Bachelorarbeit

Erweiterung eines Netzwerkmanager zur automatischen Konfiguration der sich im Fahrzeug befindenden Netzwerkgeräte anhand verschiedener Applikationsparameter

Eingereicht von:	Brunthaler Sebastian
	Matrikelnummer: 3032075
	Studiengang: Allgemeine Informatik
Durchgeführt bei:	Continental Automotive GmbH
	Siemensstr. 12, D-93055 Regensburg
	Betreuer: Christian Humig
Hochschule:	Ostbayerische Technische Hochschule Regensburg
	Erstprüfer: Prof. Dr. Thomas Waas
	Zweitprüfer: Prof. Dr. Markus Kucera

Danksagung

Mein Dank gilt der Abteilung Cross Divisional Systems & Technologyfür bei der Continental Automotive GmbH für die Möglichkeit ein solch interssantes Thema in meiner Abschlussarbeit bearbeiten zu dürfen. Besonders möchte ich mich noch für die Hilfe und Unterstützung bei der Ausarbeitung des Themas bei folgen Personen bedanken:

- Prof. Dr. Thomas Waas, welcher diese Abschlussarbeit seitens der Hochschule betreut hat und mich bei sämtlichen Fragen fachlicher und organisatorischer Art unterstützt hat.
- Christian Humig, der meine Bachelorarbeit seitig Continental Automotive GmbH betreut hat, und mir das notwendige Vertrauen entgegen gebracht hat.
- Allen Arbeitskollegen der Abteilung, ganz besonders Julian Brand und Maximilian Brückl, die mir mit Ihren fundierten Fachkentnissen aus dem Bereich automobile Netzwerke jederzeit zur Seite stehen.
- Meinen Eltern die mich in der ganzen Zeit des Studiums unterstützt habe und durch ihr Vertrauen in mich dies erst ermöglicht haben.
- Allen nicht namentlich genannten Personen, in jeglicher Form an der Arbeit beteiligt waren, sowie denen, die diese Abhandlung liest und einen Nutzen daraus zieht.

Sperrvermerk

In dieser Bachelorarbeit sind Geschäftsgeheimnisse der Continental Automotive GmbH enthalten. Eine Weitergabe oder Vervielfältigung oder Veröffentlichung der Arbeit sowie die Verwertung und Mitteilung ihres Inhaltes ist ohne ausdrückliche schriftliche Genehmigung der Continental Automotive GmbH, Siemensstraße 12, 93055 Regensburg nicht gestattet.

Inhaltsverzeichnis

Glossar

AVB Audio Video Bridging

BMC Best Master Clock

CAN Controller Area Network

CBS Credit Based Shaper

ECU Electronic Control Unit

gPTP generalized Precision Timing Protocol

ID Identifikationsnummer

IEEE Institute of Electrical and Electronics Engineers

ISO International Organization for Standardization

JSON JavaScript Object Notation

LIN Local Interconnect Network

MAC Media Access Control

MACsec Media Access Control Security

OSI Open Systems Interconnection

PSFP Per-Stream Filtering and Policing

PTP Precision Time Protocol

QoS Quality of Service

REST Representational State Transfer

SDN Software-defined networking

TAS Time Aware Shaper

TSN Time-Sensitive Networking

XML Extensible Markup Language

Kapitel

1

Einleitung

1.1 Motivation der Arbeit

Moderne Fahrzeuge wandeln sich immer mehr zum komplexen Netzwerken hin. Durch die Globalisierung und Urbanisierung, wächst das Verkehrsaufkommen weltweit schnell an. Mit diesem Mehraufwand an Fahrzeugen wird das aktuelle Verkehrsnetz an seine Kapazitätsgrenze kommen. Laut einer aktuellen Prognose, sollen im Jahr 2050 bereits 70 Prozent der Weltbevölkerung in Städten leben. Dementsprechenden wird die Zahl der Automobile in diesen Ballungsgebieten stark zunehmen [6].

Die geplante Automatisierung und Vernetzung von Fahrzeugen wird in diesen Gebieten ein große Herausforderung darstellen. Sie erschließen aber auch neue Möglichkeiten, die das Autofahren effizienter, sicherer und umweltfreundlicher machen sollen. Da deutsche Automobilhersteller und Zulieferer in diesen Bereichen bereits seit mehre Jahre tätig sind, wollen dies ihre Innovationsführerschaft weiter ausbauen. So verkündete der deutsche Automobilkonzern BMW und der chinesische Automobilhersteller Great Wall ihre Zusammenarbeit bei der Entwicklung selbstfahrender Autos und der Entwicklung gemeinsamer Standards [3].

Auch die Bundesrepublik Deutschland sieht sich in einer Vorreiter Rolle zu den Thema autonomes Fahren und hat sich, wie mehrfach in den Medien bekannt gegeben, hierzu einige Strategien überlegt. So soll in den nächsten Jahren ein Eingliederung des autonomen Fahren in den Straßenverkehr schrittweise erfolgen. Zu diesem Thema hat das Bundesministerium für Verkehr und digitale Infrastruktur, Handlungsempfehlungen zu Probebetrieb, Entwicklung zur Serienreife und Zulassungen erstellt [2].

Assistenzsysteme, die den Fahrer bei der Führung eins Automobils unterstützen, sind heute bereits der Regelfall in modernen Fahrzeugen. Die Anzahl dieser Systeme soll in den nächsten Jahren weiter ansteigen, bis diese ein vollständig autonomes Fahren eines Fahrzeuges ermöglichen. Das autonome Fahren wird sukzessive weiterentwickelt und

mehr und mehr in die nächsten Generationen von Autos integriert werden. So wird sich das selbstständige Fahren Schritt für Schritt in den Straßenverkehr etablieren.

Der Plan ist es das automatisierte Fahren zuerst auf Autobahnen und abgeschlossenen Bereichen, wie Parkanlagen, zu testen. Letzteres bietet sich an, da hier geringe Geschwindigkeiten auf komplexe Situationen aus dem Straßenverkehr treffen. Hierbei muss ein Fahrzeug sein komplettes Umfeld selbstständig mit den erwähnten Assistenzsystemen überwachen und jede Situation damit fehlerfrei beherrschen. Im Bereich autonomes Fahren ist kein Spielraum für Fehler, es müssen noch einige Innovationen erprobt werden, um eine Gefährdung für anders Verkehrsteilnehmer zur Gänze ausschließen zu können. Die Bundesregierung hat daher das Digitale Testfeld Autobahn ins Leben gerufen, welches zukünftig noch um städtische und ländliche Testfelder erweitert werden soll. Hier sollen ausgiebige Testphasen umgesetzt werden, um mögliche Gefahren frühzeitig zu erkennen und zu eliminieren [1].

1.2 Forschungsprojekt A3F

Das Projekt A3F wurde im Jahr 2016 von der Continental Automotive GmbH und der OTH Regensburg gegründet. Das Hauptforschungsthema liegt in der Entwicklung einer ausfallsichere Architekturen für autonome Fahrzeuge. Anhand von Schlüsselszenarien sollen die Herausforderungen beim Übergang hin zu einer neuen flexiblen und dynamischen Fahrzeugarchitektur herausgearbeitet werden. Hierbei geht es darum, die Anforderungen auf Applikationsebene zu verstehen, um daraus dann die Technologien und Lösungskonzepte für das Netzwerk im Fahrzeug der Zukunft abzuleiten. Ziel ist es, unter dem Gesichtspunkt der Wiederverwendung vorhandener Methoden und Technologien aus der Enterprise IT zu beleuchten und diese hinsichtlich eines möglichen Einsatzes im Fahrzeug zu bewerten. Hierzu müssen diese Methoden und Technologien neben den funktionalen Eigenschaften auch noch in Hinblick auf die speziellen automotive Anforderungen betrachtet werden. Darüber hinaus sollen erste Ansätze zu einem möglichen Migrationspfad, von bestehenden zu zukünftigen, an automotiven Anforderungen angepasste Lösungen aufgezeigt werden. Am Ende des Projekts sollen die gewonnenen Ergebnisse anhand eines Demonstrators gezeigt werden. Daraus lassen sich folgende drei globale Ziele definieren:

- Identifizierung von Schlüsseltechnologien und Methoden für zukünftige Fahrzeug-Architekturen
- Daraus Empfehlungen für zukünftige Kommunikationsnetzwerke im Fahrzeug ableiten
- Transfer von potentiellen IT-Technologien ins Fahrzeug unter Berücksichtigung von automotive Anforderungen

1.3 Aktuelle Problemstellung im Fahrzeugnetzwerk

Jede neue Anforderung an ein Fahrzeug hat aktuell eine Anpassung des Fahrzeugnetzwerks zur Folge. Dies will man in Zukunft vermeiden, indem das Netzwerk so dimensioniert wird, dass es durch beliebige Anwendungen erweitert werden kann. Außerdem muss es mit der Zunahme an Datenmengen zurecht kommen, da beispielsweise das autonome Fahren einen enormen Zuwachs an Kommunikationsdaten benötigt.

Mit den gestiegenen Anforderungen an Bandbreite, Zeitrelevanz und Skalierbarkeit ist das aktuelle Fahrzeugnetzwerk nicht mehr ausreichend dimensioniert und muss für diese Zwecke überarbeitet werden.

1.4 Ziel dieser Arbeit

Wie bereits erwähnt ist das Ziel ein Fahrzeugnetzwerk zu entwickeln, welches durch funktionelle Applikationen erweitert werden kann. Hierzu sollte das Konzept von Plug and Play verwendet werden, d.h. verschiedene Anwendungen müssen aus einem zentralen Anwendungsverzeichnis geladen werden können und ohne manuelles Eingreifen in das Fahrzeugnetzwerk lauffähig sein.

Die automatische Konfiguration soll durch ein Applikationsmanifest ermöglicht werden. Es gibt die Randbedingungen an, welche zum reibungslosen Betrieb der Anwendung benötigt werden. Das Manifest soll bereits vom Entwickler der Anwendung, anhand der spezifischen Anforderung an die Datenübertragung im Netzwerk eines Fahrzeuges, festgelegt werden. Da viel der neu entwickelten Anwendungen nicht die gleichen Ansprüche an das Netzwerk stellen, ist es wichtig bereits bei der Entwicklung an die nötigten Parameter für die Anwendung zu denken. Basierend auf diesen Parametern, wird eine bereits bestehende zentrale Netzwerkkontrolleinheit, der sog. Netzwerkmanager, funktionell weiterentwickelt.

Dieser soll in Zusammenarbeit mit weiteren Komponenten im System eine automatische Konfiguration des Netzwerkes vornehmen.

Die Veranschaulichung des Anwendungsfalles der Arbeit ist in Abb.1 dargestellt. Diese zeigt den Netzwerkmanager, hier zur Vereinfachung als Blackbox dargestellt. Anhand der Eingabeparameter aus dem Manifest allein können noch keine Netzwerkkonfigurationen erstellt werden. Der Manager muss auch über die Eigenschaften der Netzwerkteilnehmer Bescheid wissen. Nicht jedes Gerät kann die geforderten Bedingungen mit den dafür erforderlichen Technologien umsetzen. Aus diesem Grund sendet jede beteiligte Komponente seine Eigenschaften ebenfalls an den Netzwerkmanager. Dieser wertet sein Input automatisch aus und erstellt eine Konfiguration für alle Teilnehmer der Kommunikationsstrecke im Fahrzeugnetzwerk.

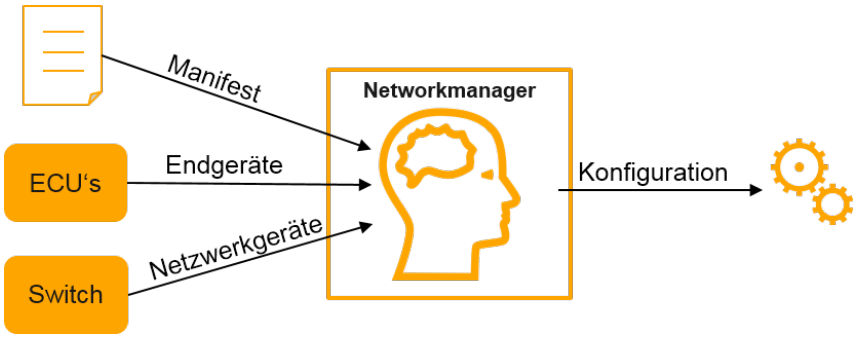

Abbildung 1: Vereinfachte Darstellung der Netzwerkkonfiguration

Im Laufe der Arbeit werden Endgeräte als Electronic Control Unit (ECU) bezeichnet. Netzwerkgeräte sind Switche und andere vermittelnde Komponenten im System.

Die Ausarbeitung des Themas erfolgt nicht nur theoretisch, sondern wird teilweise praktisch umgesetzt. Um dies zu ermöglichen müssen mehrere verschiedene Features implementiert werden. Der Praktische Teil beschränkt sich jedoch nur auf ein Fallbeispiel, da dies sonst den zeitlichen Rahmen der Arbeit sprengen würde. Anhand des Testfalles soll die Funktionalität der automatischen Konfiguration mittels eines Applikationsmanifestes belegt werden.

Kapitel

2

Grundlagen zu Echtzeitsystemen und Netzwerken in Fahrzeugen

Dieses Kapitel beschreibt die Grundlagen im Bereich Netzwerktechnik für Fahrzeuge. Im Punkt Echtzeitsysteme stützt sich diese Arbeit auf die Normen der Projektgruppe IEEE 802. Die Analyse dieser Arbeit wird auf den definierten Standardisierungen basieren. Dies umfasst die Definition der verwendeten Begriffe und die Erklärung von Ethernet-Grundlagen. Im Anschluss erfolgt die Erläuterung von Echtzeitsystemen auf Basis der Time-Sensitive Networking (TSN)-Protokolle.

2.1 Ethernet Grundlagen

In den folgenden Unterkapiteln wird kurz auf Ethernet spezifische Grundlagen eingegangen, welche im Laufe dieser Arbeit relevant werden. Es wird keine vollständige Definition von Ethernet erfolgen. Diese kann bei Bedarf in den zusätzlich angegeben Quellen recherchiert werden. Hier wird lediglich auf die Grundlagen und die für die Arbeit relevanten Eigenschaften von Ethernet eingegangen.

2.1.1 Ethernet-Frame

In Abbildung 2 wird ein kompletter Ethernet-Frame auf Layer 2 des ISO/OSI-Modell grafisch dargestellt. Das ISO/OSI-Modell ist ein standardisiertes Referenzmodell für Netzwerkprotokolle. Es ist in sieben verschiedenen Schichten gegliedert. Dies kann detailliert unter Kapitel 1.4 im Werk von A. Tanenbaum nachgelesen werden [25].
Ein Ethernet Paket ist in folgende Blöcke unterteilt:

- MAC-Adressen:
  - Destination Address: Gibt die Zieladresse des Paketes an
  - Source Address: Gibt die Adresse des Versenders an
- Ethertype: Gibt den Typen des Kommunikationsprotokolls an [13]
- Payload: In diesem Block befinden sich die zu versendenden Daten
- CRC Checksum: Prüfung des übertragenen Frames auf Fehler, wie beispielsweise Bit-Fehler

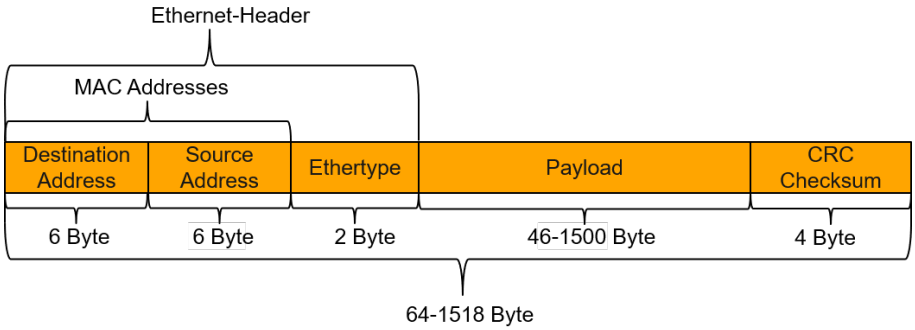

Abbildung 2: Ethernet Frame auf Layer 2 Ebene

Die ersten drei Blöcke bilden zusammen den Ethernet-Header eines Pakets. Dieser wird vom weiterleitenden Switch oder Empfänger als erstes gelesen.

2.1.2 Media Access Control Security (MACsec)

MACsec ist ein Mechanismus, der die Sicherheit einer Punkt-zu-Punkt-Ethernet-Verbindung ermöglicht. Es handelt sich um einen Mechanismus, der sich auf Layer 2 im ISO/OSI Modell befindet. Wenn MACsec verwendet wird, muss der Ethernetframe abgeändert werden.
Grundsätzlich werden in einer Ende-zu-Ende Kommunikation die Sicherheitsschlüssel der beiden Teilnehmer überprüft. Nur wenn diese übereinstimmen wird die Kommunikation erlaubt. Wenn MACsec aktiv ist, wird nicht nur die Kommunikation selbst geschützt, sondern es findet auch eine Integritätsprüfung der gesendeten Daten durch das Feldes *Integrity Check Value* statt. Diese enthält eine Prüfsumme, die über die geschützten Daten gebildet wird und sich ändert, falls diese verfälscht werden. Die Überprüfung der generellen Verbindung wird anhand der Daten im Header des Frames durchgeführt.

Besonders wichtig für das Prinzip von MACsec ist hierbei der sog. *SecTAG* [24], welcher in Abb. 3 dargestellt wird. Dieser Bereich unterteilt sich nochmals in fünf Blöcke, anhand derer verschiedene Informationen und Sicherheitsmechanismen definiert sind [24].

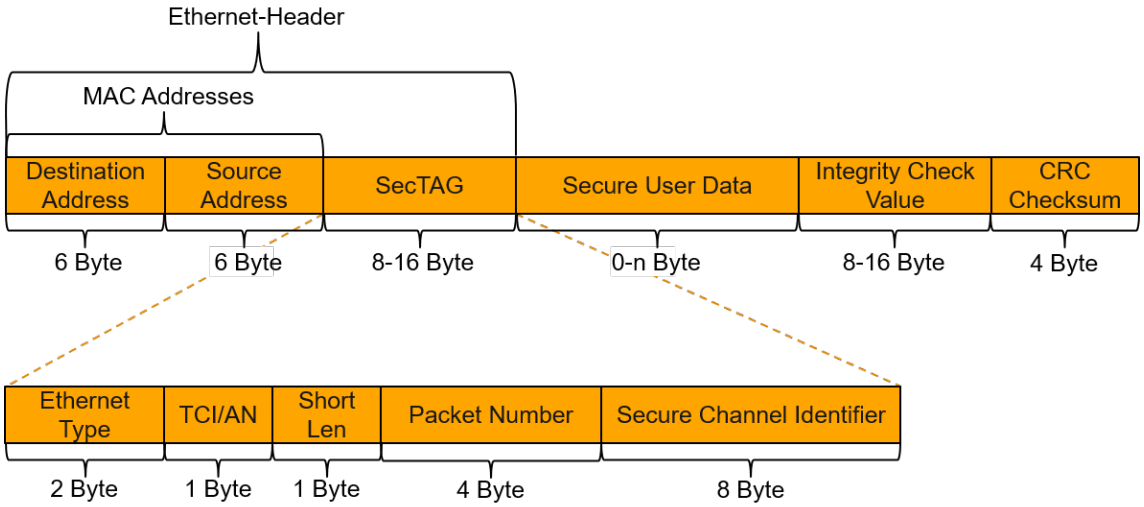

Abbildung 3: Ethernet Frame mit MACsec

Diese Auflistung beschreibt kurz die Bedeutung der einzelnen Blöcke aus dem Segment *SecTAG*:

- Ethernet Type: Gibt den Typen des Kommunikationsprotokolls an [13]
- TCI/AN:
  - TCI: Tag Control Information. In diesem Block ist unter anderem die Versionsnummer des MACsec-Protokoll enthalten.
  - AN: Association Number identifiziert bis zu vier sichere Verbindungen über einen gesicherten Kanal.
- Short Length: Ganzzahliger Wert, der die Anzahl der Oktette zwischen dem letztem Oktett von SecTAG und dem ersten Oktett von der Integrity Check Value angibt.
- Paket Number: Eindeutige Nummer des gesendeten Pakets.
- Secure Channel Identifier: Anhand dieser ID kann der sichere Kanal identifiziert werden.

2.2 Definition und Begriffe aus dem Netzwerkbereich

Im folgenden Unterkapitel werden Begriffe und Definitionen erläutert, welche in den weiteren Kapiteln relevant sind. Es wird auf deterministische Echtzeitfähigkeit, TSN im Hinblick auf den Automobil-Bereich und Quality of Service (QoS) Parameter mit deren Einflüsse und Ethernet in Fahrzeugen eingegangen. Was QoS für ein Netzwerk bedeutet, wird im nächsten Unterkapitel erläutert.

2.2.1 QoS-Parameter

Innerhalb eines Netzwerkes werden verschieden Dienste angeboten, welche durch ihre benötigten Parameter definiert werden können. Die Parametrisierung von Diensten ist im Standard ISO/IEC 13236 als Quality of Service (QoS) definiert und bezeichnet die Dienstgüte. Die vorhandenen Netzwerkdienste können anhand ihrer Güte spezifiziert und klassifiziert werden. Im Werk Computer Networks von A. Tannenbaum wird QoS wie folgt beschrieben:

> *"Die Qualität eines Services ist die Bezeichnung für die Mechanismen die die unterschiedlichen Anforderungen einr Anwendung miteinander vereinbaren."*

Im Wesentlichen ergeben sich folgende vier Hauptparameter, Latenz, Jitter, Paketverlust und Bandbreite, welche im weiteren Verlauf näher erläutert werden [37].

2.2.2 Deterministisches Fahrzeugnetzwerk

Bei den zunehmend komplexeren Aufgaben eines Kommunikationssystems in einem Fahrzeug wird es immer schwieriger die Echtzeitfähigkeit eines Systems zu garantieren. Daten die mit der Anforderung an Echtzeit versendet werden, bezeichnet man auch als kritische/zeitkritische Daten. Diese müssen innerhalb einer vorgegebenen maximalen Zeitverzögerung beim Empfänger zur Verfügung sehen. Durch die in Unterabschnitt 2.2.1 definierten QoS-Parameter, können Garantien für die Verfügbarkeit von kritischen Daten getroffen werden. Die Haupteigenschaften eines deterministischen Ethernet-Systems sind:

- Vorhersagbarkeit von Ereignissen im Netzwerk
- Berechenbarkeit des Systems
- Konsistente Antwortzeiten zwischen den Endverbrauchern

Der Netzwerk-Determinismus ermöglicht es, Daten garantiert in einem definierten Zeitfenster zwischen Endsystemen auszutauschen. Um eine deterministische Übertragung sicherzustellen, muss die Beeinflussung durch nicht planbare und nicht vorhersagbare Ereignisse im Netzwerk verhindert werden [5].

2.2.3 Latenz

Die Latenz beschreibt in erster Linie die Performance eines Systems. In dieser Arbeit wird hauptsächlich auf die Latenzzeit beim Übermitteln von Nachrichten eingegangen. Diese beschreibt die Zeit, welche zwischen der Versendung und dem Empfang eines Datenpaketes im Netzwerk benötigt wird. Durch anderen Datenverkehr im System kann es zusätzlich zur Erhöhung der Latenz kommen. Der sog. Querverkehr kann zum Problem werden, falls nicht-kritische und kritische Daten gemeinsam über eine Kommunikationsstrecke fliesen. Querverkehr (eng. *interspersing traffic*) bezeichnet in dieser Arbeit Daten, die den eigentlichen Datenfluss beeinträchtigen. Hier kann es bei jedem Hop zu Verzögerungen kommen. Ein Hop bezeichnet den Weg zwischen zwei Netzwerkknoten [31]. [25]

Abbildung 4 zeigt ein Beispiel zur Versendung von Nachrichtenpaketen $p_1 \dots p_6$ von zweier Anwendungen A und B über einen Switch (ECU). Hier kommt es an verschiedenen Stellen zu Verzögerungen der Pakete. Diese bezeichnet man als Delays. Ein Netzwerkswitch besitzt mindestens eine Warteschlange (*Queue*).Jedes Datenpaket hat zwei Media Access Control (MAC)-Adressen, die erste gibt den Absender an, die zweit den Empfänger. Die Daten werden anhand der MAC-Adresse an die angegebene Empfängeradresse weitergeleitet. In diesem Beispiel hätte das Paket p_1 die Empfänger-MAC-Adresse von C.

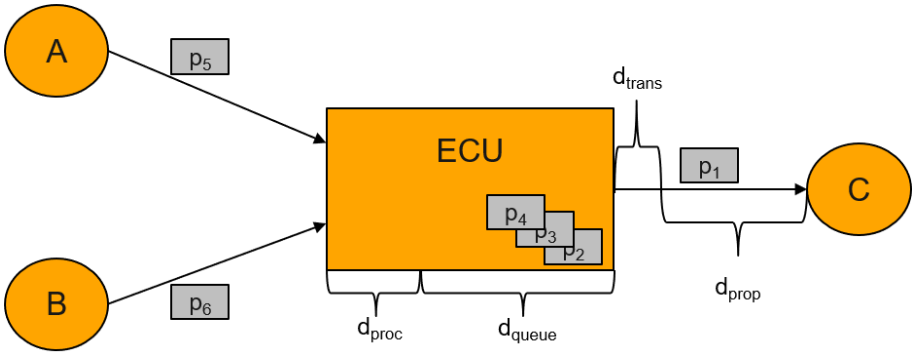

Abbildung 4: Entstehung der Latenzzeiten bei Übermittlung von Datenpaketen (Modell angelehnt an Literatur [25])

In den folgenden Punkten wird auf die verschieden Delay-Arten eingegangen:

- Verarbeitungsverzögerung (Processing Delay d_{proc})
  Beim Empfangen des Datenpakets am Switch wird es anhand der MAC-Adresse zum entsprechenden Ausgangsport weitergeleitet. Die Verzögerung entsteht bei der Verarbeitung des Paket-Headers. Hier wird neben der Zieladresse auch auf mögliche Fehler, wie beispielsweise Bit- und Übertragungsfehler, geprüft. Mit steigenden Anforderungen an die Kommunikation, können hier auch weitere Prüfungen stattfinden [33]. Die Verarbeitungszeitverzögerung ist auch abhängig von der Implementation des Switches [25].

- Warteschlangenverzögerung (Queuing Delay d_{queue})
  Jeder Port zum Datenversand hat eine Möglichkeit zur Datenzwischenspeicherung mittels verschiedener Queues. Diese ist nötig, da immer nur nacheinander Daten übertragen werden können. Ist ein Port gerade belegt, wird das Paket in die zugehörige Queue eingereiht. Die Wartezeitverzögerung berechnet sich aus der Summe von noch zu versendenden Paketen. Je mehr Pakete bereits in einer Warteschlange sind, desto höher ist die Wartezeit. Ein Port kann auch priorisierte Warteschlangen haben. Hierbei hat er mehrere Queues, die je nach Wichtigkeit der Datenpakete befüllt werden und nach diesem Prinzip abgearbeitet werden [25].

- Übertragungsverzögerung (Transmission Delay d_{trans})
  Die Übertragungsverzögerung bezeichnet die Zeit, die benötigt wird um ein Datenpaket komplett auf die Leitung zu legen. Sie steht somit in direkter Abhängigkeit zu der Paketgröße. [25]

- Ausbreitungsverzögerung (Propagation Delay d_{prop}) Die Weiterleitung zum nächsten Switch oder Endpunkt der Nachricht wird durch die Ausbreitungsverzögerung angegeben. Diese ist direkt proportional zur Leitungsgeschwindigkeit, welche wiederum von der physikalischen Beschaffenheit des Mediums abhängt [25].

Alle diese Delay-Zeiten aufsummiert, ergeben die Latenzzeit einer Nachricht. Befinden sich auf der Kommunikationsstrecke mehrere Switche, so gibt es auch mehre Wege, die bereits erwähnten Hops, so müssen diese Verzögerungen auch mit beachtet werden.

2.2.4 Jitter

Mit *Jitter* wird die Schwankung der Verzögerungszeit bezeichnet und gibt die maximale Differenz zwischen Ende-zu-Ende-Verzögerung an. Die Hauptgründe für Schwankungen sind vor allem Abweichungen beim Zugriff auf Übertragungsmedien und Querverkehr im Datenstrom. Jitter kann sowohl zu Verspätungen als auch zum verfrühten Ankommen von Datenpaketen führen. Durch diese Schwankungen sind Laufzeitgarantien weniger genau möglich. In der Netzwerktechnik wird Jitter auch oft als Varianz oder Laufzeit von Datenpaketen bezeichnet [31] [25].

2.2.5 Paketverlust

Es gibt verschiedene Ursachen die zu Paketverlust führen. Die häufigsten Verluste sind Fehler bei der Übertragung und Verluste bei der Verarbeitung in Switchen. Bei Letzterem kommt es bei zu vielen Datenpaketen zu einem Überlauf einer Queue. Dies bezeichnet man als *Buffer-Overflow*. Hierbei gehen die zuletzt eingegangen Pakete verloren, da sie nicht mehr zur Verarbeitung in der Warteschlange platziert werden können. Ein Überlauf kommt nur dann zustande, wenn die Datenrate der ausgehenden Übertragungsleitung nicht mehr ausreichend ist, um die angestauten Pakete aus den zugehörigen Queues versenden zu können [31] [25].

2.3 Netzwerkmanagement im Automotiv-Bereich

Dieser Abschnitt erklärt den aktuellen und zukünftigen Stand der Technik in einem moderneren Fahrzeug.

Da die Größe der zu sendenden Daten in einem Fahrzeug ständig weiter steigt, ist es notwendig eine neue Generation von fahrzeug-interner Netzwerkinfrastruktur zu entwickeln. Die bisher genutzten Techniken, wie Controller Area Network (CAN), Local Interconnect Network (LIN) und FlexRay sind für diese Datenmengen nicht mehr ausreichend. Aus diesem Grund versucht man Ethernet in das Fahrzeug zu integrieren, was in manchen Anwendungsbereichen bereits der Fall ist. Ethernet wurde bereits unabhängig von der Automobilbranche unter der IEEE 802.3 [23] standardisiert und ist ein vielversprechender Ansatz, da es sich durch eine hohe Wiederverwendbarkeit für Komponenten, Software und Tools auszeichnet. Ebenso verfügt es über eine ausreichend dimensionierte Bandbreite und gute Skalierbarkeit des gesamten Netzwerkes. Der Hauptvorteil von Ethernet ist, dass jeder Teilnehmer seine eigene Leitung besitzt und anders als bei den Bus-Systemen kein

geteiltes (eng. *shared*) System ist. Hierbei nutzen alle Teilnehmer dieselbe Leitung. Im Automobilbereich spielen aber auch noch andere Kriterien eine Rolle, wie beispielsweise Kosten, Robustheit und der Stromverbrauch [28]. Anhand dieser Kriterien muss sich Ethernet erst noch gegen seine bereits im Fahrzeug fest etablierten Konkurrenten durchsetzen. Diese Systeme erfüllen die benötigten Echtzeit-Anforderungen, wie eine garantierte Latenzzeit der Datenpakete, an eine Kommunikation zwischen den Endgeräten. Da Ethernet jedoch anders als Bus-Systeme ein verbindungsloses Netzwerkmedium ist, muss auf eine andere Weise sichergestellt werden, dass Daten in Echtzeit ankommen. Dies kann mittels Time-Sensitive Networking (TSN) umgesetzt werden, welches im Abschnitt 2.5 näher erläutert wird.

2.4 Audio Video Bridging

Im Jahr 2005 würde die Gruppe Audio Video Bridging (AVB) (IEEE 802.1) gegründet, um eine synchronisiertes und priorisiertes Übertragen von Audio- und Videodaten im Netzwerk zu ermöglichen. Die Gruppe erstellte Spezifikationen für die Datenübermittlung in Layer (dt. Schicht) 2, des International Organization for Standardization/Open Systems Interconnection Modells (ISO/OSI-Modell). Im Automobilbereich wurden AVB erstmal mit der Bestrebung integriert, Audio- und Videodaten im Infotainmentbereich über Ethernet zu übertragen. Das Ziel war es, für wichtige Datenpakete, eine Latenz von maximal 2ms über 7 Hops zu garantieren [11].

Für detaillierter Erklärungen zu AVB kann in [29] nachgelesen werden. Hier wird unter anderem beschrieben, dass AVB bereits die Möglichkeit bietet, Latenzen für Daten mit Echtzeit-Anforderungen zu berechnen. Jedoch wird hierzu immer noch an zusätzlichen Erweiterungen gearbeitet.

2.5 Time-Sensitive Networking (TSN)

Seit dem Jahr 2012 werden die bisherige Themen der Gruppe AVB in einer neuen Gruppe unter dem Namen Time-Sensitive Networking (TSN) weiter entwickelt. Diese beschäftigen sich weiterhin mit der Standardisierung von Erweiterungen zur Erhöhung der Güte des Gesamtsystems.

Der Grund für den Wechsel von AVB auf TSN war, dass nicht nur Audio- und Video-Daten versendet werden sollten, sondern auch Kommunikationsdaten, wie z.B. Datenpakete für autonomes Fahren, Radardaten oder große Sensordaten. Das Ziel ist es, Echtzeitfähigkeit

im Bereich Ethernet-Netzwerke zu ermöglichen. Dies soll mit einer möglichst geringen Latenzzeit von maximal 100μs auf 5 Hops bei einer Datenrate von 100MBit/s zwischen den Netzwerkgeräten ermöglicht werden [27]. Um das Versenden von Daten mit einer Latenz-Garantie zu ermöglichen, können verschieden Mechanismen verwendet werden. Einer davon wäre Datenpakete mit einer verschiedenen Priorisierungen zu versenden. So könnte für wichtige Daten ein garantiertes Ankommen in der gewünschten Zeit sicher gestellt werden. In der folgenden Tabelle 2.1 werden die bisher wichtigsten Standards der IEEE zum Thema TSN aufgelistet. Diese beinhalten weiter Technologien, wie beispielsweise das erwähnte Priorisieren.

IEEE-Standard	Titel
802.1AS-Rev	Timing and Synchronization for Time-Sensitive Applications [14]
802.1Qbu	Frame Preemption [17]
802.1Qbv	Enhancements for Scheduled Traffic [18]
802.1Qca	Path Control and Reservation [19]
802.1CB	Frame Replication and Elimination for Reliability [15]
802.1Qcc	Stream Reservation Protocol (SRP) Enhancements and Performance Improvements [20]
802.1CM	Time-Sensitive Networking for Fronthaul [16]
802.1Qci	Per-Stream Filtering and Policing [21]
802.1Qca	Path Control and Reservation [19]
802.1Qcr	Bridges and Bridged Networks Amendment: Asynchronous Traffic Shaping [22]

Tabelle 2.1: Aktuell IEEE Standards zu TSN

In den folgenden Unterabschnitten werden die wichtigsten Themen von TSN detaillierter behandelt, die für ein echtzeitfähiges Netzwerk von besonderer Bedeutung sind. Es wird nicht auf alle Standards komplett eingegangen, da dies den Rahmen der Abhandlung sprengen würde. Bei Bedarf kann dies bei den Unterlagen und den Veröffentlichungen der Gruppe IEEE 802.1 nachgelesen werden. Hier soll ein grundsätzliches Verständnis für die wichtigsten Techniken gegeben werden, um verschiedene Entscheidungen in der Arbeit nachvollziehen zu können.

2.5.1 Scheduling und Traffic Shaping

Alle teilnehmenden Netzwerkgeräte arbeiten bei der Bearbeitung und Weiterleitung von Datenpaketen nach den gleichen Regeln. TSN nutzt dafür größtenteils den IEEE 802.1Qbv Standard. Beim Shaping bezieht man sich hauptsächlich auf zwei Arten von Shapern. Diese werden in den nächsten beiden Unterpunkten kurz erläutert.

- **Time Aware Shaper (TAS)**
  Der IEEE802.1Qbv Standard definiert die Technik des TAS. Dieser stellt einen zeitlich gesteuerten Weiterleitungsmechanismus dar und ermöglicht die genaue Planung des Datenverkehrs im Netzwerk. Durch den Einsatz des Shapers können Verzögerungen kritischer Pakete durch Querverkehr vermieden werden. Er kann Zeiten bis zu 100µs garantieren. Das Prinzip von TAS basiert auf das Zeitscheibenmultiplexing-Verfahren, wodurch der Zugriff auf ein Medium zeitgesteuert abläuft. Der Shaper unterteilt die vorhandenen Zeiten in Intervalle. Hier gibt es sog. kritische- und nicht kritische-Zeitintervalle. Diese sind in verschiedene Warteschlangen unterteilt.
  Laut Definition im Standard kann ein Port bis zu acht Warteschlangen besitzen. Jede Queue bekommt ein sog. *Gate* (dt. Zeitfenster). Dieses nutzt der TAS um nur bestimmte Schlangen in einem Zeitintervall das Versenden von Daten zu erlauben. Die Übertragung kommt nur dann zustande, wenn das Gate geöffnet ist und noch genügend Zeit zur vollständigen Übermittlung des Frames bleibt.

- **Credit Based Shaper (CBS)**
  Der Credit Based Shaper ist ebenfalls in IEEE802.1 standardisiert. Die eingesetzte Technik garantiert eine Latenzzeit von 2 Millisekunden über sieben Hops bei einer Datenrate von 100 Mbit/s. Dieser arbeitet mit verschieden klassifizierten Warteschlangen an den ausgehenden Ports. So hat ein Port beispielsweise zwei Queues, eine mit Klasse A und eine weitere. Des Weiteren gibt es einen Verlauf, den sog. Credit (dt. Guthaben). Nach dem Prinzip des Credit Based Shaper wird der Port immer Daten aus der Klasse A versenden, wenn der der Credit größer oder gleich 0 ist. Sonst werden Daten aus weiteren Klasse versendet. Der Credit ändert sich nur, wenn Daten aus der zweiten Klassen versendet werden, obwohl in Klasse A noch Paket zum senden vorhanden sind. Der CBS bietet Fairness bei der Behandlung von kritischen Daten, z. B. aus Klasse A, und andern Klassen [8].

2.5.2 Frame Preemption

Der Vollständigkeit halber wird hier noch Frame Preemption kurz erläutert. Es ist ebenfalls ein IEEE Standard [17] und kann Verzögerungen bei der Latenzzeit von Paketen reduzieren. Frame Preemption ist eine weiter Möglichkeit Latenzzeiten zu senken. Zum aktuellen Zeitpunkt wird es aber noch nicht von den Endgeräten im Fahrzeug unterstützt.

Um Frame Preemption zu ermöglichen wird die Übertragung von weniger kritischen Paketen für kritische Pakete unterbrochen. Die unkritischen Datenpakete werden erst weiter versendet, wenn alle kritischen vollständig verschickt wurden. Frame Preemtion wird vor allem in Kombination mit dem Time Aware Shaper verwendet. Hiermit können sog. Überlast-Situationen vermieden werden, bei denen es zum Überlauf der Warteschlangen im Switch kommen kann. Bei diesem Verfahren wird die sichere Versendung von Daten mit der gegebenen Latenz-Anforderung garantiert. Pakete mit einer großen Latenzzeit können innerhalb der nicht kritischen Zeitintervalle des TAS versendet werden. Ein übermäßiges Senden von höher priorisierten Daten kann trotzdem zum Datenstau bei den Niedrigeren führen.

2.5.3 Per-Stream Filtering and Policing

Per-Stream Filtering and Policing (PSFP) wurd unter der IEEE 802.1Qci [21] standardisiert. Es kümmert sich um die Überwachung der Datensteams im System. PSFP definiert mehrere Filter- und Überwachungsfunktionen die an den ein- und ausgehenden Ports der Netzwerkgeräte konfiguriert werden können. In dieser Arbeit wird sich nur auf das Filtern der Datenströme bezogen. Hierzu gibt es die PSFP-Funktionen *Ingress filtering, Frame filtering und Engress filterning* [4]. Beim Filtering geht es in der Regel immer darum ein- und ausgehende Daten nach vordefinierten Regeln zu untersuchen. Falls bei einer Überprüfung eine Diskrepanz zwischen Soll- und Ist-Wert festgestellt wird, können Maßnahmen zur Behebung, oder der weitere Umgang mit den korrupten Daten eingeleitet werden. Der Punkt Ingress filtering spielt in der Netzwerktechnologie eine wichtige Rolle und wird deshalb in dieser Arbeit noch detaillierter in Abschnitt 5.1 behandelt.

2.6 Funktionale Sicherheit einer Anwendung

Der Punkt Safety ist ein großer Bereich in der Automobilbranche. Diese Arbeit betrachtet die grundsätzlich einzuhaltenden Parameter einer Anwendung nach ISO26262 [34]. Dies ist eine Norm für sicherheitsrelevante elektronische Systeme im Fahrzeugbereich. Dort werden verschiedene Vorgehensmodelle angegeben, welche die anzuwendende Methoden in der Entwicklung von Produkten im Kraftfahrzeugbereich definieren. Die Umsetzung der Norm soll eine funktionale Sicherheit im System gewährleisten. Da das Thema Sicherheit in Automobilsystemen jedoch ein großes Einzelthema ist, wird dies nur am Rande in der Arbeit behandelt. Eine ausführliche Erläuterung des Themas kann im Buch *Functional Safety for Road Vehicles* [35] nachgelesen werden.

2.7 Schutzziele einer Anwendung

In der Informationssicherheit gibt es das Konzept der Schutzziele. Dieses stellt sicher, dass sicherheitskritische Daten geschützt und unverändert übertragen werden. Der Zugriff auf die Daten ist zu beschränken und zu kontrollieren, sodass nur autorisierten Nutzern ein Zugriff gewährt wird. Die Schutzziele, die diese Anforderungen präzisieren, sind Integrität, Verfügbarkeit und Vertraulichkeit. Netzwerkgeräte, die auf kritische Daten zugreifen sollen, müssen je nach Sicherheitsanforderung eindeutig identifiziert werden können. Die entsprechende Eigenschaft der Geräte nennt man Authentizität, welches ein kombiniertes Schutzziel aus mehreren Parametern ist. Es ist eines der wichtigsten Ziele, da über die Authentifizierung der Zugriff auf Daten geregelt werden kann und somit die Verfügbarkeit gewährleistet wird [9].

Die angesprochenen und für die Arbeit relevanten Schutzziele werden in den folgenden Unterabschnitten kurz erläutert.

2.7.1 Integrität

Ein System gewährleistet die Datenintegrität, wenn es keinem Netzwerkteilnehmer möglich ist, die zu schützenden Daten unautorisiert zu verändern oder zu manipulieren. Dies kann durch Festlegen von Rechten an Daten erfolgen. So kann eine Anwendung beispielsweise nur Leserechte an den übermittelten Daten haben, während eine andere sie auch bearbeiten darf. Manipulation muss jederzeit nachgewiesen werden können und darf nicht unbemerkt bleiben. Dies kann z.B. anhand der bereits im Abschnitt 2.1.1 und 2.1.2 erwähnten Check-Summen überprüft werden. Sind die Summen nicht korrekt, weist es auf eine Veränderung der Daten hin. Eine Veränderung kann sich einerseits durch einen Übertragungsfehler ergeben oder auch durch Manipulation der Daten.

2.7.2 Verfügbarkeit

Das Netzwerk gewährleistet eine gute Verfügbarkeit, wenn authentifizierte und autorisierte Netzwerkteilnehmer bei der Ausführung ihrer Aktionen nicht unautorisiert beeinträchtigt werden können. Die Erfüllung dieses Schutzziels benötigt jedoch eine Nutzungsverwaltung von Systemressourcen und ist nicht Teil dieser Arbeit.

2.7.3 Vertraulichkeit

Ein System weist eine Informationsvertraulichkeit auf, wenn es keine unautorisierten Informationsgewinnung ermöglicht. Dies wird mittels Kontrolle der Datenflüsse im Netzwerk gewährleistet. Eine Möglichkeit hierzu ist die Verwendung von einer Ende-zu-Ende Verschlüsselung. Auch dies erfordert eine eigene Technologie, welche nicht Teil dieser Arbeit ist.

2.7.4 Authentizität

Die Authentizität eines Objekt gibt dessen Echtheit und Glaubwürdigkeit an. Anhand dieser kann sicher gestellt werden, dass es sich um einen berechtigten Nutzer der Daten handelt. Dies wird durch eindeutige Identifikationsmerkmale, wie beispielsweise Identifikationsnummer (ID), gewährleistet.

Kapitel

3

Anforderungen an die Konfiguration

Das folgende Kapitel behandelt die Anforderungen an ein Netzwerk und die Parameterdefinition für das applikationsspezifische Manifest. Dieses soll möglichst wenig Elemente beinhalten, von denen weitere Parameter automatisch abgeleitet werden können. Nach dem zuvor entwickeltem Konzept des Fahrzeugnetzwerkes, soll die Ableitung an einer zentralen Stelle im System erfolgen und nicht mehr manuell beeinflusst werden müssen. Die wiederum abgeleiteten Parameter sollen ein Netzwerk eindeutig spezifizieren und somit eine Konfiguration möglich machen. Diese erfolgt durch einen Manager, auf den im Laufe der Arbeit noch genauer eingegangen wird.

3.1 Netzwerkübersicht

In der Zieldefinition der Arbeit wurde bereits ein grober Überblick über die Funktion des Netzwerk-Managers im System gegeben, welcher jedoch nicht die einzige Komponente im Netzwerk ist. Teil der Abhandlung war es auch, das Zusammenspiel der einzelnen Komponenten in einem sog. dynamischen Netzwerk zu verstehen und die Funktionalität weiter zu vervollständigen. Um die Sicht auf das Netz etwas näher spezifizieren, wird im Folgenden näher darauf eingegangen.
Der Manager wird von anderen Netzwerkkomponenten mit Eingaben befüllt, anhand derer er Entscheidungen treffen kann. In Abb. 5 wird ein einfaches Netzwerk mit wenigen Komponenten dargestellt. Eine Anwendung soll im System gestartet werden. Dies hat die Aufgabe Daten von einem Sender zu einen Empfänger zu übermitteln. Im einfachsten Fall, bedingt dies den Datenaustausch zweier ECU die mittels einem Switches verbunden sind. Um die Kommunikationsstrecke nach den geforderten Bedingungen der Applikation zu konfigurieren, müssen der Applikations-Orchestrator, welcher für die Verwaltung der Anwendungen verantwortlich ist und der Netzwerk-Manager, der wiederum für die Konfiguration für das Netzwerk zuständig ist, ein geeignetes Setup für das System erstellen. Beide Komponenten werden detaillierter im Kapitel 4 vorgestellt.

Im Laufe der Arbeit wird noch spezifischer auf das Netzwerk eingegangen. Vorerst soll der generelle Überblick über das System genügen.

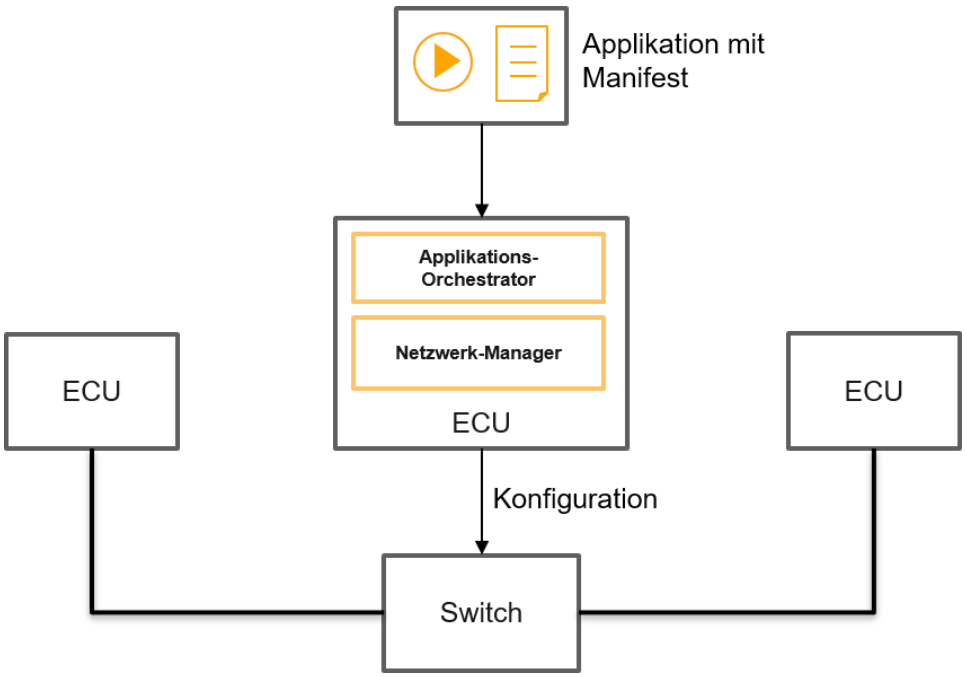

Abbildung 5: Darstellung eines vereinfachten Netzwerkes

3.2 Der Netzwerk-Manager

Eines zentrale Komponente im zukünftigen Fahrzeugnetzwerk wird der Netzwerkmanager sein. Dieser enthält alle nötigen Funktionen um das Netzwerk zu konfigurieren. Er ist eine der zentralen Bausteine im System und kennt die gesamte Topologie des Fahrzeuges. Anhand dieser Kenntnisse und der vom Orchestrator übergebenen Parameter, kann der Manager, die sich im Fahrzeug befindenden Switche und ECU konfigurieren. Die Funktionsweise des Netzwerkmanagers kann am Besten mit dem Software-defined networking (SDN) Prinzip erläutert werden [30]. Die Grundidee hinter SDN ist die zentrale Verwaltung aller Netzwerkkomponenten in einem System. Hierbei kommt es zu einer Abstraktion der Schichten. Es wird die *Control Plane* (z. dt. Kontrollschicht) von der *Data Plane* (z. dt. Datenschicht) abgespalten. Die Kontrollschicht ist für die Steuerung der Datenschicht verantwortlich, was der Abb. 6 vereinfacht dargestellt ist.

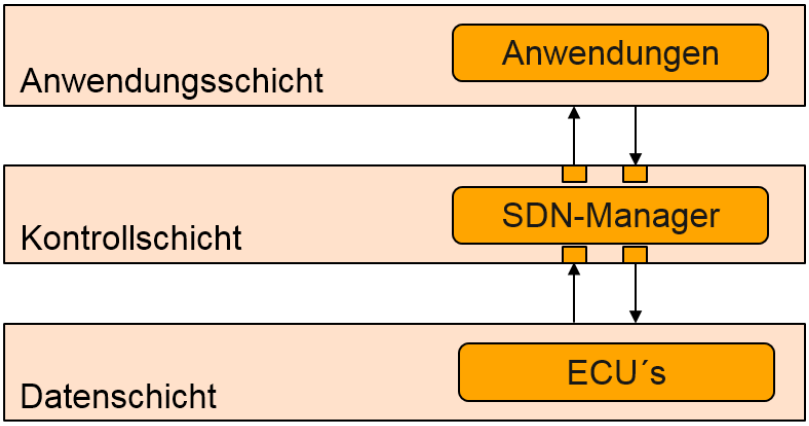

Abbildung 6: Aufbau der Schichten mittels SDN

In der mittleren Schicht befindet sich der Manager. Dieser sendet und empfängt mittels Schnittstellen Daten und Befehle. So kann beispielsweise eine Anwendung eine Netzwerkanforderung an den Manager senden. Dieser wiederum konfiguriert anhand der empfangenen Daten die entsprechenden ECU. Das Prinzip wird auch im Fahrzeugnetz weiter verfolgt und angewendet.

In Abb. 7 wird der Netzwerkmanager im Fahrzeug veranschaulicht. Er besteht aus vier Hauptkomponenten (eng. Core Functionality):

- Path finding: zu dt. Pfadfindung. Diese ist verantwortlich für das Finden des bestmöglichen Pfades im Netzwerk.
- Network Discovery: zu dt. Netzwerkerkennung. Diese kümmert sich um die Netzwerktopologie, damit diese immer aktuell ist.
- Interface to other components: zu dt. Schnittstelle nach außen.
- Remote Configuration: zu dt. automatische Konfigurationsschnittstelle. Sie konfiguriert die ECU nach den Anforderungen des Manifests.

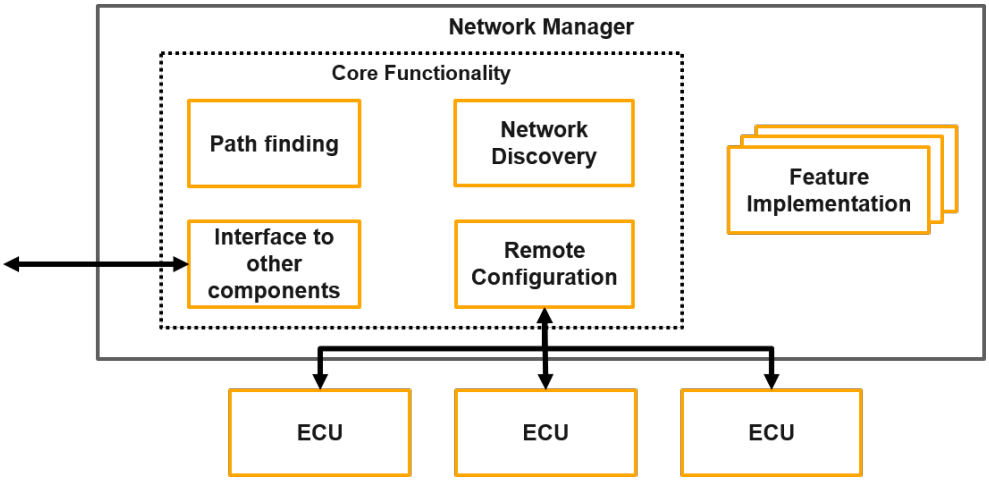

Abbildung 7: Aufbau des Netzwerkmanagers

Die ebenfalls im Bild dargestellten *Feature Implementations* stellen die einzelnen Funktionen des Managers dar, mit denen er das Netzwerk konfigurieren kann. Werden Daten aus der Anwendungsschicht an die Schnittstelle des Managers übergeben, kann dieser mittels der implementierten Funktionen die entsprechende Konfiguration für das System erstellen. In den Funktionen sind alle nötigen Mechanismen hinterlegt, die eine ECU benötigt, um die Netzwerkanforderungen zu erfüllen. Ein Beispiel hierfür sind, die in Abschnitt 2.5 beschrieben Shaper, mit denen bestimmte Latenzzeiten eingehalten werden können.

Im weiteren Verlauf dieser Arbeit werden einzelne Funktionen und die Konfigurationsschnittstelle implementiert und getestet. Die anderen Komponenten des Netzwerkmanagers werden als gegeben betrachtet und sind nicht Teil der Ausarbeitung.

3.3 Der Applikations-Orchestrator

Der Orchestrator ist im System für die Verwaltung und Verteilung der Anwendungen auf den einzelnen ECU's verantwortlich. Er kennt nicht die gesamte Netzwerktopologie, sondern weiß lediglich welche ECU und Anwendungen sich gerade im Netzwerk befinden. Der Applikations-Orchestrator ist in erster Linie für die Verwaltung der Funktionalität des Systems verantwortlich. Wenn eine neue Applikation gestartet werden soll, liest er die benötigten Parameter aus dem Applikationsmanifest aus und ermittelt eine Liste an geeigneten ECU, die sich im System befinden. Diese werden in absteigender Reihenfolge ihrer Tauglichkeit nach sortiert. In der Liste befinden sich alle Geräte, die für die Ausführung der Anwendung geeignet sind.

Die Auswahl wird anhand der benötigten Leistung einer Applikation getroffen, was einen fehlerfreien Ablauf der Funktionen garantiert. Als Leistungsparameter werden beispielsweise Speicherbedarf und CPU Leistung angegeben.

Ist die Liste vollständig, übergibt er diese mit den Parametern aus dem Manifest an den Netzwerkmanager. Dieser prüft die gewünschten Endgeräte der Reihe nach auf ihre netzwerkseitige Tauglichkeit. Sobald eine der ECU's den Anforderungen entspricht, sendet er die ID des Gerätes an den Orchestrator zurück. Wenn dies der Fall ist, kann die Anwendung auf der zugehörigen ECU gestartet werden. Sollte keines der Endgeräte in der Liste für den Betrieb geeignet sein, gibt der Manager einen Fehler zurück. Dies bedingt im Orchestrator einen Abbruch des Versuches, die Anwendung zu starten.

Abb. 8 veranschaulicht den Findungsprozess einer geeigneten ECU für eine Applikation im Netzwerk. Hierbei müssen die beiden Komponenten, Orchestrator und Netzwerkmanager, miteinander kommunizieren.

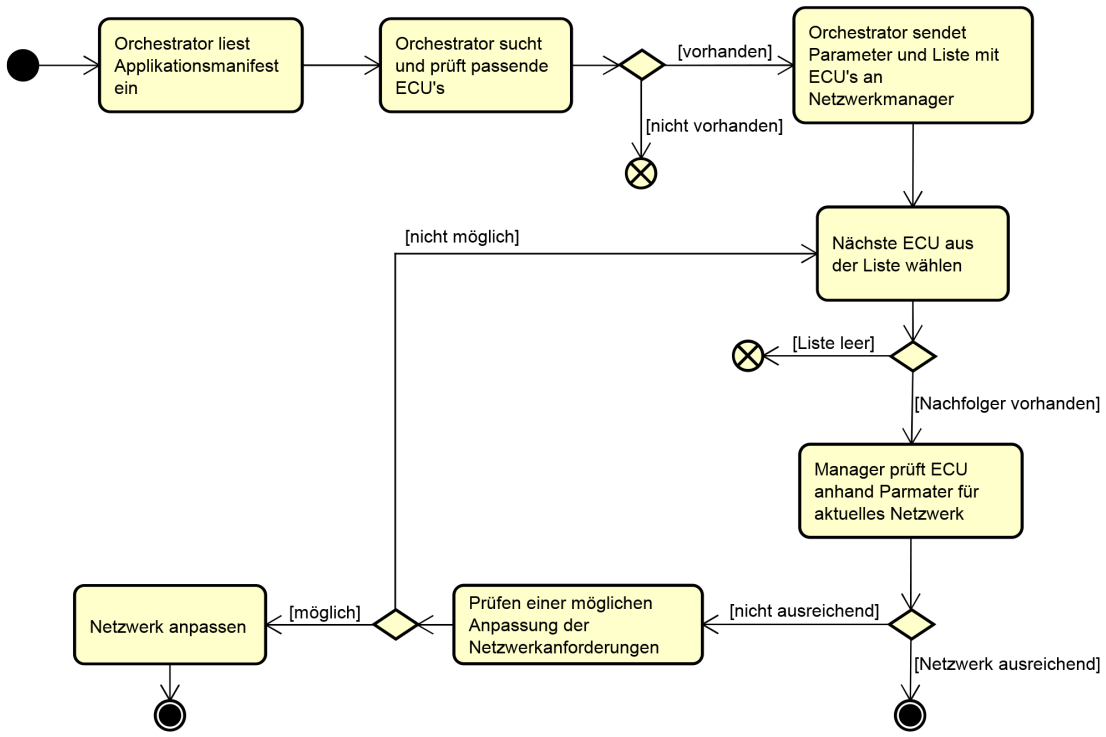

Abbildung 8: Kommunikation von Orchestrator und Manager

3.4 Anforderungen an das dynamische Netzwerk

An das Fahrzeugnetzwerk werden verschiedenste Anforderungen gestellt. So soll es beliebig erweiterbar im Hinblick auf Skalierbarkeit sein, genügend Bandbreite besitzen, um auch mit großen Datenmengen umgehen zu können, und natürlich sicher vor ungewollten Einflüssen sein. In der Automobilbranche ist Sicherheit ein besonders wichtiger Punkt, weshalb unter der Norm ISO 26262 [34] die wichtigsten Störungsmöglichkeiten definiert sind.

Generell muss ein Netzwerk alle gängigen Sicherheitsanforderungen gerecht werden. Sie müssen erfüllt sein, um ein Netzwerk nach den sog. Safety-Anforderungen zu erstellen [26]. In der Tabelle 3.1 sind die wichtigsten Störungsfälle im System aufgelistet. Anhand dieser Anforderungen, welche als Grundvoraussetzung für sichere System gelten, werden im Folgekapitel die Parameter für das Applikationsmanifest festgelegt. Wenn bereits einige dieser Anforderungen nicht abgedeckt werden, wird es schwierig ein sicheres Netzwerk aufzubauen [35].

Störung	Erläuterung
Korrupte Nachricht	Die Daten des empfangenen Paketes einer Nachricht sind falsch
Nachrichten Verzögerung	Nachricht wird später als erwartet empfangen
Nachrichtenverlust	Nachricht oder Paket geht im System verloren
Wiederholte Nachricht	Empfänger erhalten zwei oder mehr Nachrichten mit dem selben Inhalt
Nachfolgeregelung	Nachrichten werden in falscher Reihenfolge empfangen
Unterbrechungen	Empfänger erhalten unerwartet eine Nachrichten, während er noch mit dem Empfangen einer andern Nachricht beschäftigt ist
Maskierung	Nachricht wird unter Verwendung einem gefälschten Identifikationsmerkmal versandt
Asymmetrische Auskünfte	Informationen eines einzelnen Absenders werden von mehreren Empfängern unterschiedlich empfangen
Unsichere Kommunikation	Ermöglichen den unbefugten Zugriff auf Kommunikationsdaten

Tabelle 3.1: Übersicht der in ISO26262 zertifizierten Störmöglichkeiten

3.5 Manifest-Parameter

Eine Applikation hat verschiedene Anforderungen an ihre Netzwerkumgebung, die durch Parameter definiert werden. Im ersten Schritt werden gängige Netzwerk-Parameter anhand der in den Kapitel 2 definierten Werte und Techniken, insbesondere im Hinblick auf die Anforderungen an Echtzeitsystem und der Schutzziele eines Systems aus Abschnitt 2.7 ausgewählt. Diese mussten im nächsten Schritt mit den Anforderungen aus dem vorherigen Punkt 3.4 vereinigt werden. Nur wenn alle Punkte erfüllt werden, kann ein Netzwerk überhaupt im Hinblick auf Sicherheit weiter behandelt werden. Alle im Manifest definierten Parameter wurden im laufe der Abschlussarbeit erschlossen und anhand der Anforderungen spezifiziert. Hierzu wurden verschieden Quellen, wie Literatur und Experten aus dem Netzwerksegment bei der Continental Automotive GmbH konsultiert.

Der letzte Schritt in der Parameterdefinition ist das weit möglichste Abstrahieren der netzwerkspezifischen Eigenschaften. Hierdurch konnten einige wenige Parameter definiert werden, von denen wiederum der Netzwerkmanager (beschrieben in Kapitel 3.2) alle für eine Konfiguration wichtigen Parameter ableiten kann. Dies ermöglicht eine Entwicklung einer Applikation, unabhängig von einer konkreten Netzwerktechnologie. Somit können auch spätere Änderungen in der Netzwerkinfrastruktur vorgenommen werden, ohne die Anwendungen überarbeiten zu müssen. Auf diesem Weg erreicht man den Schritt von einem normalen homogenen Netzwerk zu einem heterogenen Netzwerk. Der Unterschied ist in erster Linie dass, anders als bei einem homogenen Netz, auch unterschiedliche Softwaresysteme untereinander kommunizieren können [12].

Jeder Entwickler legt für seine Anwendung die gewünschten Parameter fest, welche sich im Manifest befinden. Um einen fehlerfreien Ablauf zu garantieren, sind diese, wie schon erwähnt, lediglich abstrahierte Parameter die später softwareseitig in die konkret benötigten Parameter aufgeschlüsselt werden. So können beispielsweise aus einer Latenz Parameter zur Konfiguration, des in Kapitel 2.5.1 Traffic Shaping, abgeleitet werden. Das entstandene File wird als Applikationsmanifest bezeichnet und ist im Format JavaScript Object Notation (JSON). JSON ist ein Datenformat, welches zum Datenaustausch zwischen Anwendungen verwendet wird. Dies wurde im laufe dieser Arbeit definiert, da es mehrere Eigenschaften besitzt, welche für den Anwendungsfall der Informationsübermittlung von Vorteil sind. Einer davon ist, das es für den Entwickler einfach les- und schreibbar, was eine einfache Bedienung verspricht. Ein weiterer ist die strukturelle Form von dieser Sprache. Sie lässt in einfacher Form komplexen Strukturen abbilden. Dies ermöglicht es beispielsweise ein Array oder eine Liste an geschachtelten Elementen zu übermitteln [10].

Ein Vorteil dabei ist es, wenn eine Applikation mehrere verschieden Arten von Nachrichten übermitteln muss. Wäre diese Art der Schachtelung nicht möglich, könnte eine Anwendung immer nur genau einen Kommunikationspartner haben und eine Art von Daten versenden.

Tabelle 3.2 enthält die Parameter, welche aus Netzwerksicht gesetzt werden müssen, um eine Konfiguration der gesamten Kommunikationsstrecke zu generieren. In den folgenden Unterabschnitten werden diese näher erläutert.

Parameter	Spezifikation
name	Name der Applikation im System
id	Eindeutige Identifikationsnummer
groupId	Eindeutige Gruppenidentifikationsnummer
domain	Domain einer Anwendung
cyclic	Zyklisch- oder Event-gesteuerte Anwednung
timesync	Zeitsynchronisation erforderlich
maxLatency	Latenzzeit einer Nachricht
minLatency	Unter Grenze der Latenzzeit (Jitter)
messageSize	Größe der zu sendende Daten
frequency	Wiederholungsgeschwindigkeit einer Nachricht
integrity	Korrektheit der zu sendenden Daten

Tabelle 3.2: Übersicht der Manifest-Parameter

3.5.1 Identifikation und Gruppen

Jede Anwendung besitzt eine ID, anhand dieser sie eindeutig identifiziert werden kann. Dies wird in dem Parameter *id* hinterlegt. In dem Konzept der Erweiterbarkeit des Netzwerkes muss diese ID weltweit eindeutig hinterlegt werden, um eine Anwendung unmissverständlich einer Herkunft zuordnen zu können. Vor allem wenn es darum geht Fehler im System zu finden, ist eine eindeutige Zuordnung wichtig, da sich ein *Bugfix* der Anwendung als schwierig herausstellen kann. Dieser muss nicht nur lokal im System erfolgen, sondern weltweit für alle Fälle, in denen jene Applikation zum Einsatz kommt.

Neben der Eliminierung der Fehlermöglichkeit einer Maskierung einer Nachricht mit einer falschen ID, ist die Regelung der Kommunikation zwischen den Applikationen innerhalb eines Netzwerkes ein positiver Nebeneffekt einer eindeutigen ID. Das Format der ID ist definiert durch den Prefix *id-* und einer darauf folgenden fünfstelligen Nummer, z. B. *id-12345* und muss einmalig weltweit sein sein. Der Netzwerkmanager erhält die ID beim Start der Anwendungen. Anhand dieser können Kommunikationsstrecken aufgebaut werden.
Da es auch Applikationen gibt, die gemeinsam benachrichtigt werden müssen, existiert eine Gruppenidentifikationsvariable. Angelehnt an das Format der ID, hat die Gruppen-ID die Form *group-12345*. Anhand dieser können Multicast-Nachrichten an alle Anwendungen einer Gruppe gesendet werden. Ebenso kann eine Domain angeben werden, um eine Applikation eindeutig zuordnen zu können. Dies kann möglich sein, falls die Anwendung in mehren Subnetzen Nachrichten versenden muss, so kann diese einer Kommunikation eindeutig zugeordnet werden

3.5.2 Sender und Empfänger

Durch die eindeutige ID einer Anwendung können Regeln für ein- und ausgehende Nachrichten definiert werden. Da eine Anwendung Unicast- und Multicast-Nachrichten an einen Empfänger senden kann, oder auch von ihm empfängt, müssen nachrichtenspezifische Parameter vergeben werden. So kann die maximale Latenzzeit bei Nachricht *N1* vom Sender *id-11111* zu Empfänger *id-11112* anders sein, als die der Nachricht *N2* zu Empfänger *id-11113*. Die maximale Latenz eines Datenpakets definiert auch, wie lange dieses gültig ist. So hat Nachricht *N1* eine Gültigkeitsdauer von 200ms. Da Sender und Empfänger verschiedene Anforderungen an die Latenz haben können, müssen immer beide Seiten betrachtet werden und die geringere Zeit wird für die Kommunikation gesetzt.

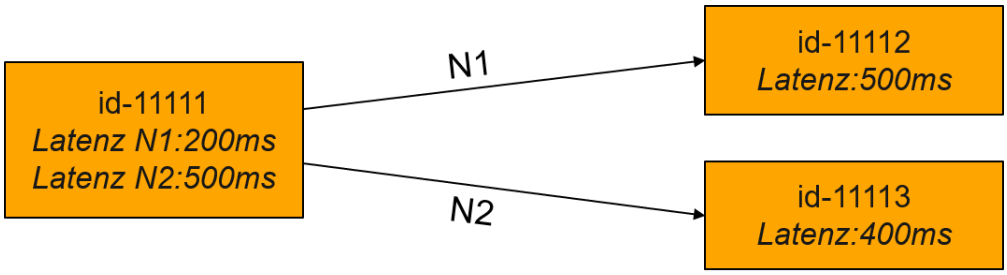

Abbildung 9: Nachrichtenparameter Latenz

Da *id-11111* die Nachricht an *id-11112* mit geringer Latenz sendet als benötigt, wird dies für *N1* als Wert gesetzt. Bei der Nachricht an *id-11113* wird eine niedrigere Latenz vom Empfänger erwartet, als *id-11111* sendet, somit muss die Latenzzeit der Nachricht *N2* auf 400 ms angepasst werden.

3.5.3 Arten der Nachrichten

Es gibt im System zwei verschiedene Arten von Nachrichten. Dies werden je nach Typ unter dem Parameter *cyclic* angeben.

- **Zyklischer Nachrichtenaustausch**
  Nachrichten die in fest definierten Zeitabständen von einer Applikation gesendet werden, bezeichnet man als zyklisch. Die Abstände werden mit der Frequenz einer Nachricht angegeben. Wenn eine zyklische Laufzeit angegeben wird, muss zusätzlich im System eine laufende Paketnummer für gesendete Daten vergeben werden, um Fehler wie wiederholte Nachrichten und vertauschte Reihenfolge der Pakete zu vermeiden.

- **Event-gesteuerter Nachrichtenaustausch**
  Sendet eine Applikation nur bei bestimmten Ereignissen eine Nachricht, wird dies als event-gesteuert bezeichnet. Genau so wie eine zyklische Nachricht, kann eine event-gesteuerte Nachricht auch eine Frequenz besitzen in der sie wiederholt auftritt. Der größe Unterschied liegt im Startzeitpunkt der Nachrichten.

3.5.4 Datenrate

Eine Applikation benötigt einen bestimmten Anteil der einer Leitung im Netzwerk. Die benötigte Datenrate kann mittels der beiden folgenden Parameter errechnet werden.

- **Nachrichtengröße**
  Die Nachrichtengröße definiert die Größe einer zu sendenden Nachricht an einen anderen Teilnehmer im System. Diese wird in *byte* unter der Variable *messageSize* abgelegt.

- **Frequenz einer Nachricht**
  Jede Nachricht wird in bestimmten Abständen nacheinander versendet. Dies wird unter *frequency* in Nachrichten pro Sekunde angegeben. Wie bereits in Abschnitt 3.5.3 beschrieben, gibt es jedoch auch Nachrichten ohne Frequenz, da diese nur einmalig gesendet werden. Hierbei ist die Frequenz mit dem Wert eins zu belegen.

Die Bandbreite pro Nachricht errechnet sich somit anhand der Formel:

$$Bandbreite = Nachrichtengroesse * Frequenz \tag{3.1}$$

Eine Applikation kann mehrere ausgehende Nachrichten versenden. Diese werden gesondert voneinander betrachtet um ein Aussage über die benötigte Bandbreite zu treffen. Im Manifest werden die verschiedenen Nachrichten einzeln angegeben und besitzen alle ihre eigenen Nachrichtenparameter, wie die ID, den Empfänger, die Nachrichtengröße und die Frequenz. Falls eine Anwendung auch event-gesteuerte Einzelnachrichten sendet, empfiehlt es sich, einen Anteil der Bandbreite der Leitung frei zu halten. Wird dies nicht gemacht, kann es zu Datenstau und den daraus resultierenden Paketverlust kommen. Dies gilt es zu berücksichtigen und zu vermeiden. Daraus wird eine ausreichend genaue Abschätzung der benötigten Datenrate ermöglicht.
Wenn mehrere Anwendungen auf einer ECU laufen, kann anhand ihrer Einzelbandbreiten die gesamte Auslastung einer Leitung berechnet werden. Mithilfe der Gesamtauslastung kann die benötigte Bandbreite einer Applikation zugesichert werden, damit diese auf einem Steuergerät laufen kann.

3.5.5 Latenz und Jitter

Die Latenzzeit (Kapitel 2.2.3) wird angegeben mit dem Parameter *maxLatency*. Dieser gibt, wie bereits beschrieben, die maximale Dauer bis zur Ankunft einer Nachricht im System an. Um die gewünschte Zeit zu erreichen, müssen Technologien wie Scheduler und Traffic Shaper (Kapitel 2.5.1) eingesetzt werden. So können aus der Latenz benötigte Parameter abgeleitet werden, wie beispielsweise nach welchen Kriterien der Shaper aus dem Traffic Shaping arbeiten soll. Dieser kann in den je nach Switchhersteller individuell definiert werden. Je kleiner die geforderte Latenzzeit, desto strenger muss der Shaper eingestellt sein, um *Burts* (dt. Ausbrüche im Datenverkehr) zu vermeiden.

Kann ein System die geforderte zeitliche Anforderung einer Anwendung nicht einhalten, kann diese Applikation nicht in dem Netzwerk betrieben werden. Aus diesem Grund sollte bereits bei der Entwicklung einer Applikation die gegebenen Eigenschaften wie die Topologie des Netzwerkes und auch die Leistungsfähigkeit der Endgeräte des Systems betrachtet werden.

Wie bereits in Abschnitt 2.2.4 erwähnt, wird die Schwankung (eng. Jitter) als die maximale Abweichung der Ankunftszeit einer Nachricht bezeichnet. Diese kann sowohl positiv als auch negativ sein, da es ein Zeitfenster mit einer oberen und unteren Schranke angibt. Die Differenz zwischen ober und unter Grenze ergibt den Jitter. Im Manifest wird die Latenz (*maxLatency*) als obere Schranke benutzt. Die untere Schranke ist im Parameter *minLatency* definiert. Das Zeitfenster des Jitters berechnet sich mit der Formel:

$$Jitter = |maximaleLatenz - minimaleLatenz| \tag{3.2}$$

Das errechnete Zeitfensters gibt an, in welchem zeitlichen Bereich die Nachricht beim Empfänger ankommen muss. Ist keine minimale Latenz angegeben, reicht es lediglich die maximale Latenz als Obergrenze der Ankunftszeit zu setzen. Dies bedeutet, dass eine Nachricht lediglich innerhalb einer bestimmten Zeit ankommen muss. Es ist jedoch egal ob sie früher ankommt. Durch Sicherstellung, das der Parameter Jitter eingehalten wird, kann ein zu frühes Ankommen der Daten verhindert werden, da hier eine Untergrenze für das eintreffen einer Nachricht existiert.

3.5.6 Zeitsynchronisation

Die Eigenschaft Zeitsynchronisation (eng. time synchronisation) ist notwendig um einen gemeinsamen Zeitbasis über verschiedene Anwendungen zu garantieren. Die Synchronisation der Applikationen kann über unterschiedliche Mechanismen erfolgen. In dieser Arbeit wird auf generalized Precision Timing Protocol (gPTP) eingegangen, welches unter der IEEE 802.1AS [14] veröffentlicht wurde. gPTP nutzt Teile von Precision Time Protocol (PTP) welches ebenfalls von der IEEE unter der Nummer 1588 standardisiert [32]. Das Prinzip von gPTP stützt sich auf die Synchronisierung aller sich im System b befindenden Uhren an deiner Master Clock (dt. Uhr). Welche Uhr die Master Uhr ist wird in der Regel durch den Best Master Clock (BMC) Algorithmus ermittelt. Hierbei übermitteln alle Teilnehmer an der Zeitsynchronisation die Genauigkeit ihre internen Uhr. Die genauest dieser Uhren wird dann die neue Best Master Clock [27].

Generell gibt es vier Level an Uhren, die im gPTP Standard festgelegt sind. Im Manifest werden zwei dieser Level umgesetzt. Ein zusätzlich eingeführtes Level 0 wird in dieser Arbeit als Deaktivierung der Zeitsynchronisation genutzt [38]. Dies kann gesetzt werden wenn zeit-unkritische Daten, wie z. B. Updates für das Fahrzeug in einer Werkstat, im System versendet werden. Level eins und zwei geben den Typ der Uhr an. Level drei ist laut Standard für IEEE 1588 reserviert und wird hier nicht mit umgesetzt. Dies entspricht auch dem *Master und Slave Port* Prinzip aus der Literatur [27].

- Level 0: Keine Zeitsynchronisation erforderlich.
- Level 1: Interne Uhr der ECU auf der die Applikation laufen soll, muss so genau sein, das im Falle eines BMC Algorithmus diese als neue Master Clock verwendet werden kann.
- Level 2: Interne Uhr der ECU synchronisiert sich auf eine Uhr aus Level 1 und sollte keine Master Clock werden.
- Level 3: Reserviert für IEEE 1588

Um eine Genauigkeit einer Anwendung mit anzugeben können verschiedene Uhrentypen angegeben werden. Die für die Arbeit wichtigen, sind in der Tabelle 3.3 aufgelistet. Es wird ebenfalls mit angegeben in welches Level dies Uhren fallen. Generell sollte Level eins nur verwendet werden, wenn ein Anspruch auf sehr hohe Genauigkeit besteht oder die Anwendung der Master für andere Uhren im System werden soll.

Uhrentyp	Spezifikation	Level
ATOM	Zeit von kalibrierter Atomuhr, auf 25ns genau	Level 1
GPS	Zeit von GPS Uhr, auf 100ns genau	Level 1
ATOM	Zeit synchronisiert auf eine Atomuhr Level 1, auf 100ns genau	Level 2
GPS	Zeit synchronisiert auf eine GPS Uhr Level 1, auf 100ns genau	Level 2
NTP	Zeit ist nach der Synchronisation auf 50ms genau	Level 2
HAND	Zeit ist nach der Synchronisation auf 10s genau	Level 2

Tabelle 3.3: Genauigkeit der Uhren im Netzwerk [38]

Im Manifest werden die gewünschten Einstellungen durch Konkatenation der einzelnen Angaben abgebildet. Dies wird im Parameter *timesync*, nach dem Schema Prefix=Level und Sufix=Uhrentyp, abgebildet. So wird z. B. eine Anwendung die nur eine Genauigkeit von maximal 100ns und keinen Anspruch darauf legt, selbst eine Master Clock zu werden, in Form des Stringes *timesync = 2ATOM* angegeben.

3.5.7 Sicherstellung der Authentizität der Daten

Bei der Sicherstellung der Echtheit der Daten, gilt es die in Kapitel 2.7 erwähnten Schutzziele zu gewährleisten. Dies kann im aktuellen Netzwerk nur anhand von Media Access Control Security umgesetzt werden, da nur dieser Mechanismus in den verwendeten Switchen implementiert ist. MACsec bietet die in Abschnitt 2.1.2 definierte Funktionalität, welche beispielsweise eine Integrität der Daten sicher stellt. Auch eine Authentifizeirung der Kommunikationspartner ist bereits anhand der MAC Adressen möglich, da diese im System eindeutig sein müssen. Die weiteren genannten Schutzziele können mit dieser Technologie noch nicht erreicht werden. Diese müssen im Nachgang noch spezifisch betrachtet werden, um die Machbarkeit im jeweiligen System sicherzustellen. Eine Möglichkeit Vertraulichkeit zu gewährleisten, wäre eine Verschlüsselung der Kommunikationsstrecke. Dies ist jedoch nur eine Möglichkeit und wird nicht weiter in dieser Arbeit verfolgt.

Der Parameter *integrity* ist lediglich eine boolische Variabel und gibt dem Entwicklern die Möglichkeit die Kommunikation zwischen den Endgräten zu überwachen. Ist dieses sog. Flag gesetzt, muss mit Performance-Einbusen im System gerechnet werden, da es einerseits Zeit kostete, aber auch Rechenleitung in den Netzwerkgräten.

Kapitel

4

Automatisiertes Konfigurationsmanagement der Netzwerkgräte

Das Kapiel automatisierte Konfiguration behandelt den bereits erwähnten Netzwerkmanager aus Kapitel 3.2 und dessen Funktionalität. Diese besteht in erster Linie darin, die sich im Netzwerk befindenden Geräte in einem Automobil für benötigte Anforderungen von Applikationen zu konfigurieren. Die zweite Komponente des Systems ist der Applikations-Orchestrator, welcher in Kapitel 3.3 erklärt wurde und die Applikationsdaten an den Manager übermitteln soll.

Des Weiteren werden die im Kapitel 3 definierten Parameter in die notwendigen Eigenschaften des Systems überführt. Jeder Eintrag im Manifest kann wieder in Unterparameter aufgebrochen werden. Hierdurch wird die automatisierte Konfiguration des Fahrzeugnetzes ermöglicht.

4.1 Verwaltung der Anwendungen im System

Der Vorgang zum Hinzufügen einer Anwendung, welcher bereits in Abschnitt 3.3 beschriebene ist, wird jedoch nur eine der nötigen Funktionen des Orchestrators im Hinblick auf Applikationsverwaltung sein. Es soll nicht nur möglich sein, neue Anwendungen zu starten, sondern auch bereits laufende auf eine andere ECU umzuziehen oder diese endgültig zu beenden. Dies kann nötig sein, falls eine Kommunikationsstrecke bereits stark ausgelastet ist, wobei andere noch genügend Kapazität hätten, um einen reibungslosen Ablauf des Systems zu garantieren. In diesem Fall würde eine Applikation umgezogen werden.

Diese Funktionalitäten sind an das Prinzip der Representational State Transfer (REST) Technologie angelehnt [7]. Dies zeichnet sich in erster Linie durch ihr vordefiniertes Aktionsprinzip *CRUD*, was für *create, retrieve, update und delete* steht, aus. Drei der Funktionen können in den Prozess des Orchestrators überführt werden:

- *Create*, eine neue Anwendung wird im System gestartet.
- *Update*, der Status einer laufenden Anwendung im System soll geändert werden. Dies kann beispielsweise ein Umzug auf eine andere ECU sein. Auch das Pausieren einer Anwendung, ohne sie endgültig aus dem System zu entfernen, kann so realisiert werden.
- *Delete*, eine laufende Anwendung soll im System beendet und entfernt werden. Die Applikation wird dabei vollständig aus dem System gelöscht.

Diese Arbeit behandelt lediglich das Hinzufügen einer neuen Applikation in ein bestehendes System. Die anderen Punkte müssen separat betrachtet werden, da sie ein Wiederherstellen des zuletzt gültigen Netzwerkzustands bedingen. Hierzu gibt es verschiedene Ansätze, wie das Speichern der letzten Konfigurationen vor einer Erweiterung des Netzwerkes oder das komplette Neukonfiguration des Systems. Die Ausarbeitung dieser Prinzipien würde jedoch den Rahmen der Abhandlung sprengen und wird hier nur der Vollständigkeit halber erwähnt.

4.2 Feature Entwicklung

Die im Abschnitt 3.2 Netzwerkmanager angesprochene Features, kümmern sich um die automatisierte Verarbeitung der Parameter aus den Applikationsmanifest und die daraus resultierenden Ableitungen zur Konfiguration des Netzwerkes. In den folgenden Unterabschnitten wir jeder Eintrag aus dem Applikationsmanifest behandelt und definiert, welche Ableitungen von diesem gemacht werden können. Ein sich im Manifest befindender Parameter kann in mehreren Features verarbeitet werden. Die Funktionen splitten nicht nur Parameter auf, sie definieren auch Mechanismen, die eine Einhaltung von applikationsspezifischen Anforderungen garantieren.

Der Aufbau der einzelnen Unterabschnitten erfolgt nach einem vorgegebenen Schema erfolgen:

- Definition des Features
- Erläuterung durch Verwendungsmöglichkeiten oder Aktivitätsdiagramm

Es wird bei jedem Parameter nur auf die Ableitungen eingegangen, die auch im Automobil-Bereich umgesetzt werden. Wenn bei der Ableitung der Parameter eine bestimmte Technik gefordert wird, die eine ECU oder Switch gewährleisten muss, wird der Einfachheit halber auf das Abprüfen ob das Endgerät diese Funktion besitzt im Diagramm verzichtet. Dies wird vor jeder Konfiguration der Geräte sowieso überprüft und liefert eine Abbruchbedingung zurück, falls die Technologie nicht zur Verfügung stehen sollte.

4.2.1 Feature Kommunikationsstrecken Aufbau

Eine Grundvoraussetzung für die Verständigung zwischen Endgeräten ist ein Aufbau einer Kommunikationsstrecke unter den Teilnehmern. Dies bedeutet eine Konfiguration aller Ports die ein Versendete Nachricht passieren muss. Je nach Anforderung aus dem Applikationsmanifest und der übermittelten Informationen aus dem Orchestrator, muss eine geeigneter Weg im System gefunden werden. Die Parameter *id, groupId, domain* und der MAC-Adressen der einzelnen Teilnehmer, können verschieden Einstellungen an den ECU's und Switchen vorgenommen werden. Nicht alle haben die selbe Technologie was bedeutet, dass endgerätspezifisch entschieden werden muss, welche Konfigurationen möglich sind.

Mögliche Verwendungen wären beispielsweise:

- Portforwarding, was eine Weiterleitung an einen anderen Port eines Endgerätes

bewirkt

- Ingress Filtering, ist eine Art Firewall und schützt vor Unerlaubter Kommunikation (Definition in Abschnitt 5.1)

Hauptsächlich dienen die Port-Konfigurationen zum reibungslosen Datenaustausch. Sie können aber auch zu einen gewissen Teil zum einhalten der in Abschnitt 2.7 definierten Schutzziele beitragen. Durch definierter Portregelungen wird ein unerlaubter Datenaustausch erschwert.

4.2.2 Feature Echtzeitfähigkeit

Um das Feature Echtzeitfähigkeit umzusetzen, werden mehre Parameter benötigt. In *maxLatency* ist die geforderte Latenzzeit angegeben, dies ist eine der wichtigsten Eigenschaften für eine Anwendung in einem Netzwerk. Aus dieser können verschieden Einstellungen gefolgert werden. Das zweite Attribut ist die *minLatency*. Dieser ergibt, wie in Abschnitt 3.5.5 beschrieben, zusammen mit der *maxLatency* den geforderten Jitter. Ein weiterer Parameter ist *timesync*, dieser wird zur Konfiguration des Time Aware Shaper (TAS) benötigt. Da sich die Anforderung nach TAS erst bei der Auswertung der Parameter ergeben kann, muss die Zeitsynchronisation nachträglich konfiguriert werden, falls diese nicht von Anfang an gesetzt ist.

Im folgenden Aktivitätsdiagramm wird grafisch veranschaulicht, wie der Netzwerkmanager anhand der Parameter entscheidet, ob eine Applikation im System laufen kann. Der erste Schritt ist die Prüfung der *maxLatency*. Die Latenz darf nicht kleiner als 100µs sein, da dies physikalisch nur bei einer Punkt-zu-Punkt Verbindung möglich ist . Diese Art der Verbindung bedingt jedoch keine Konfiguration des Netzwerkes, da dies statisch vordefiniert ist. Als nächstes muss die generelle Zeitrelevanz einer Anwendung geprüft werden. Ist die *maxLatency* größer als 200ms, kann die Applikation als zeitunrelevant betrachtet werden. Resultierend daraus, wird lediglich eine Leitung gesucht, bei der das Ankommen der Daten sicherstellt ist.

Ist Zeitrelevanz jedoch nötig, wird als nächstes geprüft, ob ein Jitter angegeben ist. Dies wird über die *minLatency* > *0* geregelt. Ist diese Null, kann an dieser Stelle der Vorgang abgebrochen werden, um das aktuelle Netzwerk zu überprüfen. Im Falle das, das System bereits ausreichend ist, wird schon die gewünschte Bestätigung zurückgegeben. Falls das aktuelle Netzwerk nicht ausreichend ist, weil beispielsweise die Latenz nicht eingehalten werden kann, muss es neu konfiguriert werden. So kann eine Latenz von kleiner 2ms den Credit Based Shaper zur Folge haben. Ist eine *minLatency* jedoch gesetzt, bedingt dies den Einsatz eines Time Aware Shaper. Welcher wiederum Zeitsynchronisation bedingt.

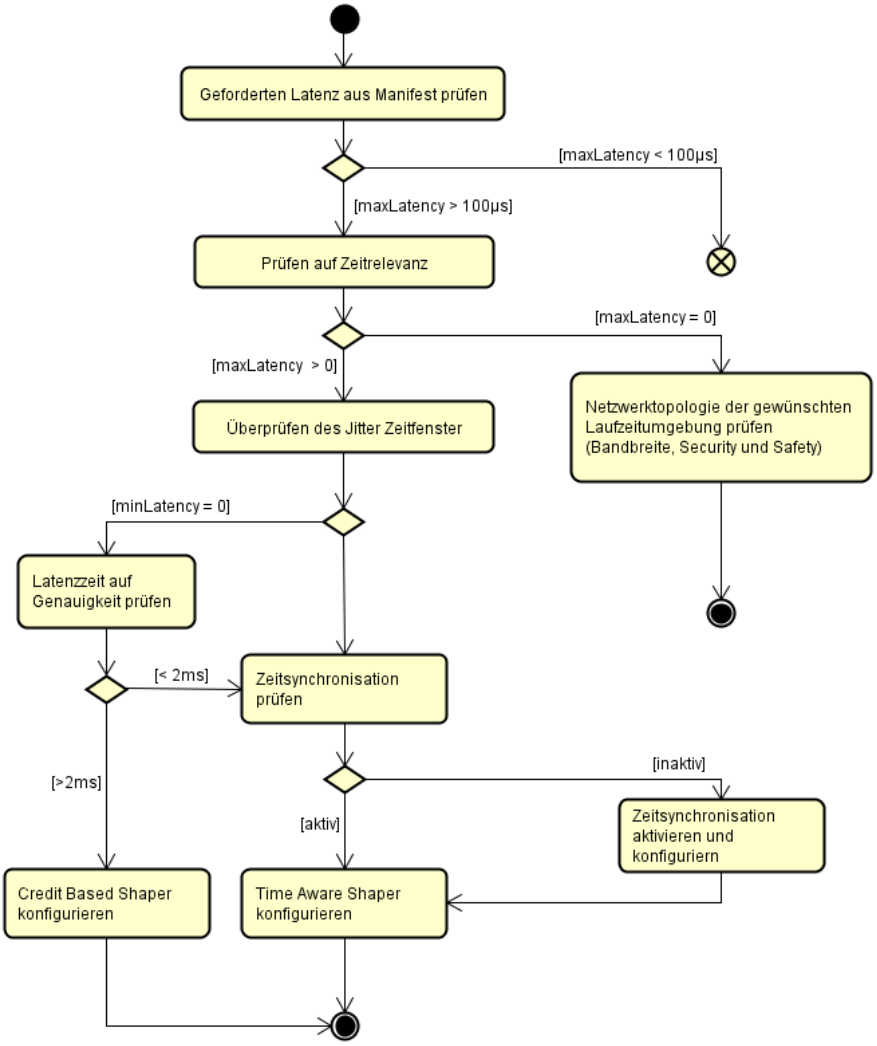

Abbildung 10: Aktivitätsdiagramm Echtzeitfähigkeit

4.2.3 Feature Datenrate

Die bereits in Abschnitt 3.5.4 erwähnte Datenrate besteht aus zwei Parametern. Das folgende Aktivitätsdiagramm zeigt den Ablauf einer Prüfung des aktuellen Netzwerkes. Diese wird anhand der Datenrate aller laufenden Prozesse durchgeführt. Reicht die restliche Bandbreite nicht mehr aus, wird eine Rekonfiguration des Netzwerkes versucht. Erst wenn diese nicht ausreichend ist, wird die Anwendung abgelehnt. In den meisten Fällen wird eine Leitung nie vollständig ausgelastet, um im Falle von *Bursts* keinen Datenverlust zu haben. Als *Bursts* werden Schwankungen in der Datenrate bezeichnet. Dies können zu einer kurzzeitigen Überlastung der Leitung führen.

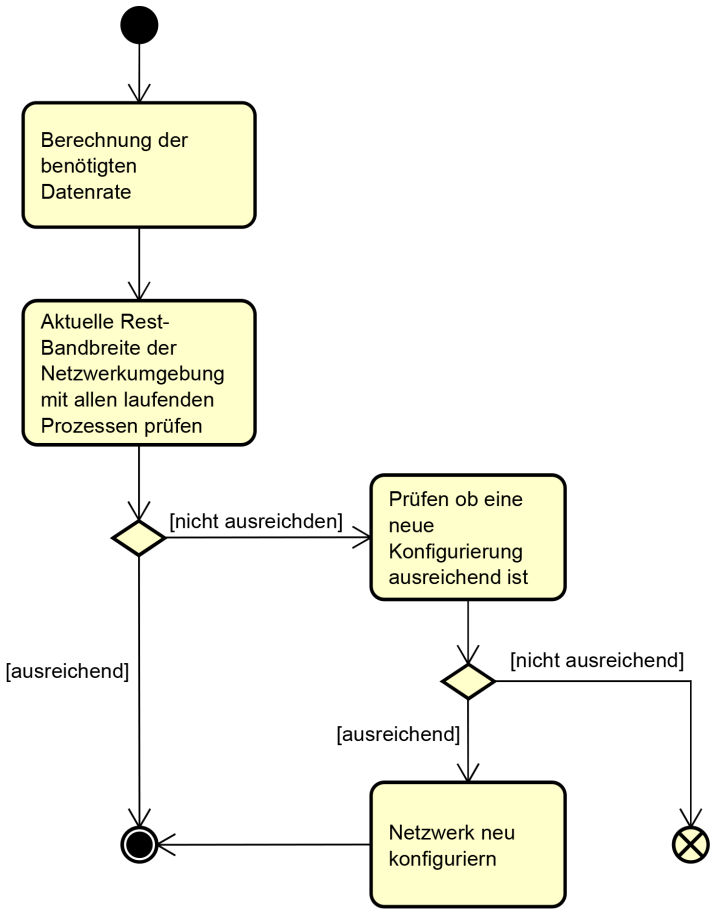

Abbildung 11: Aktivitätsdiagramm Datenrate

4.2.4 Feature Zeitsynchronisation

Die in Abschnitt 3.5.6 erwähnte Zeitsynchronisation ist nicht nur für die Echtzeitfähigkeit wichtig, sonder auch für einen synchronisierten Datenaustausch. Um sicher zu stellen, dass Daten aktuell sind, wird beispielsweise ein gemeinsamer Zeitstempel verwendet. Das Feature Zeitsynchronisation verbindet die Applikation mit einer Master-Clock im System. Ist keine passende Uhr vorhanden, können Mechanismen wie der Best-Master-Clock-Algorithmus angewendet werden [36]. Dieser sucht im System die exaktest Uhr und setzt dies als Master-Clock. Im aktuellen Netzwerk ist die jedoch nur eine Backup-Lösung bei einem Ausfalls einer Master-Clock gedacht, da dies momentan manuell vordefiniert ist. Welche Genauigkeit ein Uhr haben muss, wird durch das Manifest vorgeben und wird lediglich mit ausgelesen. Falls die aktuelle Master Clock im System den Ansprüchen nicht mehr genügt, muss ebenfalls ein neuer Master gesucht werden. Das Diagramm zeigt den Ablauf der Findung einer neuen Master Clock.

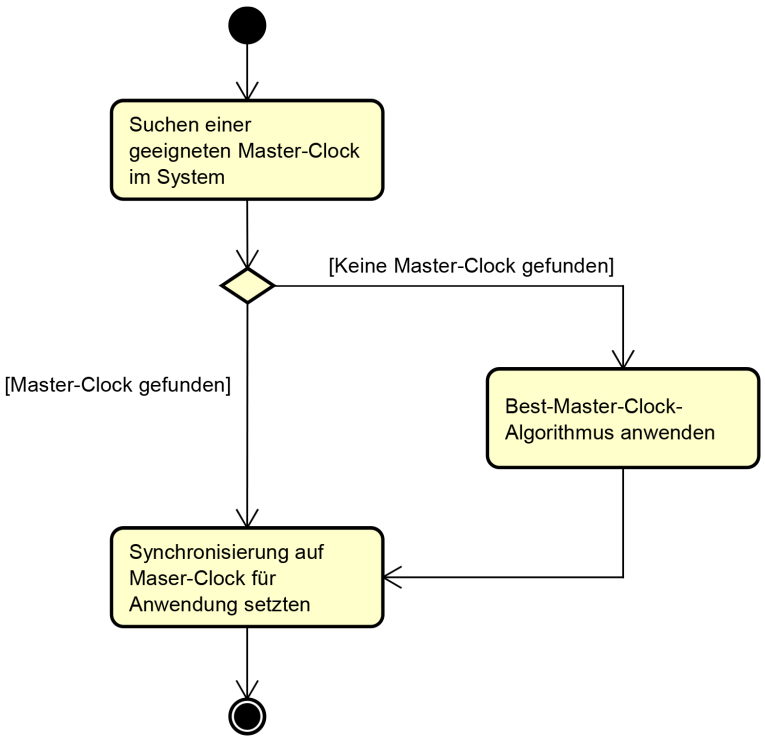

Abbildung 12: Aktivitätsdiagramm Zeitsynchronisation

4.2.5 Feature Authentizität

Das Feature Integrität soll den unbefugten Zugriff auf kritischer Daten verhindern. Im Manifest ist es in Form eines Flags modelliert. Der Wert in *integrity* entspricht einem boolischen Wert im Bereich 0 und 1. Dies ist gleichzusetzen mit einem ein- und ausschalten der Funktionalität. Falls das Flag duch eine 1 gesetzt ist, werden Sicherheitsmechanismen wie MACsec (Abschnitt 2.1.2) im System konfiguriert. So wird MACsec bei allen Teilnehmern der Kommunikationsstrecke an den jeweils verwendeten Port konfiguriert. Es gibt jedoch Endgeräte die die Technologie noch nicht unterstützen, dies muss bei der Konfiguration beachtet werden. Im nachfolgen Diagramm wird der Prozesses für alle beteiligten Netzwerkgeräte dargestellt.

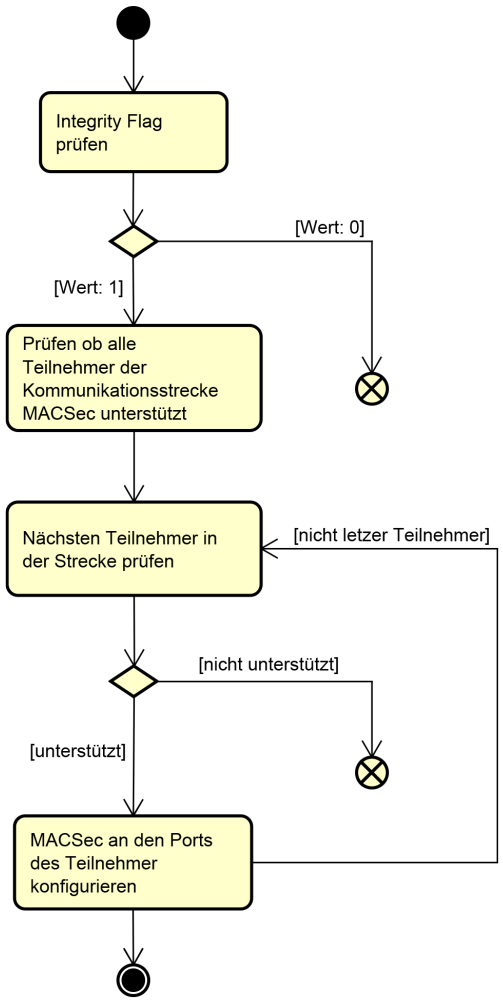

Abbildung 13: Aktivitätsdiagramm Integrität

Kapitel

5

Implementierung der Features

Das Ergebnis aus der dem Entwurf des Maifestes wird eine erste Implementation eines Testszenarios sein. Dies ist ebenfalls Bestandteil dieser Arbeit und wird mittels einiger der in Kapitel 4.2 definierten Features umgesetzt. In dieser Arbeit werden nicht alle genanten Features umgesetzt, sondern lediglich ein Szenario im System, welches bereits softwareseitig umsetzbar und testbar ist. Die benötigten Klassen hierzu werde weitestgehend neu implementiert, lediglich Schnittstellen sind bereits im System enthalten und werden als gegeben betrachtet.

Einen generellen Überblick aller Relevanten Komponenten und deren Kommunikationsschnittstellen, sog. Interfaces, werden in Folgenden Komponentendiagramm kurz dargestellt. Dies soll die Verbindungen zwischen den Komponenten, sowie die Vernetzung innerhalb der Unterkomponenten vereinfacht darstellen.

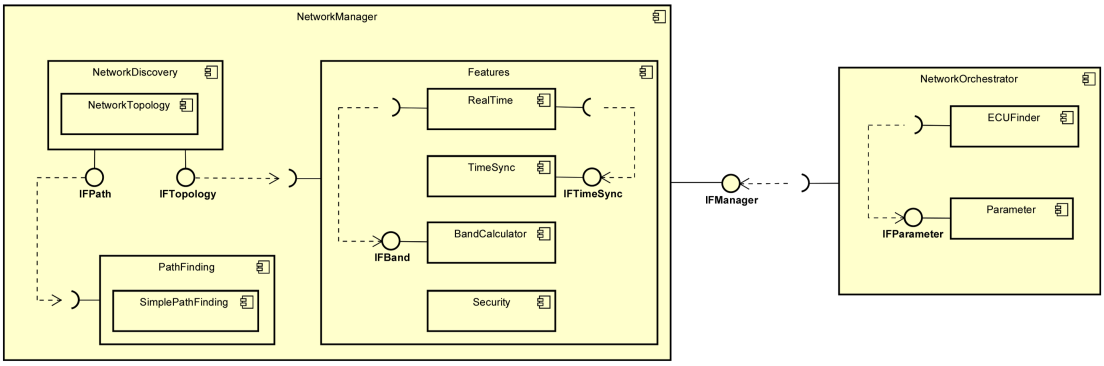

Abbildung 14: Komponentendiagramm Netzwerkmanager

Da besonders im Bereich Autonomes Fahren ein hoher Anspruch an die Ausfallsicherheit für eine Anwendungen im System gilt, werden im Nachgang noch mehrere Unit-Tests durchgeführt. Anhand dieser kann die Software auf verschiedene Eingabefälle geprüft werden. In Unit-Test werden isoliertet Softwareeinheiten, wie einzelne Klassen, mittels definierter Eingabe geprüft, um eine zuvor festgelegte Ausgabe zu erzeugen. Nur wenn dieses Ergebnis mit der gewünschten Ausgabe übereinstimmt, wird der Test als bestanden gewertet. Dies garantiert eine möglichst vielschichtige Abdeckung an Testfällen. Unit-Test geben aber keine Auskunft über das Zusammenspiel von Klassen über Schnittstellen, dies muss im Nachgang noch manuell getestet werden.

5.1 Testfall Ingress Filtering

Der gewählte Testfall bezieht sich auf das Ingress-Filtering eines Datenstroms, welches bereits in Abschnitt 2.5.3 kurz angesprochen wurde. Dieser Filter wird empfängerseitig an den jeweiligen Ports, der sich in der Kommunikationsstrecke befindenden Geräten, konfiguriert. Es handelt sich dabei um eine Filterung auf Layer 2 Eben. Ingress-Filterring ist ähnlich aufgebaut wie eine Firewall. Im Initialzustand sind alle Kommunikationen verboten. Erst wenn Regeln definiert werden, die das ein- oder ausgehen von Nachrichten erlauben, kann eine Kommunikationsstrecke aufgebaut werden. So werden bei einer Ende-zu-Ende Kommunikation lediglich die beiden Endpunkte zugelassen, die auch in den Regeln festgelegt sind. Hierbei werden die Partner anhand ihrer MAC-Adressen in den Regeln definiert. So wird sichergestellt, dass nur die gewünschten Teilnehmer die Datenpakete empfangen und versenden können.

Für die Regelung der Kommunikation haben sich hauptsächlich zwei Verfahren etabliert.

- *Application Whitelisting*
  Dieses Verfahren verbietet zuerst jegliche Kommunikation. Es müssen explizite Regeln für den Nachrichtenaustausch angelegt werden, um diese zu erlauben. In einem Netzwerk mit vielen Teilnehmern, bei denen nur wenige untereinander eine Verbindung aufbauen dürfen, ist dies ein vielversprechender Ansatz. Es werden wenige Regel benötigt, da Verbote nicht definiert werden müssen.

- *Application Blacklisting*
  Beim *Blacklisting* ist grundsätzlich jegliche Verbindung zugelassen. Ein Verbot der Kommunikation zweier Partner kann nur durch eine Regel definiert werden. Dieser Ansatz macht nur Sinn, wenn es sehr viel Netzwerkteilnehmer gibt, bei denen die meisten untereinander Kommunizieren dürfen. Es müssen somit nur wenige Regeln für Verbote definiert werden.

Für dies Arbeit wird erste Ansatz verwendet, da es eine überschaubare Anzahl an Kommunikationspartnern gibt, die wiederum aus Sicherheitsgründen nicht alle untereinander kommunizieren dürfen. Ein weiter Vorteil des Ansatzes ist, dass durch das explizite Erlauben der Kommunikation das Fehlerrisiko einer Verfälschung der Daten durch andere Teilnehmer minimiert wird. Dies kann durch einen anderen Netzwerkteilnehmer erfolgen, der die Nachricht auch empfangen konnte und sie abgefälscht weiterleitet. Ein Verfälschen der Nachrichten kann sowohl mutwillig durch eine manipulierten Netzwerkteilnehmer, als auch unwissentlich durch einen Defekt erfolgen. Beide Fälle müssen im System abgefangen werden, da durch sie ein Risiko entsteht.

5.2 Konfiguration der Endgeräte

In der Umsetzung der Arbeit wird sich auf die Konfiguration der Switche in einer Kommunikationsstrecke beschränkt, weshalb in diesem Fall ein Endgerät gleichzusetzen ist mit einem Switch. Generell gibt es zwei Möglichkeiten der Konfiguration der Endgeräte. Die erste ist eine Erstellung einer Konfigurationsdatei im Extensible Markup Language (XML) Format, welche alle generierten Einstellungen für den jeweiligen Switch enthält. Durch aufspielen dieser Datei auf ein Endgerät, kann dieser neu konfiguriert werden. Die zweite Möglichkeit, auf die sich auch dies Arbeit bezieht, ist das Nutzen eine im Projekt eigens für die Entwicklung geschriebene Library, welche alle Funktionen zur Konfiguration eines Endgerätes enthält. Diese wurde bereits im Vorfeld entwickelt und soll nun im Netzwerkmanager integriert und gesteuert werden. Hierzu wird von den in Abschnitt 4.2 definierten Featuren, eine Liste an Instruktionen erzeugt, welche am Ende einer Konfigurationsfindung von der Funktionalität der Library abgearbeitet werden soll.

5.3 Umsetzung der Parametereinlesung

Wie in Kapitel 4 bereits erwähnt, bekommt der Netzwerkmanager seine Parameter mittels JSON File übermittelt. Dies muss im ersten Schritt ausgelesen werden, und den entsprechenden Variablen im System zugeordnet werden, um im Nachgang damit zu arbeiten. Hierzu wurden die Klassen *json_translator* und *in_car_application* implementiert. Diese ermöglicht die Konvertierung der JSON-Parameter in die benötigten Variablentypen, welche in der Klasse *in_car_application* definiert sind. Der *json_translator* ist auch in der Lage zu erkennen ob eine Anwendung mehrere Nachrichten versenden will und berechnet die Gesamtbandbreite der Applikation. Dies ist wichtig, damit der Netzwerkmanager später entscheiden kann, ob noch genügend Bandbreite für die Applikation vorhanden ist.

In der Klasse *http_connector* ist die in Abschnitt 4.1 erwähne Technologie zum hinzufügen einer neuen Applikation implementiert, wird die Methode zum *adden* (dt. hinzufügen) einer neuen Anwendung zum System aufgerufen, starte diese das Einlesen der Parameter über den *json_translator* und gibt als Rückgabewert ein Objekt vom Typ *InCarApplication* zurück. In dem Objekt sind alle im Applikationsmanifest definierten Parameter abgelegt und können im sog. Netzwerkstatus (Klasse *netwerk_flow*) jederzeit von andern Klassen abgefragt werden. Für den weiteren Verlauf der Arbeit kann davon ausgegangen werden, dass eine Applikation vollständig in einem Objekt der Klasse *in_car_application* abgebildet ist.

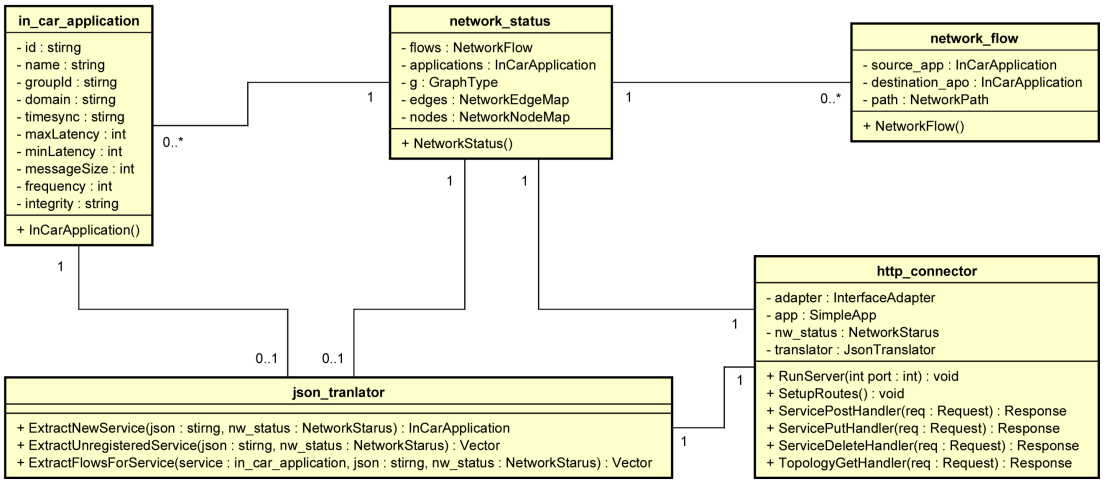

Abbildung 15: Klassendiagramm Parametereinlesung

Der bereits erwähnte Netzwerkstatus spiegelt den aktuelle Stand des Netzwerkes wieder. So sind dort beispielsweise Datenflüsse, Laufende Anwendungen und einige weitere Parameter enthalten. Der Status kann somit jederzeit von verschiedensten Funktionen abgefragt werden. Der Zusammenhang zwischen den verschieden Klassen die für die Parametereinlesung verantwortlich sind, ist im Klassendiagramm Abb. 15 dargestellt. Aufrufe von außen, wie beispielsweise durch den Orchestrator, erfolgen über den *http_connector*.

5.4 Umsetzung der Parameterauswertung mittels Feature

Nachdem die Parameter in den entsprechenden Variablen hinterlegt wurden, beginnt der Netzwerkmanager nacheinander die Features aus Abschnitt 4.2 abzuarbeiten. Diese liefern die gewünschten Konfigurationsbefehle für den Switch in Form einer Befehlsliste zurück. Im Testfall Ingress Filtering wird mit dem Feature zum Aufbau der Kommunikationsstrecke begonnen. Dazu werden die vom Orchestrator übergeben ECU's als Start- und

Endknoten verwendet. Die Kommunikationsstrecke über die Daten der Applikation übermittelt werden, wird als Flow oder Datenflow bezeichnet. In einem Flow sind die sendende und empfangende Applikation als Anfang- und Endpunkt abgespeichert, anhand dieser kann später eine zuordnen zwischen Applikation, Kommunikationspartnern und Datenflow erfolgen. Mithilfe der Klasse *simple_path_resolution* werden mögliche Pfade zwischen den Endpunkten gesucht. Ist ein geeigneter Pfad für einen Flow gefunden, werden beide in einer Map mit *Key-Value Pairs* abgelegt. So kann ein Pfad eindeutig einer Datenflow zugeordnet werden.

Ein Pfad kann aus mehren Edges bestehen, welche wiederum eine Teilstrecke eins Pfades darstellt. Eine Edge entspricht einem Hop laut Netzwerkdefinition, d. h. dem Weg zwischen zwei Netzwerkgeräten. Edge sind notwendig, da ein Pfad nur Anfang- und Endgerät kennt, jedoch nicht alle Teilnehmer die ein Datenpaket beim Versenden passieren muss. Eine Edge hat ebenfalls eine Start- und Endkonten, diese werden im System als Nodes bezeichnet.

Nodes sind nichts anders als Geräte im Netzwerk wie beispielsweise ein Switch. In einem Node besitzen Attribute wie MAC-Adresse, Ports und einem *Key-Value Pair*, welches eine Verbindung zwischen Ports und Node herstellt. Dieses Mapping ist notwendig um später, in der Verarbeitung der Features, alle dem Pfad zugehörigen Nodes den entsprechenden Ein- und Ausgangsport zuordnen zu können, da ein Node in der Regel nicht nur einen, sondern mehrere Ports besitzt.

Um die Zusammenhänge der einzelnen Klassen zu verdeutlichen, wurde in Abb. 16 ein Klassenmodel erstellt. Diese enthält in den einzelnen Komponenten lediglich die für die Arbeit relevanten Variablen und Funktionen.

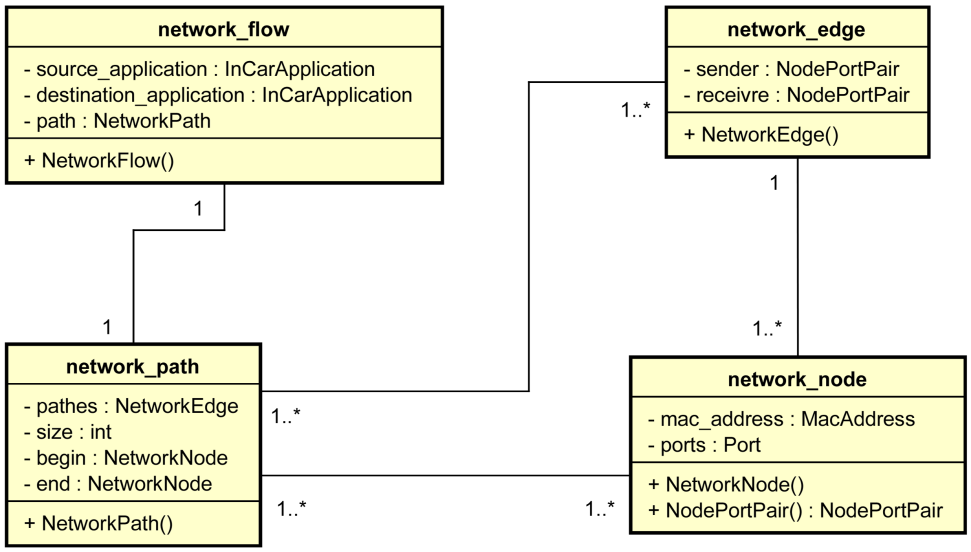

Abbildung 16: Klassendiagramm Netzwerkübersicht

Sobald die Pfadfindung abgeschlossen ist, werden aus der Klasse *feature_based_configuration_resolution* heraus, nacheinander die einzelnen Features aus Kapitel 4.2 aufgerufen. So wird das Feature Kommunikation, welches Ingress Filtering beinhaltet und in der Klasse *feature_communication_ingress_filtering* implementiert ist, ebenfalls hier durchlaufen. Dieses sucht, mittels der übergebenen Applikation, dem Flow-Pfad Mapping und dem aktuellen Netzwerkstatus den zugehörigen Pfad für die Anwendung. Ist dieser gefunden, wird über alle Edges des Pfades iteriert und für alle Nodes auf Empfängerseite die entsprechenden Portregelungen für das Ingress Filtering in der in Abschnitt 5.2 erwähnten Liste mit Instruktion abgelegt. Sobald alle Edges durchlaufen sind, gilt das Feature als abgearbeitet und es kann mit dem nächsten Feature begonnen werden. Sind alle abgearbeitet wird die Instruktion Liste an die Schnittstelle zur Konfiguration der Endgeräten übergeben. Dies konfiguriert mit den entsprechenden Funktionen, aus der erwähnten Library, das Endgerät. Im Fall Ingress Filtering werden die Portregelungen, wie Filtereigenschaften für Datenpakete, mit den entsprechenden Ein- und Ausgangsadressen der Netzwerkgeräte, übermittelt.

5.4.1 Testing der Implementation Ingress Filtering

Um sicher zu stellen das das Konzept der Features, zur automatischen Konfiguration, richtig arbeitet, müssen alle Teilelemente der Implementation mittel Unittests überprüft werden. Hierzu wurde zuerst eine Testklasse zu Parameter-Einlesung geschrieben, welche der Klasse *json_translator* ein Manifest im JSON Format übergibt und die zurückgegebenen Ist-Werte mit den zuvor definierten Soll-Werten vergleicht. Dies soll sicherstellen das alle Parameter aus den Manifest in den richtigen Variablen des Netzwerkmanager landen.

Die zweit Testklasse ist speziell für den Testfall Ingress Filtering implementiert. Es besitzt ein zuvor definierte Netzwerktopologie mit drei Netzwerkknoten. Anhand dieser wird der in Abschnitt 5.4 Prozess durchlaufen. Nach der Abbildung der Topologie in der Software, mittels Flows, Pfaden und Edges, wird die automatisierte Portregelungs-Konfiguration in der Klasse *feature_communication_ingress_filtering* angestoßen. Nach dem vollständigen Durchlauf des Features, wird die erstellte Instruktionsliste mit der erwarteten Ergebnis verglichen.

Beide Test konnten positiv bewertet werden, was sicherstellt das, das Konzept der automatischen Netzwerkkonfiguration auf diese Weise umsetzbar ist. Ein nächster Schritt ist es die Konfiguration auf eine realen Switch weiter zu leiten, um auch Hardwareseitig noch einige Tests durchführen zu können. Dies ist jedoch nicht mehr Teil dieser Arbeit, wird aber bereits im Projekt A3F weiter verfolgt.

Kapitel

6

Schlusswort

6.1 Zusammenfassung

Im Rahmen der vorliegenden Arbeit wurden bereits teilweise bestehende Netzwerkkomponenten in einem Fahrzeugnetzwerk optimiert und mit weiter Funktionalität erweitert. Ziel war es ein Möglichkeit zu entwickelt, einer Applikation die im Netzwerk gestartet werden sollte, verschiedene Applikationsparameter mit zu geben, anhand derer ein Fahrzeugnetzwerk automatisiert konfiguriert werden kann. Die Übergabe der Parameter soll mittels einem Applikationsmanifest geschehen, welches in Kapitel 3 der Arbeit eigens spezifiziert wurde. Anhand der Eigenschaften einer Anwendung wurde die zentrale Stelle des Netzwerk, der Netzwerk-Manager, funktionell so erweitert, dass er selbstständig Netzwerkkonfigurationen erstellen kann.

Die Erstellung einer Konfiguration soll mittels verschiedener Features-Erweiterungen umgesetzt werden, welche in Kapitel 4 erarbeitet wurden. So kann der Manager zukünftig beispielsweise automatisiert neu Pfade für die Kommunikation zweier Netzwerkteilnehmer festlegen. Durch die Übergabe der Anforderungen einer Applikation in Form eines Manifest, weiß der Netzwerk-Manager welche Konfigurationen für ein Netzwerk nötig sind und setzt diese mittels der Features um. Funktionen, wie die Berechnung der Datenrate und Echtzeitfähigkeit, sind unerlässlich für das neu Netzwerk, da nur so ein fehlerfreier Ablauf des Systems gewährleistet werden kann.

In Kapitel 5 wurden die bisher erarbeiteten Punkte erstmals softwaretechnisch umgesetzt. Aufgrund des definierten Umfang der Abschlussarbeit, wurde sich bei der Implementation auf einen Beispielfalles bezogen. Die Auswahl des Testfalles wurde unter Beachtung der technischen Umsetzbarkeit getroffen. Leider können zum jetzigen Zeitpunkt noch nicht alles im Projekt verwendeten Endgeräte die geforderten Technologien aus der Arbeit umsetzen. Aus diesem Grund wurde sich auf Ingress Filtering beschränkt, da dies bereits Funktionell umsetzbar ist. In der Arbeit waren softwareseitige Unit-Tests zur Gänze ausreichend, da durch die Features eine korrekte Liste an Befehlen erstellt wurde, die einen Netzwerkswitch konfigurieren können.

6.2 Fazit und Ausblick

Das Erstellen eines dynamischen Fahrzeugsnetzes für zukünftige E/E Architekturen, birgt große Herausfordungen und bedingt das Anwenden von bisher unbekannten Methoden. Diese Arbeit bildet einen Schritt hin zu einem Fahrzeugnetzwerk mit einer zentralem Manager, der remote die Netzwerkeinstellungen aller Netzwerkteilnehmer verwaltet und überwacht. Anhand der erarbeiteten Konzepte, wie QoS oder TSN, konnte ein Manifest erstellt werden, dass die wichtigsten Parameter für eine Applikation im Bezug auf ein Netzwerk enthält. Der Prozess der Parameterfindung wird in einem Netzwerkmanager für ein zukünftiges Fahrzeugnetzwerk eine zentrale Rolle einnehmen und stellt ein hochgradig nicht-triviales Problem dar, da dafür ein hohes Grad an Systemwissen notwendig ist und Konfigurationsparameter sinnvoll definiert werden müssen. Die kommenden Schritt werden die in Abhandlung erarbeiteten Erkenntnisse im Rahmen des A3F Forschungsprojekt softwareseitig umzusetzen. Parallel dazu wird in einer weiteren Forschungsarbeit ein Konzept zu Sicherstellung auf Korrektheit der Konfigurationsfindung begonnen. Dies wird notwendig sein, um eine Zulassung der Technologie nach den aktuellen Gesetzen zu ermögliche. Die Forschungsarbeit wird jedoch nicht der einzige Prozess sein, der hierzu nötig sein wird, sonder ist ein weiterer Schritt zur Etablierung der Technologie in neuen Fahrzeugnetzwerken.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] *Automatisiertes Fahren*. https://www.vda.de/de/themen/innovation-und-technik/automatisiertes-fahren/automatisiertes-fahren.html. Abgerufen: 07.06.2018.

[2] *Automatisiertes und vernetztes Fahren*. https://www.bmvi.de/DE/Themen/Digitales/Automatisiertes-und-vernetztes-Fahren/automatisiertes-und-vernetztes-fahren.html. Abgerufen: 08.08.2018.

[3] *Autonomes Fahren: Deutschland und China wollen die Marktführerschaft*. https://www.kfz-betrieb.vogel.de/autonomes-fahren-deutschland-und-china-wollen-die-marktfuehrerschaft-a-732277/. Abgerufen: 06.08.2018.

[4] *Bridges and Bridged Networks— Amendment 28: Per-Stream Filtering and Policing*. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8064221. Abgerufen: 05.08.2018.

[5] *Dan Des Ruisseaux. Designing a Deterministic Ethernet Network (Whitepaper): Industrial Communication. Hrsg. von Schneider Electric Industries SAS*. http://www.schneider-electric.co.uk/documents/solutions/process-automation/open-connectivity/Designing%20a%20Deterministic%20Ethernet%20Network.PDF. Abgerufen: 23.05.2018.

[6] *Die rapide Urbanisierung sorgt für Dynamik im Depot*. http://www.dasinvestment.com/neues-anlagethema-smart-cities-die-rapide-urbanisierung-sorgt-fuer-dynamik-im-depot/. Abgerufen: 06.08.2018.

[7] Doglio, Fernando: *Pro REST API Development with Node.js*. Springer-Verlag, Berlin Heidelberg, 1. Auflage, 2015, ISBN 978-1-4842-0918-9.

[8] *Echtzeitanwendungen mit Automotive Ethernet*. https://www.elektroniknet.de/elektronik-automotive/

bordnetz-vernetzung/echtzeitanwendungen-mit-automotive-ethernet-113002.html. Abgerufen: 20.06.2018.

[9] Eckert, Claudia: *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Walter de Gruyter GmbH & Co KG, Berlin, 9. Auflage, 2014, ISBN 978-348-685-916-4.

[10] *ECMA-404 The JSON Data Interchange Standard*. https://www.json.org/json-de.html. Abgerufen: 20.04.2018.

[11] *Heterogeneous Networks for Audio and Video: Using IEEE 802.1 Audio Video Bridging*. https://ieeexplore.ieee.org/document/6595589/. Abgerufen: 10.07.2018.

[12] *Heterogenes Netzwerk*. https://www.itwissen.info/Heterogenes-Netzwerk-heterogeneous-network.html. Abgerufen: 26.06.2018.

[13] *IEEE 802 Numbers*. https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml. Abgerufen: 30.05.2018.

[14] *IEEE 802.1AS-Rev*. https://1.ieee802.org/tsn/802-1as-rev/. Abgerufen: 16.04.2018.

[15] *IEEE 802.1CB*. https://1.ieee802.org/tsn/802-1cb/. Abgerufen: 16.04.2018.

[16] *IEEE 802.1CM*. https://1.ieee802.org/tsn/802-1cm/. Abgerufen: 16.04.2018.

[17] *IEEE 802.1Qbu*. http://www.ieee802.org/1/pages/802.1bu.html. Abgerufen: 16.04.2018.

[18] *IEEE 802.1Qbv*. http://www.ieee802.org/1/pages/802.1bv.html. Abgerufen: 16.04.2018.

[19] *IEEE 802.1Qca*. http://www.ieee802.org/1/pages/802.1ca.html. Abgerufen: 16.04.2018.

[20] *IEEE 802.1Qcc*. https://1.ieee802.org/tsn/802-1qcc/. Abgerufen: 16.04.2018.

[21] *IEEE 802.1Qci*. https://1.ieee802.org/tsn/802-1qci/. Abgerufen: 16.04.2018.

[22] *IEEE 802.1Qcr*. https://1.ieee802.org/tsn/802-1qcr/. Abgerufen: 16.04.2018.

[23] *IEEE 802.3*. http://www.ieee802.org/3/. Abgerufen: 09.04.2018.

[24] *IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Security*. http://www.ieee802.org/1/files/public/docs2010/new-seaman-1AE-markup-for-gcm-aes-256-0710-v2.pdf. Abgerufen: 30.05.2018.

[25] James F. Kurose, Keith W. Ross: *Computer networking: A top-down approach*. Pearson Education, Inc, New Jersey, 6. Auflage, 2013, ISBN 978-0-13-285620-1.

[26] *Kernel-based Architecture for Safety-critical Control*. https://pdfs.semanticscholar.org/9e71/081cbdddc54f83bdaedb743113f04b7d223b.pdf?_ga=2.225926232.335041009.1531333031-98898336.1531333031. Abgerufen: 13.06.2018.

[27] Matheus Kirsten, Thomas Königseder: *Automotive Ethernet*. Cambridge University Press, Cambridge, United Kingdom, 2. Auflage, 2017, ISBN 978-110-718-322-3.

[28] Meyer, Gereon: *Advanced Microsystems for Automotive Applications 2012*. Springer-Verlag, Berlin Heidelberg, 6. Auflage, 2012, ISBN 978-3-642-29672-7.

[29] Meyer, Philipp: *Extending IEEE 802.1 AVB with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic“*. IEEE, Boston, MA, USA, 1. Auflage, 2014, ISBN 978-1-4799-2687-9.

[30] Paul Goransson, Chuck Black, Timothy Culver: *Software defined networks: A comprehensive approach*. MA: Morgan Kaufmann, Cambridge, 2. Auflage, 2017, ISBN 978-012-804-555-8.

[31] Peter Mandl, Andreas Bakomenko, Johannes Weiß: *Grundkurs Datenkommunikation*. Vieweg+Teubner GWV Fachverlage GmbH, Wiesbaden, 1. Auflage, 2008, ISBN 978-3-8348-0517-1.

[32] *Precision clock synchronization protocol for networked measurement and control systems*. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4839002. Abgerufen: 07.06.2018.

[33] R. Ramaswamy, Ning Weng, T. Wolf.: *Characterizing network processing delay*. IEEE Global Telecommunications Conference, 2004, GLOBECOM ’04. IEEE, 1. Auflage, 2004, ISBN 0-7803-8794-5.

[34] *Road vehicles — Functional safety*. https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-1:v1:en. Abgerufen: 11.07.2018.

[35] Ross, Hans Leo: *Functional Safety for Road Vehicles*. Springer-Verlag, Berlin Heidelberg, 1. Auflage, 2016, ISBN 978-3-319-33360-1.

[36] *Specialization of IEEE 1588 Best Master Clock Algorithm to 802.1AS*. http://www.ieee802.org/1/files/public/docs2006/as-garner-bmc-060606.pdf. Abgerufen: 04.07.2018.

[37] Tanenbaum, Andrew S.: *Computer Network*. Pearson Education, Boston, 5. Auflage, 2011, ISBN 978-0-13-212695-3.

[38] *Use of IEEE 1588 Best Master Clock Algorithm in IEEE 802.1AS*. http://www.ieee802.org/1/files/public/docs2006/as-garner-use-of-bmc-061114.pdf. Abgerufen: 05.08.2018.