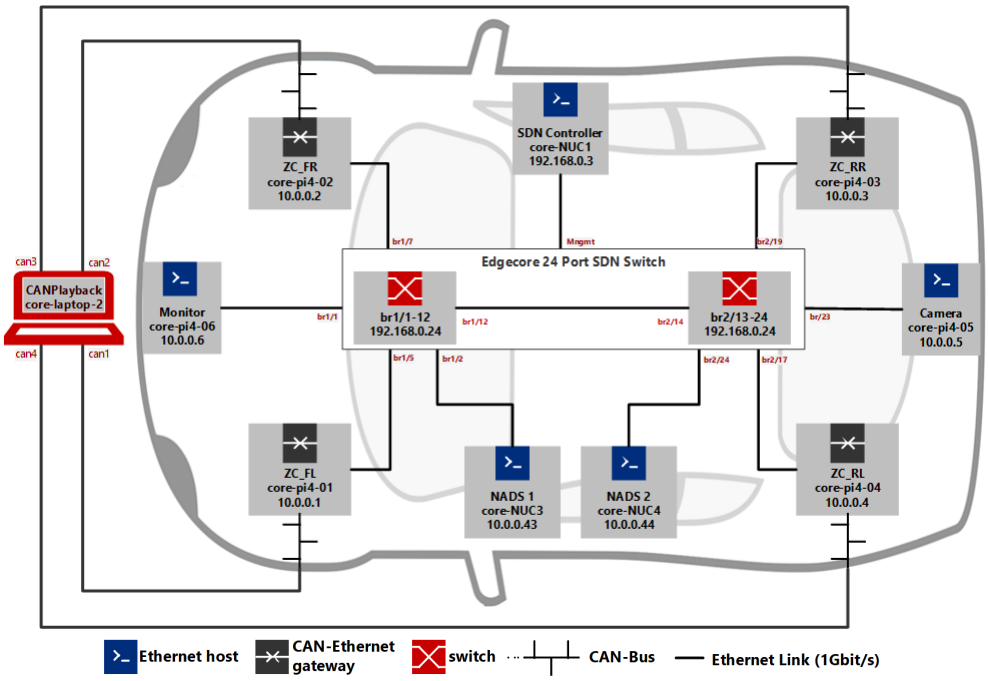

Abbildung 5.1: Netzwerktopologie des Tischaufbaus

Die CAN-Ethernet-Gateways sind mit einem Rechner verbunden, der eine Aufzeichnung vom realen CAN-Datenverkehr eines Serienfahrzeuges in einer Dauerschleife auf die vier

CAN-Verbindungen sendet. Des Weiteren existieren fünf Ethernet-Hosts mit unterschiedlichen Funktionen und Aufgabenbereichen. Der Host im hinteren Teil des Netzwerkes realisiert die Rückfahrkamera eines Fahrzeuges. Im vorderen Teil des Netzwerkes liegt ein Host, der die Videoübertragung der Rückfahrkamera empfängt und mit einem Bildschirm wiedergibt. Zusätzlich gibt es zwei Netzwerk-Anomalie-Erkennungssysteme bzw. Network-Anomaly-Detection-System (NADS). Einen für die hinteren Zonen des Fahrzeuges und einen für die vorderen Zonen des Fahrzeuges. Die NADS sind dafür da, Anomalien im Fahrzeugbordnetzwerk zu erkennen. Anomalien sind beispielsweise vom Durchschnittswert abweichende Bandbreitennutzung über eine bestimmte Zeit. Anomalien werden von den NADS entsprechend gemeldet. Der letzte Ethernet-Host auf der rechten Seite des Tischaufbaus ist der ONOS SDN-Controller. Die Kamera und deren Empfänger laufen genau wie die CAN-Ethernet-Gateways auf Raspberry Pis. Die NADS und der SDN-Controller laufen auf Intel NUCs mit einem Intel Core i3-8109U CPU @ 3,00 GHz x 2 Kerne mit 8 GB RAM. Mitten im Fahrzeug existieren zwei OpenFlow-Switches br1 und br2. Die Switches unterstützen kein TSN, dementsprechend können keine Echtzeit-Anforderungen getestet werden. Alle Netzwerkteilnehmer der vorderen Zone sind mit dem Switch br1 verbunden, während die Netzwerkteilnehmer der hinteren Zone mit dem anderen Switch br2 verbunden sind. Der ONOS SDN-Controller verwaltet die beiden Switches.

5.2 Anwendungsgebiete im Fahrzeug

In diesem Abschnitt werden die Anwendungsgebiete des SDN-Controllers im Fahrzeugbordnetzwerk beschrieben und erklärt.

5.2.1 Zugriffskontrollliste

Die Zugriffskontrolllistenapplikation erlaubt das dynamische Weiterleiten von ankommenden Datenpaketen, basierend auf einer Liste von erlaubten und nicht erlaubten Netzwerkflüssen. In modernen Fahrzeugen wird die Security aufgrund des dynamischen Netzwerkverkehrs eine hohe Rolle spielen. Wenn Fahrzeuge in der Zukunft mit der Infrastruktur oder anderen Fahrzeugen in ihrer Nähe kommunizieren, kann eine Applikation wie die Zugriffskontrollliste zum Schutz des Fahrzeugbordnetzwerkes beitragen.

Beispielsweise können Datenpakete schon zu Beginn anhand ihrer Quell- oder Zieladresse und ihres Protokolltypen blockiert oder zugelassen werden. Fahrzeuge können somit dynamisch Verbindungen von außen zulassen, ohne eine große Angriffsfläche freizulegen. In Abbildung 5.2 ist zu sehen, wie ein Eintrag der Zugriffskontrolllistenapplikation in der ONOS-Nutzoberfläche aussieht. Jeder Eintrag wird unterschieden zwischen *WHITELISTED* und *BLACKLISTED*. Einträge die *WHITELISTED* sind werden zugelassen, während Einträge die *BLACKLISTED* sind vom Netzwerkverkehr blockiert werden. Der Eintrag in Abbildung 5.2 lässt explizit TCP-Netzwerkverkehr zwischen Quelladressen im Bereich 10.0.0.37/32 und Zieladressen im Bereich 10.0.0.4/32 zu. Auch der Quellport mit 1410 und der Zielport mit 1 werden angegeben. Zusätzlich kann optional eine Liste von Geräten angegeben werden, auf die diese Regel angewendet werden soll. Mit der Applikation kann zusätzlich angegeben werden, ob ein auf einen Eintrag zutreffendes Ereignis, berichtet wird oder nicht. Einträge können priorisiert werden. Außerdem existiert ein Feld mit der Angabe, ob Pakete ohne ein Anlegen von Regeln in den Switches weitergeleitet werden Die Applikation zählt bei jeder Regeln, wie oft diese genutzt wurde.

TABLE TYPE	ENTRY	DEVICE ID
WHITELISTED	TransportTupleACLEntry matching destAddress=10.0.0.4/32, srcAddress=10.0.0.37/32, srcPort=1410, destPort=-1, protocol=TCP	DevicePortList{device='of:a4e700000000001a', ports=[3]}; DevicePortList{device='of:643a00000000001b', ports=[14]};

Abbildung 5.2: Eintrag der Zugriffskontrolllistenapplikation

5.2.2 Berichtssammlung

Die Applikation für die Berichtssammlung speichert alle Berichte der NADS in einer Datei. Die Berichtssammlung ermöglicht die zentrale Sammlung von Anomalie-Informationen im Fahrzeug. In Abbildung 5.3 sind drei Berichte für die Rückfahrkamera aufgelistet. Jeder Bericht hat einen Code, der die Berichtsart beschreibt und einen Zähler, wie oft diese Anomalie aufgetreten ist. Die 0 gibt an, dass keine Daten mehr empfangen werden, 1 ist eine unbekannte Anomalie. Der Code 3 ist für Heartbeat-Signale. Diese periodischen Signale geben an, ob eine Komponente noch funktionstüchtig ist. Heartbeat-Signale zählen nicht als Anomalien.

REPORT ID	CODE	COUNT
NADS1_VIDEOSTREAM	0	469
NADS1_VIDEOSTREAM	1	2
NADS1_VIDEOSTREAM	3	1401

Abbildung 5.3: Eintrag der Berichtssammlung

Diese Applikationen erweitert die vorhandene Aufnahme von Statistiken, die ONOS bereits für Weiterleitungsgeräte bietet. Wie in Abschnitt 3.3 erklärt, trägt jede relevante Aufnahme von Statistiken zu der Security des Fahrzeuges bei.

5.2.3 Statische Weiterleitung

Die Applikationen für statische Weiterleitung von Paketen, erlaubt das Einspielen von statischen Weiterleitungsregeln. Im Grunde genommen nutzt diese Applikation das Grundprinzip von OpenFlow (siehe Abschnitt 2.2.1), das mittels Fluss-Tabellen in Weiterleitungsgeräten, den Netzwerkverkehr vorgibt. Die statischen Flüsse werden in einer Datei angelegt und können dynamisch entfernt und hinzugefügt werden. Mit diesen Eigenschaften kommen einige Vorteile.
Beispielweise die Tatsache, dass die Netzwerkflüsse von einer zuständigen Person angelegt werden und dementsprechend verifiziert sind. Weiterleitungsregeln können sehr genau angelegt werden, was eine präzise Zuordnung von Datenpaketen ermöglicht. Abbildung 5.4 zeigt, wie Einträge von Weiterleitungsregeln in ONOS aussehen. Die Weiterleitungsregel in der Abbildung ordnet eingehende Pakete mit der Quell-MAC-Adresse 00:00:00:00:00:C4 und der Ziel-MAC-Adresse FF:00:1B:00:00:0E zu.

DEVICE ID	PACKETS	TIMEOUT	PRIORITY	ETH TYPE	INPORT	ETH MATCH	VLAN ID	VLAN PRIORITY	TREATMENT
of:643a00000000001b	2550926	0	40000	ETH_TYPE:0xeca1	IN_PORT:17	[ETH_SRC:00:00:00:00:00:C4, ETH_DST:FF:00:1B:00:00:0E]	VLAN_VID:5	VLAN_PCP:7	[OUTPUT:19, OUTPUT:14]

Abbildung 5.4: Eintrag einer statischen Weiterleitungsregel

Weiterleitungsregeln für sicherheitskritische Flüsse müssen nur einmalig angelegt werden. Durch die dynamische Einbindung von Weiterleitungsregeln kann auch ein ganzer Satz an Flüssen auf einmal gesetzt werden. Dies ermöglicht abhängig vom Zustand des Fahrzeuges einen bestimmten Satz von Flüssen zu setzen. Ein Beispiel wäre der Wechsel vom Vorwärtsgang in den Rückwärtsgang. Der Fluss von der Kamera zum Infotainment-System wäre dann nur im Rückwärtsgang aktiv. Solche Maßnahmen schränken den erlaubten Netzwerkverkehr auf die geringste Anzahl an zulässigen Flüssen ein. Diese Einschränkungen erschweren Angreifern zulässige, aber nicht genutzte Flüsse zu missbrauchen.

5.2.4 Quality of Service

In diesem Netzwerk wird QoS durch das VLAN-Feld Priority Code Point (PCP) gewährleistet. Das PCP-Feld wird in den Flüssen für die statische Weiterleitung berücksichtigt. Grundsätzlich werden Ethernet-Pakete mittels des PCP-Feldes priorisiert. Die Werte im PCP-Feld werden aufsteigend priorisiert. Eine Applikation für die Durchsetzung von QoS im Netzwerk ist wie in Abschnitt 3.1 erwähnt zwingend erforderlich. Die Lösung im Tischaufbau würde, aber auch ohne SDN-Controller funktionieren, da das PCP-Feld von den Switches bereits verwertet wird. Die Switches senden immer das höher priorisierte Paket als Erstes. Eine speziell auf QoS im Fahrzeug ausgelegte Applikation fehlt dem Tischaufbau. Die Arbeit von Nayak et al. beschreibt, wie ein SDN-Controller in Kombination mit TSN und Zeitslotvergabe genutzt werden kann, um QoS in einem Echtzeitsystem durchzusetzen [47].

5.2.5 Systemstart mit Applikationen des Tischaufbaus

Bei allen vorgestellten Applikationen stellt sich die Frage, wie lange ein Systemstart im Vergleich zu den Werten in Abbildung 4.15 dauert.

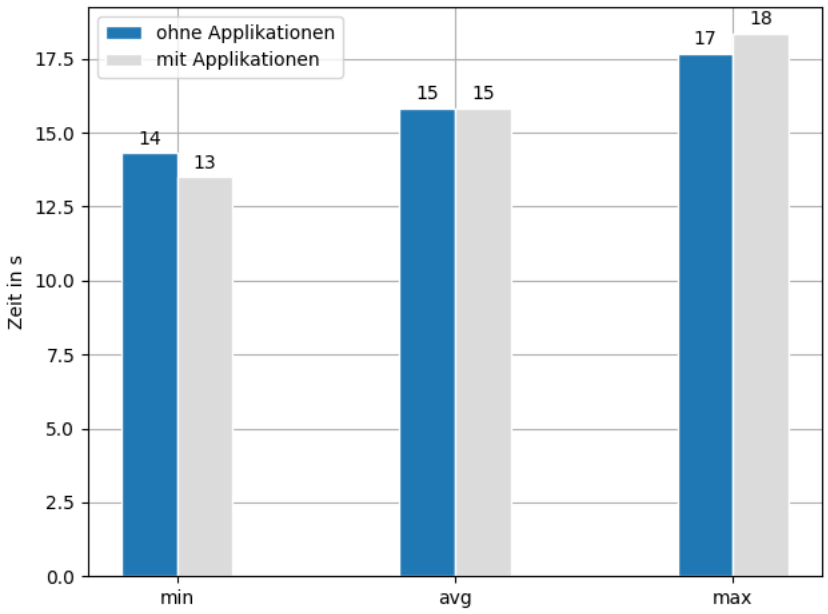

Abbildung 5.5: Systemstart mit SDN-Applikationen

Denn wie in Abschnitt 4.5 angemerkt, ist die Zeit des Systemstarts stark von der Implementierung der Applikationen abhängig. Mit drei zusätzlichen Applikationen ist zu erwarten, dass der Systemstart des ONOS SDN-Controllers etwas höher ist. Der Systemstart wird genau wie in Abschnitt 4.4 ausgelesen. Abbildung 5.5 zeigt, dass ONOS mit den zusätzlichen Applikationen nicht viel länger zur Betriebsbereitschaft braucht. Es hat auch keinen Unterschied gemacht, dass der ONOS SDN-Controller im Tischaufbau auf einem leistungsschwächeren Rechner läuft. Die Applikationen für den Tischaufbau verursachen nur eine geringe Erhöhung der Startzeit. Sofern keine Optimierung des Systemstartes ohne zusätzliche Applikationen stattfindet, braucht die Zeit zum Start der einzelnen hinzugefügten Applikationen nicht betrachtet zu werden.

5.3 Vergleich: Statische Weiterleitung und Automotive-Gateway

Im Tischaufbau werden die Applikationen für die statische Weiterleitung hauptsächlich genutzt, um einen CAN-Tunnel zu erstellen. Mittels des CAN-Tunnels werden die in Ethernet-Pakete umgewandelten CAN-Pakete der CAN-Ethernet-Gateways durch die Switches weitergeleitet. Diese Funktion wurde zusammen mit den CAN-Ethernet-Gateways getestet und in direkten Vergleich mit einem Automotive-Gateway gestellt.

5.3.1 Verzögerung des Automotive-Gateways

Der CAN-Tunnel wurde mit dem Automotive-Gateway verglichen, das ein zentrales Gateway der Firma Volkswagen ist. Die durch das Gateway erzeugte Verzögerung ist bereits in einer Arbeit von Patrick Kuncke untersucht worden [35]. Dieser hat unter anderem die Übertragungszeit von CAN-Paketen auf einem 500 Kbit/s schnellen CAN-Bus, mit zentralem Gateway gemessen und die theoretische Übertragungszeit ohne zentrales Gateway mit der Formel 5.1 berechnet.

$$T = N/V \tag{5.1}$$

wo:

T = Übertragungszeit
N = Paketgröße
V = Busgeschwindigkeit

Kuncke hat die minimale Übertragungszeit von Standard-CAN-Paketen ohne erweiterte CAN-ID zwischen zwei CAN-Endpunkten mit 94 μs und die maximale Übertragungszeit mit 260 μs angegeben. Dabei wurde der Maximalwert mit 8 Byte Daten erzeugt und der Minimalwert ohne Daten. Die Tests mit zentralem Gateway haben mit einem CAN-Paket mit 8 Byte Nutzdaten eine Übertragungszeit von 260 μs bis 520 μs ergeben. Die aus der Übertragungszeit berechnete Verzögerung durch das CAN-Paket wurde mit 58 μs bis 96 μs angegeben.

5.3.2 Testaufbau

Der Tischaufbau besitzt auf *core-laptop-2* in Abbildung 5.1 eine Möglichkeit, um die Übertragungszeit von CAN-Paketen zu messen. Für den Test wurde der aufgezeichnete Ablauf einer CAN-Kommunikation abgespielt. Es wird hierbei über den vorhandenen CAN-Bus an das CAN-Ethernet-Gateway ZC_FL gesendet, das die Pakete an alle weiteren CAN-Ethernet-Gateways weiterleitet. Die anderen CAN-Ethernet-Gateways leiten die Nachricht wieder an den Rechner weiter. CAN-Nachrichten können erweiterte CAN-IDs mit 29 Bit besitzen, im Gegensatz zum Standard mit 11 Bit. Beim Versenden von längeren monotonen Bit-Sequenzen kann es dazu kommen, dass Übertragungen auf dem CAN-Bus vom Empfänger fehlinterpretiert werden. Um eine Synchronisierung von den Kommunikationsteilnehmern auf dem CAN-Bus zu gewährleisten, werden Stopf-Bits im CAN-Paket mitgesendet. Die Stopf-Bits werden genutzt um einen Flankenwechsel bei längeren monotonen Bit-Sequenzen zu erzwingen. Diese machen aufgrund der maximalen Größe von 8 Byte Nutzdaten 19 Bit des CAN-Headers aus und stellen damit das Worst-Case-Szenario dar. Die CAN-Nachrichten in der Aufzeichnung besitzen keine erweiterte CAN-ID und sind zusammen mit den Stopf-Bits 130 Bit groß, was der maximalen Größe eines standard CAN-Paketes entspricht. Die CAN-Ethernet-Gateways erstellen mit solch einem CAN-Paket ein minimales Ethernet-Paket mit 64 Byte Größe. Die Messung beginnt unmittelbar nach der Versendung eines vollständigen CAN-Paketes vom Rechner aus und endet, sobald der Rechner das Paket vollständig wieder empfängt. Dieser Ablauf kann aus Abbildung 5.6 entnommen werden.

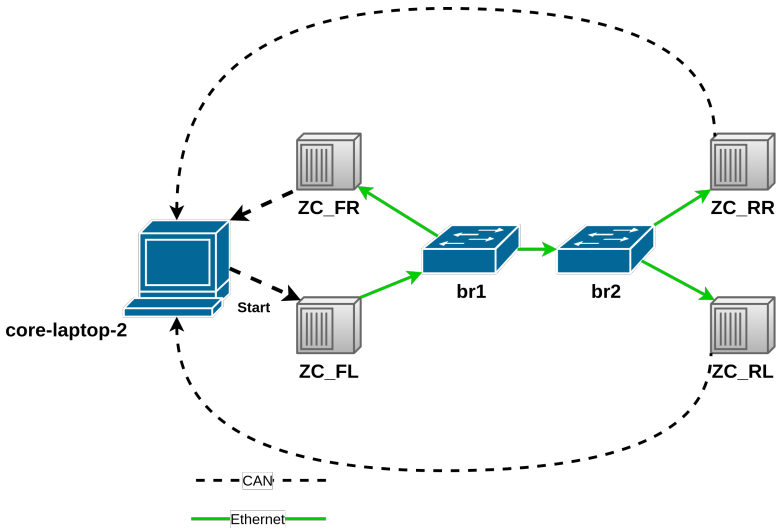

Abbildung 5.6: CAN-Tunnel Testablauf

Mittels CAN-IDs wird im CAN-Bus sichergestellt, dass die gesendeten Nachrichten priorisiert werden. Durch die Bits der CAN-IDs werden bei gleichzeitiger Übertragung, höher priorisierte CAN-Pakete über niedrig priorisierte CAN-Pakete gestellt. Niedrig priorisierte Nachrichten werden dementsprechend erst versendet, wenn die Übertragung höher priorisierter CAN-IDs beendet wurde. Die Arbitrierung auf dem CAN-Bus würde zu einer weiteren Verzögerung führen. Eine einzelne CAN-Nachricht auf dem CAN-Bus verhindert die mögliche Arbitrierung, die durch höher priorisierte Nachrichten bei einer Kollision entstehen könnte. Aus diesem Grund besitzt jedes versendete Paket im Test die gleiche CAN-ID.

5.3.3 Testergebnisse

Mit der Formel 5.2 wurde die Verzögerungszeit des CAN-Tunnels berechnet. Die Formel schließt die Übertragungszeit des CAN-Bus aus, somit wird nur die Verzögerung der CAN-Ethernet-Gateways und der Ethernet-Übertragung berücksichtigt. Die zusätzlichen Verzögerungszeiten im Rechner durch das Betriebssystem, den Treiber, oder anderen Komponenten können nur schwer oder gar nicht gemessen werden. Die Ergebnisse der Formel filtern diese Werte demzufolge nicht raus. Ohne diese Werte würden die Testergebnisse akkurater sein, dennoch sollte eine grobe Auswertung der Verzögerungszeit des CAN-Tunnels möglich sein.

$$G = L - (T \cdot 2) \tag{5.2}$$

G = Verzögerungszeit
L = gemessene Übertragungszeit
T = theoretische Bus-Übertragungszeit

Mit der Formel 5.1 wurde vorher die theoretische Zeit der Ethernet-Übertragung berechnet. Die theoretische Übertragungszeit eines minimalen Ethernet-Paketes mit 64 Byte, über einen 1 Gbit/s schnellen Ethernet-Bus dauert 0,512 μs.

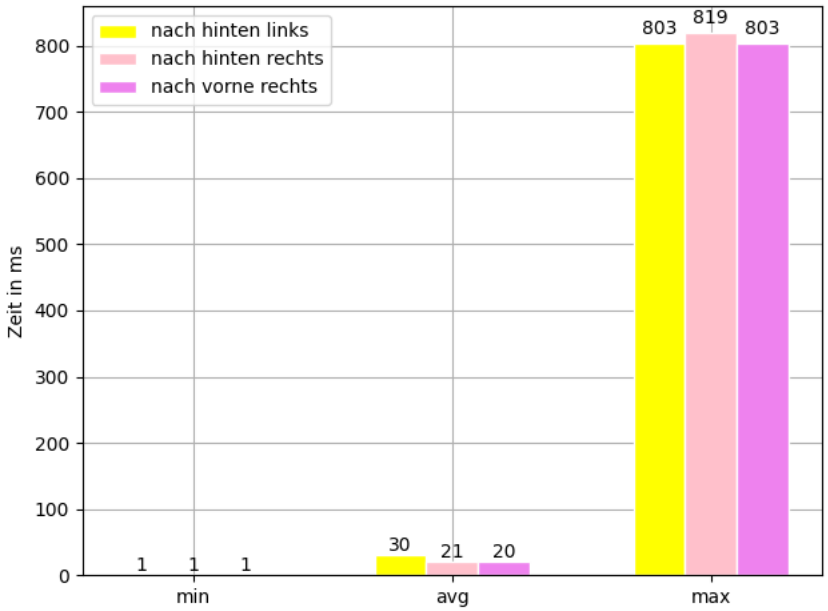

Abbildung 5.7: Mit Python-Skript aufgenommene Verzögerungszeit

Die Messergebnisse in Abbildung 5.7 sind selbst für die im Tischaufbau bestehenden Verhältnisse zu hoch. Ein Durchschnittswert von 17 ms ist nicht realistisch im Vergleich zur der theoretischen Übertragungszeit. Abgesehen davon haben die Ergebnisse wiedergegeben, dass einige Pakete auf dem Weg verloren gegangen sind. Die vorhandene Lösung auf dem Rechner ist ein Python-Skript. Python arbeitet auf einem hohen Software-Level und somit können Verzögerungen, Fehler und andere Probleme beim Empfang der Pakete nur schwer nachvollzogen werden. Die Messungen wurden deshalb wiederholt. Für die neuen

Messungen wurde ein C++-Programm entwickelt, das CAN-Treiber zum Verschicken und Empfangen von CAN-Paketen nutzt. Mit der direkten Nutzung von CAN-Treibern fallen implizite Verarbeitungsschritte weg, die beim Python-Skript zustande kommen. Die zusätzlichen Verzögerungszeiten durch das Betriebssystem und andere Hardware-nahe Komponenten bleiben trotzdem in den Testergebnissen erhalten.

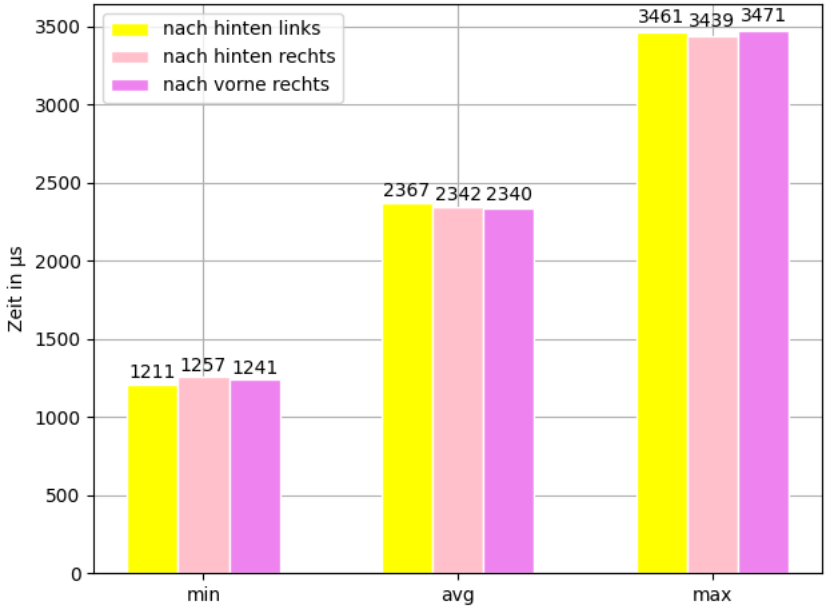

Abbildung 5.8: Mit C++-Programm aufgenommene Verzögerungszeit

Die Messergebnisse in Abbildung 5.8 mit dem C++-Programm zeigen deutlich geringere und konstantere Messergebnisse. Die Zeiten von vorne links in die anderen Zonen haben keine große Differenz. Es kann also angenommen werden, dass die Switches keinen großen Einfluss auf die Gateway-zu-Gateway Verzögerung haben. Die Verzögerungszeit durch den CAN-Tunnel beträgt durchschnittlich mehr als 2.300 μs. Darüber hinaus haben die Ergebnisse mit dem C++-Programm das Python-Skript als Fehlerquelle der ersten Messungen aufgedeckt. Während am Treiber alle Pakete ankommen, verliert das Python-Skript einige Ethernet-Pakete und erzeugt größere Verzögerung beim Empfang der Pakete.
Im Vergleich zum Automotive-Gateway ist die Verzögerung durch den CAN-Tunnel sehr hoch. Für die übliche End-zu-End Verzögerung im Fahrzeug würde dies bedeuten, dass durchschnittlich etwa 2,3 ms Verzögerung hinzukommen. Während die Verzögerung durch das Automotive-Gateway vernachlässigt werden kann, muss beim CAN-Tunnel abgewägt

werden. Die End-zu-End Verzögerungszeit darf mit der zusätzlichen Verzögerungszeit des CAN-Tunnels nicht die 150 ms der Multimediadaten übersteigen. Abhängig von den Daten ist die Toleranz noch geringer, siehe Tabelle 4.1.

Die Tests haben gezeigt, dass SDN mit den verwendeten statischen Flussregeln für die Weiterleitung von CAN-Paketen genutzt werden kann. Die gemessene Verzögerungszeit ist unter Umständen nicht mit den zeitlichen Anforderungen des Autos kompatibel. Lösungen wären die weitere Optimierung der Gateways sowie der Einsatz von TSN als Echtzeit-Ethernet Lösung.

5.4 Evaluation des Fallbeispiels

Mit dem statischen Forwarding und der Zugriffskontrollliste besitzt der Tischaufbau Applikationen, die in zukünftigen Fahrzeugen deutliche Vorteile mit sich bringen. Das statische Forwarding erlaubt präzise und strikte Flusskontrolle im Netzwerk, in der eine befugte Person den kompletten Netzwerkverkehr mittels Regeln festlegen kann. Des Weiteren können CAN-Netzwerkflüsse mit dem CAN-Tunnel durch den SDN-Controller gesteuert werden. Auch wenn dieser keine ähnliche Leistung wie ein Automotive-Gateway erbringt, wird trotzdem die Anforderung E1 erfüllt indem eine Kompatibilität mit einem bestehenden Fahrzeugkommunikationsprotokoll hergestellt wird. Die Berichtssammlung erweitert die bereits von ONOS erfüllte Anforderung der Telemetrie mit weiteren Statistiken.
Ins Besondere haben die Applikationen verdeutlicht, dass einige der beschriebenen Anforderungen aus Kapitel 3 teilweise oder vollständig durch Applikationen erfüllt werden können. Nichtfunktionale Anforderungen, wie eine Zertifizierung sind davon ausgenommen. Somit wären anhand der Anforderungen sinnvolle Erweiterungen für den Tischaufbau vorzusehen. Applikationen, die zeitbasierte Updates und Transaktionen ermöglichen wurden in Abschnitt 3.1 und Abschnitt 3.3 beschrieben. Eine wichtige Erweiterung zur Erhöhung der Security wäre eine Applikation, die Applikationen verifiziert. Die Security würde auch von einer Applikation profitieren, die Flüssen gemäß der Anforderung C3 Applikationen zuordnet. Gerade für Applikationen wie die statische Weiterleitung ist so eine Erweiterung kritisch, siehe Abschnitt 3.3. Abhängig von den Applikationen, kann die Startzeit des Controllers erhöht werden. Der Controller kann mit den vorgeschlagenen Applikationen weitere Anforderungen erfüllen. Es muss aber darauf geachtet werden, dass die Applikationen ähnlich wie im Tischaufbau keine starke Erhöhung der Startzeit

erzeugen.
Das Fallbeispiel hat verdeutlicht, welchen Mehrwert SDN-Applikationen bringen können. Ein Fahrzeugbordnetzwerk kann durch die SDN-Controller zentral überwacht und gesteuert werden. Durch Applikationen ist ein SDN-Controller in der Lage, genau für die Anforderungen in einem Fahrzeug angepasst zu werden. Bei jeder Applikation muss aber abgewägt werden, inwiefern diese das Startverhalten des Controllers beeinflusst.

6 Fazit und Ausblick

In diesem Kapitel werden die Ergebnisse der Arbeit zusammengefasst und eine abschließende Beurteilung abgegeben. Außerdem werden Optimierungen vorgeschlagen und Erweiterungen in Hinblick auf die Zukunft von SDN-Controller im Automotive-Bereich vorgestellt.

6.1 Fazit

Durch den dreistufigen Evaluationsprozess konnten mehrere Feststellungen gemacht werden. Die Ergebnisse der Anforderungsanalyse in Kapitel 3 haben wiedergegeben, dass ein Großteil der verfügbaren SDN-Controller nicht die entsprechenden Anforderungen für den Einbau in ein Fahrzeug erfüllen. Von mehr als dreißig untersuchten SDN-Controller haben nur vier aktuelle Software, angemessene Systemanforderungen und waren NETCONF-/YANG-fähig. Diese Controller sind der ONOS SDN-Controller von der ONF, der OpenDayLight SDN-Controller von der OpenDayLight Foundation, der Lumina SDN-Controller von Lumina Inc. und der Ryu SDN-Controller von der Nippon Telegraph and Telephone Corporation. Aber auch diesen Controller fehlen wichtige Eigenschaften für den Einbau in ein Fahrzeugbordnetzwerk.
Die Performanzanalyse in Kapitel 4 hat ergeben, dass von den vier Controllern der ONOS SDN-Controller der leistungsfähigste Controller ist. Von den sieben in Abschnitt 4.1 vorgestellten Controller-Metriken hat ONOS bei 4 Metriken die besten Durchschnittswerte erreicht.
Im Fallbeispiel hat sich gezeigt, was für eine Auswirkung Controller-Applikation haben können. Diese ermöglichen nicht nur eine erweiterte Steuerung des Fahrzeugbordnetzwerkes. Es sind auch Erweiterungen, die wichtige Anforderungen für SDN-Controller im Fahrzeug ergänzen können.

Auch wenn einige Anforderungen durch Applikationen erfüllt werden können, sind Anforderungen wie die Zertifizierung eines SDN-Controllers nicht durch Applikationen zu erfüllen. Bei diesen Anforderungen geht es im Grunde genommen um Anforderungen, die nur indirekt mit den Funktionen des SDN-Controllers zusammenhängen. Des Weiteren ist die Leistung der SDN-Controller im normalen Betriebszustand ausreichend, doch ein Ausfall oder Defekt ohne eine Rückfallstrategie würde in jedem Fall einen kritischen Zustand im Fahrzeug bedeuten. Eine Rückfallstrategie relativiert das Ausmaß eines SDN-Controller-Ausfalls erheblich. Ein Ausfall würde in diesem Fall jedes Mal einen Ausnahmezustand bedeuten. Es kann gesagt werden, dass ein sicherer Einbau von SDN-Controller in zukünftigen Fahrzeugen nur mit einer Rückfallstrategie und der Erfüllung aller wichtigen Anforderungen gewährleistet werden kann.
Die heutigen SDN-Controller besitzen unter passenden Umständen, genug Leistung für ein Fahrzeugbordnetzwerk, aber nicht die entsprechenden Anforderungen. Aus diesen Gründen sind die heutigen SDN-Controller noch nicht bereit für den Einbau in ein Fahrzeug.

6.2 Ausblick

In dieser Arbeit wurde ausschließlich betrachtet, was SDN-Controller bereits an Anforderungen erfüllen und was für eine Leistung diese für das Fahrzeug bringen.
Zukünftige Arbeiten könnten anhand der Ergebnisse dieser Arbeit weiterführende Themen behandeln. Es könnten Optimierungen für die Performanz von SDN-Controllern erarbeitet werden. Besonders die Ausfallsicherungszeit, war bei allen getesteten SDN-Controllern zu bemängeln.
Möglich wären auch Arbeiten, die Controller-Applikationen erstellen und untersuchen, die die Anforderungen aus der erarbeiteten Anforderungsliste erfüllen. Bei dieser Art von Forschung könnten Applikationen hervorkommen, die wertvolle Erweiterungen für SDN-Controller im Auto darstellen.

Literaturverzeichnis

[1] Abdelzaher, T. F. ; Atkins, E. M. ; Shin, K. G.: QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control. In: *IEEE Transactions on Computers* 49 (2000), November, Nr. 11, S. 1170–1183. – ISSN 0018-9340

[2] Adrichem, N. L. M. v. ; Asten, B. J. v. ; Kuipers, F. A.: Fast Recovery in Software-Defined Networks. In: *2014 Third European Workshop on Software Defined Networks*, 2014, S. 61–66

[3] Alvestrand, H.: A Mission Statement for the IETF / IETF. October 2004 (3935). – RFC

[4] Applied Research Center for Computer Networks: *RUNOS SDN/OpenFlow controller*. 2019. – URL http://arccn.github.io/runos/docs-2.0/eng/index.html

[5] AUTOSAR: SOME/IP Protocol Specification / AUTOSAR. November 2016 (696). – AUTOSAR Standard

[6] AUTOSAR: Specification of Diagnosticover IP / AUTOSAR. November 2017 (418). – AUTOSAR Standard

[7] Bhuvaneswaran, V. ; Basil, A. ; Tassinari, M. ; Manral, V. ; Banks, S.: Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance / IETF. October 2018 (8456). – RFC

[8] Bierman, A. ; Bjorklund, M. ; Watsen, K.: RESTCONF Protocol / IETF. January 2017 (8040). – RFC

[9] Bjorklund, M.: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) / IETF. October 2010 (6020). – RFC

[10] Blial, Othmane ; Ben Mamoun, Mouad ; Benaini, Redouane: An overview on SDN architectures with multiple controllers. In: *Journal of Computer Networks and Communications* 2016 (2016)

[11] Botelho, F. ; Bessani, A. ; Ramos, F. M. V. ; Ferreira, P.: On the Design of Practical Fault-Tolerant SDN Controllers. In: *2014 Third European Workshop on Software Defined Networks*, Sep. 2014, S. 73–78. – ISSN 2379-0369

[12] Brunner, Stefan ; Roder, Jurgen ; Kucera, Markus ; Waas, Thomas: Automotive E/E-architecture enhancements by usage of ethernet TSN. In: *2017 13th Workshop on Intelligent Solutions in Embedded Systems (WISES)* IEEE (Veranst.), 2017, S. 9–13

[13] Buttazzo, Giorgio C.: *Hard real-time computing systems: predictable scheduling algorithms and applications*. Bd. 24. Springer Science & Business Media, 2011

[14] Cooperation, MOST: *MOST: the automotive multimedia network*. Karlsruhe : Franzis Verlag GmbH, 2011. – ISBN 978-3-645-65061-8

[15] Cui, J. ; Zhou, S. ; Zhong, H. ; Xu, Y. ; Sha, K.: Transaction-Based Flow Rule Conflict Detection and Resolution in SDN. In: *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, July 2018, S. 1–9

[16] Enns, R. ; Bjorklund, M. ; Schoenwaelder, J. ; Bierman, A.: Network Configuration Protocol (NETCONF) / IETF. June 2011 (6241). – RFC

[17] Fielding, Roy T. ; Taylor, Richard N.: *Architectural styles and the design of network-based software architectures*. Bd. 7. University of California, Irvine Irvine, 2000

[18] Fussey, Peter ; Parisis, George: 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*. New York, NY, USA : ACM, 2017 (CarSys '17), S. 73–74. – URL http://doi.acm.org/10.1145/3131944.3131954. – ISBN 978-1-4503-5146-1

[19] IEEE 802.1 TSN Task Group: *IEEE 802.1 Time-Sensitive Networking Task Group*. – URL https://1.ieee802.org/tsn/

[20] IEEE 802.1 TSN Task Group: *IEEE P802.1Qcp – Bridges and Bridged Networks Amendment: YANG Data Model.* – URL https://1.ieee802.org/tsn/802-1qcp/

[21] IEEE 802.1 TSN Task Group: *IEEE P802.1Qcw YANG Data Models for Scheduled Traffic, Frame Preemption, and Per-Stream Filtering and Policing.* – URL https://1.ieee802.org/tsn/802-1qcw/

[22] IEEE Standards for Local and Metropolitan Area Networks / Institute of Electrical and Electronics Engineers. Geneva, CH, Oktober 1995. – Standard

[23] Institute of Electrical and Electronics Engineers: *802.3ae-2002 - IEEE Standard for Information technology - Local and metropolitan area networks - Part 3: CSMA/CD Access Method and Physical Layer Specifications - Media Access Control (MAC) Parameters, Physical Layer, and Management Parameters for 10 Gb/s Operation*

[24] Internet Technologies Research Group: *Security for Vehicular Information.* – URL https://secvi.inet.haw-hamburg.de/. – Zugriffsdatum: 2020-06-20

[25] International Organization for Standardization: Road vehicles — Controller area network (CAN) / International Organization for Standardization. Geneva, CH, Dezember 2015 (ISO 11898). – Standard

[26] International Organization for Standardization: Road vehicles — FlexRay communications system / International Organization for Standardization. Geneva, CH, Februar 2013 (ISO 17458). – Standard

[27] International Organization for Standardization: Road vehicles — Local Interconnect Network (LIN) / International Organization for Standardization. Geneva, CH, August 2016 (ISO 17987). – Standard

[28] International Organization for Standardization: Road vehicles — Functional safety / International Organization for Standardization. Geneva, CH, Dezember 2018 (ISO 26262). – Standard

[29] International Organization for Standardization: Road vehicles — Cybersecurity engineering / International Organization for Standardization. Geneva, CH, 2020 (ISO/SAE DIS 21434). – Standard

[30] Jawaharan, R. ; Mohan, P. M. ; Das, T. ; Gurusamy, M.: Empirical Evaluation of SDN Controllers Using Mininet/Wireshark and Comparison with Cbench. In: *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, 2018, S. 1–2

[31] Kopetz, Hermann: *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011

[32] Koscher, K. ; Czeskis, A. ; Roesner, F. ; Patel, S. ; Kohno, T. ; Checkoway, S. ; McCoy, D. ; Kantor, B. ; Anderson, D. ; Shacham, H. ; Savage, S.: Experimental Security Analysis of a Modern Automobile. In: *2010 IEEE Symposium on Security and Privacy*, May 2010, S. 447–462

[33] Kreutz, D. ; Ramos, F. M. V. ; Veríssimo, P. E. ; Rothenberg, C. E. ; Azodolmolky, S. ; Uhlig, S.: Software-Defined Networking: A Comprehensive Survey. In: *Proceedings of the IEEE* 103 (2015), Januar, Nr. 1, S. 14–76. – ISSN 0018-9219

[34] Kreutz, Diego ; Ramos, Fernando ; Veríssimo, Paulo: Towards secure and dependable software-defined networks, 08 2013, S. 55–60. – ISBN 9781450321785

[35] Kuncke, Patrick: *Erprobung von Echtzeit Ethernet basierten Automobil-Gateways in einem Prototypfahrzeug*. Hamburg, Hochschule für Angewandte Wissenschaften Hamburg, bachelorthesis, April 2016

[36] Laissaoui, C ; Idboufker, N ; Elassali, R ; Baamrani, K: A measurement of the response times of various OpenFlow/SDN controllers with CBench. (2015), S. 1–2

[37] Lim, Hyung-Taek ; Weckemann, Kay ; Herrscher, Daniel: Performance Study of an In-Car Switched Ethernet Network without Prioritization, 03 2011, S. 165–175

[38] Lumina Networks, Inc.: *Lumina SDN Controller*. 2014. – URL https://www.luminanetworks.com/. – Zugriffsdatum: 2019-12-27

[39] Mamushiane, L ; Lysko, A ; Dlamini, S: A comparative evaluation of the performance of popular SDN controllers. In: *Wireless Days (WD)* (2018. 2018), S. 54–59

[40] McKeown, Nick ; Anderson, Tom ; Balakrishnan, Hari ; Parulkar, Guru ; Peterson, Larry ; Rexford, Jennifer ; Shenker, Scott ; Turner, Jonathan: OpenFlow: Enabling Innovation in Campus Networks. In: *ACM SIGCOMM Computer Communication Review* 38 (2008), Nr. 2, S. 69–74

[41] Medved, J. ; Varga, R. ; Tkacik, A. ; Gray, K.: OpenDaylight: Towards a Model-Driven SDN Controller architecture. In: *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014, S. 1–6

[42] Mizrahi, T. ; Moses, Y.: Software defined networks: It's about time. In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, URL https://ieeexplore.ieee.org/abstract/document/7524418. – Zugriffsdatum: 2020-04-04, 2016, S. 1–9

[43] Mizrahi, T. ; Moses, Y.: Time Capability in NETCONF / IETF. February 2016 (7758). – RFC

[44] Mizrahi, T. ; Saat, E. ; Moses, Y.: Timed Consistent Network Updates in Software-Defined Networks. In: *IEEE/ACM Transactions on Networking* 24 (2016), Nr. 6, S. 3412–3425. – URL https://ieeexplore.ieee.org/document/7423783. – Zugriffsdatum: 2020-04-04

[45] Mizrahi, Tal ; Moses, Yoram: Time-based Updates in OpenFlow : A Proposed Extension to the OpenFlow Protocol, 2013

[46] Navet, N. ; Song, Y. ; Simonot-Lion, F. ; Wilwert, C.: Trends in Automotive Communication Systems. In: *Proceedings of the IEEE* 93 (2005), June, Nr. 6, S. 1204–1223. – ISSN 1558-2256

[47] Nayak, Naresh G. ; Dürr, Frank ; Rothermel, Kurt: Time-Sensitive Software-Defined Network (TSSDN) for Real-Time Applications. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. New York, NY, USA : Association for Computing Machinery, 2016 (RTNS '16), S. 193–202. – URL https://doi.org/10.1145/2997465.2997487. – ISBN 9781450347877

[48] Nehra, A. ; Tripathi, M. ; Gaur, M. S.: Requirement analysis for abstracting security in software defined network. In: *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, July 2017, S. 1–8

[49] Oktian, Y. E. ; SangGon Lee ; HoonJae Lee ; JunHuy Lam: Secure your Northbound SDN API. In: *2015 Seventh International Conference on Ubiquitous and Future Networks*, July 2015, S. 919–920. – ISSN 2165-8536

[50] Ongaro, Diego ; Ousterhout, John: In Search of an Understandable Consensus Algorithm. In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA : USENIX Association, Juni 2014, S. 305–319. – URL https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro. – ISBN 978-1-931971-10-2

[51] Open Network Foundation: *The Open Network Foundation*. – URL https://www.opennetworking.org/mission/. – Zugriffsdatum: 2019-12-27

[52] Open Network Foundation: *Open Network Operating System*. 2014. – URL https://www.opennetworking.org/onos/. – Zugriffsdatum: 2019-12-27

[53] Open Networking Foundation: *Atomix, A reactive Java framework for building fault-tolerant distributed systems*. – URL https://atomix.io/docs/latest/user-manual/. – Zugriffsdatum: 2020-05-10

[54] Open Networking Foundation: OpenFlow Switch Specification (ONF TS-006). – Spezifikation

[55] OpenDaylight Project a Series of LF Projects, LLC: *OpenDayLight Project*. 2018. – URL https://www.opendaylight.org/. – Zugriffsdatum: 2019-12-27

[56] Ryu SDN Framework Community: *RYU the Network Operating System(NOS)*. 2014. – URL https://ryu.readthedocs.io/en/latest/. – Zugriffsdatum: 2019-12-27

[57] Salman, O. ; Elhajj, I. H. ; Kayssi, A. ; Chehab, A.: SDN controllers: A comparative study. In: *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, 2016, S. 1–6

[58] Sherwood, R ; Yap, K: In: *Cbench controller benchmarker*

[59] Stancu, A ; Halunga, S ; Vulpe, A ; Suciu, G ; Fratu, O ; Popovici, E: A comparison between several Software Defined Networking controllers. (2015. 2015), S. 223–226

[60] Stefan Buntz, David B.: IEEE 100BASE-T1 System Implementation Specification / OPEN ALLIANCE SIG (Version 1.0). – Spezifikation

[61] Steinbach, Till: *Ethernet-basierte Fahrzeugnetzwerkarchitekturen für zukünftige Echtzeitsysteme im Automobil*. Wiesbaden : Springer Vieweg, Oktober 2018. – ISBN 978-3-658-23499-7

[62] The Faucet Organisation: *Faucet*. 2014. – URL https://faucet.nz/. – Zugriffsdatum: 2019-12-27

[63] Tootoonchian, Amin ; Ganjali, Yashar: HyperFlow: A Distributed Control Plane for OpenFlow. In: *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. USA : USENIX Association, 2010 (INM/WREN'10), S. 3

[64] Waszecki, P. ; Mundhenk, P. ; Steinhorst, S. ; Lukasiewycz, M. ; Karri, R. ; Chakraborty, S.: Automotive Electrical and Electronic Architecture Security via Distributed In-Vehicle Traffic Monitoring. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36 (2017), November, Nr. 11, S. 1790–1803. – ISSN 0278-0070

[65] Wickboldt, J. A. ; De Jesus, W. P. ; Isolani, P. H. ; Both, C. B. ; Rochol, J. ; Granville, L. Z.: Software-defined networking: management requirements and challenges. In: *IEEE Communications Magazine* 53 (2015), Nr. 1, S. 278–285

[66] Wireshark Foundation: *Wireshark*. – URL https://www.wireshark.org/. – Zugriffsdatum: 2020-01-14

[67] Zimmermann, Werner ; Schmidgall, Ralf: *Bussysteme in der Fahrzeugtechnik*. Springer, 2006. – ISBN 978-3-658-02419-2

A Anhang

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Performanzanalyse von SDN-Controllern für Ethernet-basierte Kommunikationsarchitekturen im Fahrzeug

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original