



Hochschule für Angewandte Wissenschaften Hamburg  
*Hamburg University of Applied Sciences*

# Masterthesis

Jonathan Wendeborn

Entwurf und Vergleich zweier Verfahren zur Analyse  
und Bewertung heterogener automobiler Netzwerke

*Fakultät Technik und Informatik  
Department Informations- und  
Elektrotechnik*

*Faculty of Engineering and Computer Science  
Department of Information and  
Electrical Engineering*

Jonathan Wendeborn

Entwurf und Vergleich zweier Verfahren zur  
Analyse und Bewertung heterogener  
automobiler Netzwerke

Masterthesis eingereicht im Rahmen der Masterprüfung im  
Masterstudiengang Informations- und Kommunikationstechnik  
am Department Informations- und Elektrotechnik  
der Fakultät Technik und Informatik  
der Hochschule für Angewandte Wissenschaften Hamburg

in Zusammenarbeit mit:

Vector Informatik GmbH  
Borsteler Bogen 27d  
22453 Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Aining Li

Zweitgutachter: Prof. Dr. Franz Korf

Industrieller Betreuer: Dipl.-Ing. Jörn Haase

Abgegeben am 25. August 2016

**Jonathan Wendeborn**

**Thema der Masterthesis**

Entwurf und Vergleich zweier Verfahren zur Analyse und Bewertung heterogener automobiler Netzwerke

**Stichworte**

System-Level-Simulation, Analyse, Bewertung, heterogene Netzwerke, Echtzeit-Ethernet, Time-Triggered-Ethernet, AVB, TSN, CAN, Gateway, OMNeT++, CANoe, Elektrik-/Elektronik-Architektur, Metrik, Latenz, Jitter, Datenübertragungsrate, Paketverlustrate, Puffer, Binlog, BLF, COM, .NET, CAPL

**Kurzzusammenfassung**

Durch die Einführung von Echtzeitfähigkeit bei Ethernet können neuartige Netzwerke entwickelt werden, welche neue Funktionen ermöglichen und Kosten sparen. Um die Tauglichkeit eines Netzwerks beurteilen zu können, muss eine kaum zu überblickende Datenmenge überprüft werden. Um die/den Netzwerkverantwortliche/n dabei zu unterstützen, entwickelt und untersucht diese Arbeit zwei Ansätze, wie ein in OMNeT++ simuliertes Netzwerk automatisiert analysiert und bewertet werden kann. Die Ergebnisse dieser Bewertungen werden in CANoe, einem in der Automobilindustrie weit verbreiteten Software-Werkzeug, dargestellt.

**Jonathan Wendeborn**

**Title of the paper**

Design and comparison of two methods for analyzing and evaluating heterogeneous automotive networks

**Keywords**

System Level Simulation, analysis, evaluation, heterogeneous networks, real time ethernet, time triggered Ethernet, AVB, TSN, CAN, gateway, OMNeT++, CANoe, Electric/Electronic architecture, metrics, latency, jitter, data transmission rate, packet loss rate, buffer, Binlog, BLF, COM, .NET, CAPL

**Abstract**

Introducing real time capabilities for Ethernet enables the usage of new functionalities as well as axing costs. Evaluating the serviceability requires reviewing a huge amount of data. To support the network engineer, this report develops and examines two approaches for automated analysis and evaluation of networks simulated in OMNeT++. The evaluation results are displayed in CANoe, a software tool that is popular in the automotive industry.

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>8</b>
1.1	Motivation	8
1.2	Zielsetzung	10
1.3	Verwandte Arbeiten	10
1.4	Gliederung dieser Arbeit	11
<b>2</b>	<b>Grundlagen</b>	<b>12</b>
2.1	Notation und Begriffe	12
2.1.1	Begriffe, Bussysteme und Protokolle	12
2.1.2	ISO-OSI-Referenzmodell	12
2.2	Netzwerksimulation im Automobil mit OMNeT++	13
2.3	Datenübertragung im Automobil	16
2.4	Bewertung von Netzwerken	17
2.5	Software-Werkzeug CANoe	20
2.6	Binary-Logging-Format (BLF) in CANoe	22
2.7	Entwicklung einer Elektrik-/Elektronik-Architektur	22
<b>3</b>	<b>Konzeption</b>	<b>25</b>
3.1	Analyse der Vorgaben	25
3.2	Betrachtung der Programme OMNeT++ und CANoe	27
3.3	Ansätze zur Bewertung in OMNeT++ simulierter Netzwerke	30
3.3.1	Erhebung der Metriken in OMNeT++	30
3.3.2	Erhebung der Metriken in einer externen Applikation	32
3.3.3	Erhebung der Metriken in CANoe	33
3.4	Vergleich der Konzepte	35
3.5	Auswahl und Konkretisierung der finalen Konzepte	37
<b>4</b>	<b>Umsetzung und Herausforderungen</b>	<b>40</b>
4.1	Erstes Konzept: Erhebung der Metriken in OMNeT++	40
4.1.1	Implementierung des Vector-Managers	40

4.1.2	Nutzung des BlfRecorders zur Aufzeichnung von Nachrichten-Ereignissen ..	42
4.1.3	C#-Applikation „Compagnon“ .....	43
4.1.4	BinlogNET .....	47
4.2	Zweites Konzept: Erhebung der Metriken in CANoe .....	50
4.2.1	OMNeT++ .....	50
4.2.2	C#-Applikation Compagnon .....	52
4.2.3	Gliederung der Systemvariablen .....	53
4.2.4	CANoe.....	57
4.3	Herausforderungen und Probleme .....	60
4.3.1	Erweiterung für Multicast/Broadcast .....	60
4.3.2	Erhebung der Latenzzeiten während der Weiterleitung der Nachricht in OMNeT++ .....	61
4.3.3	Detektieren von Gateway-Nachrichten in CANoe .....	63
4.3.4	Berechnung der Paketverlustrate .....	64
4.3.5	Fehlerhafte Serialisierung von TSN-Nachrichten .....	65
4.3.6	Große Anzahl von Systemvariablen .....	66
4.3.7	Langsame COM-Schnittstelle .....	68
<b>5</b>	<b>Vergleich der beiden Konzepte .....</b>	<b>70</b>
5.1	Vergleich anhand der Anforderungen .....	70
5.2	Vergleich der Konfiguration .....	73
5.3	Vergleich der Berechnungsgeschwindigkeit .....	74
5.4	Vergleich der Ergebnisse .....	77
5.5	Zusammenfassung .....	79
<b>6</b>	<b>Qualitätssicherung und Evaluation .....</b>	<b>81</b>
6.1	Qualitätssicherung .....	81
6.2	Evaluation.....	82
<b>7</b>	<b>Zusammenfassung und Ausblick .....</b>	<b>84</b>
7.1	Zusammenfassung .....	84
7.2	Fazit .....	85
7.3	Ausblick .....	86
<b>8</b>	<b>Literaturverzeichnis .....</b>	<b>88</b>

# Abbildungsverzeichnis

Abbildung 1: Die sieben Schichten des ISO-OSI-Referenzmodells nach (21).....	13
Abbildung 2: Darstellung der in CAN-Nachrichten übertragenen Signale in CANoe. Die Rohdaten (Spalte „Data“) der Nachricht „EngineData“ werden interpretiert und in reale Signal-Werte überführt, welche hier den Zustand des Motors beschreiben. ....	14
Abbildung 3: Einordnung von 14 Kommunikationssystemen nach Erscheinungsjahr, Datenübertragungsrate und Kosten pro Knoten, welche in Automobilen vorrangig eingesetzt werden. Quelle: (26).....	16
Abbildung 4: Visualisierung der verschiedenen Arbeits- und Darstellungsebenen in PREeVision: Basierend auf der Definition der Anforderungen an ein Automobil und der gewünschten Funktionen werden die logische und die Software-Architektur entworfen. Danach erfolgen die Implementation der Funktionen, der Entwurf der elektrischen Schaltungen sowie die genaue Planung für den Einbau des Systems ins Automobil. Parallel dazu (auf der rechten Seite) werden nach und nach die zu verwendenden Signale definiert und ihr Versand in Nachrichten (eventuell zu bestimmten Zeitpunkten) festgelegt. Quelle: (47 S. 6).....	24
Abbildung 5: Beispielhafte Darstellung des zeitlichen Verlaufs von Systemvariablen in CANoe .....	26
Abbildung 6: Kleines Ethernet-Netzwerk bestehend aus drei Knoten und einem Switch mit „Messspitzen“ an jedem Port. Links: node1, rechts oben: node2, rechts unten: node3, mittig: switch1. In der Tabelle wird die Beziehung zwischen dem gewählten Port im Netzwerk und dem Kanal in CANoe aufgezeigt. ....	28
Abbildung 7: Exemplarische Darstellung im Trace-Fenster in CANoe. Die Ethernet-Busse („Chn“) sind gleichzusetzen mit den Messpunkten in Abbildung 6. Es sendet node1 eine Nachricht über switch1 an node2 und node3.....	29
Abbildung 8: Schematische Abbildung der Informationsübermittlung: Die Simulationsdaten werden in OMNeT++ bereits derart aufbereitet, dass sie nur noch in CANoe abgespielt werden müssen. Als Übertragungsmedium dienen BLF-Dateien. ....	31
Abbildung 9: Schematische Darstellung zum Verlust der Gleichheit der Zeitstempel: Eine Nachricht wird vom Netzwerkport („phy“) empfangen und an dieser Stelle für das Trace-Fenster in die BLF-Datei geschrieben. Über einen Puffer („buffer“) wird die Nachricht an die Applikation („app“) weitergegeben. Diese berechnet mithilfe der Nachricht einen Metrik-Wert und schreibt ihn ebenfalls in die BLF-Datei. Durch die Verzögerung im Puffer sind die Zeitstempel der logisch miteinander verknüpften Objekte (Nachricht & Metrik-Wert) verschieden. ....	32

Abbildung 10: Die Aufbereitung der Simulationsdaten findet nicht in OMNeT++ statt, sondern in einer externen Applikation. Sie berechnet aus den in der ersten BLF-Datei enthaltenen Nachrichten die Metriken und schreibt sie wiederum in eine BLF-Datei, welche dann von CANoe eingelesen und angezeigt wird. ....	32
Abbildung 11: Die Aufbereitung der Simulationsdaten findet nicht in OMNeT++ statt, sondern in CANoe. Eine externe Applikation stellt dabei CANoe die Informationen über das simulierte Netzwerk zur Verfügung.....	34
Abbildung 12: Beispielhafte Darstellung der Bewertungsergebnisse der System-Metriken eines Netzwerks mit Gateways. Probleme wie Pufferüberläufe könnten durch rotgefärbte Knoten angezeigt werden, die Linkauslastung durch eine Färbung der Verbindungen.....	37
Abbildung 13: Schematische Darstellung der Simulation in OMNeT++ mit einer BLF-Datei, in welcher die Ergebnisse der Simulation festgehalten werden. ....	38
Abbildung 14: Schematische Darstellung der Verarbeitung der Informationen in Compagnon. Die erzeugten Dateien werden anschließend in CANoe eingespeist. ....	39
Abbildung 15: Schematische Abbildung der Verarbeitung der in einer BLF-Datei gespeicherten Nachrichten und Ablegen der Metrik-Werte in Systemvariablen.....	39
Abbildung 16: Abfrage des verwendeten Konzeptes zum Start von Compagnon .....	43
Abbildung 17: Oberfläche von Compagnon beim Hinzufügen von Dateien .....	44
Abbildung 18: Darstellung der Schnittstellen zwischen OMNeT++, Compagnon und CANoe mit Augenmerk auf die beiden letzteren. ....	44
Abbildung 19: Oberfläche von Compagnon bei der Auswahl der anzuzeigenden Links. Die Tabelle oben rechts gibt an, welcher Channel in CANoe welchen Port im Netzwerk anzeigt.....	45
Abbildung 20: Oberfläche von Compagnon bei der Erstellung der nötigen Dateien und der Eingabe von Grenzwerten für die Metriken .....	47
Abbildung 21: Schematische Darstellung des Verhaltens von CANoe beim Einlesen von BLF-Dateien: Enthalten zwei BLF-Dateien Nachrichten mit derselben Kanal-Nummer („Ch 1“), so werden diese Nachrichten in CANoe in einen Kanal gemischt und können nicht mehr unterschieden werden. Die tatsächliche Reihenfolge hängt von den Zeitstempeln der Nachrichten ab.....	48
Abbildung 22: Schematische Darstellung der Aufgabe von Compagnon bezüglich der BLF-Dateien: Die Kanäle der verschiedenen Dateien werden auf mehrere, unterschiedliche Kanäle in einer neuen BLF-Datei verteilt. Dies ermöglicht das korrekte Abspielen in CANoe. ....	48
Abbildung 23: Oberfläche der Applikation „FileInfo“, welche Informationen aus BLF-Dateien extrahiert.....	49
Abbildung 24: Nachrichten-Events in OMNeT++, welche die Übertragung eines SeqNoPackets in einem CanDataFrame zeigen. ....	51

Abbildung 25: Beispiel für die Verwendung von Namespaces bei Systemvariablen.....	53
Abbildung 26: Strukturierung der Systemvariablen in Namespaces bei der Simulation eines kleinen Netzwerkes mit Gateways.....	56
Abbildung 27: Zusammenhang der CAPL-Dateien und -Funktionen: Ankommende Nachrichten werden an die zentrale spottedMessage-Funktion weitergegeben, welche die Metriken berechnet und sie in Systemvariablen speichert.....	59
Abbildung 28: Von der Quelle (links) wird eine Nachricht zu einer Senke (rechts) übertragen. Die Latenz von ETH2 zu ETH1 beträgt $\Delta t_{1,2} = 10$ , die Latenz von ETH3 zu ETH2 beträgt ebenfalls $\Delta t_{2,3} = 10$ . Die Latenz von ETH3 zu ETH1 kann durch die Summe der beiden Latenzzeiten berechnet werden: $\Delta t_{1,3} = \Delta t_{1,2} + \Delta t_{2,3} = 20$ .....	60
Abbildung 29: Von der Quelle (links) wird eine Nachricht per Multicast zu den beiden Senken (rechts) übertragen. Die Latenz von ETH2 zu ETH1 beträgt $\Delta t_{1,2} = 10$ , die Latenz von ETH3 zu ETH1 beträgt aber $\Delta t_{1,3} = 20$ . Eine Latenz von ETH3 zu ETH2 $\Delta t_{2,3}$ kann hier nicht berechnet werden. ....	60
Abbildung 30: Beispielhafte Visualisierung der Netzwerkinformationen in CANoe. Ausgehend von einem Link (hier ETH3) werden die Nachbarschaften gemäß ihrer Entfernung in Hops gespeichert.....	61
Abbildung 31: Sequentielles Format der Gateway-Nachrichten nach (48 S. 44).....	63
Abbildung 32: Klassendiagramm des TTFrames: Das TTFrame wird vom EthernetIIFrame abgeleitet .....	66
Abbildung 33: Klassendiagramm des AVBFrames: Auch das AVB-Frame wird vom EthernetIIFrame abgeleitet.....	66
Abbildung 34: Warnung über eine zu große Anzahl an Systemvariablen, was die Arbeit mit CANoe behindern kann .....	68
Abbildung 35: Darstellung der Bewertungsergebnisse in Compagnon .....	69
Abbildung 36: Simuliertes Ethernet-Netzwerk „small“ .....	76
Abbildung 37: Simuliertes TT-Ethernet-Netzwerk „BMW“ aus (52) .....	76
Abbildung 38: Simuliertes Netzwerk „gateway“ zur Überprüfung der Analyse-Ergebnisse..	77
Abbildung 39: Verlauf der Latenzzeiten der CAN-Nachrichten mit ID 1 im Netzwerk „gateway“ in OMNeT++. Sowohl X- als auch Y-Achse sind in Sekunden aufgetragen...	78
Abbildung 40: Verlauf der Latenzzeiten der CAN-Nachrichten mit ID 1 im Netzwerk „gateway“ in CANoe, erhoben gemäß dem ersten Konzept. Sowohl X- als auch Y-Achse sind in Sekunden aufgetragen.....	78
Abbildung 41: Verlauf der Latenzzeiten der CAN-Nachrichten mit ID 1 im Netzwerk „gateway“ in CANoe, erhoben gemäß dem zweiten Konzept. ....	79
Abbildung 42: Darstellung von Hinweisen, Warnungen und Fehlern auf einer Zeitachse in Cardiogram nach (1 S. 6) .....	86



# Tabellenverzeichnis

Tabelle 1: Abschätzung der Qualität drei Konzepte hinsichtlich der Erhebung der Metriken und der damit verbundenen Möglichkeiten .....	36
Tabelle 2: Abschätzung der Qualität drei Konzepte hinsichtlich der Bewertung des Netzwerks und der Darstellung der Ergebnisse .....	37
Tabelle 3: Übersicht über die aufgezeichneten Statistik-Vektoren und deren Name in der BLF-Datei. Ausdrücke, die mit einem Dollar (\$) beginnen, sind Platzhalter für den nachfolgenden Ausdruck. ....	41
Tabelle 4: Inhalte der Statistik-Vektoren und somit der Systemvariablen .....	42
Tabelle 5: Anzahl der für eine Analyse verwendeten Systemvariablen in Abhängigkeit von der Anzahl der ausgewählten Busse ( <b>nb</b> ) und der zu analysierenden Nachrichtentypen ( <b>nc</b> ).....	67
Tabelle 6: Bewertung der Erhebung der Metriken in den beiden Konzepten gemäß Anforderung [1].....	72
Tabelle 7: Evaluierung der Anforderungen beider Konzepte .....	73
Tabelle 8: Ausführungszeiten der vier Konfigurationen des kleinen Netzwerks bei 60s Simulationszeit.....	75
Tabelle 9: Ausführungszeiten der vier Konfigurationen des großen Ethernet-Netzwerks bei 30s Simulationszeit .....	75
Tabelle 10: Ausführungszeiten der Analyse in CANoe.....	76
Tabelle 11: Vergleich der statistischen Werte des Latenzverlaufs der CAN-Nachrichten mit ID 1 im Netzwerk „gateway“ bei einer Simulationsdauer von 10s. ....	77
Tabelle 12: Übersicht über die Evaluation aller gestellten Anforderungen.....	83

# Listings

Listing 1: INI-Konfiguration der MAC-Adresse sowie einer Applikation zum Versand von Nachrichten im Modul <code>cam</code> .....	15
Listing 2: Konfiguration des Vector-Managers „VectorBlfRecorder“ in OMNeT++ .....	41
Listing 3: Hinzufügen zweier Statistik-Vektoren zu dem Standard Datarate-Channel Eth100M durch Deklaration eines neuen Channels .....	42
Listing 4: Konfiguration des BlfRecorders in OMNeT++ .....	43
Listing 5: Beispielhafter Inhalt einer BusMapper-XML-Datei über die Verknüpfung zwischen Netzwerkport und BLF-Kanal: Der simulierte Netzwerkverkehr der Ports <code>gw_ECAN.ethg[0]</code> und <code>gw_ECAN.ethg[2]</code> wurde in der Datei <code>gw.blf</code> in den Kanälen <code>ETH01</code> und <code>ETH03</code> gespeichert. ....	45
Listing 6: CAPL-Funktion, welche die Änderung einer Systemvariablen anhand der Grenzwerte überprüft.....	46
Listing 7: Anweisung an mehrere Applikationen zur Verwendung eines <code>SeqNoPacket</code> als Payload.....	51
Listing 8: Beispielhafter Inhalt einer Requirements-XML-Datei mit Angaben über die maximale Linkauslastung, Latenz und Jitter. ....	52
Listing 9: Beispielhafter Selektionspfad einer Systemvariablen, welche den aktuellen Wert der Verzögerung von <code>CAN02</code> zu <code>ETH01</code> jener Nachrichten enthält, welche vom Knoten mit der MAC-Adresse <code>EE:AA:AA:AA:AA:00</code> an den Knoten mit Adresse <code>EE:AA:AA:AA:AA:01</code> versandt werden.....	55

# 1 Einführung

In Automobilen werden fortwährend neue Funktionen integriert. Bei einem modernen Oberklassewagen sind unter anderem zahlreiche Fahrerassistenzsysteme integriert, welche selbstständig Einparken, bei Spurverlust Alarm schlagen und nachts Fußgänger erkennen, vor ihnen warnen und automatisch bremsen können. Insgesamt sind in einem modernen Fahrzeug bereits heute mehr als 300 Sensoren und Aktuatoren verbaut (1 S. 2). Die Netzwerke innerhalb eines Automobils enthalten inzwischen über 100 Steuergeräte (1 S. 2). Die Datenverarbeitung findet „zumeist an einer komplett anderen Stelle im Fahrzeug [statt] als die Sensorik, die die Daten aufnimmt. Damit kommt der Datenübertragung im Fahrzeug eine bedeutende Rolle zu.“ (2 S. 1)

Viele Experten sehen die Zukunft der Mobilität in autonomen Fahrzeugen (3). Diese stellen noch höhere Anforderungen an die Sensorik als es bei bisherigen Automobilen der Fall ist: Um ein Bild von der gesamten Umgebung zu erhalten, ist es nötig, u.a. Daten aus RADAR- und LIDAR-Systemen mit Informationen aus Kameras zu fusionieren. Da diese Sensoren mit einer hohen Datenrate arbeiten, müssen sehr viele Daten über das Netzwerk im Automobil übertragen werden können (4 S. 102).

Einen ebenso bedeutenden Einfluss auf die Mobilität der Zukunft hat die weitere Öffnung des Fahrzeugs zum Internet: Sei es der Austausch von Informationen zwischen dem Fahrzeug und anderen Fahrzeugen sowie der Infrastruktur à la Car2X (5) oder die Einbindung in das Internet-Of-Things (6); die Anzahl der Kommunikationspartner und die Menge der übertragenen Daten werden erheblich steigen.

Die im Automobilbau klassischerweise eingesetzten Bussysteme (z.B. CAN und FlexRay) sind aufgrund ihrer geringen Datenrate nicht geeignet, um diese Anforderungen zu erfüllen. Es ist stattdessen nötig, eine neue Netzwerkarchitektur auf Basis mehrerer Bussysteme zu entwickeln (4 S. 101). Ein solches heterogenes Netzwerk enthält Feldbusse mit geringen Datenübertragungsraten, welche mithilfe eines schnellen Bussystems über ein sogenanntes Backbone miteinander verbunden werden. Wie (2 S. 2) und (6) ausführen, bietet sich Ethernet als Backbone in dieser Architektur besonders an.

## 1.1 Motivation

Beim Einsatz von Ethernet im Automobil ist es möglich und auch nötig, für den Automobilbau neue Netzwerk-Strukturen einzusetzen (7). So bietet eine Sterntopologie die

bestmögliche Leistung (8 S. 10), ist jedoch um einiges teurer als komplexere Topologien (7 S. 1092) (vgl. Kapitel 2.7).

Die Kosten sind jedoch nicht das einzige Kriterium für eine gute Netzwerkstruktur: Das Netzwerk muss auch in der Lage sein, die Echtzeitanforderungen der Applikationen zu erfüllen, welche das Netzwerk nutzen. So muss z.B. das Bild einer Rückfahrkamera mit möglichst geringer Verzögerung im Cockpit des Fahrzeugs angezeigt werden, damit der Fahrer nicht irritiert wird. Noch kritischer sind die Anforderungen bei Fahrerassistenzsystemen und autonomen Autos: Werden Informationen über eine Gefahrenstelle (z.B. ein bevorstehendes Stauende) nicht innerhalb kurzer Zeit weitergeleitet, kann das Fahrzeug nicht rechtzeitig abgebremst werden und verursacht einen Unfall. Gefordert wird somit, dass Nachrichten innerhalb einer definierten Zeitspanne zugestellt werden (sog. harte Echtzeit) (2 S. 6). Da „normales“ Ethernet nach IEEE 802.3 hierzu nicht in der Lage ist, werden in Fachkreisen derzeit mehrere Protokolle diskutiert, welche Echtzeitfähigkeit nachrüsten (sog. Echtzeit-Ethernet).

An dieser Stelle setzt die Communication-over-Real-time-Ethernet-Arbeitsgruppe (CoRE) der HAW Hamburg an, in deren Kontext auch diese Arbeit entstand: Die CoRE-Arbeitsgruppe befasst sich mit Echtzeit-Ethernet-Kommunikation in unterschiedlichen Umgebungen (9). Im Forschungsprojekt „Realtime Ethernet Backbone for Cars“ (RECBAR) konzentriert sie sich auf das Design, die Entwicklung und das Testen von Ethernet im Automobil-Bereich (10). Unter anderem werden Netzwerkstrukturen und -konfigurationen simuliert (s. Kapitel 2.2), um sie hinsichtlich ihrer Eignung zu überprüfen. Im Entwurfsprozess eines Automobils könnte dies Kosten sparen, da bereits während der frühen Design-Phase mögliche Fehler korrigiert werden könnten.

Um solch eine Netzwerk-Simulation in der Automobilindustrie zu etablieren, ist es nötig, den dortigen Ingenieuren Werkzeuge zur Verfügung zu stellen, welche einfach und schnell zu bedienen sind und sich leicht in die vorhandenen Arbeitsprozesse integrieren lassen (vgl. Kapitel 2.7). Bisher steht jedoch kein Tool zur Verfügung, das diesen Ansprüchen gerecht wird: Um zu entscheiden, ob sich ein Netzwerk für die angedachten Anwendungen eignet, muss ein/e Netzwerkverantwortliche/r bisher tief in die Simulationsergebnisse eintauchen und manuell überprüfen, ob die Anforderungen, welche eine Applikation an das Netzwerk stellt, von diesem auch eingehalten werden.

Dieser Vorgang kann jedoch automatisiert werden. Hierdurch wäre es möglich, dem/der Netzwerkverantwortlichen viel Arbeit zu ersparen und direkt ein Bewertungs-Ergebnis anzuzeigen. Zusätzlich könnte dargestellt werden, an welcher Stelle ein Netzwerk Probleme aufweist, um eine Entscheidung zu ermöglichen, ob die Probleme z.B. durch bessere Hardware behoben werden können oder ob die gesamte Netzwerkstruktur verändert werden muss. Ebenso ermöglicht es eine Automatisierung durch die Zeitersparnis, mehrere Netzwerk-Varianten parallel zu bewerten und die beste auszuwählen.

## 1.2 Zielsetzung

Diese Arbeit untersucht, wie ein heterogenes Netzwerk im Automobil hinsichtlich der Eignung für Echtzeit-Applikationen bewertet werden kann. Darauf aufbauend werden zwei Programme entwickelt, welche eine solche Bewertung durchführen und die Ergebnisse grafisch darstellen kann, und vergleichend bewertet.

Die von den Applikationen erhobenen Informationen können von dem/der Netzwerkverantwortlichen genutzt werden, um das simulierte Netzwerk eigenständig weiter zu analysieren. Um eine gewohnte Darstellung und Bedienung der verwendeten Software zu garantieren, wird die Software CANoe von der Vector Informatik GmbH (s. Kapitel 2.5) eingesetzt, welche in der Automobil-Industrie bereits weit verbreitet ist.

Da CANoe bisher für Feldbussysteme mit relativ geringer Datenübertragungsrate eingesetzt wird, liegen kaum Erfahrungen bei der Arbeit mit großen Datenmengen vor. Somit wird ebenfalls ausgetestet, wo die Grenzen der Leistungsfähigkeit liegen.

## 1.3 Verwandte Arbeiten

Im Rahmen der CoRE-Arbeitsgruppe wurde bereits viel Zeit investiert, um einen grafischen Zugang zu den Simulationsergebnissen zu bieten. An dieser Stelle soll nur auf zwei Beispiele verwiesen werden: In (11) wurden verschiedene Daten wie die Busauslastung innerhalb des Netzwerks visualisiert. (12) behandelt die simulationsbasierte Analyse von Netzwerken innerhalb der Simulationssoftware OMNeT++ (s. Kapitel 2.2).

Die in (1) vorgestellte Applikation Cardiogram analysiert den Netzwerkverkehr innerhalb eines Automobils und stellt Probleme grafisch dar. Durch eine intelligente Filterung der Nachrichten und die Definition von Zustandsmaschinen können automatisch Fehler im Verhalten eines Netzwerks erkannt werden (1 S. 5). Cardiogram fokussiert sich allerdings auf ein vollständiges Netzwerk mit einer Implementation aller Applikationen, was in einer Simulation zu einem frühen Entwicklungs-Zeitpunkt nicht praktikabel ist. Es eignet sich somit vor allem für die Analyse und Bewertung realer Netzwerke.

Ethernet-Analyseprogramme wie WireShark und CableFish können zwar eingesetzt werden, um den Nachrichtenverkehr zu betrachten, nehmen allerdings keine Bewertung hinsichtlich ihrer Eignung für Echtzeit-Applikationen vor (11 S. 8f.).

Darüber hinaus wurden bereits einige Untersuchungen durchgeführt, welche bestimmte Echtzeit-Ethernet-Protokolle, Netzwerkstrukturen und -konfigurationen analysieren und vergleichend bewerten, bspw. (4), (8) und (13). Ebenfalls existieren Berichte über die Analyse eines einzelnen Netzwerkes, z.B. (14), (15) und (16).

Bisher wird jedoch noch keine Applikation entwickelt und eingesetzt, die eine solche Bewertung auf Basis einer Simulation und anhand vorgegebener Grenzwerte selbstständig vornimmt.

## **1.4 Gliederung dieser Arbeit**

Diese Arbeit ist in sieben Kapitel gegliedert. Kapitel 2 enthält die Grundlagen, welche zum Verständnis dieser Arbeit notwendig sind. In Kapitel 3 werden die Anforderungen formuliert und analysiert sowie Konzepte zur Lösung der Aufgabenstellung entwickelt und untersucht. Anschließend werden zwei Konzepte ausgewählt und ausführlicher betrachtet. Einige Informationen zur Umsetzung der ausgewählten Konzepte sowie Herausforderungen und Probleme werden in Kapitel 4 diskutiert. In Kapitel 5 werden daraufhin die beiden Konzepte hinsichtlich der Erfüllung der Anforderungen, der Konfigurationsfähigkeit, der Geschwindigkeit sowie der Bewertungsergebnisse verglichen. Kapitel 6 befasst sich mit der Qualitätssicherung und der Evaluation der Ergebnisse dieser Arbeit. Schlussendlich wird in Kapitel 7 diese Arbeit zusammengefasst und ein Ausblick auf weitere Arbeiten und Verbesserungsmöglichkeiten gegeben.

## 2 Grundlagen

### 2.1 Notation und Begriffe

Im Laufe dieser Arbeit wird für die Netzwerk-Verbindung zwischen zwei Knoten auch der englische Begriff „Link“ verwendet. Ebenso wird „Backbone“ (engl. für Rückgrat) als Bezeichnung für den „verbindenden Kernbereich eines Telekommunikationsnetzes mit sehr hohen Übertragungsraten“ (17) genutzt. Der Begriff „Byte“ wird synonym zu „Oktett“ verwendet und bezeichnet eine Gruppierung von acht Bits.

#### 2.1.1 Begriffe, Bussysteme und Protokolle

Darüber hinaus werden immer wieder die Bussysteme

- Local Interconnect Network (LIN),
- Controller Area Network (CAN),
- FlexRay,
- Media Oriented Systems Transport (MOST),
- Avionics Full Duplex Switched Ethernet (AFDX) und
- Ethernet

erwähnt. Da in dieser Arbeit mit ihnen lediglich auf abstrakter Ebene umgegangen wird, spielt das Verständnis dieser Systeme keine Rolle. Somit soll an dieser Stelle für Interessierte lediglich auf (18 S. 15ff.) verwiesen werden. Identisch verhält es sich mit den Ethernet-Technologien bzw. -Protokollen

- Time Triggered Ethernet (TTE) sowie
- Audio Video Bridging (AVB) bzw. Time Sensitive Networking (TSN).

Eine ausführliche Beschreibung hierzu findet sich in (12 S. 12ff.) bzw. (19).

#### 2.1.2 ISO-OSI-Referenzmodell

Im Verlauf dieser Arbeit wird ab und an das ISO-OSI-Referenzmodell verwendet. Dieses Modell teilt die Kommunikation in sieben Schichten auf, um sie „über unterschiedlichste technische Systeme hinweg zu ermöglichen und die Weiterentwicklung zu begünstigen“ (20). Jede Schicht übernimmt eine definierte Aufgabe und stellt Schnittstellen zu den angrenzenden Schichten bereit.

Beim Versand einer Information wird diese von der obersten Schicht immer weiter nach unten gereicht, bis sie über ein physikalisches Medium übertragen werden kann. Charakteristisch ist, dass jede Schicht Steuerdaten hinzufügt, welche sie für den Betrieb benötigt. Beim Empfänger werden die empfangenen Daten von der untersten Schicht nach oben weitergereicht, bis die Information wieder extrahiert ist (vgl. Abbildung 1).

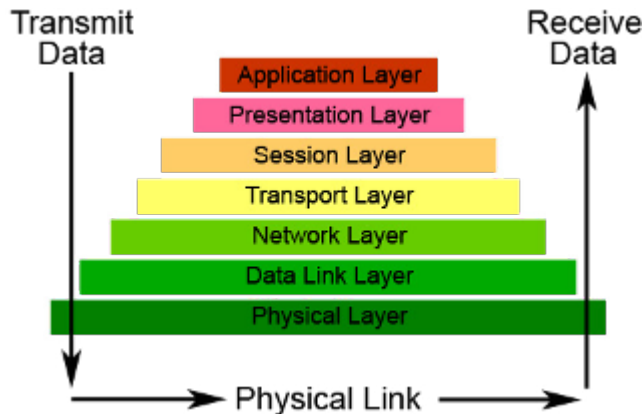


Abbildung 1: Die sieben Schichten des ISO-OSI-Referenzmodells nach (21)

Alle in Kapitel 2.1.1 aufgeführten Bussysteme spezifizieren mindestens Schicht eins und Schicht zwei. Bisher werden die höheren Schichten im Automobil meist nicht eingesetzt, um die Leistungsfähigkeit des Netzwerkes zu erhöhen. In dieser Arbeit sind vor allem die Schichten zwei (Sicherheitsschicht) und drei (Vermittlungsschicht) relevant.

Aufgabe der Sicherheitsschicht ist es, eine fehlerfreie Übertragung zu gewährleisten. Mit ihrer Hilfe ist es somit möglich, Datenpakete über ein physikalisches Medium (z.B. ein Kabel) zu übertragen.

Die Vermittlungsschicht sorgt dafür, dass Pakete zum Empfänger weitergeleitet werden. Dies ermöglicht es, dass Applikationen über ein heterogenes Netzwerk kommunizieren können, ohne sich Gedanken darüber zu machen, welchen Weg die Nachricht zum Empfänger nehmen muss. Ein populäres Beispiel für ein Protokoll dieser Schicht ist das Internet-Protocol (IP), die „Grundlage des Internets“ (22).

## 2.2 Netzwerksimulation im Automobil mit OMNeT++

Simulationen sind wichtig, um Technologien und Konzepte zu validieren, ohne sie kostspielig in Realität aufbauen zu müssen. Außerdem bieten sie den Vorteil, dass die simulierten Modelle sehr genau untersucht werden können.

Im Automobilbau sind Simulationen wie in nahezu jedem technischem Bereich bereits seit dem letzten Jahrtausend üblich (23). Beim Design der Elektrik/Elektronik wird heutzutage das gesamte System aus Steuergeräten (sogenannten ECUs) vorab simuliert (s. Kapitel 2.5). Da die üblicherweise eingesetzten Bussysteme entweder sehr lokale Anwendungsbereiche



abdecken (wie der LIN-Bus), in der Linien-Topologie aufgebaut werden (z.B. der CAN-Bus) und hauptsächlich sehr kurze Nachrichten versenden, fokussiert sich die bisherige Simulation auf die übertragenen Signale (OSI-Schicht 7). Als Signal bezeichnet man in der Autoindustrie eine einzelne Information und deren zeitlichen Verlauf, bspw. die Geschwindigkeit des Fahrzeugs oder die Temperatur des Motors (vgl. Abbildung 2). Zwar werden die verschiedenen Busse im Automobil bereits oft über ein großes Gateway zusammengeschaltet – es ergibt sich somit eine sternförmige Topologie –, Probleme wie der Verlust von Paketen oder eine zu hohe Verzögerung können jedoch dediziert an dieser Stelle untersucht werden.

Time	Chn	ID	Name	Dir	D...	Data
22.780804	CAN 2	67	Ignition_Info	Tx	2	01 00
22.751416	CAN 2	64	EngineData	Tx	8	7A 08 2D 2D AE 06 44 16
			PetrolLevel			45 l 2D
			EngPower			57.0000 kW 1644
			EngForce			1710 N 6AE
			IdleRunning			Running 0
			EngTemp			40 degC 2D
			EngSpeed			2170 rpm 87A
22.752796	CAN 2	66	EngineDataIEEE	Tx	8	D3 9A 07 45 00 C0 D5 44

Abbildung 2: Darstellung der in CAN-Nachrichten übertragenen Signale in CANoe. Die Rohdaten (Spalte „Data“) der Nachricht „EngineData“ werden interpretiert und in reale Signal-Werte überführt, welche hier den Zustand des Motors beschreiben.

Verwendet man Ethernet als Backbone im Automobil, so steht plötzlich eine Vielzahl von Topologien zur Verfügung, welche die Leistungsfähigkeit des Netzwerkes maßgeblich beeinflussen können. Somit wird es auch nötig, das Netzwerk auf Paket-Ebene (OSI-Schicht 2) zu untersuchen. An dieser Stelle setzt die CoRE-Arbeitsgruppe an: In ihrer Simulation werden Steuergeräte als reine Paket-Quellen und -Senken mit bestimmten Anforderungen (wie Datenrate, maximaler Verzögerung etc.) betrachtet. Der Inhalt dieser Pakete, also die Signale, spielt für die Funktionsweise des Netzwerkes keine Rolle und wird somit ignoriert (sog. System-Level-Simulation).

Für die Simulation wird das Framework OMNeT++ eingesetzt, welches eine diskrete, ereignisorientierte Umgebung für Netzwerksimulatoren bereitstellt (24). Da es modular aufgebaut ist, kann es sehr einfach um weitere Komponenten wie bspw. eigene Protokolle erweitert werden. OMNeT++ stellt eine Eclipse-basierte Entwicklungs- sowie eine grafische Laufzeitumgebung zur Verfügung. Hierdurch sowie durch eine eigene Modellierungssprache für Netzwerke (12 S. 26) hebt es sich von anderen Simulatoren wie ns-2, J-Sim und OPNET ab (25). Eine Einführung in die Prinzipien der Modellierung von Netzwerken mit OMNeT++ gibt (12 S. 26ff.).

Generell läuft die Kommunikation zwischen zwei Knoten in OMNeT++ wie folgt ab: In einem Netzwerk-Knoten sind eine oder mehrere Applikationen angesiedelt. Eine solche Applikation modelliert den Sendebedarf z.B. von CAN-Nachrichten oder von einem Video-

Stream. Die gesendeten Pakete werden über einen Ausgangspuffer und Module, welche die niederen OSI-Schichten simulieren (z.B. die Priorisierung nach CAN-ID), auf die Kommunikationsschnittstelle geleitet. Von dort werden die Nachrichten mit einer bestimmten Verzögerung weitergegeben; außerdem können Pakete z.B. durch Übertragungsfehler verloren gehen. Möglicherweise zwischengeschaltete Knoten empfangen das Paket, verarbeiten es und senden es wiederum auf einer anderen Schnittstelle. Im Zielknoten wird die eingehende Nachricht über einen Puffer an die Zielapplikation weitergeleitet und dort verarbeitet.

Zur Beschreibung des Netzwerks und der darin enthaltenen Module verwendet OMNeT++ die Network-Description-Sprache (NED). Sie ermöglicht es, die Knoten eines Netzwerks zu definieren und miteinander zu verbinden. Ebenso werden die Module innerhalb der Knoten definiert und miteinander verbunden, um die gewünschte Funktionalität herzustellen (12 S. 26ff.).

NED eignet sich somit einerseits zur Definition von Modulen, welche grundlegende Funktionen bereitstellen, und andererseits durch die ansteigende Generalisierung von immer komplexeren Modulen auch zur Definition von ganzen Netzwerken. Unterstützt wird dies durch folgende Techniken der Objektorientierung: Vererbung, Interfaces, Packages, innere Typen und Metadaten. Ausführlichere Informationen können in (12 S. 26ff.) eingesehen werden.

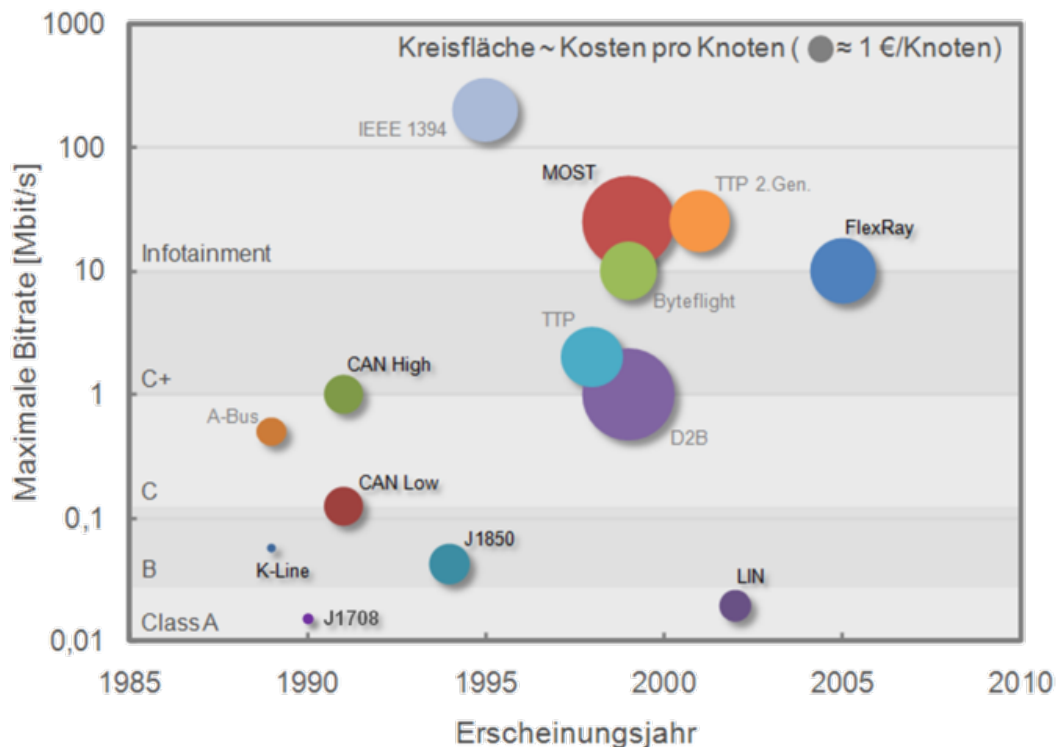
NED-Module stellen typischerweise Konfigurationsmöglichkeiten bereit, welche mittels INI-Dateien verwendet werden können (12 S. 28). Ein wichtiges Beispiel sind die Zuweisung der bei Ethernet verwendeten MAC-Adresse sowie die Zuweisung der Applikationen, welche in dem Netzwerkknoten angesiedelt ist. Mittels weniger Zeilen in der INI-Datei kann ein Knoten somit z.B. zum Sender eines Video-Streams konfiguriert werden (s. Listing 1). Da es für die Simulation unerheblich ist, welche Daten übertragen werden, werden nur die für Video-Streams charakteristischen Werte wie Paket-Intervall und -größe angegeben. Die hierfür angemessenen Werte zu ermitteln obliegt dem Nutzer/der Nutzerin der Simulations-Software.

```
1 ** .cam.phy[*].mac.address = "0A-00-00-00-00-07"  
2  
3 ** .cam.numApps = 1  
4 ** .cam.app[0].typename = "RCTrafficSourceApp"  
5 ** .cam.app[0].interval = 280us  
6 ** .cam.app[0].payload = 1428Byte  
7 ** .cam.app[0].ct_id = 3
```

**Listing 1:** INI-Konfiguration der MAC-Adresse sowie einer Applikation zum Versand von Nachrichten im Modul `cam`.

## 2.3 Datenübertragung im Automobil

Bisher wird im Automobil eine Vielzahl an verschiedenen Bussystemen eingesetzt, um Daten zwischen Sensoren, Steuergeräten und Aktoren zu übertragen. Diese Systeme entstanden über die Zeit durch verschiedene Anforderungen. So existiert bspw. mit dem LIN-Bus ein relativ langsames, aber sehr günstiges Kommunikationssystem. Um dem steigenden Datenratenbedarf von Multimediaanwendungen beizukommen, wurden auch teurere Technologien wie FlexRay und MOST entwickelt (s. Abbildung 3).



**Abbildung 3:** Einordnung von 14 Kommunikationssystemen nach Erscheinungsjahr, Datenübertragungsrate und Kosten pro Knoten, welche in Automobilen vorrangig eingesetzt werden. Quelle: (26).

Es existiert jedoch bereits seit 1995 ein Bussystem, welches hohe Datenraten zur Verfügung stellt und inzwischen in vielen Anwendungsbereichen wie Automatisierungstechnik, Bürokommunikation und Luftfahrt etabliert ist: Ethernet (27). Durch die starke Verbreitung stehen auf Personalseite bereits viele Kompetenzen zur Verfügung, was eine lange Einarbeitungsphase erspart (2), und mit der Spezifikation von „Power over Ethernet“ (PoE) können Steuergeräte mitunter sogar ohne dedizierte Energieversorgungsleitung betrieben werden und somit Material- und Kostenersparnisse erzielt werden (12 S. 12). Als erste auf Ethernet aufsetzende Technologie im Automobil wurde 2012 der weltweite Standard zur Diagnose über IP (DoIP) eingeführt (27 S. 2).

Zuerst musste jedoch mit dem Standard BroadR-Reach die Übertragung (OSI-Schicht 1) derart abgeändert werden, dass Ethernet die Richtlinien zur elektromagnetischen Verträglichkeit im Automobil einhält (28).

Nichtsdestotrotz müssen gemäß (29) noch viele Anstrengungen unternommen werden, bevor sicherheitskritische Systeme auf Ethernet aufsetzen können. So sei es bspw. nötig, die verwendeten Geräte auf elektromagnetische Verträglichkeit und Zuverlässigkeit sowie Stecker und Verlegung der Kabel ausgiebig zu testen und Erfahrungswerte zu sammeln. Immerhin ist ein Automobil – ähnlich wie ein Flugzeug – meist über mehrere Jahrzehnte im Einsatz.

## 2.4 Bewertung von Netzwerken

Die Hauptaufgabe dieser Arbeit ist die Bewertung von echtzeitfähigen Netzwerken. Eine computergestützte Bewertung erfolgt typischerweise auf Basis von Kennzahlen. Im Bereich der Netzwerke nennt man diese Kennzahlen „Metriken“.

Eine Netzwerktopologie kann durch einen mathematischen Graph beschrieben und u.a. mit dessen Grad, Durchmesser, Kantenkonnektivität und Bisektionsbreite hinsichtlich Kosten, Verzögerung und Robustheit bewertet werden (30). Diese Metriken sind statisch und müssen nicht durch eine Simulation ermittelt werden.

Neben den Topologie-Metriken werden für Netzwerke System- und die Benutzer-Metriken definiert (12 S. 45). System-Metriken werden demnach innerhalb eines Netzwerks gemessen (z.B. die Auslastung eines Links), Benutzer-Metriken bewerten das Netzwerk aus der Sicht einer Applikation (bspw. die Ende-zu-Ende-Latenz).

### System-Metriken

Im Folgenden werden vier System-Metriken erläutert und definiert, welche Fabian Kempf in Zusammenarbeit mit der IAV Industrie-Anlagen-Vertrieb GmbH für die Evaluierung von Netzwerken zusammengestellt hat (31). Deren Messung beruht – um einen zeitlichen Verlauf darstellen zu können – meistens auf Zeitfenstern. „Diese bestimmen den Zeitraum, den die gemessene Kennzahl rückwirkend beschreibt“ (12 S. 75).

Die **Datenübertragungsrate** (oft fälschlicherweise als Bandbreite bezeichnet) gibt die Datenmenge pro Zeit an, welche auf einem Medium übertragen werden kann (32). Somit wird sie in  $\text{bit/s}$  angegeben und ergibt sich aus der Spezifikation des Netzwerkes. Beispiele hierfür sind  $100 \text{ Mbit/s}$  bei Fast-Ethernet oder  $125 \text{ kbit/s}$  bei Low-speed-CAN.

Die **Linkauslastung** beschreibt die Zeit (meist relativ zum Zeitfenster), in der tatsächlich Daten übertragen wurden (12 S. 76). Dieser Wert lässt sich umrechnen in die tatsächlich übertragene Menge der Daten pro Zeit. Somit kann die Linkauslastung relativ (zum Zeitfenster sowie zur Datenübertragungsrate) in Prozent und absolut in  $\text{bit/s}$  angegeben werden. Eine sehr hohe Linkauslastung deutet darauf hin, dass das Netzwerk an der Grenze

seiner Möglichkeiten betrieben wird; eine sehr niedrige Auslastung zeigt, dass das Netzwerk überdimensioniert und damit nicht kosteneffektiv ist.

Der **Datendurchsatz** (auch Kapazität genannt) gibt die Netto-Datenmenge pro Zeit an, die über ein Netz übertragen werden kann und wird absolut in  $\text{bit/s}$  oder relativ zur Datenübertragungsrate angegeben (33). Im Gegensatz zur Datenübertragungsrate werden übertragene Steuerdaten der verwendeten Protokolle (der sogenannte Overhead) sowie Wartezeiten und durch Übertragungsfehler ungültige Daten nicht mitgezählt; diese verringern also den Datendurchsatz (12 S. 75). Da Netzwerkprotokolle gemäß OSI-Modell (s. Kapitel 2.1.2) aufeinander aufbauen und eigene Steuerdaten hinzufügen, kann für jede eingesetzte OSI-Schicht ein Datendurchsatz angegeben werden. Berechnet man diesen bspw. für Low-speed-CAN auf OSI-Schicht 2, ergibt sich ein Wert von weniger als 58% bzw.  $72 \text{ kbit/s}$  (34). Üblicherweise wird über CAN kein weiteres Protokoll gesprochen, somit ist dies der endgültige Wert für den Datendurchsatz. Bei Ethernet ist es hingegen gängig, weitere Protokolle wie bspw. das Internet-Protokoll IP und das Transmission-Control-Protokoll TCP einzusetzen. Beachtet man diese erst einmal nicht und berechnet den Datendurchsatz auf OSI-Schicht 2, so beträgt der Datendurchsatz – durch die im Vergleich zu CAN größere Menge an Nutzdaten in einem Paket – bis zu 94%. Sowohl IP als auch TCP reduzieren diesen Wert (bei großen Nutzdatenmengen) um einige wenige Prozent. Bei Protokollen, welche in höhere OSI-Schichten einzuordnen sind, wird die Angabe eines genauen Wertes immer schwieriger. Deshalb wird in dieser Arbeit nur der Datendurchsatz auf OSI-Schicht 2 verwendet.

Die **Puffergröße** beschreibt die Anzahl der zwischengespeicherten Nachrichten in einem Knoten (12 S. 76). Werden Nachrichten zwischengespeichert, so wird deren Zustellung verzögert. Zwischenspeicher (die sogenannten Puffer) sind nötig, wenn von einem Bussystem zu einem anderen mit niedrigem Datendurchsatz vermittelt wird (z.B. von Ethernet auf CAN) oder wenn mehrere Netzwerk-Verbindungen in einem vermittelnden Knoten zusammenlaufen. Da die Größe der Puffer (der Speicherplatz) in der Regel begrenzt ist, müssen unter Umständen Pakete verworfen werden; dies erhöht die **Paketverlustrate** (s.u.). Je nach eingesetztem Bussystem können zwecks der Priorisierung von Nachrichten mehrere Puffer pro Link existieren, so z.B. bei Echtzeit-Ethernet-Systemen.

## Benutzer-Metriken

Im Folgenden werden die vier wichtigsten Benutzer-Metriken gemäß (31) erläutert. Sie sind besonders für die Regelungstechnik und den Entwurf von Regelungskreisen wichtig, wie sie bspw. für die Motorsteuerung eingesetzt werden. Sollen Regelkreise ein Netzwerk beinhalten, so ist es wichtig, dass dieses Netzwerk den Anforderungen der Regelungstechnik genügt. Eine zu hohe Totzeit in einem Regelkreis kann dazu führen, dass der Regler unbrauchbar wird. Außerdem erfordern manche Systeme im Automobil eine sehr schnelle Reaktion auf ein Ereignis. „So muss z.B. bei einem Aufprall die Aufnahme der Sensorwerte, die Datenübertragung und die Aktivierung des Airbags abgeschlossen sein, bevor die Relativbewegung der Insassen beginnt“ (2 S. 5).

Die **Latenzzeit** „beschreibt die Verzögerung von Botschaften“ (12 S. 74), also die Dauer vom Erstellen der Nachricht im Quellknoten bis zum Empfang und der Verarbeitung im Zielknoten und kann für jedes einzelne empfangene Paket berechnet werden. In der Regelungstechnik wird diese Zeit auch Totzeit genannt und erschwert die Regelung eines Prozesses in einem Regelkreis. Somit ist eine möglichst geringe Latenz wünschenswert (35).

Um einen Regelkreis als lineares, zeitinvariantes System entwerfen zu können, muss die Latenzzeit darüber hinaus möglichst deterministisch sein; andernfalls wird die Modellierung erheblich aufwändiger (35). Dies führt uns zum **Jitter**, der Variabilität der Verzögerung von Paketen in Netzwerken (36). In einem idealen Netzwerk ist die Latenz stets konstant und der Jitter somit gleich null. In herkömmlichen Ethernet-Netzwerken entstehen durch verschiedene Puffergrößen in den Knoten sowie durch Paketverluste unterschiedliche Latenzen. Das Protokoll Time-Triggered-Ethernet garantiert bei TT-Nachrichten durch das Verwenden von festen Zeitslots einen sehr geringen Jitter von wenigen Mikrosekunden. Der Jitter kann auch minimiert werden, indem die Pakete im Zielknoten zwischengespeichert werden und in regelmäßigen Abständen verarbeitet werden. Somit füllt und leert sich dieser Puffer mit den Schwankungen der Verzögerung. Dies beeinflusst jedoch die Latenzzeit negativ (35).

Die **Paketverlustrate** beschreibt das Verhältnis der Anzahl verlorener zu empfangener Datenpakete (12 S. 74). Pakete können verloren gehen, wenn bei der Übertragung ein Fehler auftritt (diese werden durch Prüfsummen detektiert) oder wenn ein zwischengeschalteter Knoten im Netzwerk die Nachricht nicht weiterleitet, z.B. wegen falscher Konfiguration oder wegen des Überschreitens der Puffergröße. In der Regelungstechnik ist bei Verlust einer Information der Regelkreis nicht mehr geschlossen, was durch spezielle Techniken wie z.B. die Berechnung eines Wertes durch Modellierung der Regelstrecke (Smith Predictor) kompensiert werden muss (37).

Die **Startup-Geschwindigkeit** beschreibt die Zeit, die vergeht, bis ein Netzwerk nach dem Einschalten voll funktionsfähig ist. Hierbei können verschiedene Zustände von Interesse sein, bis zu deren Eintreffen gemessen wird: Die einzelnen Teilnehmer eines Netzes sind nach dem Einschalten meist bereits nach einigen Mikrosekunden funktionsbereit, eine etwaige Synchronisation kann mehrere Sekunden dauern. Da das Gesamtsystem spätestens bei Aktivierung der Zündung eines Automobils voll funktionsfähig sein muss, ist auch diese Dauer von Interesse.

Da die Messung der Startup-Geschwindigkeit mitunter eine vollständige Simulation eines Automobils erfordert, kann sie mit dem bisherigen Simulationsansatz der CoRE-Arbeitsgruppe nicht ermittelt werden. Es bleiben somit folgende Metriken, welche zur Bewertung eines Netzwerks herangezogen werden müssen:

- Linkauslastung
- Datendurchsatz
- Puffergröße
- Latenzzeit
- Jitter
- Paketverlustrate

## 2.5 Software-Werkzeug CANoe

CANoe von der Vector Informatik GmbH ist ein „Software-Werkzeug für die Entwicklung einzelner Steuergeräte sowie ganzer Netzwerke“ (38). Es unterstützt eine Vielzahl in der Automobilindustrie eingesetzter Bussysteme wie bspw. CAN, LIN, FlexRay und Ethernet. Reale Netzwerke können über entsprechende Hardware mit CANoe verbunden werden, wobei CANoe die Rolle eines oder mehrerer Knoten einnehmen kann. Somit ist es möglich, die empfangenen Nachrichten zu analysieren und die enthaltenen Signale zu extrahieren, angeschlossene Geräte zu stimulieren oder Diagnosefunktionen auszuführen. Ebenso ermöglicht es CANoe, diese Schritte zu automatisieren und mit dem Ergebnis ausführliche Testberichte zu erstellen. Da CANoe in der Lage ist, auf ankommende Nachrichten zu reagieren, können auch Teile des Netzwerks simuliert werden.

Eingesetzt wird dies in der Automobilindustrie meistens derart, dass ein Netzwerk vollständig in CANoe simuliert wird. Diese Simulation wird durch die Anbindung von MATLAB/Simulink unterstützt. Durch eine Vielzahl automatisierter Tests wird die Funktionsfähigkeit des simulierten Netzwerks sichergestellt. Ist ein Steuergerät soweit entwickelt, dass ein realer Prototyp produziert werden kann, so wird dieser Prototyp mit CANoe verbunden und seine Simulation deaktiviert. Mithilfe der Tests wird bei dieser sogenannten Restbussimulation überprüft, ob sich der Prototyp korrekt verhält und das Netzwerk weiterhin funktioniert. Im weiteren Verlauf der Entwicklung werden immer mehr reale Geräte im Netzwerk hinzugefügt, bis CANoe nur noch zur Analyse des Netzwerkverkehrs eingesetzt wird.

Neben dieser sogenannten Online-Analyse, während welcher CANoe im Echtzeit-Modus arbeitet, kann CANoe auch in der Offline-Analyse zuvor aufgenommenen Netzwerkverkehr abspielen. Hierbei ist es möglich, aufwändigere Operationen mit den Nachrichten durchzuführen, welche in Echtzeit nicht erledigt werden können.

Ein für diese Arbeit wichtiger Teilaspekt von CANoe sind die Systemvariablen: Sie sind vergleichbar mit globalen Variablen in diversen Programmiersprachen und können von allen Modulen in CANoe gelesen und geschrieben werden: Sie können in den CANoe-eigenen Oberflächen mit Schaltern, Schiebereglern und Textfeldern vom Nutzer/von der Nutzerin verändert werden (z.B. um bestimmte Aktionen auszulösen), ihre Werte können in Grafik-Fenstern angezeigt und auf sie kann aus in CANoe laufenden Programmen zugegriffen werden. Außerdem kann ihr Zustand von CANoe in BLF-Dateien (vgl. Kapitel 2.6)

geschrieben und aus ihnen gelesen werden. Sie eignen sich somit sehr gut dafür, Werte und Zustände zu speichern und grafisch darzustellen.

CANoe bietet eine Vielzahl an Schnittstellen. Hierzu gehören die bereits erwähnte Abstraktionsschicht für MATLAB/Simulink, die COM-Schnittstelle zum Datenaustausch und zur Steuerung von CANoe, das Datenaustausch-Protokoll FDX sowie die Möglichkeit, eigene Programme in CANoe laufen zu lassen.

Diese Programme können entweder in .NET oder in der CANoe-eigenen Programmiersprache CAPL verfasst sein und zur Laufzeit auf Ereignisse wie ankommende Nachrichten oder Änderungen von Systemvariablen reagieren und wiederum Aktionen auslösen, z.B. Nachrichten versenden oder Systemvariablen ändern. Hierdurch ist es möglich, einen Netzwerkknoten wie bspw. ein Steuergerät in CANoe zu implementieren.

Die COM-Schnittstelle (Component Object Model) „ist eine von Microsoft entwickelte Technik zur Erstellung von Softwarekomponenten, die unabhängig von der Programmiersprache eingesetzt werden können. COM-Komponenten ermöglichen unter dem Betriebssystem Windows Interprozesskommunikation und dynamische Objekterzeugung“ (39). Über den von CANoe bereitgestellten COM-Server können externe Applikationen die Konfiguration von CANoe ändern, Messungen starten und stoppen, auf Signale und Systemvariablen zugreifen sowie CAPL-Funktionen aufrufen (40). Dieser Zugriff ist durch Thread-Kontext-Wechsel jedoch nicht besonders schnell und eignet sich somit nicht zum Austausch vieler Daten.

Hierfür hat Vector Informatik das proprietäre FDX-Protokoll (Fast Data eXchange) entwickelt. Es basiert auf UDP und ermöglicht den Austausch von Systemvariablen und Signalen über IP-Netzwerke (41). Somit können diese bspw. in einer externen Applikation dargestellt oder verändert werden. Es ist über das Protokoll auch möglich, eine Messung in CANoe zu starten und zu stoppen und einen Tastendruck an CANoe weiterzuleiten. Die zu übertragenden Variablen werden vor der Laufzeit in einer XML-Datei definiert und können dann von der Applikation abgefragt oder von CANoe zyklisch oder aufgrund eines Befehls in einem CAPL-Programm übertragen werden.

CANoe unterstützt eine große Anzahl von Logging-Formaten. Bekannt ist das PCAP-Format, von welchem nur AFDX- und Ethernet-Pakete gelesen werden können (42). Besser eignet sich meistens das Binary-Logging-Format, welches nun näher betrachtet wird.



## 2.6 Binary-Logging-Format (BLF) in CANoe

Das nachrichtenorientierte Binary-Logging-Format (Binlog-Format, BLF) von Vector ist auf CANoe zugeschnitten: Es unterstützt eine große Anzahl an Bussystemen, unter anderem:

- CAN,
- Ethernet,
- AFDX,
- FlexRay,
- MOST,
- A429,
- LIN,
- WLAN,
- Events wie GPS-Positionen oder Daten einer Realtime-Clock
- sowie Werte von Systemvariablen

Für alle Bussysteme kann nicht nur der Empfang einer Nachricht aufgezeichnet werden; es besteht auch die Möglichkeit, fehlgeschlagene Übertragungen sowie Statistiken zur Busauslastung zu speichern. Pro Bussystem können mehr als 65.000 Kanäle (im Folgenden: BLF-Kanäle) in einer Datei gespeichert werden. Die BLF-Kanäle sind logisch voneinander getrennt, wodurch die darin gespeicherten Nachrichten z.B. unterschiedlichen Netzwerk Ports zugeordnet werden können. Da das Dateiformat nativ eine Komprimierung der enthaltenen Daten unterstützt, belegen gerade Log-Dateien von System-Level-Simulationen (s. Kapitel 2.2) durch die unbenutzte Payload nur sehr wenig Speicherplatz. Der Lese- und Schreibzugriff auf die Dateien ist durch eine CANoe beigelegte Win32-DLL möglich.

## 2.7 Entwicklung einer Elektrik-/Elektronik-Architektur

Durch die steigende Anzahl der im Automobil integrierten Funktionen steigt auch die Komplexität des Gesamtwerks. Um diese steigende Komplexität weiterhin beherrschen zu können, wurde im Jahr 2003 die Standardisierungsinitiative AUTOSAR („Automotive Open System Architecture“) von Automobilherstellern und Zulieferern gegründet (43). Sie bietet „eine Reihe von Spezifikationen an, die Basissoftware-Module beschreiben, Anwendungsschnittstellen definieren und eine allgemeine Entwicklungsmethodik, basierend auf einem standardisierten Austauschformat, festlegen“ (44 S. 1 Überblick). Hierdurch kann „die ständige Neuentwicklung gleicher oder ähnlicher Softwarekomponenten [eingedämmt werden]“ (44 S. 2 Motivation). In (45 S. 4ff.) wird eine ausführliche Einführung in die Konzepte von AUTOSAR gegeben.

Damit einhergehend wird die Elektrik-/Elektronik-Architektur (EEA) eines Automobils heutzutage modellbasiert entwickelt (18 S. 28). Dabei wird auf die Separierung in die Domänen Motorsteuerung, Chassis, Komfort, Fahrerassistenz, Telemetrie/Infotainment und Entertainment gesetzt (27 S. 2). „Der Einsatz von domänenspezifischen modellbasierten Entwicklungswerkzeugen bildet [...] den Standard in der Automobilentwicklung“ (18 S. 28).

Wie bereits bei der Netzwerksimulation (s. Kapitel 2.2) abstrahiert ein Modell „die Funktion, Struktur oder das Verhalten des betrachteten Systems“ (18 S. 28). Zur Beschreibung eines Modells werden – wie bei OMNeT++ (s. Kapitel 2.2) – Meta-Modelle eingesetzt, welche wiederum aus Modellen bestehen (46). Im Gegensatz zur Netzwerksimulation wird dies jedoch für den hierarchischer Entwurf des Systems genutzt, indem zuerst grob die Funktionalitäten und deren Zusammenwirken beschrieben und anschließend immer feiner definiert und implementiert werden. Die Brücke zu AUTOSAR schlägt die automatische Code-Generierung, welche große Teile der Modelle in Code umsetzen kann und die Entwurfsablauf weiter beschleunigt (18 S. 29).

Zur Modellierung der E/E-Architektur existieren mehrere Werkzeuge, unter anderem von den Herstellern dSPACE, Delphi, Mentor Graphics und Vector (18 S. 29f.). Vector verfolgt mit seinem Produkt PREEvision einen ganzheitlichen Ansatz, wodurch es bereits während der Konzeptionierung des Fahrzeugs eingesetzt werden kann (18 S. 30).

### **Architekturentwicklungswerkzeug PREEvision**

PREEvision von der Vector Informatik GmbH ist ein Computer-Aided-Software-Engineering-Werkzeug zur modellbasierten Entwicklung von E/E-Architekturen (18 S. 30). „Durch eine [d]omänenspezifische grafische Darstellung ermöglicht es die Modellierung von Anforderungen, logischem Funktionsnetzwerk, Komponenten- und Netzwerk-Architektur, Kabelsatz und zweidimensionalen räumlichen Strukturen in Form von Topologien. So ermöglicht es neben der Konzeptentwicklung auch die Evaluierung und Bewertung von Architektur-Alternativen über alle Modellierungsebenen. [...] Mithilfe von Regeln kann das Modell ebenenübergreifend auf Konsistenz überprüft werden.“ (18 S. 30) (vgl. Abbildung 4). Gemäß (29) ist der Workflow bei der Arbeit mit PREEvision oberflächlich betrachtet folgendermaßen einzuteilen:

1. Definition der gewünschten Funktionen in einem Fahrzeug
2. Aufteilen der Funktionen auf ECUs  
Daraus ergibt sich die Kommunikations-Matrix („wer kommuniziert was mit wem“), mit deren Hilfe die Verkabelung und die dazugehörigen Kosten berechnet werden.
3. Implementierung

Die Bewertung einer E/E-Architektur orientiert sich im Automobilbau stark an den Kosten: (7) nennt hierfür vier Teilkosten: Kabellänge, Anzahl der Links, benötigter Platz und Leistungsaufnahme der verbauten Geräte.

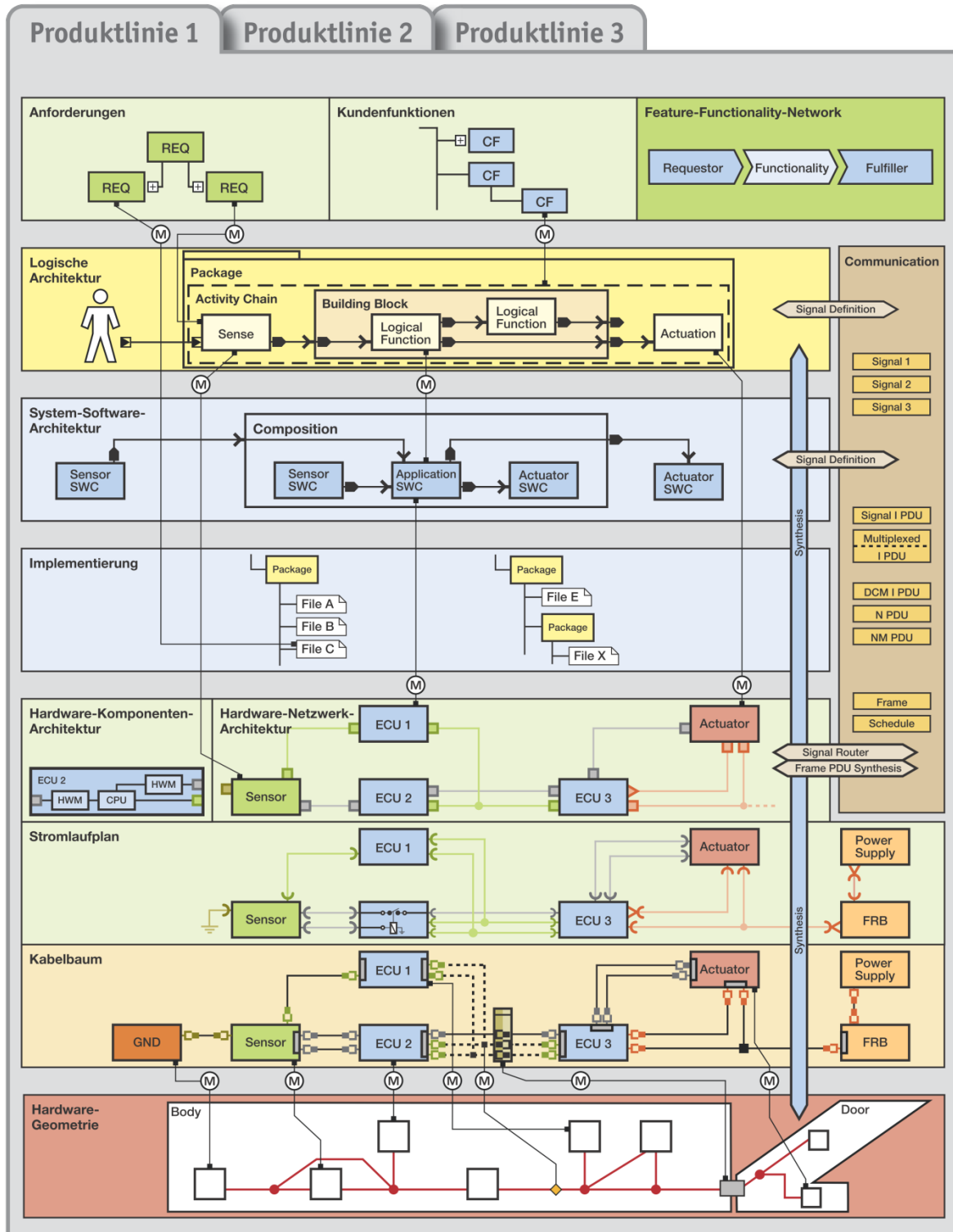


Abbildung 4: Visualisierung der verschiedenen Arbeits- und Darstellungsebenen in PREvision: Basierend auf der Definition der Anforderungen an ein Automobil und der gewünschten Funktionen werden die logische und die Software-Architektur entworfen. Danach erfolgen die Implementation der Funktionen, der Entwurf der elektrischen Schaltungen sowie die genaue Planung für den Einbau des Systems ins Automobil. Parallel dazu (auf der rechten Seite) werden nach und nach die zu verwendenden Signale definiert und ihr Versand in Nachrichten (eventuell zu bestimmten Zeitpunkten) festgelegt. Quelle: (47 S. 6)

## 3 Konzeption

Da der Kostenoptimierungsprozess bei der Entwicklung von Netzwerken wie in Kapitel 2.7 beschrieben zu mehreren günstigen Ergebnissen kommen, die Resultate jedoch nicht hinsichtlich ihrer Tauglichkeit auf Netzwerkebene bewerten kann, sollen die zu entwickelnden Lösungen dies übernehmen. Aufgabe ist es somit, ein in OMNeT++ simuliertes Netzwerk zu bewerten, um dem/der Netzwerkverantwortlichen beim Erstausrüster (OEM) die Entscheidung zwischen den möglichen Varianten zu erleichtern.

Dieser Prozess kann in drei Teile gegliedert werden:

1. Erhebung der Metriken des Netzwerks
2. Überprüfung der Metriken & Bewertung
3. Darstellung der Ergebnisse

In diesem Kapitel werden die Vorgaben und die bereits existierenden Programme analysiert, darauf aufbauend mehrere Konzepte zur Bewertung simulierter Netzwerke entworfen und schlussendlich ein Konzept ausgewählt.

### 3.1 Analyse der Vorgaben

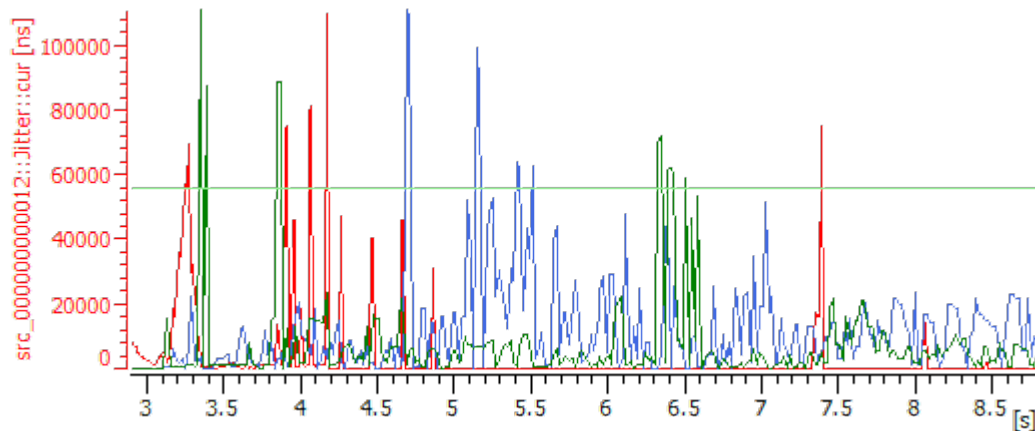
Im Folgenden werden die funktionalen und nichtfunktionalen Anforderungen erarbeitet. Für eine spätere Evaluation werden sie nummeriert.

In automobilen Ethernet-Netzwerken sind gemäß Kapitel 2.4 folgende Metriken von Bedeutung und müssen zur Bewertung eines Netzwerkes herangezogen werden [1]:

- Linkauslastung
- Datendurchsatz
- Puffergröße
- Latenzzeit
- Jitter
- Paketverlustrate

Diese Metriken sollen grafisch dargestellt werden [2]. Um dem/der Netzwerkverantwortlichen eine gewohnte Oberfläche zu bieten und ihm eine Weiterverarbeitung der Metriken zu ermöglichen, soll dies in CANoe (s. Kapitel 2.5) erfolgen. Zur Speicherung und Darstellung der Metriken eignen sich besonders und ausschließlich die Systemvariablen, denn sie können in CANoe auf einfache Weise dargestellt und weiterverarbeitet

werden (vgl. Abbildung 5). In CANoe sollen ebenfalls die Nachrichten im Trace-Fenster angezeigt werden [3].



**Abbildung 5:** Beispielhafte Darstellung des zeitlichen Verlaufs von Systemvariablen in CANoe

Die Metriken Linkauslastung, Datendurchsatz und Paketverlustrate sollen pro Link untersucht werden können [4]. Die Puffergröße soll für jeden Puffer in einem Switch dargestellt werden. Nützlich sind hierbei der Füllstand des Puffers zur Abschätzung der Verzögerungen der im Puffer gespeicherten Nachrichten sowie die Restgröße des Puffers, um abschätzen zu können, wie nah er an seiner maximalen Grenze betrieben wird [5]. Die Metriken Latenzzeit, Jitter und Paketverlustrate sollen pro Nachrichtentyp und Zwischenknoten zur Verfügung stehen, damit nachvollzogen werden kann, an welcher Stelle im Netzwerk Probleme z.B. in Form großer Verzögerungen entstehen [6]. Mit dem Nachrichtentyp werden im Folgenden alle Nachrichten gruppiert, welche dem gleichen Zweck dienen (bspw. alle Nachrichten eines Video-Streams, alle Nachrichten eines logischen Links oder alle CAN-Nachrichten mit einer bestimmten ID).

Werden in dem Netzwerk Gateways eingesetzt, so soll die Verfolgung einer Nachricht über die verschiedenen Bussysteme hinweg möglich sein [7].

Da das Netzwerk bewertet werden soll, müssen Informationen über die Anforderungen eingebracht werden. Diese Grenzwerte für die Metriken könnten zukünftig von einer externen Applikation (wie PREEvision) beigesteuert werden, sollen für den Beginn jedoch von dem/der Netzwerkverantwortlichen einstellbar sein. Es ist wünschenswert, dass mehrere Werte pro Metrik eingestellt werden können, welche Aufschluss über die Schwere des Verstoßes geben: Führt man bspw. die Schweregrade „Information“, „Warnung“ und „Fehler“ mit jeweils einem eigenen Grenzwert ein, so ist es möglich, einen Eindruck zu vermitteln, wie nahe sich das Netzwerk an den definierten Obergrenzen bewegt und wie gut es sich damit für den getesteten Einsatz eignet [8].

Zuletzt sollen die Ergebnisse der Bewertung des Netzwerkes anhand der Grenzwerte übersichtlich dargestellt werden, um eine schnelle Einschätzung zu ermöglichen, ob das Netzwerk gut oder schlecht abschneidet [9]. Wünschenswert ist ebenfalls, dass eine

gegebenenfalls zu erstellende externe Applikation in die Oberfläche von CANoe eingebunden wird [10] und der Nutzer/die Nutzerin bei der Auswahl der darzustellenden Metriken in CANoe unterstützt wird [11]. Vorstellbar ist bspw., dass nach einem Klick auf eine Verbindung in der schematischen Darstellung des simulierten Netzwerks ein CANoe-Graph mit den dazugehörigen Metriken geöffnet wird.

Da die die Bewertung eines Netzwerks teilweise ein interaktiver Prozess zwischen der Analyse-Software und dem/der Netzwerkverantwortlichen ist, sollen die Ergebnisse einer Analyse innerhalb kurzer Zeit berechnet werden können [12]. Die Simulation eines Netzwerks in OMNeT++ dauert meist mehrere Stunden und wird somit bspw. über Nacht ausgeführt. Wird die Analyse parallel zur Simulation berechnet, so sollen die Ergebnisse dem Nutzer/der Nutzerin instantan präsentiert werden können. Erfolgen die Berechnungen allerdings erst nach der Simulation und auf Geheiß des Nutzers/der Nutzerin, so dürfen sie höchstens wenige Minuten dauern, damit die Interaktion nicht beeinträchtigt wird.

Aufgrund der Tatsache, dass CANoe nur auf Windows-Systemen ausgeführt werden kann, genügt es, wenn die entwickelte Lösung ebenfalls nur auf diesen Systemen funktioniert. Eine Portierung auf andere Betriebssysteme muss somit nicht vorgenommen werden.

Es ist allerdings wünschenswert, dass die erarbeitete Lösung mit wenig Aufwand erweitert werden kann [13]. Dies betrifft einerseits neue Bussysteme und Echtzeit-Ethernet-Protokolle, andererseits aber auch das Hinzufügen von weiteren Metriken.

## 3.2 Betrachtung der Programme OMNeT++ und CANoe

Da OMNeT++ (s. Kapitel 2.2) die Quelle der Informationen über das Netzwerk und CANoe (s. Kapitel 2.5) die Senke dieser Information sein sollen, werden nun beide Programme hinsichtlich bereits vorhandener Features zur Erhebung von Metriken möglicher Einschränkungen untersucht.

### OMNeT++

Während der Simulation in OMNeT++ werden bereits einige der zu verwendenden Metriken berechnet: Unter anderem stehen die Ende-zu-Ende-Latenz für jeden Nachrichtentyp und somit auch der Jitter über sogenannte Statistik-Vektoren nach der Simulation zur Verfügung. Die Linkauslastung wird skalar als Durchschnitt über die gesamte Simulation angegeben. Eine Erhebung der Linkauslastung als Statistik-Vektor kann jedoch – wie auch für die anderen Metriken – durch Anpassungen an der Simulation hinzugefügt werden.

OMNeT++ 5.0, welches im April 2016 veröffentlicht wurde, bietet erstmals den Einsatz von sogenannten `Event-` und `Vector-Manager` an. Die `Event-Manager` werden über jedes Event in der Simulation (von der Erstellung eines Gates oder einer Nachricht über deren Versand oder Verlust bis hin zu einem neuen Eintrag im Simulations-Protokoll) informiert. Sie können somit detailliert auf die Simulationsergebnisse zugreifen.

`Vector-Manager` werden über neue Werte in Statistik-Vektoren informiert. Dies birgt gegenüber dem `Event-Manager` zwei Vorteile: Zum einen ist die Laufzeit geringer, da nicht so viele Events verarbeitet werden müssen. Zum anderen wird die Erhebung und Weiterverarbeitung (wie bspw. Filterung oder Mittelung) der Metriken von bereits vorhandenen Programmteilen übernommen, wodurch die Werte in den Vector-Dateien mit denen in den BLF-Dateien identisch sind. Außerdem ist es für den Nutzer/die Nutzerin einfacher, die Erhebung z.B. durch eine Veränderung der Durchschnittsbildung zu modifizieren, da hierbei keine erneute Kompilation notwendig ist. Mit einem `Vector-Manager` können allerdings nur Metriken aufgezeichnet werden, welche sich als Statistik-Vektor definieren lassen.

### CANoe

Da sich CANoe historisch an Bussystemen wie CAN und LIN orientiert, unterstützt das Programm bis zur aktuellen Version 9.0 keine Netzwerktopologien. Als „Bus“ wird dort das physikalische Medium betrachtet, an das alle Knoten angeschlossen sind, welche somit auch alle Nachrichten empfangen können. Mehrere „Busse“ sind logisch voneinander getrennt und können durch „Gateways“ nur aktiv während einer Online-Simulation überbrückt werden. Gerade bei Switched-Ethernet ergibt sich daraus das Problem, dass Netzwerke nicht als solche betrachtet werden können, da sie aus mehreren physikalischen Medien bestehen. Auf einem „Bus“ kann immer nur ein Punkt des Netzwerks (z.B. ein Port eines Knotens) abgebildet werden; ein Zusammenhang zwischen diesen Punkten besteht in CANoe nicht. Dies ist vergleichbar mit Messspitzen, mittels derer eine elektrische Schaltung an einem Punkt von einem Oszilloskop überwacht wird (s. Abbildung 6 und Abbildung 7).

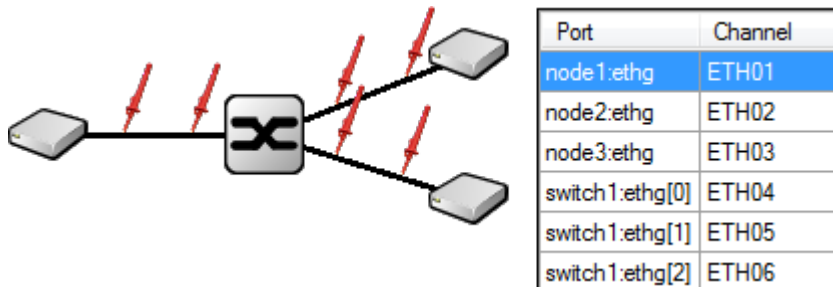


Abbildung 6: Kleines Ethernet-Netzwerk bestehend aus drei Knoten und einem Switch mit „Messspitzen“ an jedem Port. Links: node1, rechts oben: node2, rechts unten: node3, mittig: switch1. In der Tabelle wird die Beziehung zwischen dem gewählten Port im Netzwerk und dem Kanal in CANoe aufgezeigt.

Time	Source MAC	Chn	ID	Dir	DLC	Et...	Data
00:00:00	1.561006 0A:00:00:00:00:01	Eth 1		Tx	46	891D	2F 05 A2 22 3F 3F 3F 3F :
	1.561006 0A:00:00:00:00:01	Eth 4		Tx	46	891D	2F 05 A2 22 3F 3F 3F 3F :
	1.561009 0A:00:00:00:00:01	Eth 5		Tx	1500	0000	53 33 9D 49 3F 3F 3F 3F :
	1.561009 0A:00:00:00:00:01	Eth 2		Tx	1500	0000	53 33 9D 49 3F 3F 3F 3F :
	1.561031 0A:00:00:00:00:01	Eth 5		Tx	46	891D	2F 05 A2 22 3F 3F 3F 3F :
	1.561031 0A:00:00:00:00:01	Eth 6		Tx	46	891D	2F 05 A2 22 3F 3F 3F 3F :
	1.561031 0A:00:00:00:00:01	Eth 3		Tx	46	891D	2F 05 A2 22 3F 3F 3F 3F :
	1.561031 0A:00:00:00:00:01	Eth 2		Tx	46	891D	2F 05 A2 22 3F 3F 3F 3F :

Abbildung 7: Exemplarische Darstellung im Trace-Fenster in CANoe. Die Ethernet-Busse („Chn“) sind gleichzusetzen mit den Messpunkten in Abbildung 6. Es sendet node1 eine Nachricht über switch1 an node2 und node3.

Da in CANoe eine bereits durchgeführte Simulation dargestellt bzw. analysiert werden soll, muss der Offline-Modus wie in Kapitel 2.5 beschrieben eingesetzt werden. Hierbei eignet sich das nachrichtenorientierte BLF-Format (vgl. Kapitel 2.6) am besten: Dieses kann gleichzeitig Nachrichten-Ereignisse und Werte von Systemvariablen enthalten, was die Berechnung der Metriken sowohl innerhalb als auch außerhalb CANoes ermöglicht.

Sowohl in der Offline- als auch in der Online-Simulation beschränkt CANoe die Anzahl der Busse pro Bussystem auf 32 Stück. Dies ist eine Einschränkung, die bei der Konzeption der Betrachtung der Simulationsergebnisse beachtet werden muss.

Eine Unterstützung des Nutzers/der Nutzerin bei der Auswahl der darzustellenden Metriken ist mit CANoe bisher nicht möglich: Über die COM-Schnittstelle (s. Kapitel 2.5) kann CANoe konfiguriert werden, allerdings ist keine Interaktion mit der grafischen Oberfläche möglich. Somit können darüber keine Graphen geöffnet oder geschlossen werden; auch eine Auswahl der im Graphen anzuzeigenden Variablen ist nicht möglich. Selbst wenn dies möglich wäre, würde das Konzept der Graph-Datensenken eine bequeme Nutzung erschweren: Um möglichst effizient zu arbeiten, werden nur die Werte von Systemvariablen für den zeitlichen Verlauf gespeichert, welche angezeigt werden sollen. Wird eine Variable später hinzugefügt, so verfügt sie erst einmal über keinen Verlauf. Eine Darstellung kurz nach dem Klick des Nutzers/der Nutzerin wäre somit nur möglich, wenn die Metriken direkt aus der BLF-Datei gelesen oder sehr schnell in CANoe berechnet werden könnten.



### 3.3 Ansätze zur Bewertung in OMNeT++ simulierter Netzwerke

Wie bereits beschrieben, soll ein in OMNeT++ simuliertes Netzwerk analysiert und bewertet werden, um dem/der Netzwerkverantwortlichen die Entscheidung zwischen möglichen Varianten zu erleichtern.

Da sowohl Zeitverläufe der Metriken als auch Nachrichten-Events über das BLF-Format (vgl. Kapitel 2.6) in CANoe eingespeist werden können, besteht die Möglichkeit, die drei Teilaufgaben

1. Erhebung der Metriken des Netzwerks,
2. Überprüfung der Metriken & Bewertung sowie
3. Darstellung der Ergebnisse

in unterschiedlichen Programmen zu lösen. Es ist somit möglich, ein verteiltes System einzusetzen. Dies gilt besonders für den ersten Schritt: Die Werte der Metriken können

- während der Simulation in OMNeT++,
- nach der Simulation in einem externen Programm oder
- nach der Simulation in CANoe

erhoben werden. Diese drei Möglichkeiten und ihr Einfluss auf die weiteren Schritte sollen nun untersucht und ausgearbeitet werden.

#### 3.3.1 Erhebung der Metriken in OMNeT++

Findet die Erhebung der Metriken in OMNeT++ statt, so besteht die Schnittstelle zu CANoe lediglich aus den BLF-Dateien, in denen die Nachrichten-Events und Metrik-Werte gespeichert werden.

Wie bereits erwähnt ist die Erhebung der Ende-zu-Ende-Latenz in OMNeT++ bereits implementiert; ihr Verlauf wird nach der Simulation über Statistik-Vektoren zur Verfügung gestellt. Die Latenzwerte an zwischengeschalteten Switches oder Gateways werden ebenfalls erhoben, stehen allerdings nicht pro Nachrichtentyp zur Verfügung. Die Simulation muss deshalb erweitert werden, um alle benötigten Informationen zur Verfügung zu stellen. Außerdem müssen an den geeigneten Stellen neue Statistiken hinzugefügt werden, welche die anderen Metriken erheben. Mindestens in Teilen ist dies bereits durch eine Änderung der Simulationskonfiguration möglich.

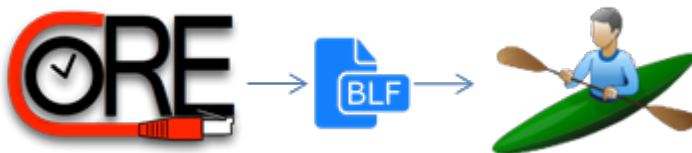
Um diese Metriken zu speichern und in CANoe weiterverarbeiten zu können, könnten entweder die Statistik-Dateien von OMNeT++ in BLF-Dateien umgewandelt oder die Werte bereits während der Laufzeit in BLF-Dateien geschrieben werden.

Für ersteres muss ein Konverter entwickelt werden, welcher sowohl die Statistik-Dateien lesen als auch BLF-Dateien schreiben kann. Nachteilig ist hierbei einerseits die enorme

Größe der ungefilterten Statistik-Dateien, welche zumindest temporär gespeichert werden müssen, sowie andererseits die verzögerte Verarbeitung der Daten: Der/die Netzwerkverantwortliche muss die Konvertierung nach der Simulation anstoßen und auf das Ergebnis warten.

Ein direktes Schreiben der Werte in BLF-Dateien ist somit angeraten. Hierfür eignen sich die in Kapitel 3.2 beschriebenen `Event-Manager` und `Vector-Manager`. Da der `Event-Manager` über alle Ereignisse informiert wird, muss anhand dieser Ereignisse manuell die Erhebung der Metriken implementiert werden. Effizienter ist deshalb wohl der Einsatz von `Vector-Managern`: Da die Erhebung der Metriken von bereits vorhandenen Programmteilen übernommen wird, sind die Werte in den Vector-Aufzeichnungen von OMNeT++ mit denen in den vom `Vector-Manager` aufgezeichneten BLF-Dateien identisch. Außerdem kann der Nutzer/die Nutzerin durch eine Anpassung der Statistik-Definition wesentlichen Einfluss auf die Erhebung der Metriken nehmen, ohne eine erneute Kompilation durchführen zu müssen. Solche Anpassungen sind bspw. Änderungen an der Methode, wie der fortlaufende Durchschnitt der Busauslastung berechnet wird.

Die Metriken werden also bisher in OMNeT++ erhoben und direkt in eine BLF-Datei gespeichert (s. Abbildung 8). Die darauf folgende Überprüfung der Metriken sowie die Bewertung des Netzwerks kann ebenfalls wieder an verschiedenen Stellen geschehen: Entweder werden die Metriken sofort in OMNeT++ mit den Grenzwerten verglichen. In diesem Fall können die Ergebnisse dieser Überprüfung als Systemvariablen ebenfalls in der BLF-Datei gespeichert werden. Hierfür existiert bereits ein Modul innerhalb der CoRE-Gruppe, das für die Ausgabe von BLF-Dateien angepasst werden könnte. Genauso verhält es sich, wenn die Überprüfung in einer externen Applikation vorgenommen wird, welche jedoch erst entwickelt werden müsste (vgl. Kapitel 3.3.2). Schlussendlich ist es ebenso möglich, dass Überprüfung und Bewertung in CANoe vorgenommen werden (vgl. Kapitel 3.3.3).



**Abbildung 8:** Schematische Abbildung der Informationsübermittlung: Die Simulationsdaten werden in OMNeT++ bereits derart aufbereitet, dass sie nur noch in CANoe abgespielt werden müssen. Als Übertragungsmedium dienen BLF-Dateien.

Probleme könnte dieser Ansatz – die Erhebung der Metriken in OMNeT++ – jedoch bei der Gleichzeitigkeit der Ereignisse (bzw. der Gleichheit der Zeitstempel) aufweisen: Werden die Nachrichten für das Trace-Fenster von CANoe an den Ports der Knoten aufgenommen, kann dies (durch Puffer etc.) zu einem anderen Simulationszeitpunkt geschehen als die Berechnung und Speicherung der Metriken, z.B. der Latenz. Bei der nachträglichen Betrachtung in CANoe könnte somit der kausale Zusammenhang zwischen Metrikerwert (z.B. Latenzwert) und eingehender Nachricht verloren gehen (vgl. Abbildung 9).

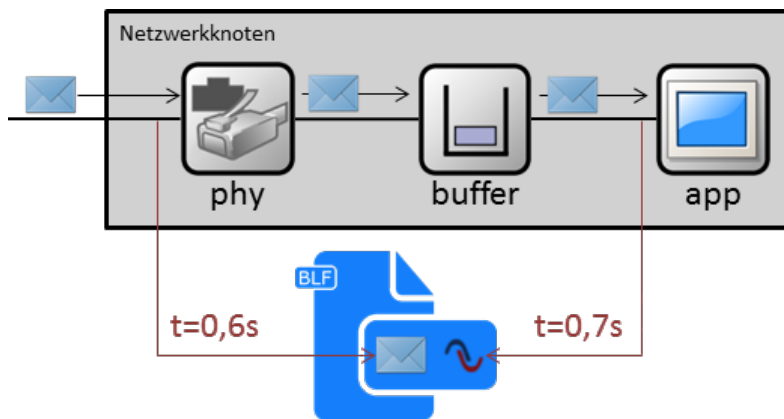


Abbildung 9: Schematische Darstellung zum Verlust der Gleichheit der Zeitstempel: Eine Nachricht wird vom Netzwerkport („phy“) empfangen und an dieser Stelle für das Trace-Fenster in die BLF-Datei geschrieben. Über einen Puffer („buffer“) wird die Nachricht an die Applikation („app“) weitergegeben. Diese berechnet mithilfe der Nachricht einen Metrik-Wert und schreibt ihn ebenfalls in die BLF-Datei. Durch die Verzögerung im Puffer sind die Zeitstempel der logisch miteinander verknüpften Objekte (Nachricht & Metrik-Wert) verschieden.

Diesem Problem wird vorgebeugt, wenn die Berechnung der Metriken ausschließlich anhand der Nachrichten-Events (vgl. Kapitel 2.2) geschieht: Hierfür werden nur Nachrichten mit ihren Zeitstempeln in die BLF-Datei geschrieben. Diese BLF-Datei wird nach der Simulation gelesen und für jede enthaltene Nachricht die Metriken berechnet. Ein solcher Ansatz birgt einen weiteren Vorteil: Der Verkehr eines realen Netzwerkes kann mithilfe von Datenloggern ebenfalls in BLF-Dateien gespeichert und anschließend mit demselben Programm analysiert werden wie zuvor das simulierte Netzwerk. Dies macht den Vergleich von Simulation und Realität sehr einfach. Erfolgt die Analyse direkt in CANoe, kann dies (bei ausreichender Rechenleistung des Systems) unter Umständen sogar live ohne den Umweg über BLF-Dateien geschehen. Die beiden nachfolgenden Ansätze setzen diese Vorgehensweise um.

### 3.3.2 Erhebung der Metriken in einer externen Applikation

Im Folgenden wird ein Konzept entwickelt, in welchem eine externe Applikation die von OMNeT++ geschriebenen BLF-Dateien mit den simulierten Nachrichtenereignissen liest, die Metriken und Analyseergebnisse berechnet und als Systemvariablen ebenfalls wieder in eine BLF-Datei speichert.



Abbildung 10: Die Aufbereitung der Simulationsdaten findet nicht in OMNeT++ statt, sondern in einer externen Applikation. Sie berechnet aus den in der ersten BLF-Datei enthaltenen Nachrichten die Metriken und schreibt sie wiederum in eine BLF-Datei, welche dann von CANoe eingelesen und angezeigt wird.

Für die Durchführung einer solchen nachträglichen Analyse (nach der Simulation) sind Informationen über die Netzwerktopologie von Nöten: Bspw. wird die Datenübertragungsrates eines jeden Links für die Berechnung der Linkauslastung benötigt. Ebenso müssen die Verbindungen der Netzwerk-Knoten innerhalb des Netzwerks bekannt sein, um Multi- und Broadcast-Nachrichten zu erkennen (s. Kapitel 4.3.1). Diese Informationen sind in den NED- und INI-Dateien gespeichert, welche das Netzwerk in OMNeT++ beschreiben. Sie müssen somit von der externen Applikation eingelesen und verarbeitet werden.

Darüber hinaus ist es nötig, in OMNeT++ den Nachrichten beim Speichern eine Information beizufügen, welche sie eindeutig identifiziert: Nur anhand eines solchen eindeutigen Merkmals kann die Nachricht über die verschiedenen Kanäle in der BLF-Datei (s. Kapitel 2.6) hinweg wiedererkannt werden. Das BLF-Format enthält einige ungenutzte Felder, welche für diesen Zweck verwendet werden können.

Es unterstützt ebenfalls die Aufzeichnung von Übertragungsfehlern, weshalb eine Berechnung der Paketverlustrate auf diesem Weg durchaus möglich wäre. Da fehlerhafte Übertragungen in OMNeT++ jedoch nur auf Empfangsseite (im empfangenden Netzwerkport) registriert werden, kann nur der Wert für eingehende Nachrichten angegeben werden. Teilweise gelöst werden könnte dieses Problem, wenn die Nachrichten und die Übertragungsfehler in OMNeT++ nicht im Port sondern in der Verbindung selbst aufgezeichnet werden (vgl. Kapitel 4.3.4).

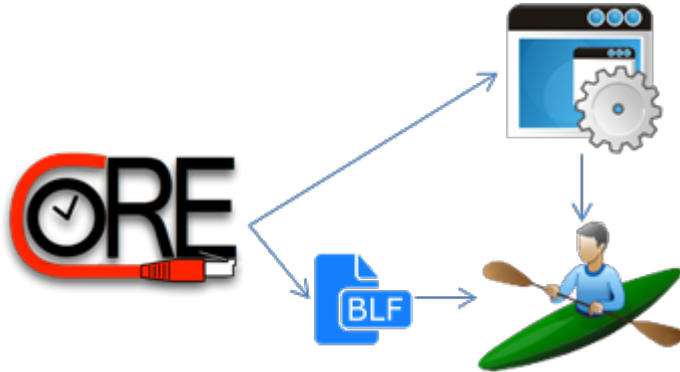
Das Überprüfen der erhobenen Werte der Metriken an den Grenzwerten kann ebenfalls in der externen Applikation geschehen. Die Ergebnisse dieser Analyse würden – so wie bei der Analyse in OMNeT++ in Kapitel 3.3.1 – als Systemvariablen in der BLF-Datei gespeichert und somit an CANoe weitergegeben werden.

### **3.3.3 Erhebung der Metriken in CANoe**

Die Verarbeitung der Simulationsdaten in CANoe ist der Verarbeitung in einer externen Applikation (s. Kapitel 3.3.2) in manchen Teilen sehr ähnlich, da auch CANoe von Haus aus über keine Informationen über das simulierte Netzwerk verfügt. Statt die Metriken jedoch in einer dritten Applikation zu erheben, finden diese Berechnungen direkt in CANoe (in einem CAPL-Programm) statt. Da das Lesen von Nachrichten aus einer BLF-Datei in CANoe durch realen Netzwerkverkehr ersetzt werden kann, birgt dieses Konzept den Vorteil, dass unter Umständen auch eine Echtzeit-Analyse von Netzwerken durchgeführt werden könnte. Ebenso ist es von Vorteil, dass CANoe auf die Verarbeitung von Nachrichten spezialisiert ist und somit eine hohe Geschwindigkeit verspricht.

Es existieren jedoch einige Einschränkungen: So ist bspw. das Lesen von Dateien in CAPL nicht möglich. Die NED- und INI-Dateien, welche das Netzwerk beschreiben, können somit nicht eingelesen werden. Die darin enthaltenen Daten können jedoch – wie in Kapitel 3.3.2 – von einer externen Applikation gelesen, aufbereitet und anschließend an CANoe weitergereicht werden. Dies ist bspw. über die Generierung von Programm-Code möglich.

Ebenso können diese Informationen aber auch in der Payload von Pseudo-Nachrichten in BLF-Dateien abgelegt oder über die COM-Schnittstelle in Systemvariablen übertragen werden (s. Abbildung 11).



**Abbildung 11:** Die Aufbereitung der Simulationsdaten findet nicht in OMNeT++ statt, sondern in CANoe. Eine externe Applikation stellt dabei CANoe die Informationen über das simulierte Netzwerk zur Verfügung.

Auf die ungenutzten Felder des BLF-Formats, welche im vorherigen Kapitel zur eindeutigen Identifizierung von Nachrichten über verschiedene BLF-Kanäle hinweg verwendet wurden, kann aus einem CAPL-Programm nicht zugegriffen werden. Deshalb muss ein solcher Identifier in den Nutzdaten der Nachrichten verpackt werden. Diese Nutzdaten spielen bei der (hier angewandten) System-Level-Simulation keine Rolle, weshalb eine Änderung derselben kein Problem darstellt. In einem realen Steuerungsnetzwerk werden Nachrichten jedoch oft zyklisch übertragen, wodurch die Nutzdaten mehrerer aufeinanderfolgender Pakete meistens identisch sind. In diesem Fall können die Nachrichten nicht mehr korrekt verfolgt werden und die berechneten Metriken können fehlerhaft sein. Dieser Fall wurde jedoch nicht näher untersucht, da er zu weit von der Aufgabenstellung entfernt ist.

Der Typ einer Nachricht (im Folgenden: Nachrichtentyp) kann in einem heterogenen Netzwerk anhand mehrerer Merkmale identifiziert werden. Bei Ethernet bieten sich hierfür bspw. die Quell- und Ziel-MAC-Adressen an. Die Echtzeit-Ethernet-Protokolle führen weitere Identifier ein, mit welchen ein Datenstrom gekennzeichnet wird (vgl. Kapitel 4.3.5). Die Nutzung der MAC-Adresse stößt an ihre Grenze, wenn in dem Netzwerk Router zum Einsatz kommen, da diese die MAC-Adressen bei der Weiterleitung ändern. Sollte dies vorkommen, muss der Einsatz von IP-Adressen implementiert werden. Sollen mehrere Streams zwischen zwei IP-Knoten unterschieden werden können, müssen noch Identifier von den eingesetzten Multiplex-Protokollen (wie TCP oder UDP) hinzugefügt werden. Um Nachrichten auf Feldbussen wie CAN oder FlexRay zu identifizieren, kann auf die Identifier (z.B. CAN-ID und Slot-ID) des jeweiligen Bussystems zurückgegriffen werden.

Eine Unterstützung von Gateways kann aufgrund der unterschiedlichen Adressierungsparadigmen der Bussysteme (Quelladressierung vs. Zieladressierung) sehr kompliziert werden. In der CoRE-Arbeitsgruppe werden Gateways jedoch bisher nur eingesetzt, um Feldbus-Nachrichten über Ethernet zu übertragen. Ebenso wird hierbei darauf verzichtet,

den Identifier nach der Übertragung zu verändern. Somit kann auf eine Unterstützung unterschiedlicher Identifier auf verschiedenen Bussystemen für denselben Nachrichtentyp verzichtet werden.

Die Berechnung der Metriken in CANoe bietet weitere Möglichkeiten, welche unter anderem eine besser Darstellung der Bewertungsergebnisse ermöglichen: Durch die in Kapitel 2.5 beschriebene COM-Schnittstelle oder das FDX-Protokoll können die Metrik-Werte an eine externe Applikation weitergereicht werden, welche die Bewertung übernimmt und wiederum an CANoe zurück meldet. Der Vorteil hierbei liegt darin, dass die Bewertung der Metriken wesentlich komplexer ausfallen kann, als in der Programmiersprache CAPL (s. Kapitel 2.5) möglich ist. Nachteilig ist jedoch, dass insbesondere die Kommunikation über die COM-Schnittstelle viele Thread-Kontext-Wechsel enthält und somit sehr langsam ist.

### 3.4 Vergleich der Konzepte

Im Rahmen dieser Arbeit werden zwei der drei vorgestellten Konzepte umgesetzt und miteinander verglichen. Deshalb werden an dieser Stelle zwei Konzepte ausgewählt und konkretisiert.

Wie in den vorherigen Kapiteln deutlich wird, ergeben sich durch eine spätere Erhebung der Metriken immer mehr Einschränkungen: Wird sie in OMNeT++ vorgenommen, so kann auch die Bewertung in OMNeT++ und somit zur Simulationszeit durchgeführt werden; ebenso ist es jedoch auch möglich, die Bewertung in einer externen Applikation oder in CANoe vorzunehmen. Werden die Metriken jedoch erst in der externen Applikation erhoben, kann das Netzwerk nur noch dort oder in CANoe bewertet werden. Eine Ausnahme bildet die Metrik-Erhebung in CANoe: Da diese über seine Schnittstellen (s. Kapitel 2.5) mit anderen Programmen kommunizieren kann, bleiben eine Bewertung sowie die Darstellung der Ergebnisse sowohl in einer externen Applikation als auch in CANoe möglich.

Im Folgenden wird erarbeitet, wie gut sich welche Vorgehensweise für die Erhebung der Metriken eignet. Zur Auswahl stehen der `Event-Manager` in OMNeT++, eine externe Applikation sowie CANoe. Die Nachrichten-Ereignisse werden in allen drei Konzepten in BLF-Dateien gespeichert; hierfür muss ein entsprechendes Modul in OMNeT++ integriert werden.

Die Definition und Erhebung der Metriken sind in OMNeT++ über die Statistik-Vektoren relativ einfach zu implementieren, während dies im `Event-Manager`, in der externen Applikation sowie in CANoe neu programmiert werden muss. Somit lässt sich dieser Vorgang in OMNeT++ bei Verwendung des `Vector-Managers` auch leichter verändern; die Netzwerkverantwortlichen werden jedoch mit CANoe vertraut sein und somit auch im dortigen Code Änderungen vornehmen können. Eine Anpassung in einer externen Applikation verlangt jedoch nach einer Einarbeitung in dessen Quelltext sowie nach der Verfügbarkeit von Visual Studio und ist somit aufwändig.

Die Statistik-Vektoren von OMNeT++ können verschiedene Ereignisse innerhalb eines Moduls zusammenführen und darauf aufbauend eine Metrik berechnen; jedoch ist es bspw. nicht möglich, die Ereignisse mehrerer Module miteinander zu verknüpfen, wie es für die Berechnung der Paketverlustrate nötig ist (s. Kapitel 4.3.4). Hier ist der `Event-Manager` dem `Vector-Manager` überlegen, jedoch ist auch der Entwicklungs-Aufwand höher und die Möglichkeiten der Anpassung sind kleiner. Schwierig ist ebenfalls die Aufzeichnung von Diese Überlegungen werden in Tabelle 1 visualisiert.

	OMNeT++		Externe App.	CANoe
	Events	Vektoren		
Aufzeichnen der Nachrichten		2	-	-
Einfachheit der Metriken-Berechnung	3	1	3	3
Aufzeichnen der Metriken in BLF		2	2	-
Support von Multi-/Broadcast		2	3	3
Support von Gateways		2	2	2
Mögliche Komplexität der Metriken	1	2	2	2
Anpassungsmöglichkeit der Metriken	3	1	3	2
Entwicklungsaufwand	3	2	3	2

**Tabelle 1: Abschätzung der Qualität drei Konzepte hinsichtlich der Erhebung der Metriken und der damit verbundenen Möglichkeiten**

Nun werden die weiteren Schritte des Analyse- und Bewertungsprozesses betrachtet. Für eine Bewertung der Metriken müssen Anforderungen an das Netzwerk eingebracht werden, bspw. Maximalwerte für Latenz und Jitter. Hierfür ist es nötig, ein entsprechendes Dateiformat zu entwickeln. Denkbar ist bspw. die Verwendung von XML oder JSON als Grundlage des Formats. Bei der Verwendung von Statistik-Vektoren kann in OMNeT++ ein bereits vorhandenes Modul der CoRE-Gruppe zur Bewertung verwendet werden. In den anderen Fällen muss die Bewertung selbst implementiert werden. Nach der Bewertung müssen die Werte unter Umständen in eine BLF-Datei gespeichert werden. Die Darstellung der Ergebnisse kann – wie oben beschrieben – in CANoe als auch in einer externen Applikation erfolgen. In der externen Applikation ist jedoch eine ausgefeiltere Visualisierung möglich, z.B. indem die System-Metriken (s. Kapitel 2.4) innerhalb des Netzwerks angezeigt werden (s. Abbildung 12). Diese Überlegungen werden in Tabelle 2 visualisiert.

Ein generelles Hindernis ist, dass CANoe aktuell nur bis zu 32 Busse pro Bussystem unterstützt (s. Kapitel 2.5). Gerade bei Netzwerken mit CAN-Verbindungen könnte durch die Aufzeichnung einer jeden Verbindung diese Grenze leicht gesprengt werden, wodurch die Nachrichten auf den restlichen Bussen von CANoe ignoriert würden. Es liegen bereits

jetzt Netzwerke mit über 50 CAN-Links vor. Eine Auswahl der zu betrachtenden Verbindungen ist somit nur auf zwei Wegen möglich: Entweder wird direkt in OMNeT++ ausgewählt, welcher Link in CANoe dargestellt werden sollen (z.B. mittels Optionen in INI-Dateien oder durch ein Plugin). Dann werden auch nur diese Verbindungen aufgezeichnet und können ohne weitere Verarbeitung in CANoe eingelesen werden. Eine Änderung der Auswahl würde dann jedoch auch eine erneute Simulation des Netzwerkes erfordern.

Die zweite Möglichkeit ist eine nachgelagerte Auswahl der Links. Da CANoe beim Einlesen von BLF-Dateien bisher keine Möglichkeit bietet, die zu nutzenden Kanäle auszuwählen, muss hierfür ein externes Programm eingesetzt werden, welches die BLF-Dateien analysiert und nötigenfalls die umschreibt.

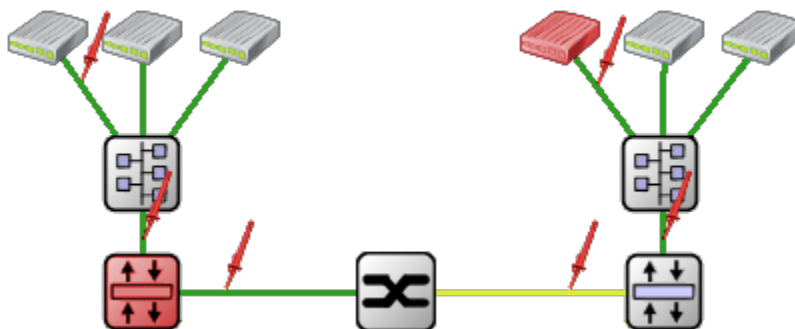


Abbildung 12: Beispielhafte Darstellung der Bewertungsergebnisse der System-Metriken eines Netzwerkes mit Gateways. Probleme wie Pufferüberläufe könnten durch rotgefärbte Knoten angezeigt werden, die Linkauslastung durch eine Färbung der Verbindungen.

	OMNeT++		Externe App.	CANoe
	Events	Vektoren		
Einlesen der Grenzwerte		2	1	2
Aufwand der Metriken-Bewertung	3	2	2	2
Aufzeichnen der Bewertung		2	1	-
Darstellung der Ergebnisse	-	-	1	2

Tabelle 2: Abschätzung der Qualität drei Konzepte hinsichtlich der Bewertung des Netzwerkes und der Darstellung der Ergebnisse

### 3.5 Auswahl und Konkretisierung der finalen Konzepte

Wie in Tabelle 1 und Tabelle 2 ersichtlich wird, verfügt jedes Konzept über Vor- und Nachteile. Am besten schneidet jedoch die Erhebung der Metriken in OMNeT++ mittels eines `Vector-Manager`s ab (s. Kapitel 3.3.1), weshalb die Wahl darauf fällt. Ebenfalls ausgewählt wird die Erhebung der Metriken in CANoe (s. Kapitel 3.3.3), da diese den zweiten Platz belegt und sich vom ersten Ansatz deutlich unterscheidet.



Diese beiden Konzepte werden im Folgenden konkretisiert.

Während im ersten Konzept die Metriken direkt in OMNeT++ erhoben werden, verwendet das zweite Konzept nur die Nachrichten-Ereignisse und die Puffergrößen von OMNeT++ und speichert diese in BLF-Dateien (s. Kapitel 3.3.3). Die Bewertung der Metriken erfolgt in beiden Fällen in CANoe selbst. Eine Begleit-Applikation generiert die für CANoe notwendigen Dateien.

Zum Aufzeichnen der Nachrichten wird ein passendes OMNeT++-Modul verwendet. Es wird derart konfiguriert, dass ein- und ausgehende Nachrichten in den Links oder an den Netzwerkports registriert und in BLF-Dateien geschrieben werden (s. Abbildung 13). Als Payload der Nachrichten wird eine eindeutige Zahl verwendet (Unique ID, UID, vgl. Kapitel 3.3), um die weitere Verarbeitung in CANoe zu ermöglichen. Das erste Konzept erstellt mit Hilfe des `Vector-Managers` eine weitere BLF-Datei, welche die erhobenen Metriken enthält.



**Abbildung 13:** Schematische Darstellung der Simulation in OMNeT++ mit einer BLF-Datei, in welcher die Ergebnisse der Simulation festgehalten werden.

Eine externe Applikation übernimmt in beiden Fällen die Aufgabe der Konfiguration von CANoe: Da die Anzahl der Busse in CANoe beschränkt ist (vgl. Kapitel 2.5), wird dem Nutzer/der Nutzerin mithilfe der NED-Dateien die Möglichkeit gegeben, grafisch die Verbindungen des Netzwerks auszuwählen, welche er anzeigen bzw. überprüfen möchte (s. Abbildung 14).

Werden die Metriken in CANoe erhoben, ist es zu Beginn einer Analyse sinnvoll, die Endpunkte eines Netzwerks, also die Netzwerkports der Endknoten zu wählen, um die Benutzer-Metriken (s. Kapitel 2.4) berechnen zu können. Für eine weitergehende Analyse kann bspw. der Pfad einer Nachricht gewählt werden, um zu prüfen, an welcher Stelle welches Problem (z.B. eine zu hohe Latenz) entsteht. Bei der Erhebung der Metriken in OMNeT++ ist die Wahl der Links für die Bewertung des Netzwerkes irrelevant und dient nur der Anzeige im Trace-Fenster in CANoe.

Die Nachrichten der ausgewählten Verbindungen werden derart in eine neue BLF-Datei verpackt, dass sie sich unter den ersten 32 Kanälen befinden und somit von CANoe

verarbeitet werden können. Zusätzlich werden Definitionen für Systemvariablen erzeugt, damit diese in CANoe verwendet werden können.

Beim ersten Konzept wird ein CAPL-Programm erzeugt, welches die Überprüfung der Systemvariablen anhand von Grenzwerten vornimmt, welche ebenfalls in Systemvariablen gespeichert sind. Beim zweiten Konzept werden weitere CAPL-Dateien erzeugt, welche u.a. Aufschluss über die Netzwerkkonfiguration und über die zu analysierenden Nachrichtentypen geben (s. Abbildung 14).

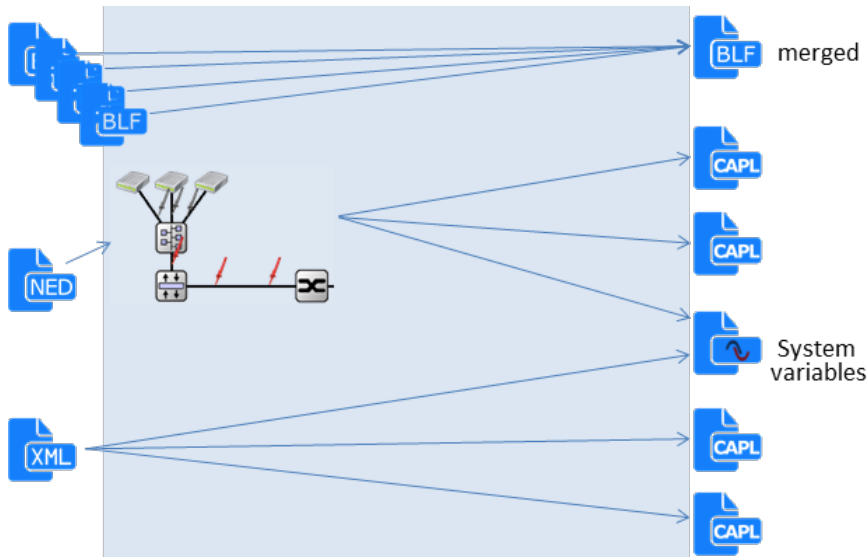


Abbildung 14: Schematische Darstellung der Verarbeitung der Informationen in Compagnon. Die erzeugten Dateien werden anschließend in CANoe eingespeist.

Bei der Erhebung der Metriken in CANoe erfolgt anschließend die Auswertung der Nachrichten: Es werden der Typ der Nachricht (Quell- und/oder Zieladresse, z.B. CAN-ID 7), die UID und weitere Informationen wie die Paketgröße extrahiert und darauf basierend die Metriken berechnet und in Systemvariablen geschrieben (vgl. Abbildung 15). Deren Werte lassen sich wie gewünscht in CANoe anzeigen und auch in BLF-Dateien abspeichern, um eine mehrmalige Berechnung zu vermeiden. Anhand der Metrikerwerte wird das Netzwerk bewertet; diese Ergebnisse werden ebenfalls in Systemvariablen gespeichert.

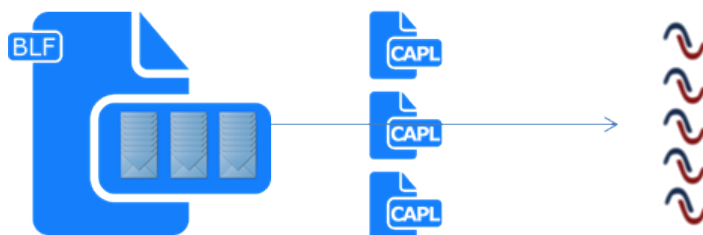


Abbildung 15: Schematische Abbildung der Verarbeitung der in einer BLF-Datei gespeicherten Nachrichten und Ablegen der Metrik-Werte in Systemvariablen.

## 4 Umsetzung und Herausforderungen

Im Folgenden wird beschrieben, wie die in Kapitel 3.4 gewählten Konzepte umgesetzt wurden. Dies gewährt einerseits einen Einblick in die verwendeten Komponenten und Techniken, andererseits stellt es abschnittsweise eine Anleitung für Anwender und Entwickler dar. Anschließend wird auf aufgetretene Probleme und – wenn möglich – auch auf deren Lösung eingegangen.

Die in Kapitel 3.4 beschriebenen Verfahren zur Bewertung von Netzwerken bestehen beide aus mehreren Komponenten, welche über eine Vielzahl von Schnittstellen miteinander kommunizieren. Der grundsätzliche Ablauf beider Konzepte ist identisch: OMNeT++ stellt Netzwerkbeschreibungs- und Simulations-Log-Dateien zur Verfügung, welche von einer externen Applikation (im Folgenden: Compagnon) gelesen werden. Diese wandelt die enthaltenen Daten um und erstellt Definitionen der Systemvariablen, CAPL-Code und ebenfalls Log-Dateien. Über die COM-Schnittstelle (s. Kapitel 2.5) konfiguriert Compagnon CANoe derart, dass die generierten Dateien verwendet und eine Offline-Analyse gestartet werden. Während der Analyse meldet CANoe über die COM-Schnittstelle Änderungen in den Systemvariablen, welche die Ergebnisse der Überprüfung der Grenzwerte enthalten. Diese Ergebnisse werden dann in Compagnon übersichtlich dargestellt.

### 4.1 Erstes Konzept: Erhebung der Metriken in OMNeT++

Zuerst wird auf die Umsetzung des ersten Konzeptes eingegangen, welches die Metriken in OMNeT++ erhebt und in CANoe bewertet.

#### 4.1.1 Implementierung des Vector-Managers

Wie bereits in Kapitel 3.3.1 erläutert wird ein `Vector-Manager` implementiert, welcher von OMNeT++ über die Existenz aller Statistik-Vektoren und während der Simulation über deren Änderungen informiert wird (OMNeT++-Projekt: `RecordedNetworks`).

Der Vector-Manager wird für ein Netzwerk gemäß Listing 2 in einer INI-Datei aktiviert und konfiguriert.

```
1 outputvectormanager-class = "RecordedNetworks::VectorBlfRecorder"
2 outputvectormanager-recorder-path = "results/vectors.blf"
3 outputvectormanager-recorder-recordStartValues = false
```

**Listing 2: Konfiguration des Vector-Managers „VectorBlfRecorder“ in OMNeT++**

Zu Beginn der Simulation werden dem `Vector-Manager` die Namen der Statistik-Vektoren und des Moduls, das den Vektor definiert, übergeben. Anhand dieses Namens wird das zugehörige Objekt gesucht, um überprüfen zu können, um welchen Typ es sich handelt. Hierdurch ist es nicht nötig, sich auf vorher festgelegte Namen zu verlassen. Anschließend wird entschieden, ob der Vektor aufgezeichnet werden soll. Bei einer positiven Entscheidung wird der Name der Systemvariable erstellt, unter dem die Werte in die BLF-Datei gespeichert. Alle Statistik-Vektoren, welche aufgezeichnet werden, können in Tabelle 3 eingesehen werden. Ihr Inhalt wird in Tabelle 4 aufgeschlüsselt.

Der Name der Systemvariablen setzt sich aus mehreren Komponenten zusammen, bspw. dem Pfad des Moduls in der Hierarchie des Netzwerks oder dem Start- und Endpunkt eines Links. Da bisher keine weiteren Anwendungsfälle zum Aufzeichnen von Statistik-Vektoren bekannt sind, die einen bestimmten Systemvariablen-Namen erfordern, ist keine Konfiguration der aufzuzeichnenden Module und Vektoren möglich: Eine vollständige Flexibilität des Namens ist zu komplex, um sie in einem Prototypen umzusetzen.

Modul-Typ	Vektor	Name der Systemvariablen (ohne beginnendes <code>statistics::</code> )
omnetpp::cChannel	utilization	Bus::\$fromNode \$toNode::Busload
	throughput	Bus::\$fromNode \$toNode::Throughput
inet::MACBase	droppedPk BitError	Buffer::\$node::\$macName::Dropped
CoRE4INET::Buffer	queueSize	Buffer::\$node::\$buffer::Size
	dropped	Buffer::\$node::\$buffer::Dropped
FiCo4OMNeT::Buffer	queueSize	Buffer::\$node::\$buffer::Size
CoRE4INET:: TrafficSinkApp	rxLatency	Type::\$node::\$app::Latency
	rxJitter	Type::\$node::\$app::Jitter
FiCo4OMNeT:: ISinkApplication	rxDFPayload Latency	Type::\$node::\$app::Latency
	rxRFLatency	Type::\$node::\$app::Latency
	rxDFPayload Jitter	Type::\$node::\$app::Jitter
	rxRFJitter	Type::\$node::\$app::Jitter

**Tabelle 3: Übersicht über die aufgezeichneten Statistik-Vektoren und deren Name in der BLF-Datei. Ausdrücke, die mit einem Dollar (\$) beginnen, sind Platzhalter für den nachfolgenden Ausdruck.**

Name	Wertebereich	Inhalt
Bus:: ... ::Busload	[0; 1]	0: Unbenutzt, 1: Voll ausgelastet
Bus:: ... ::Throughput	[0; 1]	0: Keine Nutzdaten, 1: Kein Overhead
Buffer:: ... ::Size	$N^0$	Anzahl der Nachrichten im Buffer
Buffer:: ... ::Dropped	$N^0$	Anzahl der verworfenen Nachrichten
Type:: ... ::Latency	$R^+$	Latenz in Sekunden
Type:: ... ::Jitter	$R^+$	Jitter in Sekunden

**Tabelle 4: Inhalte der Statistik-Vektoren und somit der Systemvariablen**

Es besteht die Möglichkeit, den Recorder anzuweisen, dass die Start-Werte der Metriken nicht aufgezeichnet werden (s. Listing 2, Z. 3). Da die Vektoren zu Beginn der Simulation alle den Wert 0 aufweisen, ist diese Information redundant und für eine Bewertung des Netzwerkes unnötig. Ihre Aufzeichnung zu deaktivieren spart deshalb einerseits etwas Speicherplatz und beschleunigt die Verarbeitung in CANoe.

Manche der aufgezeichneten Statistik-Vektoren existieren in der Standard-Konfiguration nicht und müssen vom Nutzer/von der Nutzerin hinzugefügt werden. Hierzu gehören die Statistiken des Kanals (s. Listing 3),

```

1  channel Eth100M_S extends Eth100M {
2      @statistic[utilization] (
3          source="floatingIntervalAvgSampleAndHold(channelBusy)";
4          record="vector");
5      @statistic[datathroughput] (
6          source="floatingIntervalAvg(dataThroughput(messageSent))";
7          record="vector");
8  }

```

**Listing 3: Hinzufügen zweier Statistik-Vektoren zu dem Standard Datarate-Channel Eth100M durch Deklaration eines neuen Channels**

#### 4.1.2 Nutzung des BlfRecorders zur Aufzeichnung von Nachrichten-Ereignissen

Während der Simulation kommt zusätzlich der `BlfRecorder` von Mulici (48) zum Einsatz. Dies ist ein Modul, welches sich für die Event-Signale „empfangenes Paket“ und „gesendetes Paket“ anderer Module (z.B. der Netzwerkports) registriert, die jeweiligen Pakete entgegen nimmt und in eine BLF-Datei schreibt. Für letzteres ist es nötig, die Nachrichten-Objekte aus OMNeT++ in einen Bitstrom umzuwandeln (zu serialisieren). Diese Aufgabe übernehmen die dem `BlfRecorder` beigelegten Serializer (48 S. 38).

Da der `BlfRecorder` in seiner aktuellen Implementierung in der BLF-Datei nur einen BLF-Kanal (s. Kapitel 2.6) beschreiben kann, muss er in den verwendeten Modulen innerhalb der Netzwerkports angebracht werden, um nur einen Netzwerkport abzudecken. Andernfalls würden die Nachrichten mehrerer Ports in einem Kanal vermischt. Die von der CoRE-Gruppe entwickelten Host-, Switch- und Gateway-Module wurden entsprechend angepasst (OMNeT++-Projekt: `RecordedNetworks`). Diese Module müssen in einem vorhandenen

Netzwerk eingesetzt werden, um eine Aufzeichnung zu ermöglichen. Der `BlfRecorder` kann gemäß Listing 4 konfiguriert werden. Ausführlichere Informationen hierzu finden sich in (48 S. 48).

```
1 **.node1.blfRecorder.recording = true
2 **.node1.blfRecorder.blfFile = "results/n1.blf"
3
4 **.switch1.phy[0].blfRecorder.recording = true
5 **.switch1.phy[0].blfRecorder.blfFile = "results/sw1.0.blf"
6
7 **.gateway2.tte[0].blfRecorder.recording = true
8 **.gateway2.tte[0].blfRecorder.blfFile = "results/gw2.eth.blf"
9 **.gateway2.can[0].canNodePort.blfRecorder.recording = true
10 **.gateway2.can[0].canNodePort.blfRecorder.blfFile = "results/gw2.can.blf"
```

Listing 4: Konfiguration des `BlfRecorder`s in OMNeT++

### 4.1.3 C#-Applikation „Compagnon“

Um eine Brücke zwischen den Daten von OMNeT++ und der Verarbeitung von CANoe zu schlagen, wurde eine C#-Applikation mit dem Namen „Compagnon“ erstellt. Ihre Aufgabe ist es, CANoe vorzubereiten und die Ergebnisse der Bewertung anzuzeigen. Da Compagnon beide Konzepte unterstützt, wird hier auf den Bereich eingegangen, welche nach dem Start mit einem Klick auf „Ja“ erreicht wird (s. Abbildung 16). Die meisten der folgenden Erläuterungen gelten jedoch für beide Bereiche.

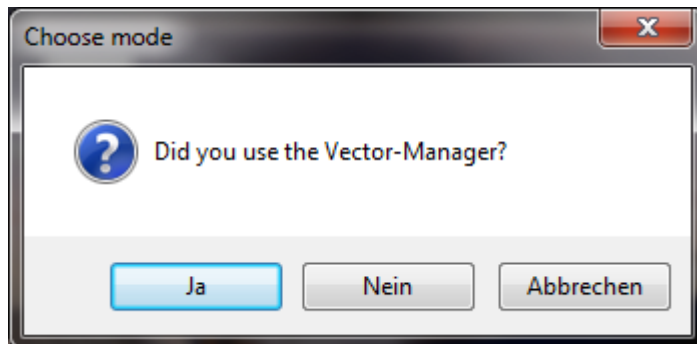


Abbildung 16: Abfrage des verwendeten Konzeptes zum Start von Compagnon

Nach dem Start von Compagnon muss der Nutzer/die Nutzerin die Eingangs-Dateien hinzufügen (s. Abbildung 17). Dies sind die NED-Datei, in welcher das Netzwerk definiert ist, die BLF-Dateien sowie eine XML-Datei, welche später in diesem Kapitel erläutert wird. Alle Dateien können einzeln oder gemeinsam per Drag&Drop in Compagnon gezogen werden. Die Informationen aus diesen Dateien werden anschließend zusammengeführt und an CANoe weitergegeben (s. Abbildung 18).

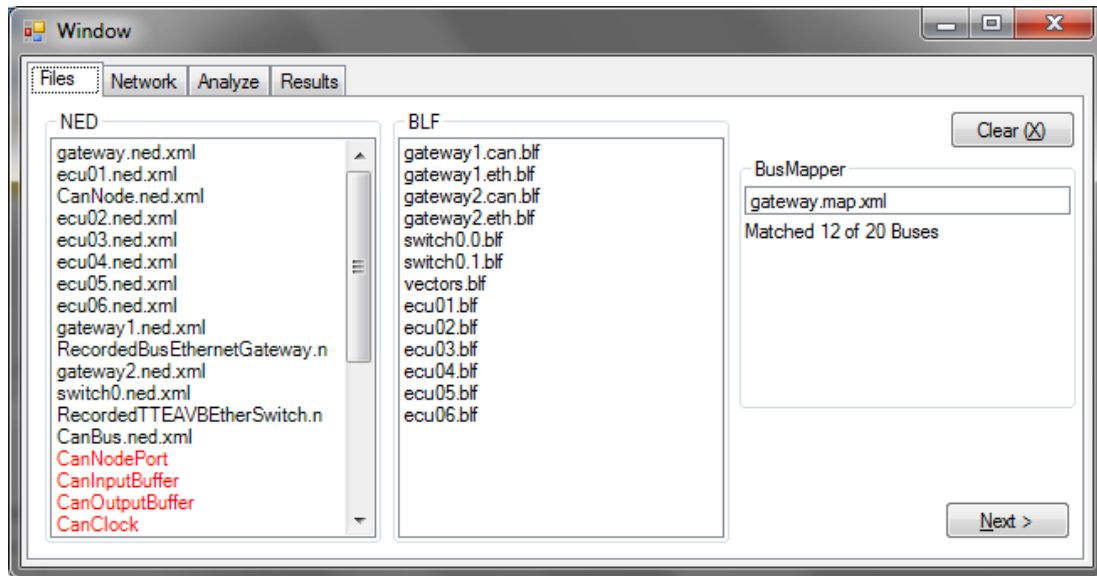


Abbildung 17: Oberfläche von Compagnon beim Hinzufügen von Dateien

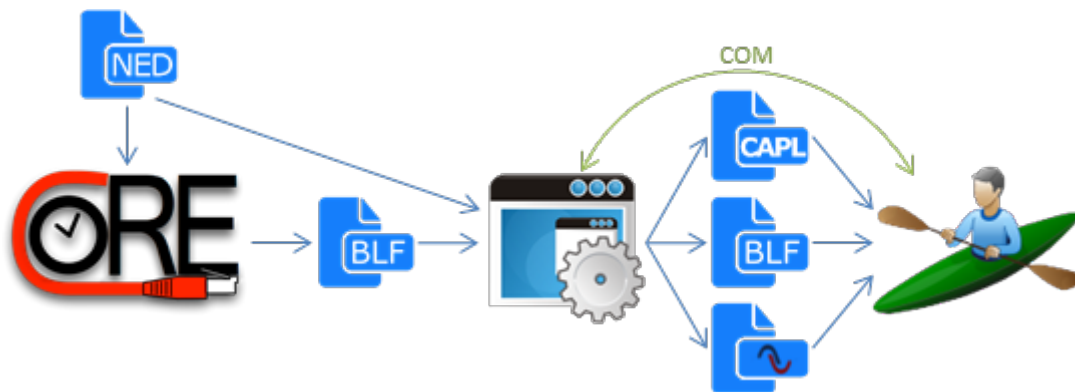


Abbildung 18: Darstellung der Schnittstellen zwischen OMNeT++, Compagnon und CANoe mit Augenmerk auf die beiden letzteren.

Die NED-Sprache von OMNeT++ (s. Kapitel 2.2) verfügt über sehr viele Features und ist ziemlich komplex. Da keine NED-Parser-Bibliothek für .NET existiert, wurde ein Parser in C# erstellt. Um die Komplexität beim Parsen zu reduzieren, unterstützt dieser nur das NEDXML-Format, welches aus OMNeT++ exportiert werden kann. In NEDXML-Dateien werden alle NED-Anweisungen in XML verpackt. Diese Vorgehensweise reduziert die Möglichkeit schwerwiegender Fehler beim Einlesen der Datei.

Der NEDXML-Parser liest die Dateien mit Hilfe des C#-XML-Parsers und verarbeitet die für die Anwendung relevanten Teile: Um das Netzwerk zeichnen zu können, ist es nötig, die Icons und die Position der enthaltenen Module sowie deren Verbindungen untereinander zu kennen. Da die Icons in den meisten Fällen von einer Basis-Klasse definiert werden, ist es unter anderem nötig, die Vererbung von NED zu implementieren. Die Datei mit der

Definition der Basis-Klasse wird dabei anhand des Namens der Klasse und des Dateinamens im Ordner sowie in allen Unterordnern der Netzwerk-NED-Datei gesucht.

Mithilfe der Netzwerkdefinition in den NEDXML-Dateien kann nun dem Nutzer/der Nutzerin das Netzwerk so präsentiert werden, wie es auch in OMNeT++ dargestellt wird (s. Abbildung 19). Es ist nun seine Aufgabe, die Netzwerkports auszuwählen, deren Nachrichten in CANoe im Trace-Fenster angezeigt werden sollen. Um die logische Verbindung zwischen diesen Ports und den BLF-Kanälen (s. Kapitel 2.6), in welchen die Nachrichten des Ports gespeichert wurden, herzustellen, benötigt Compagnon jedoch noch weitere Informationen. Hierfür wurde für diese Arbeit das BusMapper-XML-Format definiert (s. Listing 5).

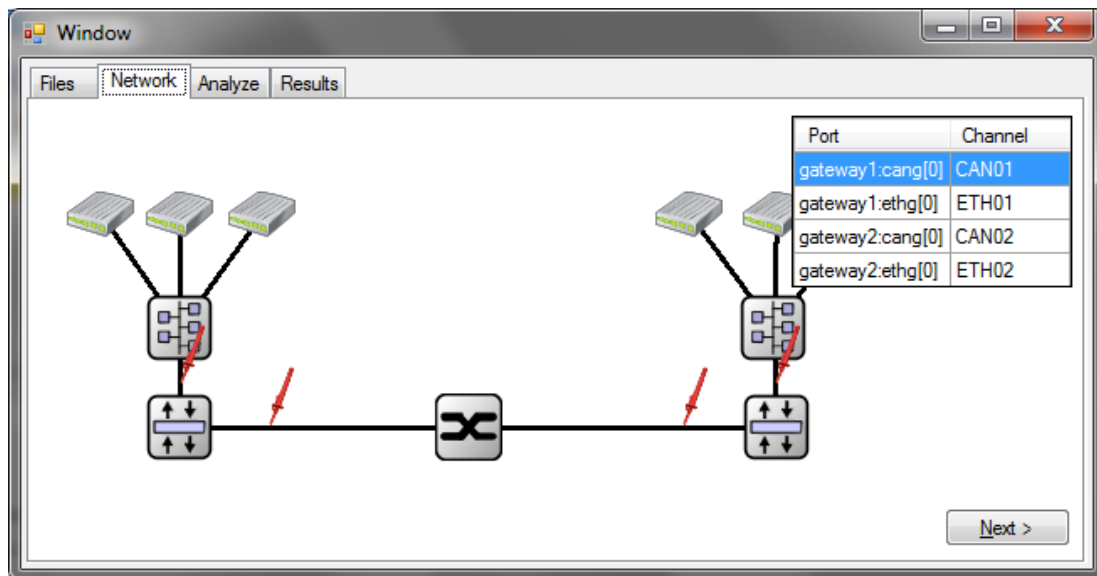


Abbildung 19: Oberfläche von Compagnon bei der Auswahl der anzuzeigenden Links. Die Tabelle oben rechts gibt an, welcher Channel in CANoe welchen Port im Netzwerk anzeigt.

```

1 <?xml version="1.0" encoding="utf-8" standalone="yes"?>
2 <busMapper version="1">
3   <file name="gw.blf">
4     <bus type="ETH">
5       <channel nr="1" device="gw_FCAN" port="ethg[0]" />
6       <channel nr="3" device="gw_ECAN" port="ethg[2]" />
7     </bus>
8   </file>
9 </busMapper>

```

Listing 5: Beispielhafter Inhalt einer BusMapper-XML-Datei über die Verknüpfung zwischen Netzwerkport und BLF-Kanal: Der simulierte Netzwerkverkehr der Ports gw\_ECAN.ethg[0] und gw\_ECAN.ethg[2] wurde in der Datei gw.blf in den Kanälen ETH01 und ETH03 gespeichert.

Eine BusMapper-XML-Datei ist – wie auch NEDXML – ein XML-Dokument, das jedoch bestimmte Knotennamen enthält. Es ordnet BLF-Dateien, welche anhand ihres Namens identifiziert werden, und den darin enthaltenen BLF-Kanälen einen Netzwerkport zu.



Um dem Nutzer/der Nutzerin bei der Erstellung dieser Dateien eine gewisse Unterstützung zu bieten, wurde eine kleine Hilfs-Applikation namens „INI2BusMapper“ entwickelt, welche die INI-Dateien des Netzwerks nach `BlfRecorder`-Definitionen (s. Listing 4) durchsucht und daraus die BusMapper-XML-Datei generiert. Das Ergebnis muss jedoch nachgearbeitet werden, da einige Informationen aus den INI-Dateien nicht ersichtlich sind: So werden bspw. in den NED-Dateien die internen TTE- und TSN-Ports auf externe ETH-Ports gemappt. Aus diesem Grund wurde das Lesen der INI-Dateien auch nicht direkt in Compagnon integriert.

Mit den Informationen aus der BusMapper-XML-Datei und den durch den Nutzer/die Nutzerin gewählten Verbindungen können nun die BLF-Dateien zusammengeführt werden: Die Nachrichten werden (je nach Kanal) aus den BLF-Dateien gelesen und in einer großen BLF-Datei gespeichert. Dies ermöglicht es einerseits dem Nutzer/der Nutzerin, während der Simulation nicht auf die eingestellte Kanal-Nummer des `BlfRecorders` achten zu müssen, adressiert andererseits aber auch die Einschränkung von CANoe, nur die Kanäle 1 bis 32 aus BLF-Dateien zu lesen (s. Kapitel 4.1.4).

Compagnon durchsucht außerdem die hinzugefügten BLF-Dateien nach derjenigen, welche von dem `Vector-Manager` aufgezeichnet wurde. Um diesen Prozess zu beschleunigen, schreibt der `Vector-Manager` einen festgelegten Identifier an den Beginn der Datei. Somit muss Compagnon nur den Anfang der BLF-Dateien überprüfen, um diese bestimmte Datei zu finden.

Anschließend wird die gesamte BLF-Datei mit den Systemvariablen eingelesen. Anhand deren Namen wird ein Baum aufgebaut, welcher die Namespaces repräsentiert (s. Kapitel 2.5). Dieser Baum wird um Variablen für die Grenzwerte erweitert und in eine Systemvariablen-Definitionsdatei umgewandelt. Dieses XML-Format ist von CANoe vorgegeben. Ebenso werden die Namen der Variablen verwendet, um eine CAPL-Datei zu generieren, in welcher auf die Änderungen der Systemvariablen reagiert und mit den Grenzwerten überprüft wird (s. Listing 6).

```
1 on sysvar Statistics::Bus::ecu01_canbus1::Busload {
2   double req;
3   req = @Requirements::Bus::ecu01_canbus1::Busload::error;
4   if (req > 0 && @this > req) {
5     @Requirements::Bus::ecu01_canbus1::Busload::Active = 3; return;
6   }
7   req = @Requirements::Bus::ecu01_canbus1::Busload::warn;
8   if (req > 0 && @this > req) {
9     @Requirements::Bus::ecu01_canbus1::Busload::Active = 2; return;
10  }
11  req = @Requirements::Bus::ecu01_canbus1::Busload::info;
12  if (req > 0 && @this > req) {
13    @Requirements::Bus::ecu01_canbus1::Busload::Active = 1; return;
14  }
15  }
16  @Requirements::Bus::ecu01_canbus1::Busload::Active = 0;
17 }
```

Listing 6: CAPL-Funktion, welche die Änderung einer Systemvariablen anhand der Grenzwerte überprüft

Darüber hinaus baut Compagnon eine COM-Verbindung zu CANoe auf (vgl. Kapitel 2.5) und konfiguriert CANoe für eine Offline-Messung. Ebenfalls über die COM-Schnittstelle werden die zusammengeführte BLF-Datei und die Systemvariablen-Definitionen in CANoe hinzugefügt.

Der Nutzer/die Nutzerin hat nun die Möglichkeit, in Compagnon Grenzwerte für die Grenzwerte einzugeben (s. Abbildung 20). Diese Werte werden zu Beginn der Messung über die COM-Schnittstelle an CANoe übertragen.

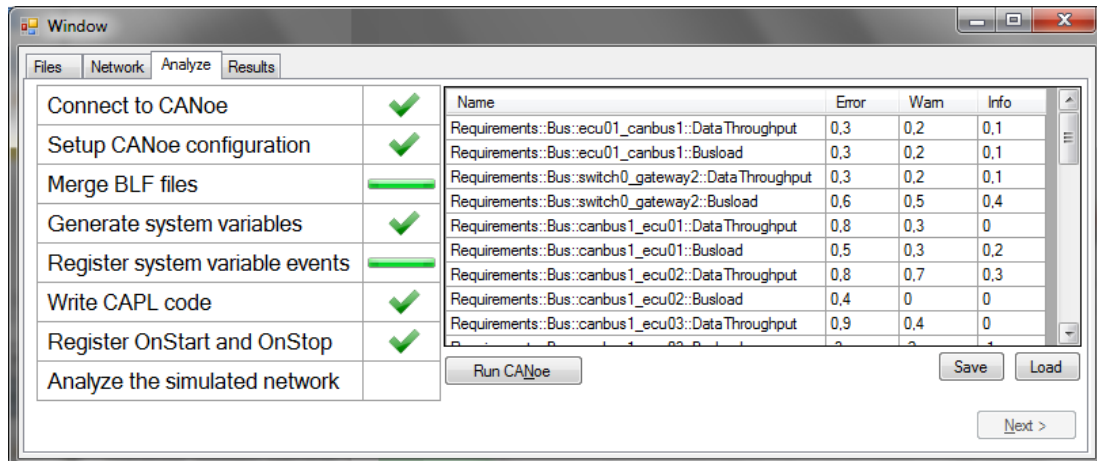


Abbildung 20: Oberfläche von Compagnon bei der Erstellung der nötigen Dateien und der Eingabe von Grenzwerten für die Metriken

#### 4.1.4 BinlogNET

Wie bereits in Kapitel 3.4 beschrieben, wurde das Binlog-Format (BLF, s. Kapitel 2.6) als Datenschnittstelle zu CANoe ausgewählt, da es eine Vielzahl an Möglichkeiten bietet. Sowohl der `BlfRecorder` als auch der implementierte `Vector-Manager` verwendet die von der Vector Informatik GmbH zur Verfügung gestellte `binlog.dll`, um die BLF-Dateien zu erstellen (48 S. 31). Da BLF ein proprietäres Format ist, stellt diese Win32-DLL die einzige Möglichkeit dar, auf BLF-Dateien zuzugreifen.

Aus zweierlei Gründen ist es nötig, die von OMNeT++ generierten BLF-Dateien in Compagnon zu bearbeiten: In der aktuellen Implementierung kann eine BLF-Datei jeweils nur von einem `BlfRecorder`-Modul beschrieben werden. Diese Module koordinieren sich nicht; stattdessen legt der Nutzer/die Nutzerin fest, welcher BLF-Kanal verwendet wird. Es ist somit nicht ausgeschlossen, dass zwei Instanzen (in zwei verschiedenen Dateien) dieselbe Kanalnummer verwenden. Da es CANoe nicht ermöglicht, diese beiden Kanäle beim Abspielen auf unterschiedliche Kanalnummern aufzuteilen, werden die Nachrichten vermischt (s. Abbildung 21). Um dem Nutzer/der Nutzerin die Konfiguration zu erleichtern, sorgt Compagnon nachträglich dafür, dass jeder Port einen eindeutigen BLF-Kanal erhält (s. Abbildung 22).

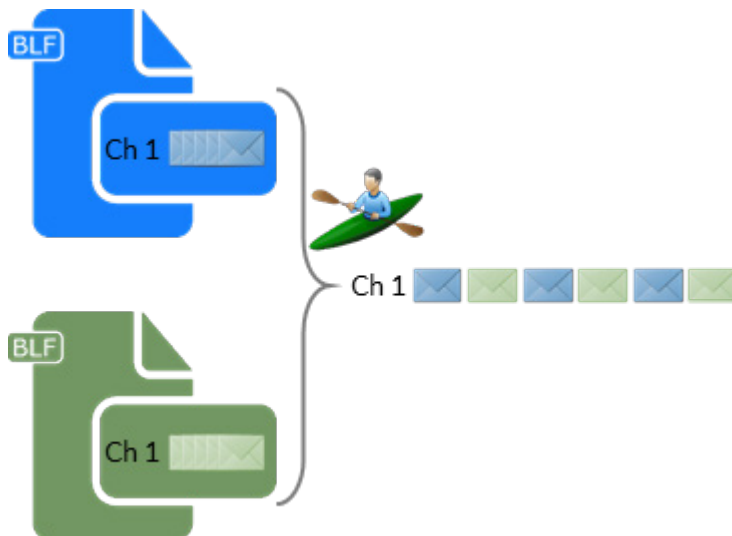


Abbildung 21: Schematische Darstellung des Verhaltens von CANoe beim Einlesen von BLF-Dateien: Enthalten zwei BLF-Dateien Nachrichten mit derselben Kanal-Nummer („Ch 1“), so werden diese Nachrichten in CANoe in einen Kanal gemischt und können nicht mehr unterschieden werden. Die tatsächliche Reihenfolge hängt von den Zeitstempeln der Nachrichten ab.

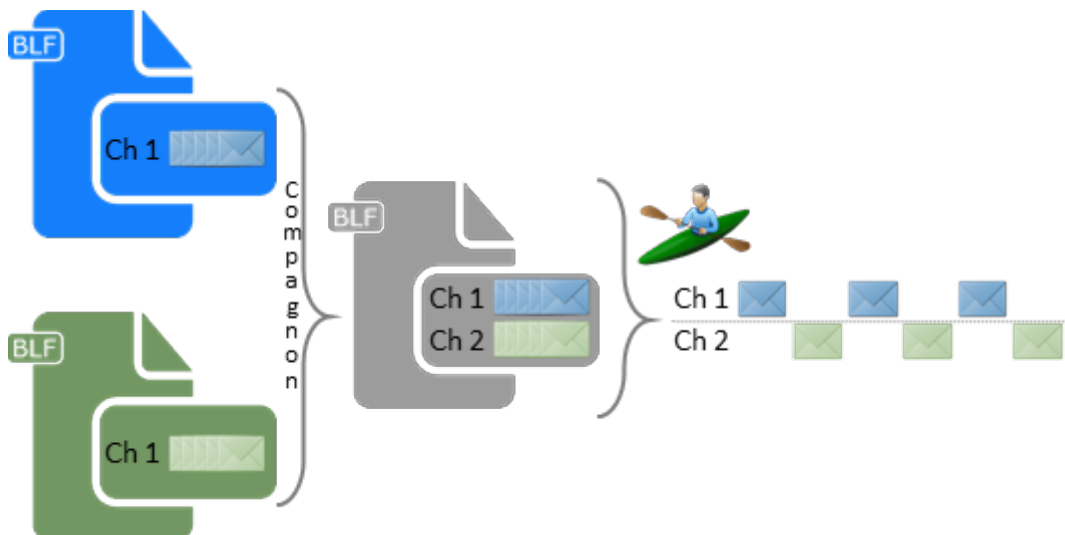


Abbildung 22: Schematische Darstellung der Aufgabe von Compagnon bezüglich der BLF-Dateien: Die Kanäle der verschiedenen Dateien werden auf mehrere, unterschiedliche Kanäle in einer neuen BLF-Datei verteilt. Dies ermöglicht das korrekte Abspielen in CANoe.

Zusätzlich enthält CANoe die Einschränkung, dass nur 32 Busse pro Bussystem verwendet werden können (s. Kapitel 2.5). Dies wird in der aktuellen Version von CANoe derart auf BLF abgebildet, dass nur die Kanäle 1 bis 32 eingelesen werden; alle Nachrichten mit anderen Kanalnummern werden verworfen. Wie in Kapitel 3.4 erwähnt, liegen bereits jetzt Netzwerke mit über 50 CAN-Links vor. Um alle ausgewählten CAN-Links in CANoe

analysieren zu können, muss Compagnon dafür sorgen, dass sich in der neuen Datei (vgl. Abbildung 22) die zu analysierenden Kanäle unter den ersten 32 BLF-Kanälen befinden. Außerdem muss dem Nutzer/der Nutzerin angezeigt werden, dass er nicht mehr als 32 Netzwerkports desselben Bussystems auswählen kann.

Da die `binlog.dll` eine Win32-DLL ist, ist es naheliegend, ein C++-Programm für diese Teilaufgabe zu entwickeln. Weil die BLF-Dateien einer Simulation unter Umständen jedoch sehr viele Nachrichten enthalten können und ihre Verarbeitung somit einige Minuten dauern kann, soll dem Nutzer/der Nutzerin währenddessen der Fortschritt angezeigt werden. Hierfür ist es nötig, dass selbiger vom C++-Programm an Compagnon gemeldet wird. Eine solche Interprozesskommunikation (z.B. über COM oder die Windows-API) wurde als zu aufwändig eingeschätzt.

Stattdessen wurde eine C#-Library (genannt „BinlogNET“) erstellt, welche die `binlog.dll` einbindet. Sie stellt deren Methoden in C# zur Verfügung, nimmt die Umwandlung zwischen verwaltetem in nicht verwaltetem Speicher vor und stellt Klassen zum einfachen Umgang mit BLF-Dateien und -Nachrichten zur Verfügung.

Hiermit war es z.B. ein leichtes, eine grafische Oberfläche zu bauen, welche detaillierte Informationen über die in einer BLF-Datei enthaltenen Objekte anzeigt. Dies ist bei der Überprüfung der von OMNeT++ generierten Dateien sehr nützlich (s. Abbildung 23).

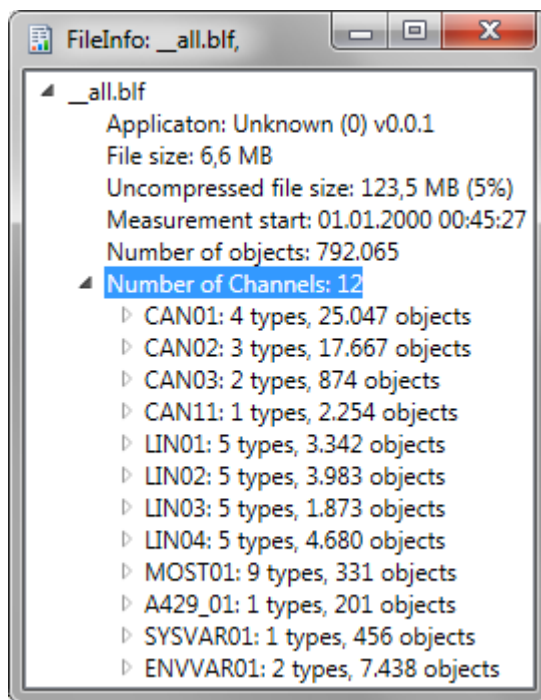


Abbildung 23: Oberfläche der Applikation „FileInfo“, welche Informationen aus BLF-Dateien extrahiert.

Darüber hinaus stellt BinlogNET auch eine Klasse zur Verfügung, welche die für Compagnon bedeutende Aufgabe erfüllt: Nachrichten aus mehreren Eingangsdateien – separiert nach Kanal – in eine neue BLF-Datei zu schreiben. Diese Operation wird asynchron, aber innerhalb von Compagnon durchgeführt, sodass mit Hilfe von C#-Events dem Nutzer/der Nutzerin sehr einfach ein Fortschritt angezeigt werden kann.

## 4.2 Zweites Konzept: Erhebung der Metriken in CANoe

Im Folgenden wird auf die Umsetzung des zweiten Konzeptes eingegangen, welches die Metriken in CANoe erhebt und bewertet.

Der in Kapitel 4.1.1 beschriebene `Vector-Manager` wird hierbei nicht verwendet, der `BlfRecorder` aus Kapitel 4.1.2 kann jedoch ohne Anpassungen übernommen werden. Das in Kapitel 4.1.3 eingeführte Programm Compagnon wird für dieses Konzept erweitert, sodass die Bewertung in CANoe vorgenommen werden kann.

### 4.2.1 OMNeT++

Um eine Analyse in CANoe vornehmen zu können wird – wie bereits in Kapitel 3.3 beschrieben – eine eindeutige Sequenznummer benötigt: Werden mehrere solche kurz hintereinander abgeschickt (ein sogenannter Burst), ist ohne diese Nummer nicht zu erkennen, welche der Nachrichten wann weitergeleitet wurde. Da der Nachrichtentyp bspw. durch die MAC-Adresse des Absenders definiert werden kann, kann nicht garantiert werden, dass die Reihenfolge der Nachrichten beibehalten wird, wenn z.B. eine Priorisierung durch VLAN-IDs vorgenommen wird.

Auf die Sequenznummer soll während der Analyse in CANoe zugegriffen werden können. Da im BLF-Format bis auf das Nutzdaten-Feld alle Felder, welche in CANoe von CAPL-Programmen gelesen werden können, relevant sind, muss die Sequenznummer in den Nutzdaten verpackt werden.

Basierend auf dem modularen Konzept von OMNeT++ wurde entschieden, eine neue Nachrichtenart zu definieren, welche zwar keine eigenen Daten enthält, jedoch über einen Serializer verfügt, welcher die OMNeT++-interne Sequenznummer in das Nutzdaten-Feld schreibt (OMNeT++-Projekt: `SignalsAndGateways`). Diese ID ist vier Byte groß und wird von OMNeT++ bei jeder neuen Nachrichten-Instanz inkrementiert. Packt man diese Nachricht in die bereits vorhandenen Nachrichten (z.B. Ethernet- oder CAN-Nachricht) als Payload ein, rufen die Serializer der Nachrichten jenen der Sequenznummern-Nachricht auf. Der `BlfRecorder` von (48) musste somit nicht verändert werden.

```
#77 0.002222 canbus1 --> ecu03    message SeqNoPacket:8 bytes  CanDataFrame:14 bytes
#77 0.002222 canbus1 --> gateway1 message SeqNoPacket:8 bytes  CanDataFrame:14 bytes
#83 0.002333 ecu01 --> canbus1    message SeqNoPacket:4 bytes  CanDataFrame:10 bytes
#94 0.002491 canbus1 --> ecu01    message SeqNoPacket:4 bytes  CanDataFrame:10 bytes
#94 0.002491 canbus1 --> ecu02    message SeqNoPacket:4 bytes  CanDataFrame:10 bytes
#94 0.002491 canbus1 --> ecu03    message SeqNoPacket:4 bytes  CanDataFrame:10 bytes
#94 0.002491 canbus1 --> gateway1 message SeqNoPacket:4 bytes  CanDataFrame:10 bytes
#102 0.00257 ecu01 --> canbus1    message SeqNoPacket:2 bytes  CanDataFrame:8 bytes
#113 0.002696 canbus1 --> ecu01    message SeqNoPacket:2 bytes  CanDataFrame:8 bytes
#113 0.002696 canbus1 --> ecu02    message SeqNoPacket:2 bytes  CanDataFrame:8 bytes
#113 0.002696 canbus1 --> ecu03    message SeqNoPacket:2 bytes  CanDataFrame:8 bytes
```

**Abbildung 24:** Nachrichten-Events in OMNeT++, welche die Übertragung eines SeqNoPackets in einem CanDataFrame zeigen.

Da in OMNeT++ die Payload einer Nachricht von jener Applikation festgelegt wird, welche die Nachricht versendet, müssen alle sendenden Applikationen angepasst werden. Um die Anpassung nicht an eine einzige Nachricht zu binden und eine gewisse Flexibilität zu gewährleisten, wurde ein neuer NED-Parameter eingeführt, welcher es ermöglicht, die zu versendende Payload in der INI-Datei zu spezifizieren (s. Listing 7). Somit kann der Nutzer/die Nutzerin wählen, welche Nachrichten versendet werden sollen. Mittels Wildcards können auch alle Applikationen gleichzeitig konfiguriert werden (s. Listing 7, Z. 5).

```
1 **.node1.app[0].payloadContentType = "SignalsAndGateways::SeqNoPacket"
2 **.node1.app[1].payloadContentType = "SignalsAndGateways::SeqNoPacket"
3 **.node2.app[0].payloadContentType = "SignalsAndGateways::SeqNoPacket"
4
5 **.app[*].payloadContentType = "SignalsAndGateways::SeqNoPacket"
```

**Listing 7:** Anweisung an mehrere Applikationen zur Verwendung eines SeqNoPacket als Payload

Soll die Nachricht weniger als vier Byte Payload transportieren (vgl. Abbildung 24), so muss die Nachrichten-ID verkürzt werden. Da OMNeT++ innerhalb der Knoten ebenfalls Nachrichten verwendet, um Informationen auszutauschen (bspw. zwischen einem Puffer und einer Applikation), beträgt der Abstand zwischen zwei aufgezeichneten Nachrichten-IDs normalerweise mehr als 1. Soll eine nur ein Byte große Payload verwendet werden und wird die ID hierfür lediglich gekürzt, so tritt ein systematischer Fehler auf, wenn genau  $x \cdot 256 - 1$ ,  $x \in \mathbb{N}$  Nachrichten zwischen den beiden aufgezeichneten Nachrichten erstellt wurden. Es besteht somit die Wahrscheinlichkeit, dass sich dieser Fehler vermehrt für bestimmte Nachrichtentypen durch die gesamte Simulation zieht und die Analyse dieser Typen stark beeinträchtigt.

Um in diesem Fall einen systematischen Fehler zu vermeiden, wird eine Hash-Funktion von Wang eingesetzt, welche die enthaltene Information gleichmäßig über alle Bits verteilt (49). Da bei der Reduktion auf weniger als vier Bytes jedoch Information verworfen werden muss, bleibt die relative Anzahl der Fehler genauso groß wie wenn keine Hash-Funktion verwendet wird. Diese Fehler sind jedoch pseudozufällig über die gesamte Simulation verteilt und die nachfolgende Analyse kann für jeden Nachrichtentyp viele (wenn auch nicht unbedingt alle) Metrik-Werte berechnen.

## 4.2.2 C#-Applikation Compagnon

Glücklicherweise können viele der bereits vorhandenen Programmteile von Compagnon weiter benutzt werden: Die Kommunikation über die COM-Schnittstelle ist ebenso identisch wie die Anzeige der Netzwerke und das Zusammenführen der BLF-Dateien.

Die Erhebung der Metriken in CANoe benötigt allerdings noch mehr Informationen über das Netzwerk, als wie in Kapitel 4.1.3 behandelt generiert werden. So muss bspw. die Datenübertragungsrate der Links aus den NEDXML-Dateien gespeichert werden, da nur mit dieser Angabe die Linkauslastung berechnet werden kann.

Ebenso ist es nötig, das Netzwerk hinsichtlich der Nachbarschaftsbeziehungen der Netzwerkteilnehmer zu analysieren, um Broad- und Multicast-Nachrichten zu unterstützen. Nähere Informationen hierzu finden sich in Kapitel 4.3.1.

Diese Ergebnisse sowie die Datenübertragungsraten werden in separate CAPL-Dateien geschrieben, damit CANoe mit diesen arbeiten kann (vgl. Abbildung 14).

Um nicht nur eine Analyse sondern auch eine Bewertung des Netzwerkes durchzuführen, müssen Grenzwerte für die verschiedenen Metriken definiert werden. Dies geschieht bei diesem Konzept über eine Requirements-XML-Datei. Dies ist unter anderem deshalb nötig, da vor der Analyse nicht bekannt ist, welche Nachrichtentypen analysiert werden sollen. In dieser Datei können die Grenzwerte für die bisher unterstützten Metriken Linkauslastung, Latenzzeit und Jitter angegeben werden (s. Listing 8). Die Werte der Requirements werden bei der Generierung der Systemvariablen-Definitionsdatei als Initialwerte eingetragen und müssen somit nicht bei jedem Messungsstart übertragen werden.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <requirements version="1">
3   <thresholds>
4     <busloadDefault type="ETH" error="60%" warn="40%" info="30%" />
5     <busload from="canbus1" to="gateway1" error="90%" info="70%" />
6     <latency id="1" error="23ms" warn="80%" info="65%" />
7     <jitter src="EE:AA:AA:AA:AA:00" dst="EE:AA:AA:AA:AA:01" error="1ms" />
8   </thresholds>
9 </requirements>
```

**Listing 8: Beispielhafter Inhalt einer Requirements-XML-Datei mit Angaben über die maximale Linkauslastung, Latenz und Jitter.**

Um nicht für jeden Link einen einzelnen Eintrag anlegen zu müssen, können auch Werte pro Bussystem angegeben werden (s. Listing 8, Z. 4). Ein einzelner Link wird anhand der Namen der beiden Module selektiert, welche er miteinander verbindet. Die Benutzermetriken (s. Kapitel 2.4) werden – wie in Kapitel 3.3.3 beschrieben – anhand ihrer Quell- oder Zieladresse oder aus einer Kombination der beiden ausgewählt. Dadurch ist es möglich, CAN- und TSN-Nachrichten mit ihrer Ziel-Adresse bzw. Stream-ID genauso zu erfassen wie Ethernet-Multicast-Nachrichten von verschiedenen Absendern und ein Paketstrom von einem Knoten zu einem anderen.

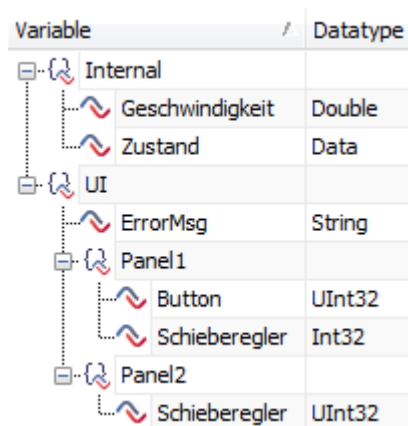
Den angegebenen Adressen werden intern acht Byte lange Identifier zugewiesen, welche in CANoe weiterverarbeitet werden können. Hierfür wird wie bereits zuvor CAPL-Code generiert.

Dies bedeutet, dass die Analyse nur für Nachrichtentypen durchgeführt wird, für welche ein Grenzwert definiert wurde, was in doppelter Hinsicht sinnvoll ist: Zum einen würden andernfalls für jede Nachricht die Metriken mehrmals berechnet werden müssen und den Rechenaufwand somit deutlich erhöhen, zum anderen liegt der Fokus dieser Arbeit auf der Bewertung eines Netzwerks, für welche Metriken, die keinen Grenzwerten unterliegen, unerheblich sind.

### 4.2.3 Gliederung der Systemvariablen

Die Systemvariablen nehmen in diesem Konzept eine noch wichtigere Position ein: Sie sind zum einen die Datenbank für Werte der Metriken, zum anderen sind sie für den Nutzer/die Nutzerin sichtbar und müssen für ihn leicht verständlich sein. Deshalb wird im Folgenden kurz darauf eingegangen, wie sie mithilfe von Namespaces (s. Kapitel 2.5) strukturiert werden.

Namespaces sind ein sehr praktisches Feature der Systemvariablen in CANoe, welche es ermöglichen, letztere in einer Baumstruktur anzuordnen. Hierdurch kann auch bei einer großen Anzahl an Systemvariablen eine gewisse Übersichtlichkeit gewahrt werden. Es lassen sich bspw. Variablen für Bedienelemente in CANoe und Variablen für CAPL-Berechnungen separieren (s. Abbildung 25).



Variable	Datatype
Internal	
Geschwindigkeit	Double
Zustand	Data
UI	
ErrorMsg	String
Panel1	
Button	UInt32
Schieberegler	Int32
Panel2	
Schieberegler	UInt32

Abbildung 25: Beispiel für die Verwendung von Namespaces bei Systemvariablen.

Wie auch in anderen Bereichen der Technik ist es wichtig, den Systemvariablen semantische Namen zu geben. Das bedeutet, dass der Name der Variable den Inhalt beschreibt (50). In Abbildung 25 sind somit die Namen „Geschwindigkeit“ und „ErrorMsg“ Beispiele für gute Namen, da sie angeben, was in ihnen gespeichert wird. Ein schlechtes Beispiel sind die „Schieberegler“ und der „Button“, da sie spezifizieren, mit welchem UI-Element sie benutzt werden sollen. Es wird aber nicht deutlich, was ihr Wert bedeutet.



Für die Bewertung eines Netzwerkes muss eine Vielzahl an Systemvariablen eingeordnet werden: Zum einen bietet das CAPL-Programm verschiedene Konfigurationsmöglichkeiten. Hierzu gehören bspw. die Größe des Zeitfensters für die Berechnung der Linkauslastung (vgl. Kapitel 2.3) oder die Zykluszeit für den Timer, welcher alte Nachrichten aus dem internen Speicher entfernt (s. Kapitel 4.2.4). Darüber hinaus können während der Simulation in CANoe kritische Fehler auftreten, welche ebenfalls mittels Systemvariablen über die COM-Schnittstelle an Compagnon gemeldet werden sollen. Letztlich existieren noch die eigentlichen Latenz-, Jitter- und Busauslastungs-Werte, welche jeweils über den Link und/oder den Netzwerktypen abgefragt werden können sollen.

Die Systemvariablen werden somit zuerst in vier Namespaces unterteilt:

- **Config**  
Dieser Namespace enthält Variablen, welche eine Konfiguration des Analyse-Programms ermöglichen.
- **Signals**  
Die hier enthaltenen Systemvariablen dienen dazu, Compagnon über die COM-Schnittstelle über kritische Fehler während der Simulation zu informieren. Diese Fehler können dann dem Nutzer/der Nutzerin angezeigt und erläutert werden.
- **Requirements**  
Hier werden die Werte für die Grenzwerte sowie der Status der Überprüfung gespeichert, also ob einer der angegebenen Schwellwerte aktuell überschritten wurde.
- **Statistics**  
In diesem Bereich werden die Metrik-Werte gespeichert.

Sowohl im Namespace „Requirements“ als auch in „Statistics“ ist eine weitere Untergliederung der enthaltenen Systemvariablen nötig. Beide werden deshalb in die Namespaces „Type“ und „Bus“ untergliedert:

- Im Bus-Namespace werden die Metriken nach Links sortiert gespeichert. Dies sind einerseits die Datenübertragungsrate, der Datendurchsatz und die Anzahl der Nachrichten, welche auf diesem Link gesendet wurden, andererseits aber auch die Latenz und der Jitter zu anderen Links, welche jeweils ebenfalls einen eigenen Namespace aufweisen. In die beiden letzten Systemvariablen fließen die Metrik-Werte unabhängig vom Nachrichtentyp ein. Dies ermöglicht es, einen Netzwerkknoten z.B. als Quelle großer Verzögerungen zu identifizieren, ohne sich auf einen bestimmten Nachrichtentyp einschränken zu müssen.
- Im Type-Namespace sind alle Metriken nach dem Nachrichtentyp geordnet. Diese Namespaces enthalten einerseits einen Zähler, welcher die Anzahl der analysierten Nachrichten dieses Typs angibt, andererseits aber auch die Metriken Latenz und Jitter als Ende-zu-Ende-Wert.

Zusätzlich soll es gemäß Kapitel 3.1 möglich sein, die Metriken eines Nachrichtentyps auf den Link genau untersuchen zu können. Deshalb werden – wie im Bus-Namespace – weitere Namespaces für Links und deren Nachbarn eingefügt, welche ebenfalls die Metriken Latenz und Jitter enthalten.

Für die Weiterverarbeitung ist aber nicht immer der letzte Wert der Latenz oder des Jitters gewünscht. Teilweise genügt es auch, die minimalen und maximalen Werte als Zahl anzeigen zu lassen oder den Durchschnitt zu betrachten. Um diese Bedürfnisse ebenfalls zu befriedigen, wird statt einer Latenz- bzw. Jitter-Variable ein Namespace mit dem Namen „Latency“ bzw. „Jitter“ eingesetzt, welcher jeweils fünf Variablen enthält:

- „cur“: Aktueller Wert
- „min“: Minimaler Wert
- „max“: Maximaler Wert
- „avg“: Durchschnitt über die gesamte Simulationszeit
- „expAvg“: Exponentieller Durchschnitt über die gesamte Simulationszeit

Es wird deutlich, dass durch die große Anzahl an Analysemöglichkeiten sehr viele Variablen angelegt werden müssen. Durch die Einordnung in vier Bäume bleibt die Übersichtlichkeit dennoch bestehen (s. Abbildung 26). Der Statistics-Namespace ist mit Grad 6 (fünf Unter-Namespaces) der größte Baum. Deutlich wird dies an einem beispielhaften Selektionspfad, wie er u.a. auch im Grafikfenster angezeigt wird:

```
Statistics::Type::EEAAAAAAAAA00->EEAAAAAAAAA01::CAN02::to_ETH01::Latency::cur
```

**Listing 9: Beispielhafter Selektionspfad einer Systemvariablen, welche den aktuellen Wert der Verzögerung von CAN02 zu ETH01 jener Nachrichten enthält, welche vom Knoten mit der MAC-Adresse EE:AA:AA:AA:00 an den Knoten mit Adresse EE:AA:AA:AA:01 versandt werden.**

Variable	Unit	Datatype
Config		
Requirements		
Signals		
Statistics		
Bus		
Type		
dst_EEAAAAAAAA01		
EEAAAAAAAA00->EEAAAAAAAA01		
CAN01		
CAN02		
LastSeenHere	-	UInt64
msgCount	-	UInt64
to_CAN01		
to_ETH01		
Jitter		
Latency		
avg	ns	Double
cur	ns	UInt64
expAvg	ns	Double
max	ns	UInt64
min	ns	UInt64
msgCount	-	UInt64
to_ETH02		
ETH01		
ETH02		
Jitter		
Latency		
msgCount	-	UInt64
src_000000000001		
src_000000000002		
src_000000000003		
src_000000000004		
src_000000000005		
src_000000000006		
src_000000000007		

Abbildung 26: Strukturierung der Systemvariablen in Namespaces bei der Simulation eines kleinen Netzwerkes mit Gateways.

#### 4.2.4 CANoe

Die Erhebung der Metriken (s. Kapitel 2.3) erfolgt in CANoe. Hierfür wird im Messaufbau ein CAPL-Programm zwischengeschaltet, welches die Nachrichten-Events verarbeitet.

Compagnon kann über die COM-Schnittstelle einige Einstellungen in CANoe ändern; bspw. können die Systemvariablen-Definitionen (s. Kapitel 4.2.3) eingebunden und die Offline-Simulation aktiviert werden. Einige notwendige Schritte wie bspw. das Einfügen des CAPL-Programms können nicht über die COM-Schnittstelle geschehen. Somit muss zur Analyse die vorbereitete Analyse-Konfiguration geladen werden.

Im CAPL-Programm werden bei Ankunft einer Nachricht die Meta-Informationen extrahiert:

- Zeitstempel,
- Kanal,
- Größe in Bit,
- Nachrichtentyp und
- UID.

Zunächst wird die Linkauslastung durch Aufsummieren der Größe der Nachrichten und mithilfe eines Zeitfensters ermittelt: Ist ein solches Zeitfenster verstrichen, wird die Anzahl der insgesamt übertragenen Bits durch die Datenübertragungsrate und die Länge des Zeitfensters dividiert. Analog dazu wird der Wert für den Datendurchsatz gebildet.

Als Nachrichtentyp werden – je nach Bussystem – die möglichen Adressen (Quelladressen, Zieladressen sowie eine Kombination aus beiden) verwendet. Ist für den Nachrichtentyp ein Grenzwert vorhanden, so wird die Nachricht weiter analysiert.

Anhand des Nachrichtentyps und der Sequenznummer wird ein vorheriges Auftauchen dieser Nachricht gesucht. Mithilfe der beiden Zeitstempel kann die Latenzzeit berechnet werden; die Gesamtlatenz ergibt sich aus der Summe der Einzellatenzen. Durch Differenzbildung zweier Latenzwerte kann auch der aktuelle Jitter ermittelt werden (s. Kapitel 2.3).

Alle ermittelten Metriken werden in Systemvariablen gespeichert und mit ihren Grenzwerten verglichen. Wird ein Verstoß gegen einen der Grenzwerte festgestellt, so wird dies ebenfalls in der entsprechenden Systemvariable notiert.

Um den Speicherbedarf des Programms gering zu halten und somit die Geschwindigkeit zu optimieren, wurde eine Routine implementiert, welche die Informationen über veraltete Nachrichten löscht (quasi ein Garbage-Collector). Hierbei wird der Zeitstempel der letzten Nachricht einer Sequenznummer verwendet, also die letzte Wiederholung einer bestimmten Nachricht. Das Intervall, ab wann eine Sequenznummer nicht mehr benutzt wird, hängt jedoch vom simulierten Netzwerk ab: Es muss sichergestellt sein, dass die Verzögerung in einem Knoten nicht größer ist als der eingestellte Wert. In einem normalen Ethernet-Netzwerk kann zwar nicht garantiert werden, bis wann eine Nachricht in einem Switch weitergeleitet wurde. Eine großzügige Abschätzung für einen Maximalwert ist aber

möglich. Ist diese Abschätzung zu gering, sind zwei Berechnungen der Metriken für diesen Nachrichtentyp fehlerhaft.

Zu Problemen bei der Darstellung kann es kommen, wenn bei einem Nachrichtentyp die Verzögerung in einem Knoten größer ist als die geringste Latenz. Da der Garbage-Collector den Wert für die Ende-zu-Ende-Latenz in die Systemvariablen überträgt, kann hier nicht mehr ausgeschlossen werden, dass mehrere Latenzwerte gespeichert werden sollen. Die Graphen unterstützen dies, jedoch weicht hier der Verlauf der Latenz vom eigentlichen Verlauf ab (s. Abbildung 41). Eine grundsätzliche Verzögerung kann aber auch sonst nicht verhindert werden, da nur durch den Garbage-Collector sichergestellt werden kann, dass eine Nachricht wirklich nicht weitergeleitet wird.

Um eine gewisse Übersichtlichkeit zu wahren, wurden die CAPL-Funktionen gemäß ihrer Aufgabe in zehn Dateien verteilt (s. Abbildung 27). Hinzu kommen die von Compagnon generierten CAPL-Dateien, welche hauptsächlich globale Variablen befüllen.

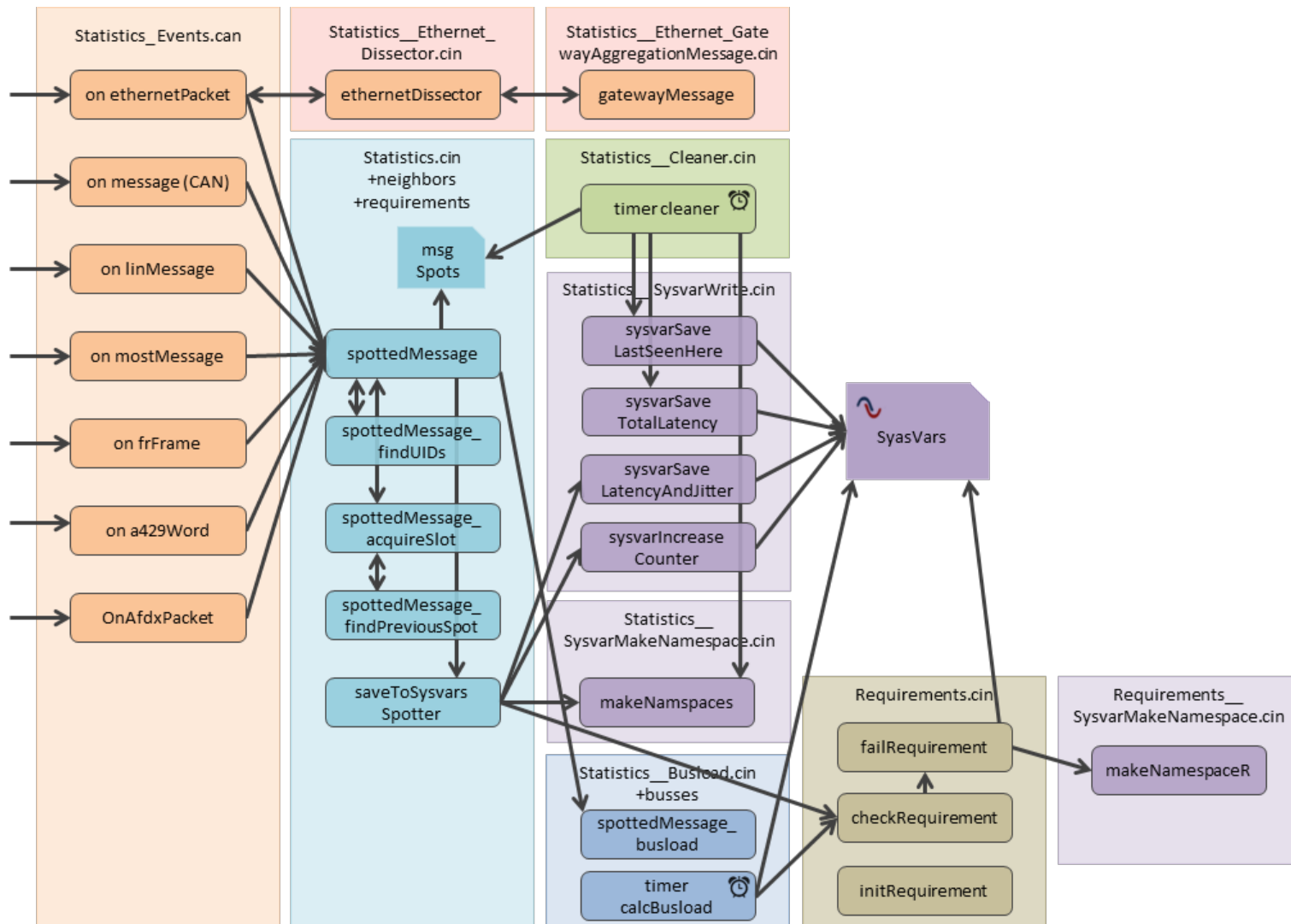


Abbildung 27: Zusammenhang der CAPL-Dateien und -Funktionen: Ankommende Nachrichten werden an die zentrale `spottedMessage`-Funktion weitergegeben, welche die Metriken berechnet und sie in Systemvariablen speichert.

### 4.3 Herausforderungen und Probleme

Im Folgenden wird auf einige Punkte eingegangen, welche sich im Verlauf der Arbeit als Schwierigkeit heraus gestellt haben. Nicht alle dieser Punkte konnten abschließend gelöst werden; sie bleiben somit (vorerst) als Einschränkungen bestehen.

#### 4.3.1 Erweiterung für Multicast/Broadcast

Gerade bei der Übertragung von Multimedia über Ethernet wird gerne das sogenannte Multicasting eingesetzt, um den Traffic zu reduzieren. Eine Multicast-Nachricht wird anhand einer einzigen Adresse an mehrere Knoten verteilt. Auch bei TTE und TSN kommt diese Adressierungstechnik zum Einsatz. Die Analyse muss deshalb damit umgehen können.

Ein Problem stellt dies im zweiten Konzept dar, da CANoe von Haus aus keine Informationen über die Verbindungen der Knoten im Netzwerk hat; es wird lediglich über das Auftauchen einer Nachricht auf einem Bus informiert. Werden die Metriken also in CANoe erhoben und wird eine Nachricht nun an mehrere Knoten verteilt, werden die Latenzzeiten falsch berechnet, da die vorher detektierte Nachricht nicht die tatsächlich vorhergegangene Nachricht ist. Deutlich wird dies an einem Beispiel: In Abbildung 28 geht der Nachricht auf dem Bus `ETH3` die Nachricht auf `ETH2` voraus. In Abbildung 29 hingegen ist der Vorgänger der Nachricht auf `ETH3` die Nachricht auf Bus `ETH1`. Beide Szenarien werden in den BLF-Dateien jedoch vollkommen identisch abgespeichert und können von CANoe somit nicht unterschieden werden.

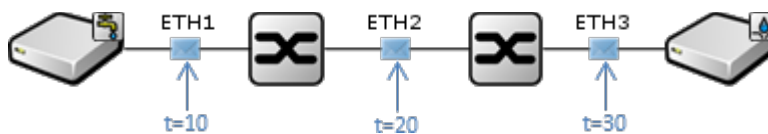


Abbildung 28: Von der Quelle (links) wird eine Nachricht zu einer Senke (rechts) übertragen. Die Latenz von `ETH2` zu `ETH1` beträgt  $\Delta t_{1,2} = 10$ , die Latenz von `ETH3` zu `ETH2` beträgt ebenfalls  $\Delta t_{2,3} = 10$ . Die Latenz von `ETH3` zu `ETH1` kann durch die Summe der beiden Latenzzeiten berechnet werden:  $\Delta t_{1,3} = \Delta t_{1,2} + \Delta t_{2,3} = 20$

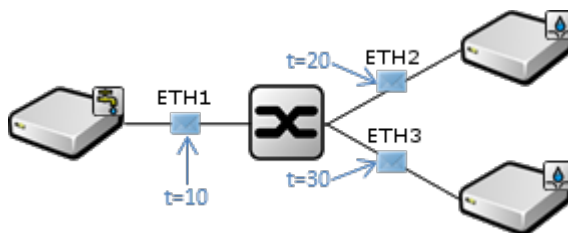


Abbildung 29: Von der Quelle (links) wird eine Nachricht per Multicast zu den beiden Senken (rechts) übertragen. Die Latenz von `ETH2` zu `ETH1` beträgt  $\Delta t_{1,2} = 10$ , die Latenz von `ETH3` zu `ETH1` beträgt aber  $\Delta t_{1,3} = 20$ . Eine Latenz von `ETH3` zu `ETH2`  $\Delta t_{2,3}$  kann hier nicht berechnet werden.

Es ist somit nötig, dass CANoe über die Topologie des analysierten Netzwerkes Bescheid weiß: Für die Berechnung der Latenz muss die älteste jener Nachrichten gefunden, welche die geringste Anzahl an Hops zum Ausgangspunkt aufweisen. In einem Netzwerk ohne Schleifen reicht es somit aus, wenn bekannt ist, wie groß der Abstand von einem Link zu den anderen Links ist.

Nach dem Einlesen eines Netzwerks aus der NED-Datei erstellt nun Compagnon für jeden Link und für jeden Abstand von diesem eine Liste mit Bussen. Hierfür wird ein Baum gemäß Abbildung 30 aufgebaut, welcher als Wurzel den jeweiligen Link besitzt (hier `ETH3`). Anschließend werden die Baum-Struktur verworfen und nur die Busse in die jeweilige Liste eingetragen (blaue Streifen in Abbildung 30).

Vor dem Start der Analyse werden diese Listen ausgedünnt, sodass nur noch die vom Nutzer/von der Nutzerin ausgewählten und somit zu analysierenden Busse enthalten sind. Diese Liste wird über CAPL-Code zu CANoe übertragen und während der Laufzeit verwendet, um die vorangegangene Nachricht zu einem aktuellen Nachrichtenevent zu finden.

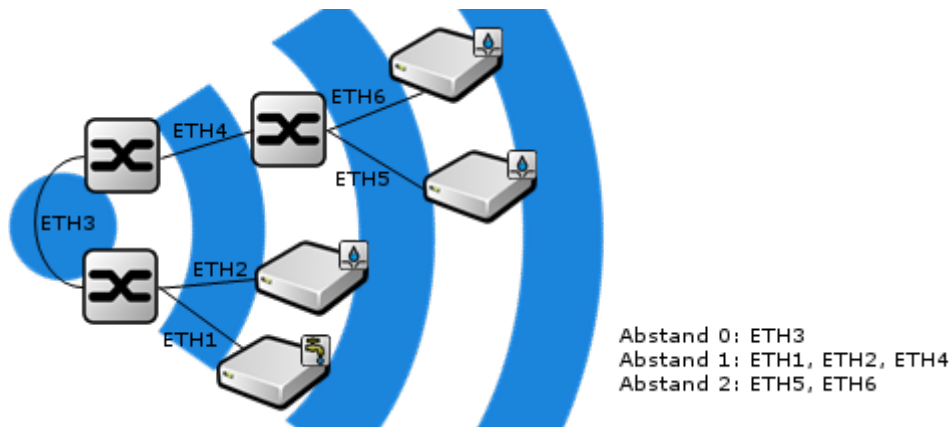


Abbildung 30: Beispielhafte Visualisierung der Netzwerkinformationen in CANoe. Ausgehend von einem Link (hier `ETH3`) werden die Nachbarschaften gemäß ihrer Entfernung in Hops gespeichert.

#### 4.3.2 Erhebung der Latenzzeiten während der Weiterleitung der Nachricht in OMNeT++

Eine Anforderung an die entwickelte Lösung ist, dass die „Metriken Latenzzeit, Jitter und Paketverlustrate [...] pro Nachrichtentyp und Zwischenknoten zur Verfügung stehen [sollen], damit nachvollzogen werden kann, an welcher Stelle im Netzwerk Probleme z.B. in Form großer Verzögerungen entstehen“ (s. Anforderung [6]). Dies gestaltet sich insbesondere bei der Erhebung der Statistiken in OMNeT++ schwierig:

OMNeT++ ermöglicht es mittels sogenannter *Statistic-Templates*, Statistiken dynamisch – also während der Simulation – zu erstellen. Hierdurch kann bspw. eine Statistik pro Verbindung, pro Client o.ä. erzeugt werden, ohne dass jede Statistik vor Simulationsbeginn



einzelnen definiert werden muss. Diese Templates lassen sich nutzen, um in den Switches Statistiken über die Latenz und den Jitter eines Nachrichtenstroms zu erstellen.

Da dieser Nachrichtenstrom im ersten Konzept jedoch anhand der Ziel-Applikation definiert wird (dies ist der Name der Systemvariablen), müssen im Switch die entsprechenden Applikationen für eine Nachricht gefunden werden. Da auch Multi- und Broadcast-Nachrichten unterstützt werden sollen, können dies sogar mehrere Applikationen sein.

Die Switches in einem Netzwerk müssen somit für jede sie passierende Nachricht abfragen, welche Applikationen als Empfänger in Frage kommen. In der aktuellen Umsetzung der Module in OMNeT++ bedeutet dies, dass ein zweiter Algorithmus aufgebaut werden muss, welcher die Zustellung von Nachrichten simuliert. Zum zusätzlichen Wartungsaufwand kommt hinzu, dass diese Vorgehensweise nicht aufwärtskompatibel wäre: Wird in einem Netzwerk auch IP eingesetzt, so kann die Zustellung nicht mehr auf Basis der MAC-Adresse erfolgen. Ein Ethernet-Switch, wie er jedoch weiterhin in einem solchen Netzwerk eingesetzt werden könnte, müsste für die Erhebung der Metriken jedoch folglich das Internet-Protokoll implementieren. Identisch müsste mit jedem weiteren Protokoll verfahren werden, welches auf Ethernet aufsetzt. Dies widerspricht dem Grundgedanken des ISO-OSI-Referenzmodells (s. Kapitel 2.1.2) und ist deshalb nicht akzeptabel.

Wünschenswert wäre es, wenn die Applikations-Module Methoden zur Verfügung stellen, mit Hilfe derer abgefragt werden kann, ob sie eine solche Nachricht empfangen würden. Die Überprüfung hinsichtlich der MAC-, der IP-Adresse oder der CAN-ID könnte somit an dieser Stelle und somit in der korrekten OSI-Schicht implementiert werden.

Schwierig gestaltet sich dieser Ansatz jedoch, wenn Adressen mehrfach verwendet werden. Dies ist bspw. der Fall, wenn mehrere CAN-Netzwerke über ein Ethernet-Backbone miteinander verknüpft werden. Die Verwendung derselben CAN-ID für zwei unterschiedliche Nachrichten ist in diesem Fall valide. Es müsste somit überprüft werden, ob die Nachricht des einen Netzwerkes das andere Netzwerk überhaupt erreichen kann.

Der Ansatz muss somit abgeändert werden, dass nun die Zwischenknoten befragt werden können, an welche anderen Knoten sie eine Nachricht weiterleiten würden. Die Erhebung bspw. der Latenzzeit während der Weiterleitung würde demnach folgendermaßen funktionieren:

- Einem Switch (bspw. `Switch1`) wird eine Nachricht übergeben.
- Er bestimmt, an welchen Knoten er diese Nachricht weiterleitet (bspw. `Switch2`).
- `Switch1` ruft deshalb eine Methode von `Switch2` auf, um abzufragen, an welche Knoten dieser die Nachricht weiterleiten wird.
- `Switch2` antwortet mit einer Liste dieser Knoten (bspw. `Switch30` und `Switch31`)
- `Switch1` fragt diese Knoten dasselbe wie zuvor `Switch2` und erhält ggf. wieder Listen von Knoten.
- Dieser Vorgang wird fortgesetzt, bis die Nachricht in allen Verzweigungen bei einer Applikation angekommen ist.

- Anschließend erstellt `Switch1` für jede Zielapplikation einen Statistik-Vektor und speichert darin den aktuellen Latenzwert der Nachricht.

Dieser Ansatz weist in mehreren Punkten noch Mängel auf:

Die Konfiguration des Netzwerkes könnte sich über die Zeit (während der Weiterleitung der Nachricht) verändern: So ist es bspw. möglich, dass sich ein Knoten bei einem Switch für eine Multicast-Adresse registriert und somit Empfänger einer Nachricht wird, noch während diese Nachricht unterwegs ist. Die Suche nach Zielapplikationen zu Beginn der Zustellung würde diese Applikation deshalb noch nicht berücksichtigen können.

Eine Anforderung an die Analyse war, dass CAN-Nachrichten, welche über das Ethernet-Backbone übertragen werden, erkannt werden (s. Kapitel 3.1). Dies ist wichtig, da gerade das Ethernet-Backbone eine große Herausforderung für die Echtzeitfähigkeit darstellt und die Übermittlung der CAN-Daten auch auf diesem Medium genau analysiert werden können soll.

Aus diesem Grund muss die Verwendung von Gateways berücksichtigt werden. Wenn eine Ethernet-Gateway-Nachricht an ein Gateway zugestellt wird, so transformiert dieses die Nachricht und leitet die enthaltenen Nachrichten an die angeschlossenen CAN-Knoten weiter. Ein CAN-Knoten kann nicht der Empfänger eine Gateway-Nachricht sein, jedoch der Empfänger einer darin verpackten Nachrichten. Eine eventuelle Transformation der Nachricht in einem Knoten (wie einem Gateway) muss deshalb bei der Suche nach den Zielapplikationen ebenfalls berücksichtigt werden.

Die Umsetzung dieses Konzeptes wurde als zu zeitintensiv eingeschätzt und deshalb nicht vorgenommen. Somit sind im ersten Konzept nur die Ende-zu-Ende-Latenzzeiten verfügbar. Doch auch im zweiten Konzept gestaltet sich die Detektion von Gateway-Nachrichten als schwierig.

### 4.3.3 Detektieren von Gateway-Nachrichten in CANoe

Im `BlfRecorder` werden die Nachrichten gemäß Abbildung 31 serialisiert (48 S. 44). Dasselbe Protokoll wurde auch in CANoe implementiert. Da in der Simulation keine Fragmentierung der Gateway-Nachrichten eingesetzt wird, wurde dieses Feature in CANoe ebenfalls außen vor gelassen.

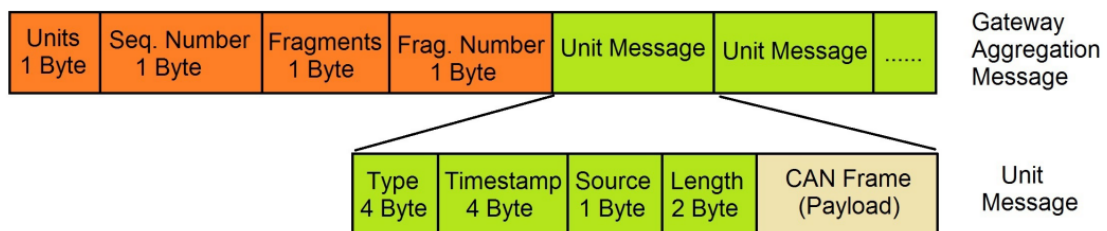


Abbildung 31: Sequentielles Format der Gateway-Nachrichten nach (48 S. 44).

Als problematisch stellte sich heraus, dass für die Gateway-Nachrichten kein eigener EtherType definiert ist, anhand dessen eine Gateway-Nachricht von anderen Ethernet-Nachrichten unterschieden werden kann. Durch die fehlerhafte Serialisierung der TSN-Nachrichten und damit der TSN-Header (s. Kapitel 4.3.5) war es jedoch auch nicht hilfreich, einen EtherType zu definieren, da das entsprechende Feld nicht in der Nachricht in CANoe enthalten ist. Somit kann in CANoe nicht erkannt werden, ob eine Ethernet-Nachricht von einem Gateway stammt.

Um sie dennoch verarbeiten zu können, wird in der aktuellen Implementierung versucht, jede Ethernet-Nachricht als Gateway-Nachricht zu interpretieren. Ist dies nicht möglich, wird die Nachricht als normale Ethernet-, TSN- oder TTE-Nachricht behandelt.

Da die Nachrichten aus einer System-Level-Simulation stammen und somit eine konstante Payload enthalten sowie durch die hohe Redundanz im Gateway-Format (z.B. vier Byte Type-Feld, welches durch die alleinige Unterstützung von CAN immer 1 ist) und die Pointer auf die nächste Unit-Message in der Nachricht (Length-Feld, s. Abbildung 31) können False-Positives ausgeschlossen werden. Normale Ethernet-Nachrichten werden also nicht fälschlicherweise als Gateway-Nachrichten interpretiert.

#### 4.3.4 Berechnung der Paketverlustrate

Wie bereits angedeutet, ist die Berechnung der Paketverlustrate (s. Kapitel 2.4) in beiden Konzepten nicht implementiert. Dies hat mehrere Gründe: Zum einen speichert der `BlfRecorder` in seiner aktuellen Implementierung keine fehlerhaften Nachrichten. Dies könnte jedoch relativ einfach ergänzt werden.

Zum anderen wird eine Applikation nicht darüber informiert, dass eine Nachricht auf dem Weg zu ihr verloren ging. Somit kann auch in der Applikation hierzu keine Statistik erstellt werden. Eine Ausnahme bilden Protokolle wie TTE, da diese den Zeitpunkt des Versands einer Nachricht fest vorschreiben. Bleibt eine der zyklischen Nachricht aus, kann die Applikation von einem Verlust ausgehen. In allen Fällen ist es jedoch nicht möglich, die Quelle des Problems (z.B. ein falsch dimensionierter Puffer in einem Switch) zu finden.

Zusätzlich ist die Aufzeichnung der Nachrichten an den Netzwerkports hinderlich: Ob bei der Übertragung einer Nachricht ein Fehler auftritt, wird im Link beschlossen, also nachdem die Nachricht von einem Netzwerkport gesendet und bevor sie von einem anderen Port empfangen wurde. Zu diesem Zeitpunkt wurde die Nachricht vom sendenden Netzwerkport schon in seine BLF-Datei gespeichert; er wird auch nicht nachträglich über einen Übertragungsfehler benachrichtigt. Es können also nur jene Nachrichten als fehlerhaft übertragen aufgezeichnet werden, welche von einem Netzwerkport empfangen werden.

Im Durchschnitt ergibt sich dadurch in der Aufzeichnung nur der halbe Wert für die Paketverlustrate. Als Workaround könnte also der berechnete Wert verdoppelt werden. Dies wäre jedoch nur im statistischen Mittel korrekt: Bei Netzwerken, welche stark asymmetrisch sind (wie bspw. einer Medienquelle auf der einen Seite des Netzwerks und

Lautsprechern auf der anderen Seite), würde das Ergebnis verfälscht, da der Netzwerkverkehr hauptsächlich von der einen zur anderen Seite des Netzwerks fließen würde. Korrekter wäre es, die Verlustrate nur für die eingehenden Nachrichten zu berechnen. Jedoch sind auch die dadurch ermittelten Werte unvollständig.

Der vierte Grund dafür, dass die Paketverlustrate nicht erhoben wird, ist der Verlust von Paketen in den Switches durch Pufferüberläufe: Ist ein Empfangs- oder Sendepuffer voll, so muss die neu empfangene Nachricht verworfen werden. Sie wird also nicht weiter übertragen. Für eine spätere Analyse müsste dieser Verlust jedoch in der BLF-Datei notiert werden. Da sich ein BLF-Kanal aktuell auf den Verkehr an einem Netzwerkport bezieht, müsste diese Verknüpfung aufgeweicht werden. Gespeicherte Nachrichten können im BLF-Format jedoch während des Schreib-Vorgangs nicht geändert werden, weshalb der `BlfRecorder` im Netzwerkport mit dem Schreiben warten müsste, bis der Puffer im Switch bestätigt hat, dass sie erfolgreich verarbeitet wurde.

Aus diesen Gründen wurde für das zweite Konzept in CANoe eine indirekte Bestimmung der Paketverlustrate implementiert: Es wird gezählt, wie oft eine Nachricht bestimmten Typs als letztes am jeweiligen Bus gesichtet wurde (hierfür ist der Garbage-Collector verantwortlich, s. Kapitel 4.2.4). Somit kann durch Überprüfung der Zählerstände am Ende der Analyse überprüft werden, an welcher Stelle im Netzwerk Nachrichten nicht mehr weitergeleitet wurden und somit verloren gingen. Da hierbei jedoch auch das Multicasting mit einbezogen werden muss (vgl. Kapitel 4.3.1), konnte aus Zeitgründen keine Routine entworfen werden, welche einen Wert berechnet und etwaige Grenzwerte überprüft. Nichts-desto-trotz wurde mit Hilfe dieser Methode bereits ein Fehler in der Simulation des Gateways entdeckt, welches immer nur die letzte CAN-Nachricht einer ID weiterleitete und die anderen verwarf.

Im ersten Konzept werden die Statistiken zum Verlust von Paket ebenfalls erhoben. Es ist in OMNeT++ jedoch nicht ohne weiteres möglich, die Anzahl der Verluste in den Puffern und auf den Links zu kombinieren. Aus diesem Grund werden verschiedene Systemvariablen verwendet (s. Kapitel 4.1.1).

### 4.3.5 Fehlerhafte Serialisierung von TSN-Nachrichten

Die Serializer in der aktuellen Version des `BlfRecorders` weisen einen Fehler auf, welches im zweiten Konzept die Analyse von TSN-Nachrichten anhand einer TSN-ID – im Gegensatz zu TTE-Nachrichten – unmöglich macht:

In OMNeT++ wird die TTE-Nachricht-Klasse von der Ethernet-Nachricht-Klasse abgeleitet (s. Abbildung 32). Um die sogenannte CT-ID zu speichern wird ein Teil der MAC-Adresse verwendet. Diese MAC-Adresse wird von der Basisklasse, also der Ethernet-Nachricht, definiert. Da der Serializer die komplette MAC-Adresse in die BLF-Dateien speichert, kann die CT-ID in der Analyse als Nachrichtentyp verwendet werden.

Bei TSN heißt der relevante Identifier „Stream-ID“ und ist von der MAC-Adresse unabhängig. Eine TSN-Nachricht (bzw. eine AVB-Nachricht) wird ebenfalls von der Ethernet-Nachricht abgeleitet (s. Abbildung 33), fügt dieser aber die Stream-ID als weiteres Feld hinzu; dieses Feld befindet sich aus der Perspektive des Ethernet-Frames in den Nutzdaten. Da der Serializer jedoch nur erkennt, dass die vorliegende TSN-Nachricht eine Ethernet-Nachricht ist, werden die zusätzlichen Felder wie die Stream-ID nicht in die BLF-Datei gespeichert, was eine Verwendung als Nachrichtentyp verhindert.

Auch wenn die Stream-ID in der Analyse nicht verwendet werden kann, können TSN-Nachrichten noch anhand der MAC-Adressen analysiert werden. Somit ist zumindest auf diesem Weg eine Selektion von TSN-Nachrichten möglich.

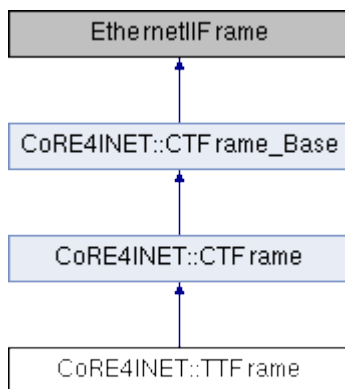


Abbildung 32: Klassendiagramm des TTFrames: Das TTFrame wird vom EthernetIIFrame abgeleitet

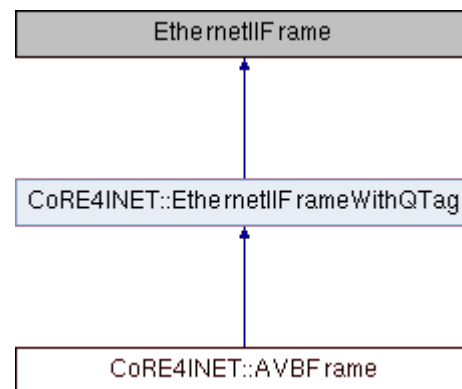


Abbildung 33: Klassendiagramm des AVBFrames: Auch das AVB-Frame wird vom EthernetIIFrame abgeleitet.

### 4.3.6 Große Anzahl von Systemvariablen

Vor der Analyse in CANoe werden Definitionen für Systemvariablen als XML generiert, welche anschließend von CANoe eingelesen werden (vgl. Kapitel 3.4 und Kapitel 4.2.3). Gerade beim zweiten Konzept ist es mitunter möglich, dass die Anzahl der Systemvariablen zu groß ist, um von CANoe verwaltet werden zu können. Beim ersten Konzept sind bisher nicht so viele Systemvariablen vorhanden, da die Latenzzeiten während der Weiterleitung der Nachrichten nicht erhoben werden kann (s. Kapitel 4.3.2)

Die genaue Anzahl hängt von der Anzahl der in Compagnon ausgewählten Busse ( $n_b$ ) sowie der Anzahl der zu analysierenden Nachrichtentypen ( $n_c$ ) ab, welche sich von den vom Nutzer/von der Nutzerin eingegebenen Grenzwerten ableitet. Sie berechnet sich wie folgt (vgl. Kapitel 4.2.3):

- Fünf Variablen werden für Konfiguration und COM-Kommunikation benötigt.  
 $N_C = 5$
- Im Statistics-Bus-Namespace werden pro Bus zwei Variablen für die Linkauslastung und den Datendurchsatz verwendet. Zusätzlich werden von jedem Bus zu jedem anderen ausgewählten Bus jeweils fünf Variablen für die Latenzzeiten und den Jitter definiert. Hinzu kommt ein Zähler für die Anzahl der Nachrichten.  
 $N_S = n_b \cdot (2 + (n_b - 1) \cdot (5 + 5 + 1))$
- Im Statistics-Type-Namespace werden zusätzlich zu einer Zählvariablen ebenfalls die Latenzzeiten und der Jitter mit fünf Variablen pro Nachrichtentyp definiert. Darüber hinaus werden ein Zähler und ein Zeitstempel pro Bus gespeichert sowie – wie zuvor – Latenz, Jitter und ein Zähler vom jeweiligen Bus zu allen anderen Bussen.  
 $N_T = n_c \cdot (5 + 5 + 1 + n_b \cdot (1 + 1 + (n_b - 1) \cdot (5 + 5 + 1)))$
- Im Requirements-Namespace werden pro Nachrichtentyp jeweils zwei Variablen für Latenz und Jitter definiert, bei der Linkauslastung sind es ebenfalls zwei.  
 $N_R = n_c \cdot (2 + 2) + n_b \cdot 2$

Insgesamt ergibt sich somit eine Anzahl von

$$N = N_C + N_S + N_T + N_R = 5 - 7n_b + 11n_b^2 + 15n_c - 9n_b n_c + 11n_b^2 n_c$$

und damit als Größenordnung von  $N$

$$O(N) = 11 \cdot n_b^2 \cdot n_c.$$

Die Anzahl der ausgewählten Busse  $n_b$  geht somit näherungsweise quadratisch ein, ist aber derzeit durch die maximale Anzahl der Kanäle in CANoe auf  $2 \cdot 32$  (Ethernet und CAN) begrenzt, während die Anzahl der Nachrichtentypen nach oben offen ist. In Tabelle 5 werden die Ergebnisse einiger Kombinationen aufgeführt.

$n_b \backslash n_c$	10	50	100	500	1000
1	179	859	1.709	8.509	17.009
5	2.695	12.495	24.745	122.745	245.245
10	11.285	52.285	103.535	513.535	1.026.035
20	46.615	216.015	427.765	2.121.765	4.239.265
30	106.145	491.945	974.195	4.832.195	9.654.695
64	489.563	2.269.363	4.494.113	22.292.113	44.539.613

**Tabelle 5:** Anzahl der für eine Analyse verwendeten Systemvariablen in Abhängigkeit von der Anzahl der ausgewählten Busse ( $n_b$ ) und der zu analysierenden Nachrichtentypen ( $n_c$ )

Die Anzahl der Systemvariablen steigt also sehr schnell. Dies führt zu mehreren Problemen: Zum einen muss die generierte XML-Datei sehr groß sein. Bei 500.000 Variablen ist die Datei ungefähr 100 MB groß, bei 2,1 Millionen Variablen belegt sie schon fast 400 MB Speicherplatz. Die für die Generierung verwendete C#-Bibliothek musste deshalb derart

angepasst werden, dass die Definitionen direkt in die Datei geschrieben und nicht als Zeichenkette zwischengespeichert werden.

Zum anderen ist es möglich, dass CANoe beim Einlesen die Speichergrenze von 32-Bit-Prozessen überschreitet (ca. 2 GB). Da CANoe offenbar mehrere Kilobyte pro Systemvariable belegt, ist dies ab einigen 100.000 Systemvariablen der Fall. Seit CANoe 8.5 steht eine 64-Bit-Version zur Verfügung, wodurch sich dieses Problem umgehen lässt. Nichts-desto-trotz kann das Erstellen und Einlesen der Variablen mehrere Minuten dauern und stellt damit eine große Verzögerung im Analyse-Prozess dar.

Aus diesem Grund berechnet Compagnon die Anzahl der Systemvariablen und warnt den Nutzer/die Nutzerin, wenn er zu viele Grenzwerte eingespeist und/oder zu viele Busse aktiviert hat (s. Abbildung 34).

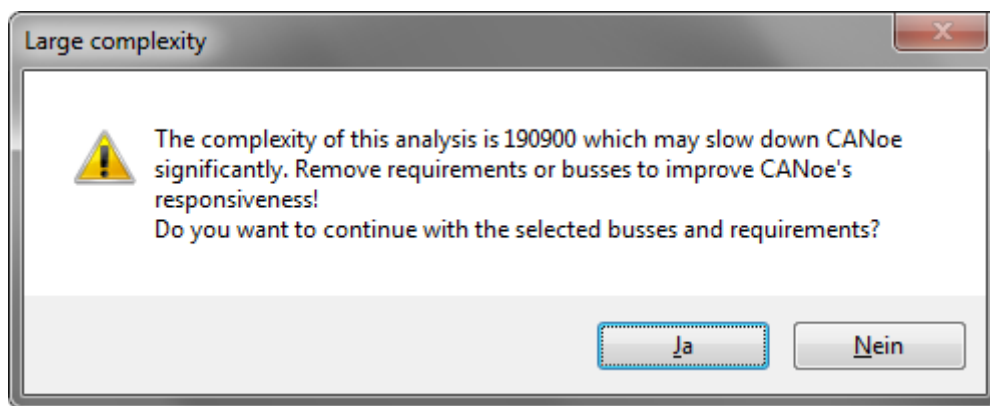


Abbildung 34: Warnung über eine zu große Anzahl an Systemvariablen, was die Arbeit mit CANoe behindern kann

#### 4.3.7 Langsame COM-Schnittstelle

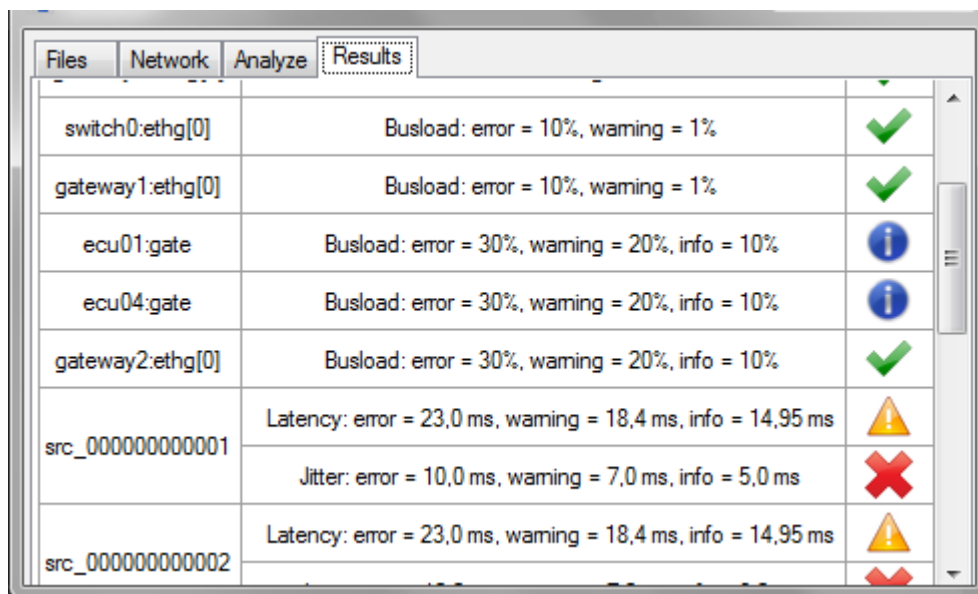
CANoe stellt die Möglichkeit zur Verfügung, in .NET über die COM-Schnittstelle (s. Kapitel 2.5) über Wertänderungen von Systemvariablen informiert zu werden. Die COM-Schnittstelle ist wegen der benötigten Thread-Kontext-Wechsel jedoch nicht besonders schnell und kann die Bewertung des Netzwerks stark ausbremsen.

In Compagnon wird zur Anzeige der Bewertung ein Symbol für jeden Grenzwert dargestellt (s. Abbildung 35). Hierfür wird auf eine Änderung der Systemvariablen reagiert, welche den aktuellen Wert der Überprüfung der Grenzwerte enthält. Bei Netzwerken, welche die Grenzwerte sehr häufig übertreten, konnte bei Testläufen ein deutlicher Einbruch in der Analyse-Geschwindigkeit beobachtet werden, welcher auf die COM-Kommunikation zurückgeführt werden konnte:

Meldet CANoe die Änderung einer Systemvariablen, so wartet es darauf, dass die externe Applikation dieses Ereignis verarbeitet hat. Dies ist nötig, damit sie zum gleichen Simulationszeitpunkt Aktionen (wie bspw. das Ändern einer anderen Variablen) durchführen kann.

Während Compagnon das passende Symbol zum Grenzwert sucht und überprüft, ob es geändert werden muss (also ob der Schweregrad höher ist als der bereits angezeigte), kann CANoe die Verarbeitung somit nicht fortsetzen.

Da die Anzeige des Symbols jedoch keine Synchronität mit CANoe benötigt, wurde der Vorgang entkoppelt: Das Ereignis einer Änderung wird von Compagnon lediglich notiert und die Analyse fortgesetzt. Parallel dazu (und eventuell auch etwas später) wird die Änderung von einem separaten Thread verarbeitet und die Anzeige aktualisiert. Diese Änderung beschleunigte den Analysevorgang deutlich.



Component	Analysis Results	Status Icon
switch0:ethg[0]	Busload: error = 10%, warning = 1%	Green checkmark
gateway1:ethg[0]	Busload: error = 10%, warning = 1%	Green checkmark
ecu01:gate	Busload: error = 30%, warning = 20%, info = 10%	Blue information icon
ecu04:gate	Busload: error = 30%, warning = 20%, info = 10%	Blue information icon
gateway2:ethg[0]	Busload: error = 30%, warning = 20%, info = 10%	Green checkmark
src_000000000001	Latency: error = 23,0 ms, warning = 18,4 ms, info = 14,95 ms	Yellow warning triangle
	Jitter: error = 10,0 ms, warning = 7,0 ms, info = 5,0 ms	Red X
src_000000000002	Latency: error = 23,0 ms, warning = 18,4 ms, info = 14,95 ms	Yellow warning triangle
		Red X

Abbildung 35: Darstellung der Bewertungsergebnisse in Compagnon



# 5 Vergleich der beiden Konzepte

Im Folgenden werden die beiden Konzepte aus Kapitel 3.3.1 und Kapitel 3.3.3 miteinander verglichen. Da hierfür mehrere Aspekte von Bedeutung sind, wird der Vergleich anhand der Anforderungen, der Konfigurationsmöglichkeiten, der Berechnungsgeschwindigkeiten und der Bewertungsergebnisse vorgenommen.

## 5.1 Vergleich anhand der Anforderungen

Zunächst geschieht dies anhand der in Kapitel 3.1 gestellten Anforderungen. Die Anforderungen [2], [3], [9], [10] und [11] betreffen die Umsetzung von Compagnon und spielen für den Vergleich der beiden Konzepte keine Rolle. Anforderung [12] wird in Kapitel 5.2 eingehend betrachtet. Somit werden hier nur die übrigen Anforderungen diskutiert.

Die wohl wichtigste Anforderung ist, dass alle zur Bewertung eines Netzwerkes nötigen Metriken erhoben werden (Anforderung [1]). Gemäß Kapitel 2.4 sind dies

- die Linkauslastung,
- der Datendurchsatz,
- die Puffergröße,
- die Latenzzeit,
- der Jitter sowie
- die Paketverlustrate.

Die Linkauslastung wird in beiden Konzepten erhoben, verwendet jedoch unterschiedliche Zeitfenster: Während in OMNeT++ für jede Nachricht die Linkauslastung für das zurückliegende Zeitintervall berechnet wird („floating interval“), geschieht dies in CANoe in festen Intervallschritten. Beide Methoden werden in (12 S. 77ff.) näher beschrieben.

In beiden Fällen bereitet die Erhebung der Auslastung von CAN-Links Probleme: Da OMNeT++ nicht über die Möglichkeit verfügt, einen CAN-Bus als geteiltes Medium zu simulieren, wird dies von der CoRE-Gruppe durch einen Knoten simuliert, welcher die empfangenen Nachrichten an alle angeschlossenen Geräte sendet (sozusagen ein Hub). Um ein korrektes Timing simulieren zu können, muss die Nachricht vom Sender sofort und somit mit möglichst hoher Datenrate beim Hub angelangen, welcher ihn dann mit langsamerer CAN-Geschwindigkeit (z.B. 500 kBit/s) weiterverteilt. Diese Geschwindigkeit wird in der INI-Datei angegeben. Die Links zwischen den CAN-Knoten und dem Hub werden

somit mit einer anderen Datenrate konfiguriert, als sie logisch auf dem CAN-Bus verwendet wird.

Dies führt an zwei Stellen zu Problemen: Im ersten Konzept darf nicht der CAN-Link verwendet werden, um die Linkauslastung zu erheben. Stattdessen müssen die Statistiken im Hub berechnet werden. Allerdings muss in diesem Fall von der Nomenklatur der Systemvariablen abgewichen werden. Im zweiten Konzept wird die falsche (möglichst hohe) Datenrate aus den NEDXML-Dateien verwendet. Diese Dateien müssen deshalb nach dem Exportieren durch den Nutzer/die Nutzerin bearbeitet werden.

Der Datendurchsatz wird ebenfalls in beiden Konzepten erhoben. Bei der Erhebung der Metriken in OMNeT++ ist er allerdings als Verhältnis der Netto- zur Bruttodatenmenge definiert, während er in CANoe relativ zur Datenübertragungsrate angegeben wird. Ebenso ist es problematisch, dass in beiden Konzepten fehlerhafte Übertragungen nicht beachtet werden. In CANoe liegt dies daran, dass der `BlfRecorder` keine fehlerhaften Nachrichten aufzeichnet, was aber leicht geändert werden könnte (vgl. Kapitel 4.3.4). In OMNeT++ liegt die Schwierigkeit darin, dass ein Statistik-Vektor mehrere Datenquellen nur in einer arithmetischen Formel kombinieren kann (51 S. 4.15.2.1). Somit können die beiden Signale „fehlerfreie Übertragung“ und „fehlerhafte Übertragung“ nicht gleichzeitig als Datenquelle benutzt werden. Abhilfe könnte schaffen, wenn ein weiteres Signal eingeführt wird, welches in beiden Fällen aufgerufen wird. Danach müsste jedoch ein eigens angepasster Filter den Datendurchsatz berechnen.

Der Datendurchsatz birgt eine weitere Schwierigkeit, welche in keinen der beiden Konzepten gelöst wurde: Es besteht die Möglichkeit, dass für diese Metrik ein Grenzwert sowohl nach oben als auch nach unten gewünscht ist. So könnte eine Anforderung lauten, dass die Netto-Datenrate auf einem Link nicht zu hoch ist, um eine gewisse Reserve beizubehalten. Der Datendurchsatz ist hierbei nach oben begrenzt. Andererseits könnte gefordert werden, dass der Overhead nicht zu groß sein darf, um die Effizienz des Netzwerks beizubehalten. Dann wäre der Datendurchsatz nach unten begrenzt. Ein Minimum-Grenzwert ist bisher jedoch in keiner der beiden Lösungen implementiert.

Die Puffergröße kann mit dem `Vector-Manager` aus dem ersten Konzept vollständig erhoben werden und steht pro Switch zur Verfügung (Anforderung [5]). Da der `Vector-Manager` beim zweiten Konzept nicht zum Einsatz kommt, stehen dort keine Puffergrößen zur Verfügung. Dies zu ändern wäre jedoch nur eine Frage der Konfiguration. Bei entsprechender Anpassung stünden auch im zweiten Konzept die Puffergrößen in CANoe zur Verfügung.

Die Paketverlustrate konnte in beiden Konzepten nicht wie gewünscht verwendet werden (s. Kapitel 4.3.4). Bei der Erhebung der Metriken in OMNeT++ werden deshalb mehrere Systemvariablen generiert, die den Verlust eines Pakets in einem Knoten anzeigen. Diese Verluste können jedoch nicht auf einen Nachrichtentyp bezogen werden (Anforderung [6]). In CANoe wurde aufgrund der fehlenden Informationen über fehlerhafte Übertragungen

und Verluste in den Puffern ein indirekter Indikator implementiert: Indem die letzte Sichtung eines Nachrichtentyps pro Link vermerkt wird, kann der Nutzer/die Nutzerin manuell überprüfen, ob ein ungewünschter Paketverlust vorliegt.

Die Benutzermetriken Latenzzeit und Jitter werden in beiden Konzepten erhoben. Doch auch hier wird im ersten Konzept Anforderung [6] nicht erfüllt; die Metriken stehen nicht für die Zwischenstationen zur Verfügung (s. Kapitel 4.3.2).

Beide Konzepte unterstützen den Einsatz von Gateways vollständig (Anforderung [7]). Zur Erhebung der Metriken müssen bei OMNeT++ neue Bussysteme, welche von Gateways unterstützt werden, nicht implementiert werden. Dies ist ein Vorteil gegenüber dem zweiten Konzept.

Auf die Erweiterbarkeit der Lösung (Anforderung [13]) wurde während des Entwicklungsprozesses großer Wert gelegt. Beim ersten Konzept können neue Metriken im Vector-Manager hinzugefügt werden, jedoch muss dieser daraufhin neu kompiliert werden. Da OMNeT++ offen und eine Kompilation zur Simulation der Netzwerke ohnehin notwendig ist, kann die hierfür nötige Software vorausgesetzt werden.

Beim zweiten Konzept kann ein neues Ethernet-Echtzeit-Protokoll mit eigenen Positionen für Identifier in der CAPL-Datei `Statistics_Ethernet_Dissector.cin` mit nur wenigen Zeilen implementiert werden. Da diese Lösung auch alle bisher von CANoe unterstützten Bussysteme abdeckt (z.B. FlexRay und MOST), muss bei der Simulation eines neuen Bussystems keine Änderung durchgeführt werden. Neue Metriken müssen hingegen in mehreren Dateien hinzugefügt werden; unter anderem in dem Template für die Systemvariablen in Compagnon. Dies bedingt auch eine erneute Kompilation der Applikation.

Eine Zusammenfassung der Evaluierung von Anforderung [1] ist in Tabelle 6 gegeben. Die Evaluierung aller hier diskutierten Anforderungen findet sich in Tabelle 7.

	Erhebung der Metriken in	
	OMNeT++	CANoe
Linkauslastung	1	1
Datendurchsatz	2	2
Puffergröße	1	3
Latenzzeit	1	1
Jitter	1	1
Paketverlustrate	2	2

Tabelle 6: Bewertung der Erhebung der Metriken in den beiden Konzepten gemäß Anforderung [1]

	Erhebung der Metriken in	
	OMNeT++	CANoe
[1]: Erhebung aller Metriken	2	3
[4]: Linkauslastung, Datendurchsatz und Paketverlustrate pro Link	2	2
[5]: Puffergröße pro Switch	1	3
[6]: Latenz, Jitter und Paketverl. pro Nachrichtentyp und Knoten	3	2
[7]: Unterstützung von Gateways	1	1
[13]: Einfach zu erweitern	2	2

Tabelle 7: Evaluierung der Anforderungen beider Konzepte

## 5.2 Vergleich der Konfiguration

Neben ihrer Funktionalität ist es ebenfalls von Bedeutung, wie gut sich die beiden Konzepte konfigurieren und bedienen lassen.

Zuerst soll auf den Aspekt der Inbetriebnahme eingegangen werden. Hierbei wird der derzeitige Stand der Arbeit beschrieben; viele der Schritte könnten jedoch auch in eine größere Lösung integriert werden.

Das Hinzufügen des `BlfRecorders` (s. Kapitel 4.1.2) erfordert das Ersetzen der Knoten im Netzwerk durch jene, welche im OMNeT++-Projekt `RecordedNetworks` bereitgestellt werden. Anschließend müssen alle Instanzen des `BlfRecorders` in der INI-Konfiguration aktiviert werden. Dazu gehört auch die Erstellung einer `BusMapper-XML`-Datei, welche die Verknüpfung zwischen den NED- und den BLF-Dateien herstellt (s. Kapitel 4.2.2). Dieser Aufwand ist jedoch bei beiden Konzepten identisch.

Im ersten Konzept muss in der INI-Datei zusätzlich der `Vector-Manager` eingefügt werden; dies ist mit einigen wenigen Zeilen erledigt. Um wirklich alle Metriken zu erheben, müssen aber auch einige Änderungen an NED-Dateien vorgenommen werden: Bspw. muss in dem Netzwerk ein neuer Kanal verwendet werden, welche die entsprechenden Statistik-Vektoren implementiert. Ebenso ist eine Änderungen an dem MAC-Modul im INET-Projekt nötig, um die Anzahl der fehlerhaft übertragenen Pakete zu erfassen (s. Kapitel 4.1.1).

Der Vorteil ist hierbei, dass durch eine simple Änderung an den NED-Dateien die Berechnung der Metriken verändert werden kann. So ist es unter anderem sehr einfach möglich, die Art des Zeitfensters anzupassen. In CANoe bedarf eine solche Änderung zusätzlicher Programmierarbeit.

Dies führt uns zur Konfiguration: Es gibt mehrere Optionen, auf die der Nutzer/die Nutzerin Einfluss nehmen kann, bspw. die Größe des Zeitfensters zur Berechnung der Metriken. Während der Nutzer/die Nutzerin hierfür in CANoe lediglich den Wert einer

Systemvariablen ändern muss, ist eine Beeinflussung der Größe in OMNeT++ derzeit nicht implementiert. Es ist aber auch hier eine Konfiguration über eine INI-Option denkbar.

Ein weiterer wichtiger Aspekt ist die Konfiguration der Anforderungen an das Netzwerk, also der Grenzwerte für die Metriken. Hierfür wurden zwei verschiedene Ansätze umgesetzt: Bei der Erhebung der Metriken in OMNeT++ hat der Nutzer/die Nutzerin die Möglichkeit, die Grenzwerte in die Oberfläche von Compagnon einzutragen (s. Abbildung 20). Diese Werte werden dann zu Beginn der Auswertung in CANoe über die COM-Schnittstelle übertragen. Beim zweiten Konzept stellt der Nutzer/die Nutzerin hingegen eine XML-Datei mit den Grenzwerten bereit, welche anschließend in die Definitionen der Systemvariablen einfließt.

Diese Unterschiedlichkeit leitet sich maßgeblich von den Unterschieden der Konzepte ab: Während Compagnon im ersten Ansatz aus der BLF-Datei bereits ablesen kann, welche Metriken existieren, definiert der/die Netzwerkverantwortliche mit den Grenzwerten die Nachrichtentypen, für welche die Metriken berechnet werden.

Der erste Ansatz bietet den Vorteil einer grafischen Oberfläche; dem Nutzer/der Nutzerin werden die Metriken der Apps präsentiert und er/sie muss nur noch den Wert eines Grenzwertes eingeben. Allerdings müssen die Werte bei jedem Start von CANoe übertragen werden, was einige Sekunden dauern kann. Der zweite Ansatz bedeutet einen sehr viel höheren Aufwand, da sich der/die Netzwerkverantwortliche mit dem Netzwerk und den darin versandten Nachrichten auseinandersetzen muss. Es muss identifiziert werden, welcher Nachrichtentyp von Interesse ist und mit einem Grenzwert versehen werden muss.

### 5.3 Vergleich der Berechnungsgeschwindigkeit

Ebenfalls maßgeblich ist eine kurze Berechnungszeit und somit eine gute Interaktivität der Applikation (Anforderung [12]). Hier sind zwei Aspekte wichtig: Einerseits die Beeinflussung der Simulation in OMNeT++, andererseits die Dauer der Bewertung in CANoe. Der Entwickler des `BlfRecorders` gibt an, dass die Verwendung seines Moduls die Simulationsdauer in OMNeT++ um 23% erhöht (48 S. 57). Im Rahmen dieser Arbeit wurden jedoch weitere Testreihen durchgeführt, um die Geschwindigkeit genauer beurteilen zu können.

Insgesamt müssen pro Netzwerk vier Konfigurationen simuliert werden, um den Einfluss der verwendeten Komponenten in OMNeT++ abschätzen zu können. Konfiguration 2 entspricht hier dem ersten Konzept (s. Kapitel 3.3.1), Konfiguration 4 dem zweiten Konzept (s. Kapitel 3.3.3). Gemessen wurde die Ausführungsdauer der Simulation in der Cmdenv-Umgebung von OMNeT++.

1. Simulation ohne Statistiken, `Vector-Manager` und `BlfRecorder`
2. Simulation ohne Statistiken und `Vector-Manager` aber mit `BlfRecorder`  
→ Rechenzeit des `BlfRecorders`  $t_{BlfRecorder} = t_{Conf\ 2} - t_{Conf\ 1}$
3. Simulation mit Statistiken, aber ohne `Vector-Manager` und `BlfRecorder`  
→ Rechenzeit der Statistiken durch  $t_{Statistiken} = t_{Conf\ 3} - t_{Conf\ 1}$
4. Simulation mit Statistiken, `Vector-Manager` und `BlfRecorder`  
→ Rechenzeit des `Vector-Managers` durch  
 $t_{Vector-Manager} = t_{Conf\ 4} - t_{Conf\ 3} - t_{BlfRecorder}$

Zuerst wurde ein kleines Netzwerk simuliert, welches aus drei Knoten und einem Switch besteht (s. Abbildung 36). Die Simulationszeit betrug 60s. Die Ergebnisse können in Tabelle 8 eingesehen werden, die Zeiten aller 36 Durchläufe finden sich in Anhang 2.1.

	Conf 1	Conf 2	Conf 3	Conf 4
<b>Median:</b>	285s	390s	9147s	9342s
<b>Mittelwert:</b>	285s	390s	9147s	9341s

Tabelle 8: Ausführungszeiten der vier Konfigurationen des kleinen Netzwerks bei 60s Simulationszeit

Bei diesem Netzwerk verlangsamt der `BlfRecorder` die Simulation um 100 Sekunden (s. Tabelle 8: Conf1 → Conf2). Sehr viel aufwändiger ist die Berechnung der Statistik-Vektoren in OMNeT++ (s. Tabelle 8: Conf1 → Conf3); sie verlangsamt die Ausführung um 2,5 Stunden, also um das mehr als das Dreißigfache! Der `Vector-Manager` verbraucht gerade mal anderthalb Minuten Rechenzeit (s. Tabelle 8: Conf3 → Conf4).

Anschließend wurde ein großes Ethernet-Netzwerk mit 15 Knoten und 7 Switches simuliert, welches für (52) entwickelt wurde (s. Abbildung 37). Die Simulationszeit betrug 30s; die Ergebnisse können in Tabelle 9 und Anhang 2.2 eingesehen werden.

	Conf 1	Conf 2	Conf 3	Conf 4
<b>Median:</b>	256s	359s	5398s	5559s
<b>Mittelwert:</b>	256s	359s	5393s	5558s

Tabelle 9: Ausführungszeiten der vier Konfigurationen des großen Ethernet-Netzwerks bei 30s Simulationszeit

Wie auch hier deutlich wird, ist die Erhebung der Statistiken in OMNeT++ herausragend aufwändig (+5.143s). Der `BlfRecorder` (+104s) benötigt fast doppelt so viel Rechenzeit wie der `Vector-Manager` (+57s).

Die Erhebung der Statistiken ist so aufwändig, da alle Statistiken berechnet werden, welche innerhalb des Netzwerks definiert werden. Hierzu gehören nicht nur die Statistik-Vektoren, welche aufgezeichnet werden, sondern auch noch Histogramme, Skalare, Vektoren anderer Module usw. Leider ist es in der aktuellen Version von OMNeT++ nicht möglich, nur die Berechnung der Statistik-Vektoren zu aktivieren. Sobald dies möglich ist, werden sich die Simulationsdauern von Conf 3 und Conf 4 wohl in einem sehr viel niedrigeren Bereich bewegen.

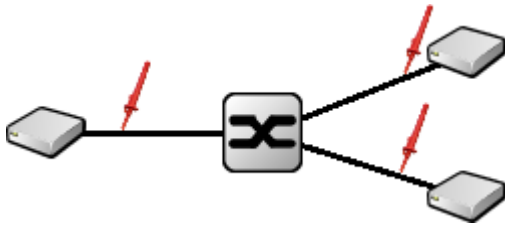


Abbildung 36: Simuliertes Ethernet-Netzwerk „small“

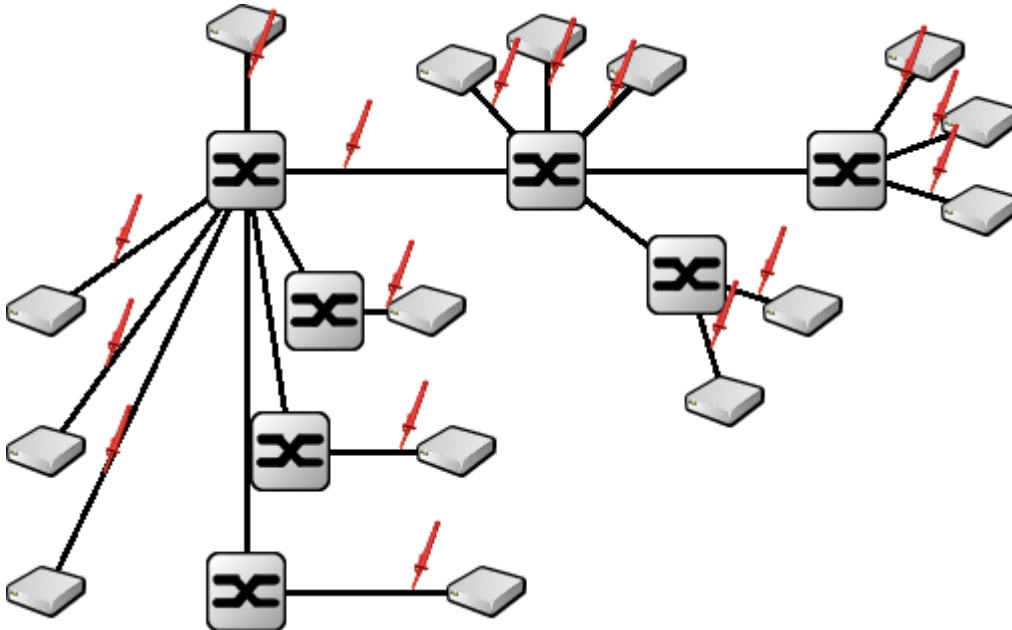


Abbildung 37: Simuliertes TT-Ethernet-Netzwerk „BMW“ aus (52)

Zusätzlich zur Ausführungsgeschwindigkeit in OMNeT++ ist es auch von Bedeutung, wie lange die Analyse in CANoe dauert. Hierfür wurden die Simulationsergebnisse der beiden Netzwerke aus Conf 4 jeweils 200 Mal analysiert. Die Ergebnisse für das kleine Netzwerk können in Anhang 2.3, jene für das große Netzwerk in Anhang 2.4 eingesehen werden. Sie werden in Tabelle 10 zusammengefasst.

	kleines Netzwerk		großes Netzwerk	
	Conf 2	Conf 4	Conf 2	Conf 4
<b>Median:</b>	165s	47s	435s	48s
<b>Mittelwert:</b>	168s	47s	432s	48s

Tabelle 10: Ausführungszeiten der Analyse in CANoe

Die Analyse mit Konfiguration 2 ist offensichtlich aufwändiger als mit Konfiguration 4. Dies überrascht nicht, da die Erhebung der Metriken rechenintensiver ist als das bloße Abspielen der BLF-Dateien.

## 5.4 Vergleich der Ergebnisse

Um die Funktion der beiden Konzepte zu überprüfen, wurden die Ergebnisse der Analyse miteinander verglichen. Wie sich dabei herausstellte, stimmt die Analyse des zweiten Konzeptes sehr genau mit der des ersten überein (s. Tabelle 11). Beim Vergleich der Statistik der Latenz der CAN-Nachrichten mit ID 1 im Netzwerk `gateway` (s. Abbildung 38) ergaben sich lediglich Abweichungen von genau  $1 \mu\text{s}$ . Diese Abweichung ist konstant über die gesamte Simulationsdauer. Erzeugt wird sie innerhalb des sendenden Knotens beim Übergang von der Applikation zum Puffer (s. Anhang 3). Da der `BlfRecorder` die Nachricht erst dort aufzeichnet und das erste Konzept mit dem Zeitstempel aus der Applikation arbeitet, ergibt sich somit eine konstante Abweichung.

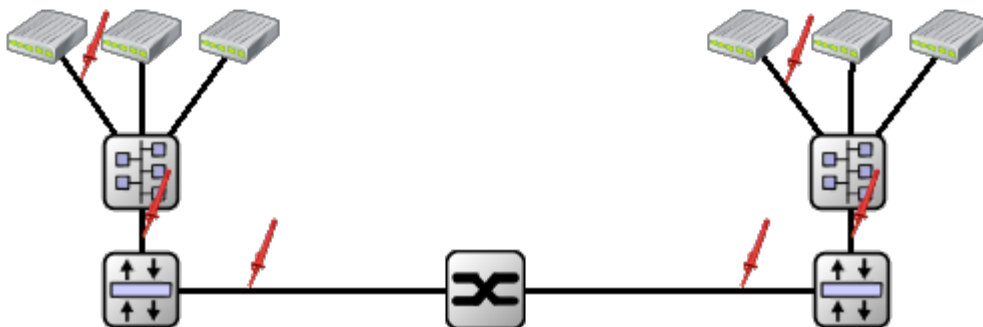


Abbildung 38: Simuliertes Netzwerk „gateway“ zur Überprüfung der Analyse-Ergebnisse

	tatsächlich	1. Konzept	1. Diff	2. Konzept	2. Diff
Anzahl Nachrichten	3327	3327	0	3333	6
Maximum [s]	0,021196	0,021196	0,000000	0,021195	0,000001
Durchschnitt [s]	0,012645	0,012645	0,000000	0,012611	0,000034
Minimum [s]	0,004194	0,004194	0,000000	0,004193	0,000001

Tabelle 11: Vergleich der statistischen Werte des Latenzverlaufs der CAN-Nachrichten mit ID 1 im Netzwerk „gateway“ bei einer Simulationsdauer von 10s.

Das Ergebnis des zweiten Konzepts gibt an, dass mehr Nachrichten analysiert wurden als im ersten Konzept (s. Tabelle 11). Dies ist darin begründet, dass zu dem Zeitpunkt, als die Simulation gestoppt wurde, noch sechs weitere Nachrichten im Netzwerk unterwegs und noch nicht endgültig zugestellt waren. Da OMNeT++ die Latenz nur beim Eintreffen im Zielknoten berechnet, wurden diese Nachrichten nicht mehr mit einbezogen. Im zweiten Konzept wurden sie jedoch bereits registriert und die Latenz-Werte für den bereits zurückgelegten Weg der Nachrichten erhoben.

Die etwas höhere Abweichung in der Berechnung des Durchschnitts ergibt sich durch Rechenungenauigkeiten: Während OMNeT++ alle einzelnen Latenzwerte speichert, darüber den Durchschnitt berechnet und somit eine sehr hohe Genauigkeit erreicht, wird im zweiten Konzept in CANoe für jede Nachricht der Durchschnittswert angepasst. Bei jeder solchen Anpassung entsteht – bedingt durch die endliche Genauigkeit von Fließkomma-



zahlen im Computer – ein kleiner Fehler, welcher sich über die Simulation summiert. Er ist jedoch sehr gering und somit akzeptabel.

Auch die grafischen Verläufe der Latenzzeiten lassen nach Sichtprüfung den Schluss zu, dass die Berechnungen vergleichbar sind (s. Abbildung 39, Abbildung 40 und Abbildung 41).

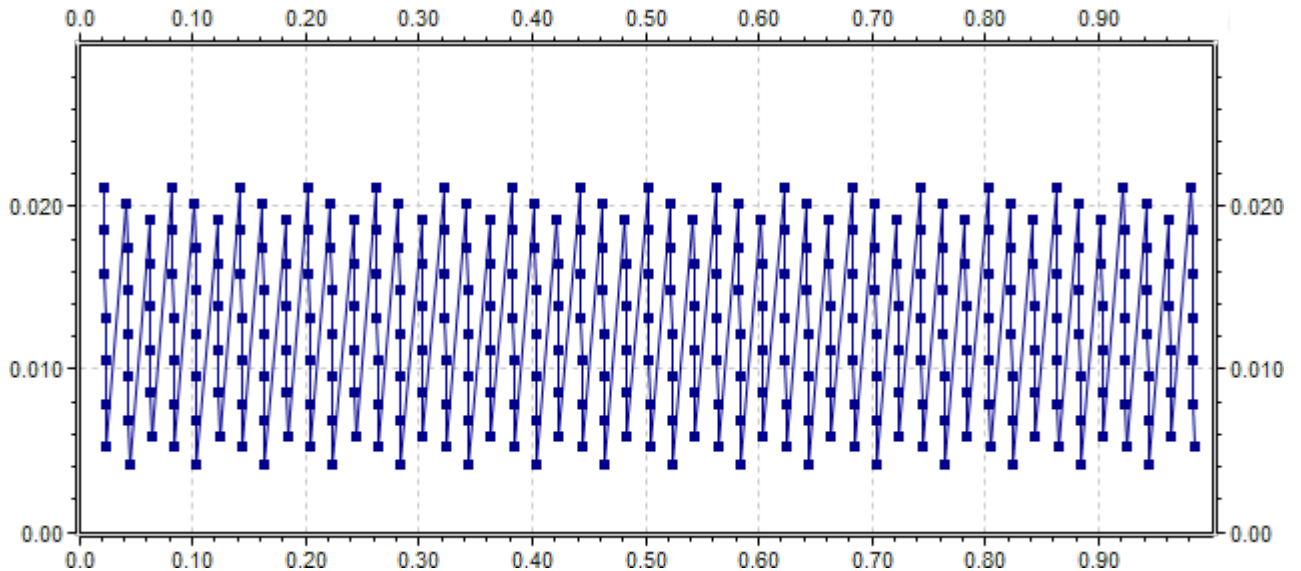


Abbildung 39: Verlaufsdiagramm der Latenzzeiten der CAN-Nachrichten mit ID 1 im Netzwerk „gateway“ in OMNet++. Sowohl X- als auch Y-Achse sind in Sekunden aufgetragen.

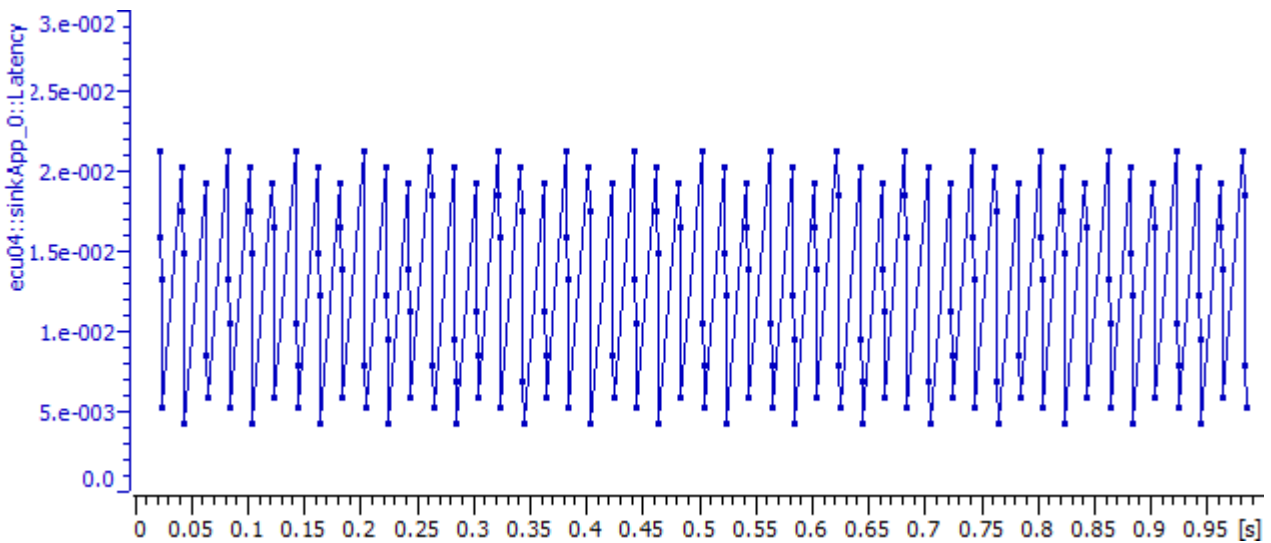


Abbildung 40: Verlaufsdiagramm der Latenzzeiten der CAN-Nachrichten mit ID 1 im Netzwerk „gateway“ in CANoe, erhoben gemäß dem ersten Konzept. Sowohl X- als auch Y-Achse sind in Sekunden aufgetragen.

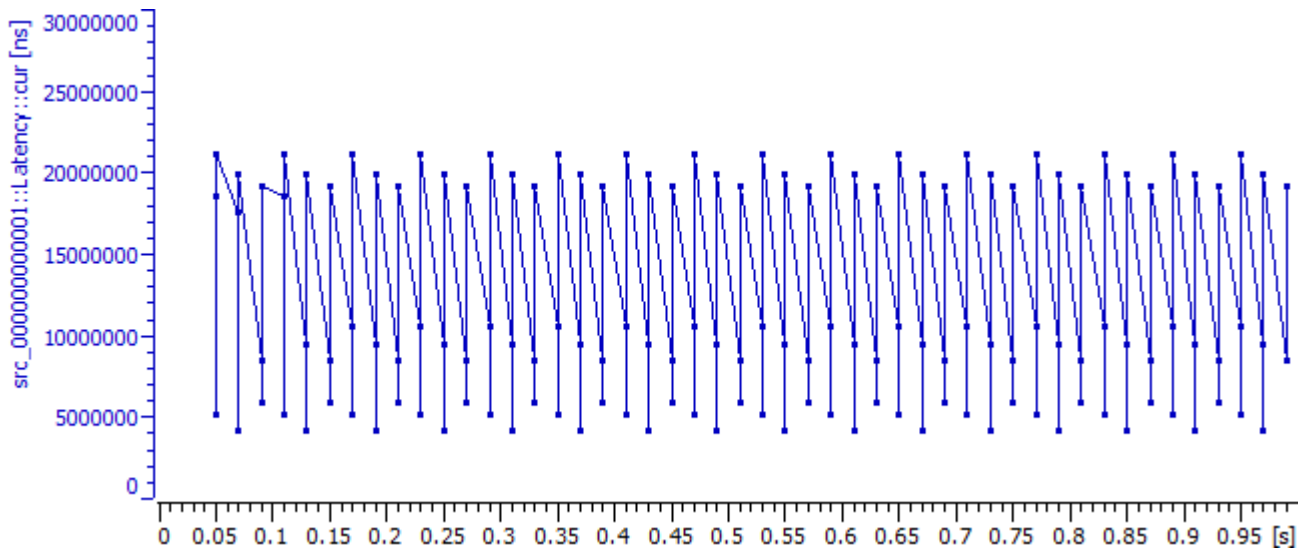


Abbildung 41: Verlauf der Latenzen der CAN-Nachrichten mit ID 1 im Netzwerk „gateway“ in CANoe, erhoben gemäß dem zweiten Konzept.

In Abbildung 40 sind weniger Datenpunkte zu erkennen als in Abbildung 39, jedoch ist die nur der Tatsache geschuldet, dass CANoe mitunter Datenpunkte ausblendet, wenn diese zu dicht beieinander liegen.

In Abbildung 41 ist vor allem anhand der senkrecht verlaufenden Linien zu erkennen, dass die Latenz-Werte erst nach Ablauf eines Zeitfensters eingetragen werden. Dies ist nötig, da anders nicht sichergestellt werden kann, ob die Zustellung der Nachricht erfolgt ist. Deshalb werden mehrere Werte auf einen Zeitpunkt gelegt. Auch wenn sich hierdurch die Form des Graphen verändert, bleiben die statistischen Werte korrekt (s. Tabelle 11).

## 5.5 Zusammenfassung

Sowohl das erste (s. Kapitel 3.3.1) als auch das zweite Konzept (s. Kapitel 3.3.3) weisen Schwächen bei der Erfüllung der Anforderungen auf (s. Kapitel 5.1): Die Paketverlustrate kann von beiden nicht direkt angegeben werden; ebenso wird der Datendurchsatz nur unvollständig erhoben. Im zweiten Konzept fehlen zusätzlich die Puffergrößen; OMNeT++ ist es hingegen nicht möglich, Latenzzeit und Jitter pro Link aufzuschlüsseln.

Das Hinzufügen des `BlfRecorders` in OMNeT++ ist bei beiden Konzepten von Nöten (s. Kapitel 5.2). Beim ersten Konzept ist es unter Verzicht auf Ansicht der Nachrichten im Trace-Fenster jedoch möglich, keine BusMapper-XML-Datei zu erstellen. Ebenso ist dort das Definieren von Anforderungen an das Netzwerk einfacher, hinsichtlich der Laufzeit jedoch nicht so effizient. Dies fällt wiederum nicht besonders ins Gewicht, da die Zeitspanne, welche die Analyse in CANoe benötigt, beim zweiten Konzept erheblich länger ist als beim ersten (s. Kapitel 5.3). Nichtsdestotrotz dauert die Analyse bei beiden Konzepten deutlich zu lange, um von Interaktivität sprechen zu können.

Ein großes Problem des ersten Konzeptes ist noch die sehr hohe Ausführungsdauer der Simulation, welche teilweise mehr als 30 Mal größer ist als beim zweiten Konzept.

Das zweite Konzept muss bei der Ermittlung der Ende-zu-Ende-Latenz auf ein Zeifenster zurückgreifen (s. Kapitel 5.4). Da nur hierdurch detektiert werden kann, dass eine Nachricht endgültig zugestellt wurde, können die Latenz-Werte erst nachträglich in die entsprechende Systemvariable aufgenommen werden. Während die Werte selbst hierdurch nicht beeinflusst werden, weicht der optische Verlauf doch stark vom Original ab.

Insgesamt schneidet somit das erste Konzept besser ab. Gerade wenn OMNeT++ nachbessert, die gezielte Aktivierung von Statistik-Vektoren ermöglicht und somit die Dauer der Simulation wieder auf ihr ursprüngliches Niveau sinkt, bietet das erste Konzept eine kürzere Analysezeit und ist somit für den Anwender besser zu nutzen. Auch die Entkoppelung der Anzeige von Nachrichten im Trace-Fenster und der Bewertung der Metriken ist von großem Vorteil, da bei der Auswahl der Links in Compagnon keine Kenntnisse über das Netzwerk und die darin versandten Nachrichten eingebracht werden müssen.

# 6 Qualitätssicherung und Evaluation

Im Folgenden wird dargestellt, wie die Qualität der entwickelten Lösungen gesichert und ihre Ergebnisse evaluiert wurden.

## 6.1 Qualitätssicherung

Bei jeder Entwicklung ist es wichtig, dass die Qualität des Produkts überprüft werden kann. Ein Ansatz hierfür ist das Test-Driven-Development (53). Durch die dabei verwendeten Unit-Tests, welche die Funktionalitäten im Kleinen testen, kann jedoch nicht überprüft werden, ob das Programm seinen Zweck erfüllt. So ist bspw. die Kommunikation mit CANoe über die COM-Schnittstelle oder das visuelle Anzeigen nicht mit Unit-Tests überprüfbar; hierfür sind Systemtests nötig.

Im Laufe der Arbeit wurde eine Vielzahl an Unit-Tests verfasst, welche jene Klassen und Funktionen testen, auf denen die Gesamtlösung basiert. Parallel dazu wurde ein Ablauf für einen Systemtest entworfen, welcher möglichst viele Fehlerquellen abdecken soll.

### Unit-Tests

Insgesamt wurden 63 Test-Methoden geschrieben, welche 51 % des Codes von Compagnon abdecken. Rechnet man jene Klassen heraus, welche überhaupt nicht von Unit-Tests überprüft werden (da z.B. ihre Funktionalität nicht auf diese Weise überprüft werden kann), beträgt die Codeabdeckung 86 %. In der Bibliothek BinlogNET (s. Kapitel 4.1.4) beträgt die Codeabdeckung 92 %.

Es ist selbstverständlich, dass mit Fertigstellung dieser Arbeit alle Unit-Tests erfolgreich ausgeführt werden konnten.

### Systemtest

Bei einem Systemtest wird das „gesamte System gegen die gesamten Anforderungen (funktionale und nicht-funktionale Anforderungen) getestet“ (54). Die dazugehörigen Testschritte (s. Anhang 1) wurden vor, während und nach der Arbeit festgelegt.

Die größte Anzahl der Punkte in diesem Systemtest überprüfen die Bedienung von Compagnon und der Darstellung dessen verschiedener Zustände, da diese Punkte nicht mit

Unit-Tests automatisiert getestet werden können bzw. da der Aufwand einer etwaigen Automatisierung den des manuellen Testens übersteigt. Eine manuelle Überprüfung eines Latenz-Verlaufs wurde bereits in Kapitel 5.4 vorgenommen. Somit wurde auch die Funktionalität des gesamten Systems getestet.

## 6.2 Evaluation

Im Folgenden wird überprüft, ob die in Kapitel 3.1 gestellten Anforderungen erfüllt wurden und ggf. untersucht, warum eine Anforderung nicht erfüllt werden konnte. Der Fokus wird hierbei auf jene Anforderungen gelegt, welche noch nicht in Kapitel 5.1 diskutiert wurden.

Wie gefordert können durch die Verwendung von Systemvariablen die erhobenen Metriken grafisch in CANoe dargestellt und weiterverarbeitet werden (Anforderung [2]). Der simulierte Nachrichtenverkehr ist im Trace-Fenster sichtbar (vgl. Abbildung 7, Anforderung [3]). Schwierig ist hierbei jedoch, dass der Nutzer/die Nutzerin die Verbindung zwischen den Links im Netzwerk und den im Trace-Fenster angezeigten Bussen selbst herstellen muss. Die Ergebnisse der Bewertung werden in Compagnon übersichtlich dargestellt (s. Abbildung 35, Anforderung [9]).

Gewünscht war ebenfalls, die Applikation Compagnon in die Oberfläche von CANoe einzubinden (Anforderung [10]). CANoe ermöglicht dies, wenn die Applikation `Windows Forms` verwendet. Aus diesem Grund wurde Compagnon mit eben dieser Oberflächenbibliothek erstellt. Wie sich später herausstellte, funktioniert das Vergrößern und Verkleinern des Fensters der Anwendung in CANoe nicht korrekt, was sich vor allem bei großen Netzwerken negativ auswirkte. Somit ist die Einbindung von Compagnon in CANoe zwar möglich, aber nicht ratsam.

Auch eine Hilfestellung bei der Auswahl der darzustellenden Metriken (Anforderung [11]) kann Compagnon nicht bieten: CANoe stellt keine Möglichkeiten zur Verfügung, den Messaufbau und die Oberfläche zu verändern. Somit können weder neue Graphen-Datensenken in CANoe eingerichtet werden, noch können die dazugehörigen Fenster geöffnet oder konfiguriert werden. Der Nutzer/die Nutzerin muss sich deshalb leider die anzuzeigenden Systemvariablen selbst heraussuchen.

Auf die Erweiterbarkeit der Lösung (Anforderung [13]) wurde großer Wert gelegt. Somit kann ein neues Ethernet-Echtzeit-Protokoll mit eigenen Positionen für Identifier in der CAPL-Datei `Statistics_Ethernet_Dissector.cin` mit nur wenigen Zeilen implementiert werden. Da die entwickelte Lösung alle bisher von CANoe unterstützten Bussysteme behandelt (z.B. FlexRay und MOST), muss bei der Simulation eines neuen Bussystems keine Änderung durchgeführt werden. Neue Metriken müssen hingegen in mehreren Dateien hinzugefügt werden; unter anderem in dem Template für die Systemvariablen in Compagnon. Dies bedingt auch eine erneute Kompilation der Applikation.

Der `Vector-Manager` des ersten Konzeptes kann einfach erweitert werden, sodass er weitere Statistik-Vektoren aufzeichnet. Das Hinzufügen weiterer Statistiken zur Simulation ist ohnehin sehr einfach möglich.

Eine Übersicht darüber, inwiefern alle Anforderungen erfüllt wurden, wird in Tabelle 12 gegeben.

	Erhebung der Metriken in	
	OMNeT++	CANoe
[1]: Erhebung aller Metriken	2	2
[2]: Grafische Darstellung der Metriken	1	
[3]: Anzeige der Nachrichten im Trace-Fenster	2	
[4]: Linkauslastung, Datendurchsatz und Paketverlustrate pro Link	2	2
[5]: Puffergröße pro Switch	1	3
[6]: Latenz, Jitter und Paketverl. pro Nachrichtentyp und Knoten	3	2
[7]: Unterstützung von Gateways	1	1
[8]: Einstellung von Grenzwerten	2	
[9]: Übersichtliche Bewertung eines Netzwerks	1	
[10]: Applikation innerhalb CANoes	2	
[11]: Unterstützung bei Auswahl der Metriken in CANoe	3	
[12]: Kurze Berechnungszeit	3	3
[13]: Einfach zu erweitern	1	2

Tabelle 12: Übersicht über die Evaluation aller gestellten Anforderungen

# 7 Zusammenfassung und Ausblick

In diesem Kapitel werden die Arbeit zusammengefasst, ein Fazit gezogen sowie mehrere Möglichkeiten zur Verbesserung vorgestellt.

## 7.1 Zusammenfassung

Im Rahmen dieser Arbeit konnten zwei Lösungen entworfen werden, welche in OMNeT++ simulierte, heterogene Echtzeit-Ethernet-Netzwerke analysieren und bewerten. Hierfür werden die Nachrichten während der Simulation in OMNeT++ mittels eines `BlfRecorder`-Moduls (48) in BLF-Dateien gespeichert. Eine externe Applikation verarbeitet die Netzwerkdefinitionen und konfiguriert CANoe.

In der ersten Lösung werden die Metriken von OMNeT++ mittels Statistik-Vektoren erhoben und von einem `Vector-Manager` in BLF-Dateien gespeichert. Anschließend werden sie in CANoe anhand der gestellten Grenzwerte bewertet (s. Kapitel 3.3.1, vgl. Abbildung 11). In der zweiten Lösung erfolgt die Erhebung der Metriken in CANoe mit Hilfe der gespeicherten Nachrichtenergebnisse. Zeitgleich wird ebenfalls anhand von Grenzwerten überprüft, ob sich das Netzwerk für die gestellten Anforderungen eignet (s. Kapitel 3.3.3).

In beiden Fällen werden die Verläufe der Metriken in CANoe dargestellt, einer Umgebung, die dem/der Netzwerkverantwortlichen beim Erstausrüster (OEM) vertraut ist (s. Kapitel 2.5). Hier können die erhobenen Werte betrachtet und weiterverarbeitet werden.

Schließlich wurden im Rahmen dieser Arbeit beide Lösungen bewertet und miteinander verglichen. Dabei stellte sich heraus, dass jeder der Ansätze über Vor- und Nachteile verfügt. Im Verlauf der Arbeit kristallisierte sich jedoch heraus, dass der erste dem zweiten überlegen ist, wenn er auch aktuell noch unter einer sehr großen Simulationsdauer leidet (s. Kapitel 5.5).

## 7.2 Fazit

Im Verlauf der Arbeit konnten zahlreiche Erfahrungen im Umgang mit OMNeT++ und CANoe gesammelt werden. Vor allem die COM-Schnittstelle zu CANoe und das BLF-Format (vgl. Kapitel 2.6) wurden genau untersucht. Ebenso wurde ein interessanter Einblick in die Anforderungen und Hindernisse beim Entwurf und bei der Bewertung von echtzeitfähigen Automobil-Netzwerken gewährt.

Die entwickelten Lösungen erfüllen die meisten der Anforderungen (s. Kapitel 6.2) und nehmen eine Bewertung von Netzwerken hinsichtlich ihrer Echtzeit-Fähigkeit vor. Sie weisen dabei jedoch auch einige markante Probleme auf:

Der Aufwand, die Simulationsdaten aus OMNeT++ zu exportieren, ist noch recht hoch. Er kann jedoch verringert werden, wenn der `BlfRecorder` standardmäßig in die CoRE-Module eingefügt wird. Der Aufwand beim Importieren der Daten in Compagnon ist ebenfalls hoch und bedarf einiger Zeit, muss jedoch nur einmal pro Netzwerk vorgenommen werden. Ebenso nehmen Simulation und Analyse eine zu große Zeit in Anspruch (s. Kapitel 5.3).

Bei der Analyse nach dem zweiten Konzept ist unter anderem die große Anzahl der Systemvariablen hinderlich (s. Kapitel 4.3.6). Sie macht es unmöglich, große Netzwerke mit vielen Nachrichtentypen vollständig bewerten zu lassen. Stattdessen muss die Analyse manuell auf mehrere Durchläufe aufgeteilt werden, wobei vom Anwender darauf geachtet werden muss, dass Nachrichtenpfade nicht auseinander gerissen werden. Ebenso fehlt die Möglichkeit, ein Netzwerk automatisch auf Paketverluste zu überprüfen (s. Kapitel 4.3.4). Die Analyse ist somit zwar funktionsfähig, jedoch ist der gesamte Prozess zur Bewertung eines Netzwerks noch zu kompliziert und vor allem zu langwierig, um ihn produktiv einzusetzen.

Es konnte gezeigt werden, dass CANoe ein sehr universell einsetzbares Programm ist, mit welchem sich viele verschiedene Aufgaben erledigen lassen. Die Stärke von CANoe liegt allerdings beim Interpretieren der Nachrichten-Inhalte, welche bei einer System-Level-Simulation keine Rolle spielen. Durch die eingeschränkten Möglichkeiten zur Steuerung der Oberfläche konnte kein vereinfachtes Bedienkonzept umgesetzt werden, weshalb dem Nutzer/der Nutzerin einiges über das Netzwerk und die versandten Nachrichten bekannt sein muss, welches er analysiert. Da es sich hierbei meistens um ein generiertes Netzwerk handelt (vgl. Kapitel 2.7), sind solche Kenntnisse wahrscheinlich nicht vorhanden.

All diese manuellen Schritte müssen entfernt werden, um eine Integration in PREEvision (s. Kapitel 2.7) zu ermöglichen. Die Möglichkeit, Ethernet-Netzwerke auf Basis der bisherigen Prozesse zu generieren und mit Hilfe einer Simulation zu bewerten, wird in Zukunft eine wichtige Lücke füllen.

Insgesamt stellt diese Arbeit eine gute Grundlage dar, um die automatische Bewertung von in OMNeT++ simulierten Netzwerken mit einer möglichen Weiterverarbeitung in CANoe voran zu treiben.



## 7.3 Ausblick

Es existieren einige Punkte, durch welche die Komplexität der Bedienung verringert und der Bewertungsvorgang verbessert werden könnte:

Wie in Kapitel 4.2.3 erläutert wurde, werden im zweiten Konzept in den Namespaces der Systemvariablen die Namen der Busse verwendet (z.B. `CAN01` oder `ETH05`, vgl. Listing 9). Da sich die Nummern der Busse mit jeder Auswahl in Compagnon ändern können, ist es intuitiver und passender, hier einen anderen Text zu verwenden. Eine Verbindung könnte z.B. anhand des Namens des jeweiligen Netzwerkports identifiziert werden. Die hierfür nötige Änderung in Compagnon und im CAPL-Code ist von relativ geringem Ausmaß. Dennoch würden z.B. im Trace-Fenster weiterhin die Namen der Busse verwendet werden.

Compagnon könnte ebenfalls derart angepasst werden, dass die Bewertungsergebnisse grafisch im Netzwerk dargestellt werden, um einen noch schnelleren Überblick zumindest über die System-Metriken (s. Kapitel 2.4) zu verschaffen (s. Abbildung 12). Die Darstellung von Benutzer-Metriken stellt sich schwieriger dar, da Nachrichten in einem Netzwerk nicht angezeigt werden. Hier könnte man sich an Cardiogram (1) orientieren, welches Probleme auf einer Zeitachse aufträgt (s. Abbildung 42). Überschreitungen von Latenz oder Jitter könnten somit übersichtlich aufgetragen werden; außerdem kann mit einem Blick erfasst werden, ob viele oder wenige Probleme aufgetreten sind. Kombiniert könnte somit ein guter Gesamteindruck des Netzwerks vermittelt werden.

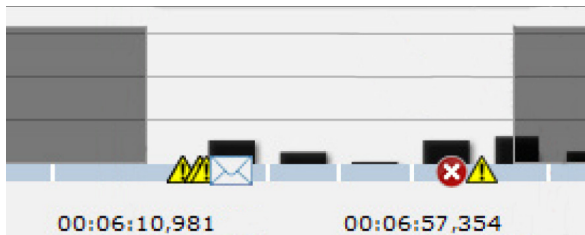


Abbildung 42: Darstellung von Hinweisen, Warnungen und Fehlern auf einer Zeitachse in Cardiogram nach (1 S. 6)

Eine weitere Änderung an den Systemvariablen, welche das in Kapitel 4.3.6 beschriebene Problem mit dem hohen Speicherverbrauch adressiert, wäre die Nutzung von Strukturen in den Systemvariablen: Dieses in CANoe 9.0 eingeführte Feature ermöglicht es, Strukturen für die aktuellen, minimalen, maximalen und durchschnittlichen Werte von Latenz und Jitter einzusetzen. Hierdurch könnte Speicherplatz sowohl in der Definitionsdatei als auch im Arbeitsspeicher gespart werden. Gemäß Aussagen vonseiten Vectors sind diese Einsparungen aber wahrscheinlich eher gering.

Das in Kapitel 4.1.3 beschriebene BusMapper-XML-Format und das dazugehörige Vorgehen zur Herstellung der Verknüpfung zwischen den BLF-Kanälen und den Punkten im Netzwerk, an welchem der Verkehr aufgezeichnet wurde, sind umständlich und unschön. Besser wäre es, diese Informationen direkt in die BLF-Dateien zu integrieren. Dort würden sie von dem `BlfRecorder` eingetragen werden, welcher über alle nötigen Informationen verfügt.

Außerdem wären sie fest einem Kanal zugeordnet und könnten somit auch nicht verloren gehen. Die Vector Informatik GmbH überlegt derzeit, ob Meta-Daten aus dem BLF-Format über die `binlog.dll` an externe Programme herausgegeben werden soll. Diese Meta-Daten könnten dann zum Speichern solcher Zuordnungen verwendet werden.

Alternativ könnte ein `AppText`-Objekt benutzt werden, welches es ermöglicht, Zeichenketten in eine BLF-Datei zu schreiben. Es wird aktuell von CANoe eingesetzt, um Verweise auf Datenbank-Dateien in BLF-Dateien einzufügen. Der `BlfRecorder` könnte somit in diesem String die Informationen kodieren, welcher Netzwerkport in welchem Kanal aufgezeichnet wurde.

Die während dieser Arbeit entwickelte Bibliothek BinlogNET (s. Kapitel 4.1.4) wird von der Vector Informatik GmbH übernommen. Es ist geplant, BinlogNET gemeinsam mit Vector-Produkten wie CANoe auszuliefern. Die aktuelle Version kann dann bspw. mit der CANoe-Demo von der Vector-Webseite heruntergeladen werden.

Weitere Neuerungen sind von Seiten Vectors geplant: So soll das Binlog-Format in naher Zukunft derart erweitert werden, dass nicht mehr die gesamte Payload von Nachrichten gespeichert werden muss. Gerade bei großen Ethernet-Frames aus der System-Level-simulation werden damit unnötig Speicherplatz und Rechenzeit belegt.

Ebenso ist bereits angedacht, dass CANoe mittelfristig Ethernet-Simulationen besser unterstützen wird. Dies beinhaltet unter anderem, dass komplette Netzwerke angezeigt werden können sowie die Verfolgung von Nachrichten über mehrere Links hinweg, was in dieser Arbeit über Sequenznummern realisiert wurde (s. Kapitel 4.2.1). Für beide Konzepte könnte ein solches Feature erhebliche Vorteile bieten.

Es bleibt abzuwarten, wie sich diese Änderungen auf die Bewertung von Echtzeit-Ethernet-Netzwerken auswirken werden. Ebenso bleibt es spannend, welches Echtzeit-Ethernet-Protokoll sich durchsetzen wird und wie sich die Elektrik-/Elektronik-Architektur im automobilen Bereich weiter entwickelt.

## 8 Literaturverzeichnis

1. *Cardiogram: Visual Analytics for Automotive Engineers*. **Sedlmair, Michael, et al., et al.** 2011. Proceedings of ACM CHI 2011 Conference on Human Factors in Computing Systems. S. 1727-1736. 10.1.1.386.1375.
2. **Steinbach, Till.** *Ethernet als Bus für Echtzeitanwendungen im Automobil*. CoRE Research Group, Hochschule für Angewandte Wissenschaften. Hamburg : s.n., 2008.
3. *Autonomes Fahren – Mobilität und Auto in der Welt von morgen*. **Fraedrich, Eva und Lenz, Barbara.** 04 2014, Technikfolgenabschätzung – Theorie und Praxis 23. Jg, Bd. Heft 1. [http://elib.dlr.de/93592/1/Technikfolgenabsch%C3%A4tzung\\_Autonomes%20Fahren\\_Fraedrich%20Lenz\\_2014.pdf](http://elib.dlr.de/93592/1/Technikfolgenabsch%C3%A4tzung_Autonomes%20Fahren_Fraedrich%20Lenz_2014.pdf).
4. *Performance evaluation of the inter-domain communication in a switched Ethernet based in-car network*. **Lim, Hyung-Taek, et al., et al.** Bonn : IEEE, 2011. IEEE 36th Conference on Local Computer Networks (LCN). S. 101-108. [http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6115156&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs\\_all.jsp%3Farnumber%3D6115156](http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6115156&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs_all.jsp%3Farnumber%3D6115156). 10.1109/LCN.2011.6115156.
5. **Wendeborn, Jonathan.** Car2X - When Cars Talk. [PDF]. Hamburg : s.n., 05. 07 2015.
6. *Automotive Ethernet: In-vehicle Networking and Smart Mobility*. **Hank, Peter, et al., et al.** San Jose, CA, USA : EDA Consortium, 2013. Proceedings of the Conference on Design, Automation and Test in Europe (DATE). S. 1735-1739. 978-1-4503-2153-2.
7. *Automotive Network Planning - a genetic approach*. **Müller-Rathgeber, Bernd.** [Hrsg.] IEEE. Xi'an : s.n., 2009. S. 1088-1092. 978-1-4244-3503-6.
8. *Challenges in a Future IP/Ethernet-based In-Car Network for Real-Time Applications*. **Lim, Hyung-Taek, Völker, Lars und Herrscher, Daniel.** New York : s.n., 2011. Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE. S. 7-12. 978-1-4503-0636-2.
9. **CoRE Group.** About CoRE. [Online] 2016. [Zitat vom: 25. 05 2016.] <https://core.informatik.haw-hamburg.de/about/core-group.html>.
10. —. RecBar. [Online] 2016. [Zitat vom: 25. 05 2016.] <https://core.informatik.haw-hamburg.de/projects/recbar.html>.
11. **Burchard, Benjamin.** Visualisierung von Kommunikationsdaten im Automotive Echtzeit-Ethernet Netzwerk. [Online] 02 2015. [Zitat vom: 25. 05 2015.] <https://core.informatik.haw-hamburg.de/bib/eigene/b-vkaen-15.pdf>.

12. **Kempf, Fabian.** *Simulationsbasierte Analyse heterogener Fahrzeugnetzwerke: Generierung, Simulation und Evaluation.* Hamburg : s.n., 2014.
13. *Introducing support for scheduled traffic over IEEE audio video bridging networks.* **Alderisi, Giuliana, Patti, Gaetano und Lo Bello, Lucia.** Cagliari : IEEE, 2013. Conference on Emerging Technologies & Factory Automation (ETFA). S. 1-9. 10.1109/ETFA.2013.6647943. 978-1-4799-0862-2.
14. *The Use of Ethernet for Single On-board Train Network.* **Aziz, M., et al., et al.** Sanya : IEEE, 2008. IEEE International Conference on Networking, Sensing and Control (ICNSC). S. 1430-1434. 10.1109/ICNSC.2008.4525444. 978-1-4244-1686-8.
15. *Ethernet-Based Car Control Network.* **Daoud, Ramez M., et al., et al.** Ottawa, Ont. : IEEE, 2006. Canadian Conference on Electrical and Computer Engineering (CCECE). S. 1031-1034. 10.1109/CCECE.2006.2777777. 1-4244-0038-4.
16. *Fault-Tolerant Ethernet-Based Vehicle On-Board Networks.* **Daoud, Ramez M., et al., et al.** Paris : IEEE, 2006. S. 4662-4665 . 10.1109/IECON.2006.347266.
17. **Wikipedia.** Backbone (Telekommunikation). [Online] 19. 08 2015. [Zitat vom: 04. 03 2016.] [https://de.wikipedia.org/wiki/Backbone\\_%28Telekommunikation%29](https://de.wikipedia.org/wiki/Backbone_%28Telekommunikation%29).
18. **Heinz, Matthias.** *Modellbasierte Entwicklung und Konfiguration des zeitgesteuerten FlexRay Bussystems.* Karlsruhe : KIT Scientific Publishing, 2012. 978-3866448162.
19. **Salathé, Jan Jasper.** *Konzepte zur Reduzierung der Übertragungslatenz im AVB v2 Standard.* Hamburg : s.n., 2013.
20. **Wikipedia.** OSI-Modell. [Online] 26. 04 2016. [Zitat vom: 27. 05 2016.] <https://de.wikipedia.org/wiki/OSI-Modell>.
21. **Cola, Jack.** The Seven Layers of OSI. [Online] 18. 06 2013. [Zitat vom: 27. 05 2016.] <http://www.jackcola.org/wp-content/uploads/2012/11/osireferencemodel.jpg>.
22. **Wikipedia.** *Internet Protocol.* [Online] 17. 06 2016. [Zitat vom: 24. 06 2016.] [https://de.wikipedia.org/wiki/Internet\\_Protocol](https://de.wikipedia.org/wiki/Internet_Protocol).
23. *Modelling And Simulation Of Automotive Vehicle Systems.* **Jones, R. P.** 02. 04 1990.
24. **OMNeT++.** What is OMNeT++. [Online] 2015. [Zitat vom: 26. 02 2016.] <https://omnetpp.org/intro/what-is-omnet>.
25. **Köksal, Murat Miran.** *A Survey of Network Simulators Supporting Wireless Networks.* Ankara : s.n., 2008.
26. **EMOTIVE GmbH & Co. KG.** Diagnosesysteme im Automobil - MOST - Media Oriented System Transport. [Online] 17. 11 2014. [Zitat vom: 26. 02 2016.] <http://www.emotive.de/wiki/index.php?title=MOST>.

- 
27. **Lo Bello, Lucia.** The case for Ethernet in Automotive Communications. *ACM SIGBED Review - Special Issue on the 10th International Workshop on Real-time Networks (RTN 2011)*. 2011, Bd. 8, 4.
28. **Wikipedia.** BroadR-Reach. [Online] 05. 02 2016. [Zitat vom: 04. 03 2016.] <https://de.wikipedia.org/wiki/BroadR-Reach>.
29. **Fellmeth, Peter.** *Telefoninterview: Workflow der E/E-Architektur-Entwicklung und Zukunft von CANoe*. Bremervörde, 30. 10 2015.
30. **Tichy, Prof. Dr. Walter F., Pankratius, Dr. Victor und Jannesari, A.** Netzwerk-Kenngrößen und -Topologien. [Online] 14. 03 2012. [Zitat vom: 24. 02 2016.] <http://www.ipd.uka.de/Tichy/uploads/fohlen/138/Cluster06Netztopologien.pdf>.
31. **Kempf, Fabian, Korf, Franz und Steinbach, Till.** *RECBAR-Bericht: Metriken in Ethernet-basierten Fahrzeugnetzen*. Hamburg : s.n., 2014.
32. **Wikipedia.** Datenübertragungsrate. [Online] 05. 02 2016. [Zitat vom: 24. 02 2016.] <https://de.wikipedia.org/wiki/Daten%C3%BCbertragungsrate>.
33. —. Datendurchsatz. [Online] 15. 02 2016. [Zitat vom: 24. 02 2014.] <https://de.wikipedia.org/wiki/Datendurchsatz>.
34. **Vector Informatik GmbH.** *Schulung CANoe*. Stuttgart : s.n., 2015.
35. *Analysis of Real-Time Control Systems with Time Delays.* **Nilsson, Johan und Bernhardsson, Bo.** Kobe, Japan : IEEE, 1996. 35th IEEE Conference on Decision and Control. S. 3173-3178. <http://www.control.lth.se/documents/1996/nilj+ber96.pdf>. 10.1109/CDC.1996.574247.
36. **Mansour, Yishay und Patt-Shamir, Boaz.** Jitter Control in QoS Networks. [Hrsg.] H. Zhang. *IEEE/ACM Transactions on Networking*. 1998, Bd. 9, 4, S. 492-502.
37. **Colombo, Armando Walter, et al., et al.** Distributed Control Network (DCN) - Ein Realisierungsansatz für ein SoA-basiertes verteiltes Regelungssystem. [Online] 03. 08 2012. [Zitat vom: 24. 02 2016.] [http://www.hs-emden-leer.de/fileadmin/user\\_upload/Forschung\\_Transfer/Projekte/IMCAESOP/41\\_AT2012\\_Distributed\\_Control\\_Network\\_DCN\\_20120117\\_Bekel\\_Wehs\\_Vortrag.pdf](http://www.hs-emden-leer.de/fileadmin/user_upload/Forschung_Transfer/Projekte/IMCAESOP/41_AT2012_Distributed_Control_Network_DCN_20120117_Bekel_Wehs_Vortrag.pdf).
38. **Vector Informatik GmbH.** CANoe - Steuergeräte-Entwicklung und -Test. [Online] [Zitat vom: 30. 03 2016.] [http://vector.com/vi\\_canoe\\_de.html](http://vector.com/vi_canoe_de.html).
39. **Wikipedia.** Component Object Model. [Online] 11. 01 2016. [Zitat vom: 30. 03 2016.] [https://de.wikipedia.org/wiki/Component\\_Object\\_Model](https://de.wikipedia.org/wiki/Component_Object_Model).
40. **Vector Informatik GmbH.** CANalyzer/CANoe as a COM Server. [Online] 26. 03 2013. [Zitat vom: 30. 03 2016.] [http://vector.com/portal/medien/cmc/application\\_notes/AN-AND-1-117\\_CANoe\\_CANalyzer\\_as\\_a\\_COM\\_Server.pdf](http://vector.com/portal/medien/cmc/application_notes/AN-AND-1-117_CANoe_CANalyzer_as_a_COM_Server.pdf).
41. —. CANoe FDX Protocol. 2013. 1.4.

- 
42. —. CANoe Hilfe - Logging Block/Replay Block - Ethernet. 2016.
43. —. Lernmodul AUTOSAR. [Online] 2016. [Zitat vom: 04. 03 2016.] [https://elearning.vector.com/vl\\_autosar\\_introduction\\_de.html](https://elearning.vector.com/vl_autosar_introduction_de.html).
44. **Wikipedia**. AUTOSAR. [Online] 18. 02 2016. [Zitat vom: 04. 03 2016.] <https://de.wikipedia.org/wiki/AUTOSAR>.
45. *Kopplung einer AUTOSAR eventbasierten Simulation mit der OMNET++ Simulation von Automotive-Netzwerken*. **Schrade, Sebastian**. 2015.
46. **Hillenbrand, M., et al., et al.** Ontology-Based Consideration of Electric/Electronic Architectures of Vehicles. *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme*. 2011, Bd. VII.
47. **Vector Informatik GmbH**. Broschüre - PREEvision - Modellbasierte E/E-Entwicklung. [Online] 03 2016. [Zitat vom: 27. 06 2016.]
48. **Mulici, Besnik**. Integration von CANoe in die OMNeT++ System Level Simulation für automobile Netzwerke. [Online] 01. 05 2016. [Zitat vom: 01. 05 2016.] <https://core.informatik.haw-hamburg.de/bib/eigene/m-icosl-16.pdf>.
49. **Jenkins, Bob**. 4-byte Integer Hashing. [Online] 22. 08 2013. [Zitat vom: 04. 04 2016.] <http://burtleburtle.net/bob/hash/integer.html>.
50. **Coyer, Chris**. CSS-Tricks - What Makes For a Semantic Class Name? [Online] 04. 08 2011. [Zitat vom: 18. 04 2016.] <https://css-tricks.com/semantic-class-names/>.
51. **OMNeT++**. Simulation Manual. [Online] 14. 04 2016. [Zitat vom: 30. 07 2016.] <https://omnetpp.org/doc/omnetpp/manual/>.
52. *Beware of the Hidden! How Cross-traffic Affects Quality Assurances of Competing Real-time Ethernet Standards for In-Car Communication*. **Steinbach, Till, et al., et al.** Clearwater Beach, FL : IEEE, 2015. S. 1-9. <https://core.informatik.haw-hamburg.de/bib/eigene/slksh-bhcan-15.pdf>. 10.1109/LCN.2015.7366277.
53. **Beck, Kent**. *Test Driven Development. By Example*. s.l. : Addison Wesley, 2002. 978-0321146533.
54. **Wikipedia**. Softwaretest. [Online] 13. 04 2016. [Zitat vom: 06. 05 2016.] <https://de.wikipedia.org/wiki/Softwaretest>.
55. **Müller, Sebastian**. Simulationsmodell eines Multi-Bus Realtime Ethernet Gateways. [Online] 06. 10 2014. [Zitat vom: 03. 05 2016.] <https://core.informatik.haw-hamburg.de/bib/eigene/m-smreg-14.pdf>.

# Anhang 1: Test-Prozeduren

## Dateien laden

Auszuführen bei beiden Konzept-Oberflächen:

### NED-Files

Überprüfe nach jedem Schritt die Anzeige des Netzwerks auf Fehler. Lösche nach jedem Schritt die Dateien mit dem Button „Clear“.

- Öffne small
- Öffne gateway
- Öffne BMW
- Öffne KM\_RecBar\_Car
- Öffne missingDependencies
- Öffne NED zweimal nacheinander (ohne Clear)
- Öffne zwei verschiedene NEDs nacheinander (ohne Clear)
- Öffne gleiches NED nachdem Busse aktiviert worden waren → Busse bleiben aktiviert
- Öffne anderes NED nachdem Busse aktiviert worden waren → Busse verschwinden

### BLF-Dateien

- Irgendwelche BLF-Dateien hinzufügen

### BusMapper

Überprüfe im Netzwerk, ob die definierten Busse aktiviert werden können

- Füge Busmapper hinzu, wenn NED geladen
- Füge Busmapper hinzu, bevor NED geladen
- Füge Busmapper ein zweites Mal hinzu
- Lade fehlerhaftes BusMapper von missingDependencies

## Requirements

- Lade Requirements von small
- Lade Requirements von gateway
- Lade Requirements von BMW.
- Lade fehlerhafte Requirements von missingDependencies

## Oberfläche

- Doppelklick auf Felder öffnen Datei-Öffnen-Dialoge
  - Filter ist auf richtiges Dateiformat eingestellt
  - Dort ausgewählte Dateien werden auch geöffnet
  - Dateien können mit DEL-Taste entfernt werden
- Netzwerk verschieben
  - Busse können geklickt werden, auch wenn Netzwerk verschoben
  - Probes werden an richtiger Stelle angezeigt und mitverschoben
- Ausführung starten

## COM-Schnittstelle

- CANoe ist geschlossen → Compagnon startet CANoe
- CANoe hat arbeitet in einer nicht gespeicherten Konfiguration → Fehlermeldung
- CANoe wird geschlossen → Wird von Compagnon wieder geöffnet
- Wenn CANoe fertig ist, wird das in Compagnon angezeigt
- Die Icons bei Results sind entsprechend gesetzt

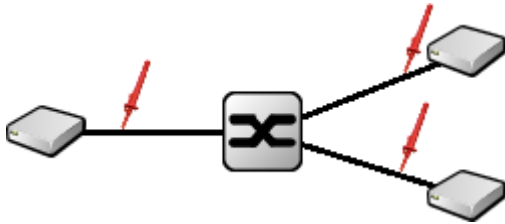
## 1. Konzept: Ausführung starten

Aufgaben bei jedem Netzwerk:

- Merged.blf überprüfen
- Compiled.vsysvar überprüfen
- CAPL-Dateien überprüfen

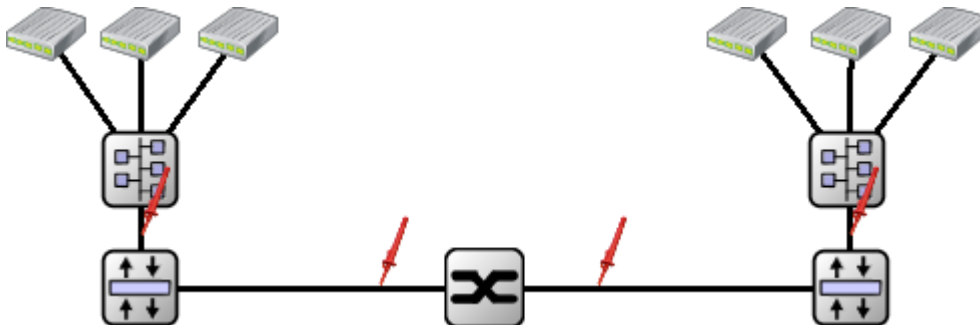


## Netzwerk „small“



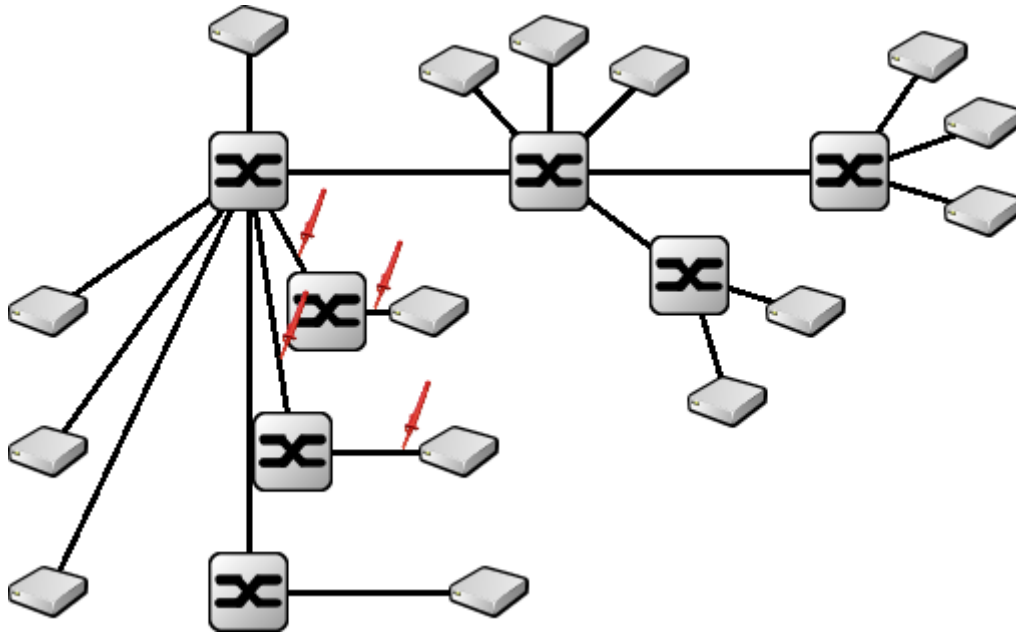
- Erwartet: merged.blf mit 3 Eth-Kanälen, auch in CANoe
- Erwartet: generiertes vSysvar mit 3 Eth-Bussen und 10 IDs
- Erwartet: generierte CAPL-Dateien:
  - busses\_generated: Enthält 3 Busse
  - neighbors\_generated: Enthält 3 Busse
  - requirements\_generated: Enthält 20 Grenzwerte, inkl. Defaults
  - source2target\_generated: Enthält überwachte IDs

## Netzwerk „gateway“



- Erwartet: merged.blf mit 2 Eth- und 2 CAN-Kanälen, auch in CANoe
- Erwartet: generiertes vSysvar mit 2 Eth- und 2 CAN-Bussen und 11 IDs
- Erwartet: generierte CAPL-Dateien:
  - busses\_generated: Enthält 4 Busse
  - neighbors\_generated: Enthält 4 Busse
  - requirements\_generated: Enthält 20 Grenzwerte (inkl. Defaults)
  - source2target\_generated: Enthält überwachte IDs

## Netzwerk „BMW“



- Erwartet: merged.blf mit 4 Eth-Kanälen, auch in CANoe
- Erwartet: generiertes vSysvar mit 4 Eth -Bussen und 6 IDs
- Erwartet: generierte CAPL-Dateien:
  - busses\_generated: Enthält 4Busse
  - neighbors\_generated: Enthält 4Busse
  - requirements\_generated: Enthält 8 Grenzwerte, inkl. Defaults
  - source2target\_generated: Enthält überwachte IDs

## 2. Konzept: Ausführung

### Netzwerk „small“

- Erwartet: Aufzeichnung in OMNeT++ erfolgreich
- Erwartet: merged.blf mit 3 Eth-Kanälen, auch vectors.blf CANoe
- Erwartet: vSysvar-Datei mit Metriken
- Erwartet: generierte CAPL-Datei mit on-sysvar-Handlern

### Netzwerk „gateway“

- Erwartet: Aufzeichnung in OMNeT++ erfolgreich
- Erwartet: merged.blf mit 2 Eth- und 2 CAN-Kanälen, auch vectors.blf in CANoe
- Erwartet: vSysvar-Datei mit Metriken
- Erwartet: generierte CAPL-Datei mit on-sysvar-Handlern

### Netzwerk „BMW“

- Erwartet: Aufzeichnung in OMNeT++ erfolgreich
- Erwartet: merged.blf mit 4 Eth-Kanälen, auch vector.blf in CANoe (s.o.)
- Erwartet: vSysvar-Datei mit Metriken
- Erwartet: generierte CAPL-Datei mit on-sysvar-Handlern

Erfolgreich durchgeführt am 31.07.2016

# Anhang 2: Performance-Ergebnisse

Konfiguration (s. Kapitel 5.3)

1. Simulation ohne Statistiken, `Vector-Manager` und `BlfRecorder`
2. Simulation ohne Statistiken und `Vector-Manager` aber mit `BlfRecorder`  
→ Rechenzeit des `BlfRecorders` durch  $(Conf\ 2) - (Conf\ 1)$
3. Simulation mit Statistiken, aber ohne `Vector-Manager` und `BlfRecorder`  
→ Rechenzeit der Statistiken durch  $(Conf\ 3) - (Conf\ 1)$
4. Simulation mit Statistiken, `Vector-Manager` und `BlfRecorder`  
→ Rechenzeit des `Vector-Managers` durch  
 $(Conf\ 4) - (Conf\ 3) - ((Conf\ 2) - (Conf\ 1))$

## Anhang 2.1: Ausführungsdauer der Simulation von „small“

Ausgeführte Befehle:

```
time ../../src/RecordedNetworks.exe -u Cmdenv -c With_Crosstraffic -l
CANoe4OMNeT --cmdenv-express-mode=true omnetpp.conf1.ini > /dev/null
time ../../src/RecordedNetworks.exe -u Cmdenv -c With_Crosstraffic -l
CANoe4OMNeT --cmdenv-express-mode=true omnetpp.conf2.ini > /dev/null
time ../../src/RecordedNetworks.exe -u Cmdenv -c With_Crosstraffic -l
CANoe4OMNeT --cmdenv-express-mode=true omnetpp.conf3.ini > /dev/null
time ../../src/RecordedNetworks.exe -u Cmdenv -c With_Crosstraffic -l
CANoe4OMNeT --cmdenv-express-mode=true omnetpp.conf4.ini > /dev/null
```

	Conf 1	Conf 2	Conf 3	Conf 4
	284s	388s	9169s	9333s
	284s	392s	9114s	9342s
	286s	392s	9128s	9345s
	293s	388s	9170s	9338s
	288s	390s	9182s	9327s
	284s	390s	9139s	9316s
	283s	389s	9150s	9304s
	284s	390s	9136s	9356s
	287s	389s	9139s	9370s
	286s	393s	9126s	9305s
	284s	390s	9144s	9303s
	282s	388s	9107s	9328s

---

	284s	391s	9115s	9321s
	285s	390s	9123s	9337s
	285s	389s	9188s	9374s
	285s	392s	9119s	9328s
	287s	391s	9176s	9374s
	286s	391s	9150s	9354s
	285s	390s	9159s	9382s
	286s	389s	9131s	9348s
	284s	391s	9118s	9314s
	288s	389s	9124s	9303s
	285s	390s	9165s	9361s
	282s	389s	9140s	9332s
	285s	388s	9136s	9302s
	285s	389s	9162s	9374s
	285s	389s	9184s	9399s
	286s	389s	9153s	9344s
	286s	389s	9149s	9378s
	287s	390s	9163s	9347s
	284s	389s	9129s	9341s
	285s	390s	9177s	9317s
	285s	391s	9157s	9369s
	287s	391s	9125s	9304s
	285s	388s	9175s	9368s
	288s	391s	9152s	9351s
<b>Median</b>	285s	390s	9147s	9342s
<b>Mittelwert</b>	285s	390s	9147s	9341s

## Anhang 2.2: Ausführungsdauer der Simulation von „BMW“

Ausgeführte Befehle:

```
time ../../src/RecordedNetworks.exe -u Cmdenv -l CANoe4OMNet --cmdenv-
express-mode=true omnetpp.base.ini omnetpp.conf1.ini > /dev/null
time ../../src/RecordedNetworks.exe -u Cmdenv -l CANoe4OMNet --cmdenv-
express-mode=true omnetpp.base.ini omnetpp.conf2.ini > /dev/null
time ../../src/RecordedNetworks.exe -u Cmdenv -l CANoe4OMNet --cmdenv-
express-mode=true omnetpp.base.ini omnetpp.conf3.ini > /dev/null
time ../../src/RecordedNetworks.exe -u Cmdenv -l CANoe4OMNet --cmdenv-
express-mode=true omnetpp.base.ini omnetpp.conf4.ini > /dev/null
```

	<b>Conf 1</b>	<b>Conf 2</b>	<b>Conf 3</b>	<b>Conf 4</b>
	258s	365s	5411s	5537s
	260s	367s	5408s	5548s
	254s	361s	5398s	5577s
	254s	358s	5391s	5558s
	255s	356s	5387s	5559s
	253s	359s	5390s	5558s
	253s	357s	5365s	5560s
	256s	351s	5400s	5549s
	257s	358s	5386s	5574s
	253s	362s	5386s	5552s
	252s	358s	5365s	5536s
	259s	357s	5375s	5566s
	257s	355s	5374s	5541s
	256s	361s	5409s	5561s
	260s	356s	5402s	5571s
	256s	360s	5412s	5565s
	257s	361s	5402s	5561s
	253s	352s	5399s	5548s
	255s	363s	5368s	5579s
	260s	359s	5410s	5524s
	253s	356s	5406s	5549s
	263s	359s	5377s	5546s
	253s	362s	5398s	5595s
	251s	359s	5421s	5586s
<b>Median</b>	256s	359s	5398s	5559s
<b>Mittelwert</b>	256s	359s	5393s	5558s

### Anhang 2.3: Ausführungsdauer der Analyse von „small“

Die Ergebnisse der 200 Durchführungen werden in zwei Spalten abgedruckt, um Platz zu sparen. Bei Konfiguration 4 ist es möglich, keine Nachrichteneignisse in CANoe einzuspeisen. Deshalb wurden hier beide Möglichkeiten durchgespielt.

Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace	Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace	Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace
458s	48s	69s	431s	48s	77s	437s	48s	75s
392s	48s	70s	431s	47s	77s	440s	47s	75s
389s	49s	78s	432s	47s	78s	438s	48s	75s
391s	48s	73s	432s	47s	78s	436s	47s	75s
391s	48s	75s	431s	48s	77s	436s	48s	75s
396s	48s	73s	432s	48s	77s	437s	47s	75s
394s	48s	69s	431s	48s	76s	438s	48s	76s
396s	47s	74s	432s	48s	78s	438s	47s	75s
397s	48s	70s	438s	48s	77s	438s	48s	75s
395s	48s	72s	434s	46s	78s	439s	47s	76s
428s	46s	70s	433s	49s	78s	439s	48s	75s
420s	48s	71s	432s	48s	77s	437s	47s	75s
421s	48s	71s	434s	48s	77s	437s	48s	75s
420s	47s	70s	433s	47s	77s	438s	48s	79s
423s	47s	70s	433s	48s	78s	438s	48s	76s
421s	47s	70s	433s	47s	78s	437s	47s	75s
422s	47s	71s	434s	48s	77s	438s	47s	75s
420s	48s	70s	435s	46s	79s	436s	47s	75s
421s	47s	73s	433s	47s	78s	439s	48s	75s
421s	46s	71s	436s	47s	78s	437s	48s	77s
421s	48s	73s	434s	47s	78s	437s	48s	75s
421s	48s	71s	435s	47s	78s	437s	48s	77s
421s	47s	71s	433s	47s	78s	436s	47s	75s
422s	47s	71s	434s	47s	78s	437s	47s	76s
425s	47s	71s	434s	48s	78s	438s	48s	76s
425s	49s	71s	435s	47s	78s	437s	48s	76s
421s	48s	71s	433s	49s	78s	438s	47s	76s
422s	48s	72s	434s	46s	78s	439s	47s	77s
422s	47s	71s	434s	49s	78s	438s	48s	97s
421s	47s	70s	435s	47s	78s	444s	48s	77s
423s	47s	70s	435s	48s	78s	438s	47s	77s
429s	47s	71s	434s	47s	75s	438s	47s	77s
424s	48s	71s	434s	48s	75s	442s	47s	77s

Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace	Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace	Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace
427s	48s	70s	435s	47s	73s	440s	48s	76s
426s	48s	71s	434s	48s	73s	439s	48s	77s
425s	47s	72s	435s	47s	74s	440s	47s	76s
424s	48s	71s	434s	47s	74s	440s	48s	77s
425s	48s	71s	436s	47s	74s	440s	48s	77s
427s	48s	71s	436s	49s	75s	440s	46s	77s
426s	47s	71s	439s	47s	74s	439s	48s	77s
425s	48s	71s	436s	47s	74s	441s	47s	77s
425s	47s	72s	436s	47s	74s	440s	48s	77s
425s	48s	71s	436s	48s	74s	440s	47s	78s
425s	48s	71s	437s	47s	74s	440s	48s	77s
426s	48s	71s	436s	47s	74s	441s	48s	77s
426s	47s	72s	440s	47s	74s	441s	48s	77s
427s	48s	75s	436s	48s	75s	441s	48s	78s
427s	46s	78s	438s	48s	74s	441s	47s	77s
427s	48s	75s	438s	48s	74s	448s	47s	77s
428s	48s	77s	437s	48s	74s	441s	48s	77s
427s	48s	76s	436s	48s	74s	441s	48s	77s
427s	48s	76s	443s	48s	74s	442s	48s	78s
432s	47s	76s	435s	47s	75s	440s	47s	77s
429s	46s	77s	436s	48s	75s	441s	48s	78s
429s	48s	77s	436s	48s	74s	442s	48s	77s
427s	46s	76s	438s	46s	74s	441s	49s	77s
430s	48s	77s	438s	47s	75s	442s	48s	77s
430s	47s	76s	437s	47s	75s	441s	48s	77s
430s	47s	78s	436s	48s	75s	442s	48s	77s
431s	48s	77s	438s	47s	75s	443s	47s	78s
431s	48s	77s	436s	48s	75s	442s	48s	77s
433s	48s	78s	437s	47s	75s	443s	49s	80s
431s	48s	78s	437s	48s	75s	442s	47s	77s
431s	47s	77s	436s	49s	97s	442s	48s	78s
432s	48s	78s	437s	47s	75s	444s	48s	77s
431s	47s	77s	436s	47s	76s	444s	48s	77s
431s	47s	77s	440s	46s	75s			
					<b>Median</b>	435s	48s	76s
					<b>Mittelwert</b>	432s	48s	75s



## Anhang 2.4: Ausführungsdauer der Analyse von „BMW“

Die Ergebnisse der 200 Durchführungen werden in zwei Spalten abgedruckt, um Platz zu sparen. Bei Konfiguration 4 ist es möglich, keine Nachrichteneignisse in CANoe einzuspeisen. Deshalb wurden hier beide Möglichkeiten durchgespielt.

Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace	Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace	Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace
165s	48s	71s	165s	47s	77s	174s	48s	75s
163s	50s	70s	164s	47s	77s	173s	47s	73s
163s	48s	74s	163s	48s	77s	173s	47s	76s
165s	47s	78s	164s	47s	76s	174s	48s	77s
164s	47s	75s	164s	47s	78s	173s	47s	78s
164s	47s	78s	164s	47s	77s	173s	47s	76s
163s	47s	76s	164s	47s	77s	173s	47s	72s
164s	48s	76s	164s	48s	77s	173s	47s	74s
164s	47s	75s	164s	47s	78s	173s	47s	77s
164s	47s	79s	164s	48s	77s	173s	47s	76s
164s	47s	78s	164s	47s	77s	173s	47s	76s
164s	48s	77s	164s	47s	77s	172s	47s	76s
164s	47s	76s	164s	47s	75s	172s	47s	77s
163s	47s	78s	164s	47s	75s	173s	47s	77s
163s	47s	77s	164s	48s	78s	173s	48s	76s
163s	48s	78s	173s	47s	73s	173s	47s	77s
163s	48s	78s	164s	47s	77s	173s	47s	77s
164s	48s	78s	164s	47s	77s	173s	47s	77s
164s	47s	77s	165s	47s	77s	172s	47s	77s
163s	47s	76s	165s	47s	75s	173s	47s	77s
164s	47s	76s	164s	47s	77s	180s	47s	77s
163s	48s	75s	164s	47s	78s	179s	47s	76s
164s	47s	71s	165s	47s	73s	173s	47s	77s
165s	47s	77s	164s	47s	76s	173s	47s	77s
164s	48s	76s	164s	47s	78s	173s	47s	77s
164s	47s	76s	164s	47s	76s	173s	47s	77s
164s	47s	76s	165s	47s	77s	173s	47s	76s
164s	48s	76s	163s	47s	77s	173s	47s	77s
163s	47s	77s	164s	47s	78s	174s	47s	77s
164s	47s	73s	164s	47s	77s	173s	47s	77s
164s	47s	75s	164s	47s	77s	173s	47s	77s
164s	47s	78s	164s	47s	74s	172s	47s	77s
164s	47s	78s	164s	47s	76s	173s	47s	77s

Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace	Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace	Conf 2	Conf 4 ohne Trace	Conf 4 mit Trace
164s	47s	77s	165s	48s	73s	172s	47s	78s
164s	47s	76s	165s	47s	77s	173s	47s	77s
165s	47s	77s	159s	47s	78s	173s	47s	77s
164s	47s	76s	165s	47s	77s	173s	47s	77s
164s	47s	76s	164s	47s	77s	173s	48s	74s
164s	47s	76s	164s	47s	78s	173s	47s	71s
164s	47s	78s	164s	47s	77s	173s	47s	77s
165s	47s	77s	164s	47s	71s	173s	47s	74s
165s	47s	76s	172s	47s	74s	172s	47s	78s
164s	47s	72s	172s	47s	77s	168s	47s	75s
164s	47s	77s	173s	47s	77s	173s	47s	76s
164s	47s	74s	173s	47s	76s	173s	47s	76s
164s	47s	75s	168s	47s	76s	173s	47s	77s
164s	47s	77s	174s	47s	65s	180s	47s	76s
164s	47s	77s	173s	47s	75s	180s	47s	78s
165s	47s	75s	173s	47s	76s	172s	47s	77s
164s	48s	77s	174s	47s	77s	172s	48s	76s
164s	47s	77s	174s	47s	76s	174s	47s	75s
164s	47s	78s	173s	47s	78s	173s	47s	77s
163s	47s	73s	174s	48s	77s	180s	47s	76s
164s	47s	77s	172s	47s	77s	172s	47s	76s
164s	47s	74s	173s	47s	76s	173s	47s	77s
158s	47s	77s	173s	47s	76s	176s	47s	76s
164s	47s	76s	172s	47s	73s	176s	47s	77s
164s	47s	76s	173s	47s	76s	178s	47s	73s
164s	47s	77s	173s	47s	76s	176s	47s	77s
164s	47s	75s	174s	47s	77s	177s	47s	77s
164s	47s	71s	173s	47s	77s	176s	47s	75s
164s	47s	76s	172s	47s	77s	175s	47s	77s
164s	47s	77s	180s	47s	76s	173s	47s	72s
164s	47s	78s	168s	47s	77s	172s	47s	78s
164s	47s	77s	174s	47s	72s	176s	47s	77s
163s	47s	77s	168s	47s	73s	180s	47s	77s
165s	47s	77s	173s	47s	77s			
					<b>Median</b>	165s	47s	77s
					<b>Mittelwert</b>	168s	47s	76s

## Anhang 3: Validierung „gateway“

### Weg einer Nachricht mit CAN ID 1 in Netzwerk gateway

0,001000	sourceApp[0]	bufferOut
0,001001	bufferOut	canNodePort
0,001001	ecu1	canbus1
0,001223	canbus1	gateway1
0,021556	gateway1	switch0
0,021602	switch0	gateway2
0,021641	gateway2	canbus2
0,021863	canbus2	ecu4
0,022196	canNodePort	bufferIn
0,022196	bufferIn	sinkApp[1]

#### Soll:

Ende-zu-Ende-Latenz:	0,021196	0,000000
GW-zu-GW-Latenz:	0,020418	0,000000

#### VEC-Datei:

Ende-zu-Ende-Latenz:	0,021196	0,000000
GW-zu-GW-Latenz:	-	

#### 1. Konzept

Ende-zu-Ende-Latenz:	0,021196	0,000000
GW-zu-GW-Latenz:	-	

#### 2. Konzept

Ende-zu-Ende-Latenz:	0,021195	-0,000001
GW-zu-GW-Latenz:	0,020417	-0,000001

# Danksagungen

Ich möchte mich an dieser Stelle bei allen, die mir bei der Erstellung dieser Abschlussarbeit geholfen und mich begleitet haben, bedanken.

Zuerst danke ich meinen beiden Prüfern von der HAW Hamburg, Prof. Dr.-Ing. Li und Prof. Dr. Korf, welche mich mit großer Geduld, guten Ratschlägen und den richtigen Fragen unterstützten.

Weiterhin danke ich der Vector Informatik GmbH und besonders Jörn Haase, die mir nun für mehrere Jahre eine sehr angenehme berufliche Heimat boten und mir auch bei dieser Arbeit durch ihre Kontakte zur Automobilindustrie sehr hilfreich waren.

Am meisten danke ich meiner Frau Laura Wendeborn und meiner Tochter Maja Wendeborn, die mir in allen Phasen dieser Arbeit mit viel Geduld, aufmunternden Worten und tatkräftiger Hilfe zur Seite standen, sowie meiner Tochter Klara Wendeborn, die es hingegen oft genug schaffte, mich mit ihren kleinen blauen Augen vom Arbeiten abzuhalten.

# Versicherung über Selbstständigkeit

*Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.*

Hamburg, den \_\_\_\_\_