



FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
TECHNISCHE FAKULTÄT

# Lehrstuhl für Informatik 7

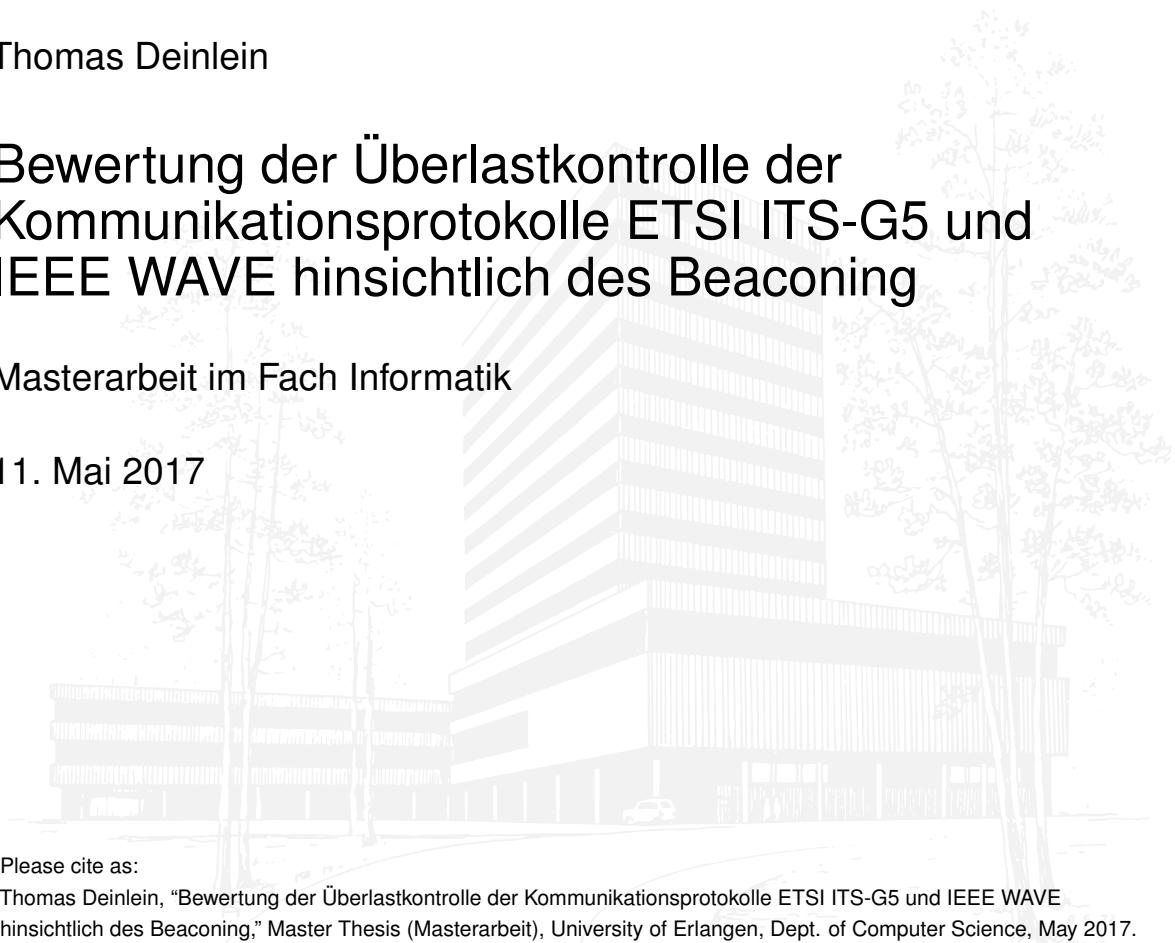
## Rechnernetze und Kommunikationssysteme

Thomas Deinlein

# Bewertung der Überlastkontrolle der Kommunikationsprotokolle ETSI ITS-G5 und IEEE WAVE hinsichtlich des Beacons

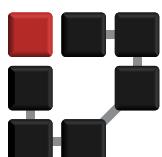
Masterarbeit im Fach Informatik

11. Mai 2017



Please cite as:

Thomas Deinlein, "Bewertung der Überlastkontrolle der Kommunikationsprotokolle ETSI ITS-G5 und IEEE WAVE hinsichtlich des Beacons," Master Thesis (Masterarbeit), University of Erlangen, Dept. of Computer Science, May 2017.



Friedrich-Alexander-Universität Erlangen-Nürnberg  
Department Informatik  
Rechnernetze und Kommunikationssysteme  
Martensstr. 3 · 91058 Erlangen · Germany  
<http://www7.cs.fau.de/>

# **Bewertung der Überlastkontrolle der Kommunikationsprotokolle ETSI ITS-G5 und IEEE WAVE hinsichtlich des Beaconing**

Masterarbeit im Fach Informatik

vorgelegt von

**Thomas Deinlein**

geb. am 17. Mai 1983  
in Kemnath

angefertigt am

**Lehrstuhl für Informatik 7  
Rechnernetze und Kommunikationssysteme**

**Department Informatik  
Friedrich-Alexander-Universität Erlangen-Nürnberg**

Betreuer: **Dr.-Ing. Anatoli Djanatliev  
Christina Stadler (Audi AG)**

Betreuer Hochschullehrer: **Prof. Dr.-Ing. Reinhard German**

Abgabe der Arbeit: **11. Mai 2017**

## Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.

Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

## Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.

I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Thomas Deinlein)

Erlangen, 11. Mai 2017

---

## Abstract

---

The Car2X communication by WLAN is described in communication protocols. These protocols are defined by standardization committees depending on the region. The standards ETSI ITS-G5 for Europe and IEEE WAVE for the USA are the focus of this master thesis. A basic functionality of the Car2X communication is the frequently sending of vehicle data to all surrounding vehicles which are in the reach of the transmitter. This function is called Beaconing. Single-Hop-Beaconing means that received Beacons are not forwarded. Multi-Hop-Beaconing allows forwarding of received Beacons optionally. Since Beacons are broadcasted with a frequency of up to 10 Hz, the communication channel can be congested in scenarios with high vehicle density. Therefore, the efficiency of the Beaconing depends on congestion control mechanisms which should reduce the channel load and prevent the channel of overloading, respectively. Both standards include an own definition of the beaconing and a congestion control mechanism.

The European standard broadcasts beacons with the message format Cooperative Awareness Message (CAM). Its decentralized congestion control (DCC) is distributed across the whole protocol stack. The US-American standard uses the format Basic Safety Message (BSM) to broadcast Beacons. The generating process of BSMs is combined with an individual congestion control mechanism.

By implementing both mechanisms within the simulation environment Omnet++ with the framework VEINS, different Single-Hop scenarios were looked into and an achievement assessment was provided concerning the congestion control mechanisms and the Beaconing. On the one hand, the results show clearly that on the part of WAVE the congestion control mechanism has to be seen critically, since the channel load is not adequately reduced or prevented. On the other side, WAVE guarantees a high broadcasting rate and a high number of received Beacons. On the part of ETSI ITS-G5 the contrary results stand out. The DCC is performing very well, however, it leads to a lower number of sent and transferred Beacons.

---

## Kurzfassung

---

Die Car2X-Kommunikation per WLAN ist in Kommunikationsprotokollen beschrieben. Diese Protokolle werden für verschiedene Regionen der Welt von dort ansässigen Standardisierungsgremien definiert. Im Fokus dieser Masterarbeit stehen die beiden Standards ETSI ITS-G5 für Europa und IEEE WAVE für die USA. Eine Grundfunktionalität der Car2X-Kommunikation ist das regelmäßige Senden der eigenen Position und weiterer Daten des Fahrzeugs an alle umliegenden Fahrzeuge, die sich in der Reichweite des Senders befinden. Diese Funktion wird Beaconing genannt. Zu unterscheiden sind hierbei Single-Hop- (keine Weiterleitung empfangener Beacons) und Multi-Hop-Beaconing (Weiterleitung empfangener Beacons optional).

Da Beacons mit einer Frequenz von bis zu 10Hz versendet werden, kann es in Situationen mit vielen Fahrzeugen in Funkreichweite zur Überlastung des Kommunikationskanals führen. Die Leistungsfähigkeit des Beaconing steht demnach in unmittelbarem Zusammenhang mit sogenannten Überlastmechanismen, die eine Überlast reduzieren bzw. von vornherein vermeiden sollen. Die beiden betrachteten Standards enthalten jeweils eine Definition des Beaconing und einer Überlastkontrolle. Für den europäischen Standard werden Beacons in Form des Nachrichtenformats Cooperative Awareness Message (CAM) versendet. Die entsprechende Überlastkontrolle, Decentralized Congestion Control (DCC), ist über den gesamten Protokollstack verteilt. Der US-amerikanische Standard verwendet das Nachrichtenformat Basic Safety Message (BSM) zum Versenden von Beacons, worin in dem Generierungsprozess eine Überlastkontrolle integriert ist.

Durch Implementierung beider Mechanismen innerhalb der Simulationsumgebung Omnet++ mit dem Framework VEINS wurden verschiedene Single-Hop Szenarien betrachtet und eine Leistungsbewertung hinsichtlich der Überlastkontrolle und des Beaconing erstellt. Es zeigt sich deutlich, dass auf Seiten von WAVE der Mechanismus in Bezug auf die Überlastkontrolle kritisch zu betrachten ist und die Kanallast nicht adäquat reduziert bzw. verhindert wird. Allerdings garantiert er eine hohe Senderate und eine hohe Anzahl empfangener Beacons. Auf Seiten von ETSI ITS-G5 zeichnet sich das gegenteilige Bild ab. Hier greift der Überlastmechanismus sehr gut, was allerdings zu einer geringeren Anzahl versendeter und übertragener Beacons führt.

---

# Inhaltsverzeichnis

---

<b>Abstract</b>	<b>iii</b>
<b>Kurzfassung</b>	<b>iv</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	2
1.3 Themenverwandte Arbeiten . . . . .	3
1.4 Aufbau der Arbeit . . . . .	3
<b>2 Grundlagen</b>	<b>5</b>
2.1 WAVE . . . . .	5
2.1.1 Frequenzbandallokierung und Kanalübersicht . . . . .	5
2.1.2 Überblick über den Protokollstack . . . . .	6
2.1.3 Basic Safety Message (BSM) . . . . .	8
2.1.4 Generierung und Congestion Control einer Basic Safety Message	10
2.1.4.1 Channel Busy Percentage (CBP) . . . . .	12
2.1.4.2 Packet Error Ratio (PER), Channel Quality Indicator und Vehicle Density in Range . . . . .	12
2.1.4.3 Tracking Error . . . . .	15
2.1.4.4 Transmission Probability . . . . .	15
2.1.4.5 Maximum BSM Generation Interval MaxITT . . . . .	16
2.1.4.6 Transmission Decision . . . . .	17
2.1.4.7 Schedule Transmission . . . . .	18
2.1.4.8 Radiated Power Calculation . . . . .	18
2.1.4.9 Generierung einer BSM und Festlegung der darauf- folgenden BSM Generierung . . . . .	19
2.2 ETSI ITS-G5 . . . . .	20
2.2.1 Frequenzbandallokierung und Kanalübersicht . . . . .	20
2.2.2 Überblick über den Protokollstack . . . . .	20
2.2.3 Cooperative Awareness Message (CAM) . . . . .	22

2.2.3.1	CAM Basic Service . . . . .	22
2.2.3.2	Das CAM-Format . . . . .	23
2.2.4	Decentralized Congestion Control (DCC) . . . . .	24
2.2.4.1	Architektur . . . . .	25
2.2.4.2	DCC Access . . . . .	26
2.2.5	Generierung von CAMs . . . . .	32
<b>3</b>	<b>Implementierung in VEINS</b>	<b>34</b>
3.1	Verwendete Tool-Chain . . . . .	34
3.1.1	Omnet++ . . . . .	34
3.1.2	Vehicles in Network Simulation (VEINS) . . . . .	35
3.1.3	Simulation of Urban Mobility (SUMO) . . . . .	36
3.2	Herangehensweise und Festlegungen . . . . .	36
3.3	Integration der WAVE-BSM-Generierung in VEINS . . . . .	38
3.3.1	Erweiterung der MAC-Schicht . . . . .	38
3.3.2	Erweiterung der Applikationsschicht . . . . .	38
3.3.2.1	Integrierung des Intervalls <i>vCBPMeasInt</i> . . . . .	39
3.3.2.2	Integrierung der Intervalle <i>vPERInterval</i> und <i>vPER-SubInterval</i> . . . . .	40
3.3.2.3	Integrierung des Intervalls <i>vTxRateCntrlInt</i> . . . . .	42
3.3.2.4	Generierung und Versendung einer Basic Safety Message (BSM) . . . . .	43
3.4	Integration der ETSI-CAM-Generierung inklusive DCC in VEINS . . . . .	44
3.4.1	Erweiterung der MAC-Schicht . . . . .	45
3.4.1.1	DCC mit Standardparameter . . . . .	45
3.4.1.2	DCC Profile . . . . .	48
3.4.1.3	LIMERIC . . . . .	48
3.4.2	Erweiterung der Applikationsschicht . . . . .	49
3.4.2.1	Prüfung der Änderung von dynamischen Fahreignissen . . . . .	49
3.4.2.2	Generierung und Versenden einer CAM . . . . .	50
3.5	Testen der Implementierung . . . . .	50
<b>4</b>	<b>Leistungsbewertung beider Mechanismen</b>	<b>52</b>
4.1	Erste Erkenntnisse nach der Implementierung . . . . .	52
4.2	Vorgehensweise . . . . .	53
4.3	Evaluierung anhand von Szenarien . . . . .	56
4.3.1	Szenario Autobahnkreuz . . . . .	56
4.3.1.1	Gegenüberstellung der Kennwerte . . . . .	57
4.3.1.2	Zusammenfassung der erkannten Tendenzen . . . . .	59

4.3.1.3	Erkenntnisse bei längerer Simulationszeit . . . . .	60
4.3.2	Szenario <i>Stadt</i> . . . . .	62
4.3.2.1	Gegenüberstellung der Kennwerte . . . . .	63
4.3.2.2	Zusammenfassung der erkannten Tendenzen . . . . .	64
4.3.2.3	Erkenntnisse bei längerer Simulationszeit . . . . .	65
4.3.3	Szenario <i>Stau</i> . . . . .	66
4.3.3.1	Gegenüberstellung der Kennwerte . . . . .	66
4.3.3.2	Zusammenfassung der erkannten Tendenzen . . . . .	68
4.3.3.3	Erkenntnisse bei längerer Simulationszeit . . . . .	69
4.4	Individuelle Betrachtung beider Mechanismen . . . . .	69
4.4.1	WAVE . . . . .	70
4.4.1.1	Auffälligkeiten hinsichtlich des Generierungsintervalls	70
4.4.1.2	Einfluss des <i>TxDynamics</i> -Flags . . . . .	71
4.4.2	ETSI ITS-G5 . . . . .	73
4.4.2.1	Auswirkung der DCC Transmit Queue . . . . .	73
4.4.2.2	Gegenüberstellung der ETSI-Varianten . . . . .	73
<b>5</b>	<b>Zusammenfassung</b>	<b>78</b>
<b>A</b>	<b>Übersicht aller Testfälle</b>	<b>80</b>
<b>B</b>	<b>Grafische Gegenüberstellung aller Evaluierungskennwerte</b>	<b>84</b>
<b>C</b>	<b>Inhalt des Datenträgers</b>	<b>102</b>
	<b>Abkürzungsverzeichnis</b>	<b>104</b>
	<b>Algorithmusverzeichnis</b>	<b>107</b>
	<b>Abbildungsverzeichnis</b>	<b>111</b>
	<b>Listings</b>	<b>112</b>
	<b>Tabellenverzeichnis</b>	<b>113</b>
	<b>Literaturverzeichnis</b>	<b>114</b>

---

# Kapitel 1

---

## Einleitung

---

Im Mai 2016 kam es zum ersten tödlichen Unfall eines Fahrzeuges mit sogenannter Selbstfahrrvorrichtung [1]. Die On-Board-Sensorik hatte einen LKW fälschlicherweise nicht richtig erkannt und deshalb keine Notbremsung durchgeführt. Der Hersteller des Fahrzeugs titulierte diese Fahrfunktion als "Autopilot". Auch wenn sich durch eine Untersuchung des Unfalls herausstellte, dass auch menschliches Versagen zum Unfallhergang beigetragen hatte [2], wurde deutlich, dass die Sensorik eines Fahrzeugs allein nach heutigem Stand unzureichend ist, um autonom fahren zu können. In diesem Zusammenhang stellt sich die Frage, ob der Unfall anders verlaufen wäre, wenn beide Fahrzeuge per Funk ihre Fahrzeugdaten rechtzeitig vor dem Aufprall hätten austauschen können.

### 1.1 Motivation

Dieses aktuelle Beispiel soll verdeutlichen, dass die Vernetzung von Fahrzeugen, Car-to-Car, insbesondere für Sicherheitsaspekte des Straßenverkehrs von großer Bedeutung ist. Auch wird in diesem Kontext sehr oft die Kommunikation von Fahrzeugen mit Infrastrukturkomponenten (z.B. Ampel) mit einbezogen (Car-to-X). Sogenannte Cooperative Intelligent Transport Systems (C-ITSs) sind bereits seit Ende der 1990er Jahre im Fokus der Forschung und wurden in zahlreichen Projekten praktisch getestet (siehe [3], S. 526f.). Durch die kommunizierten Daten können sozusagen die Sensoren umliegender Fahrzeuge in das eigene integriert werden, wodurch auch Bereiche aus dem nicht sichtbaren Umfeld erfasst werden können, was im Eingangs erwähnten Beispiel von großem Nutzen gewesen wäre. Damit die kommunizierten Fahrzeugdaten möglichst aktuell sind, müssen diese periodisch an alle umliegenden Fahrzeuge gesendet werden, was allgemein als Beaconing bezeichnet wird. Dieses ist für den europäischen und den nordamerikanischen Raum unterschiedlich spezifiziert. Allgemein ist beim Beaconing noch zwischen Single-Hop und Multi-Hop zu

unterscheiden. Single-hop bezeichnet die Variante, in der ein Beacon nur von seinem Erzeuger versendet wird (es erfolgt eine einmalige Übertragung). Bei Multi-Hop können Beacons über mehrere Stationen (bzw. Fahrzeuge) weitergeleitet werden. Themenschwerpunkt dieser Arbeit sind die auf WLAN 802.11p (siehe [4]) basierenden Kommunikationsprotokolle ETSI ITS-G5 (Europa) und IEEE WAVE (USA), die für Car-to-X-Kommunikation vorgesehen sind. Beide Bezeichnungen stehen für einen bestimmten Protokollstapel basierend auf WLAN 802.11p, womit Vehicular Ad Hoc Networks (VANETs) realisiert werden können. Dabei werden ausschließlich deren Single-Hop Beaconing Mechanismen betrachtet.

Auf europäischer Seite (von dem European Telecommunications Standards Institute (ETSI) spezifiziert) werden (Single-Hop) Beacons als Cooperative Awareness Messages (CAMs) in einer dynamisch berechneten Frequenz zwischen 1 Hz bis 10 Hz periodisch versendet. Das nordamerikanische Pendant hierzu sind Basic Safety Messages (BSMs), welche von der Society of Automotive Engineers (SAE Intern.) spezifiziert wurden und von WAVE verwendet werden. BSMs setzen ebenfalls eine dynamische Frequenzberechnung ein. Ein weiterer Unterschied ist die Überlastkontrolle, die beide Generierungsmechanismen beeinflussen. ETSI ITS-G5 baut auf eine Decentralized Congestion Control (Decentralized Congestion Control (DCC)), die durch verschiedene Parameter (insbesondere durch Vorgabe der Zeit, die zwischen zwei CAMs verstreichen muss) Einfluss nimmt. Bei WAVE ist eine zentrale Congestion Control nicht vorgesehen bzw. zum Zeitpunkt der Erstellung dieser Arbeit nicht spezifiziert. Hier werden verschiedene Parameter direkt bei der Generierung einer BSM berechnet und herangezogen.

## 1.2 Zielsetzung

Insbesondere bei einem sehr hohen Fahrzeugaufkommen kann das häufige Broadcasten zu einer Überlastung des Funkkanals führen. Sogenannte Congestion Control Mechanismen dienen dazu Kanallasten zu reduzieren bzw. von vornherein zu vermeiden. Im Zusammenhang mit der Beacon-Generierung werden die jeweils vorgesehenen Congestion Control Mechanismen der Protokolle betrachtet und analysiert.

Es wird aufgezeigt, in welchen Situationen und in Abhängigkeit bestimmter Parameter, wie z.B. Datengröße eines Beacons, Überlastsituationen in einer Car-to-X-Kommunikation auftreten können und wie performant beide Protokolle mit diesen hinsichtlich des Beaconing umgehen. Zur Evaluierung der Leistungsfähigkeit beider Protokolle wird das VEINS-Simulationsframework (siehe [5], [6], [7] und [8]) herangezogen. Dieses wird durch die in den jeweiligen Standards beschriebenen

Mechanismen erweitert. Anhand verschiedener Szenarien erfolgt eine Gegenüberstellung beider Protokolle und Bewertung der Leistungsfähigkeit.

### 1.3 Themenverwandte Arbeiten

Auf Seiten des europäischen Standards ETSI ITS-G5 sind schon umfassende Forschungen betrieben worden.

In [9] wurde das Beaconsing unter ETSI ITS-G5 und WAVE bereits 2013 verglichen. Allerdings wurde hier eine konstante Rate von 10 Hz für das Beaconsing verwendet, was nur in bestimmten Situationen auftreten kann. Es wurde dabei die Standardparametrisierung aus [10] herangezogen. Trotz eines oszillierenden Verhaltens der Kanallast seitens ETSI ITS-G5 bedingt durch den Zustandswechsel der DCC wurde festgestellt, dass dieser Ansatz zu einer besseren Performance als bei WAVE führt. Das Oszillieren der Kanallast in Verbindung der Generierung von CAM wurde in [11] untersucht. Dabei wurde festgestellt, dass die Kombination des Generierungintervalls einer CAM mit weiteren Parametern, wie Sendeleistung, Einfluss auf die Stabilität der DCC hat.

Das zustandsabhängige reaktive Verhalten der DCC wurde in [12] verglichen. Auch hier konnte das Oszillieren der DCC und instabiles Verhalten beobachtet werden. Weitere Untersuchungen in [13] und [14] kamen zu ähnlichen Ergebnissen. [15] stellte darüber hinaus fest, dass die DCC zu Instabilität und einer niedrigen Anzahl übertragener CAMs führen kann.

Bei WAVE wurde in [16] bereits 2011 die Notwendigkeit eines Congestion Control Mechanismus für BSMs festgestellt. Mit Einführung von [17] wurde dieser 2016 veröffentlicht.

### 1.4 Aufbau der Arbeit

Diese Arbeit gliedert sich in fünf Kapitel. Das erste Kapitel, das mit diesem Abschnitt endet, beinhaltet die Einführung in das behandelte Themengebiet, verwandte Forschungsarbeiten sowie die Beschreibung der Zielsetzung der Arbeit.

In Kapitel 2 werden Grundlagen, die für das weitere Lesen der Arbeit benötigt werden, erklärt. Zunächst erfolgt eine allgemeine Vorstellung beider Kommunikationsprotokolle, mit Fokus auf einer detaillierten Beschreibung der jeweiligen Beaconsing-Generierungs- und Congestion Control Mechanismen.

Kapitel 3 stellt die Architektur und die erfolgte Implementierung in VEINS im Detail vor. Hier wird auf die in Kapitel 2 beschriebenen Prozesse Bezug genommen und

erläutert, wie bei der Implementierung vorgegangen wurde. Mit einer Erläuterung des Testens der Implementierung endet dieses Kapitel.

Im vorletzten Kapitel 4 werden zunächst die ausgewählten Szenarien vorgestellt, die für die Bewertung der Leistungsfähigkeit herangezogen wurden. Anhand verschiedener Parameter, die während der Simulation aufgezeichnet wurden, erfolgt eine statistische Analyse beider Protokolle.

Diese Arbeit schließt mit einer Zusammenfassung der gewonnenen Erkenntnisse und gibt einen Ausblick auf mögliche Verbesserungen sowie weitere Fragestellungen für künftige Forschungsarbeiten.

---

## Kapitel 2

---

# Grundlagen

---

Dieses Kapitel stellt beide behandelten Kommunikationsprotokolle sowie deren Beaconing-Konzept und Überlastkontrollmechanismen anhand der relevanten Standards vor.

### 2.1 WAVE

Bereits 2004 wurde von IEEE der WLAN-Standard 802.11 für die Verwendung im Kontext von Car-to-X erweitert und mit 802.11p bezeichnet [4]. Ebenso wurden weitere Standards entwickelt, die zusätzliche Netzwerkschichten abdecken. Es entstanden mehrere Spezifizierungen, die alle unter die Bezeichnung IEEE 1609.x fallen (siehe [18], [19], [20], [21], [22] und [23]). Zusammenfassend werden die genannten Regelwerke als Wireless access in vehicular environments (WAVE) bezeichnet [24].

#### 2.1.1 Frequenzbandallokierung und Kanalübersicht

WAVE basiert auf Dedicated Short Range Communications (DSRC) und deren festgelegtem Frequenzband. Die Federal Communications Commission (FCC) hat einen Frequenzbereich von 5.850 GHz bis 5.925 GHz bestimmt [25]. Dieser ist in sieben Kanäle mit je 10 MHz Spektren unterteilt. Hierunter sind sechs sogenannte Service Channels (SCHs) und ein Control Channel (CCH). Auf dem CCH werden ausschließlich hochpriorisierte Nachrichten versendet (siehe [16], [24] und [26]). In Abbildung 2.1 ist diese Kanalaufteilung dargestellt. Dabei ist ersichtlich, dass die Kanäle 174 und 176 sowie 180 und 181 zu je einem 20 MHz Kanal (Nr. 175 bzw. 181) kombiniert werden können.

Eine Besonderheit bei WAVE ist, dass es erlaubt, Nachrichten auf verschiedenen Kanälen (d.h. auf dem CCH und einem der SCHs) zu versenden und zu empfangen, wenn ein Single-Channel Transceiver im Fahrzeug verbaut ist. Ein solches System

kann Nachrichten auf einem einzelnen 10 MHz Kanal senden oder empfangen, allerdings nicht simultan beides zugleich (siehe [9] und [26]). Die Alternating Access genannte Methode wechselt dazu alle 50 ms zwischen dem CCH und SCH. In diesem Falle werden BSMs auf dem CCH versendet. Während des SCH-Intervalls kann auf einen SCH umgeschaltet werden oder auf dem CCH verblieben werden. Durch das Kanalswitching reduziert sich die Bandbreite laut [9] allerdings um die Hälfte. Im Rahmen dieser Arbeit wurde diese Methode nicht betrachtet und als Grundlage ein Dual-Transceiver vorausgesetzt, welcher kontinuierlich auf dem Kanal verbleibt, auf dem die BSMs gesendet und empfangen werden, was ein periodisches Kanalswitching nicht verlangt.

Erwähnenswert in Bezug auf die Nutzung der Kanäle ist, dass alle in Abbildung 2.1 aufgeführten Kanäle US-weit rein für die Verkehrssicherheit allokiert wurden (siehe [3], S. 527).

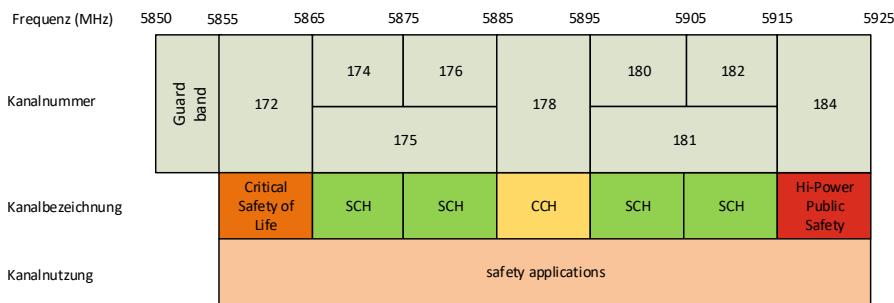


Abbildung 2.1 – Kanalallokierung bei WAVE [26]

### 2.1.2 Überblick über den Protokollstack

Der Protokollstack bei WAVE basiert auf dem WLAN-Standard 802.11. Die an VANETs angepasste Version 802.11p bildet die Grundlage für die untersten beiden Netzwerkschichten. Höhere Schichten werden durch weitere IEEE Standards beschrieben. Eine Übersicht über den gesamten Protokollstack ist in Abbildung 2.2 dargestellt.

Die für diese Arbeit besonders relevanten Komponenten wurden farblich hervorgehoben. Auf Seiten der MAC-Layer ist besonders der Kanalzugriff von Interesse. 802.11p übernimmt hierbei den in 802.11e eingeführten Quality of Service (QoS)-Mechanismus Enhanced Distributed Channel Access (EDCA) [4]. Hierbei werden die Nachrichten, die versendet werden sollen, in Prioritätskategorien (Voice, Video, BestEffort, Background) eingeordnet. Für jede der vier Kategorien und für jeden Kanaltyp (CCH bzw. SCH) wird eine Nachrichtenqueue eingerichtet. Die von höheren Schichten eintreffenden Nachrichten werden darin zunächst zwischengespeichert.

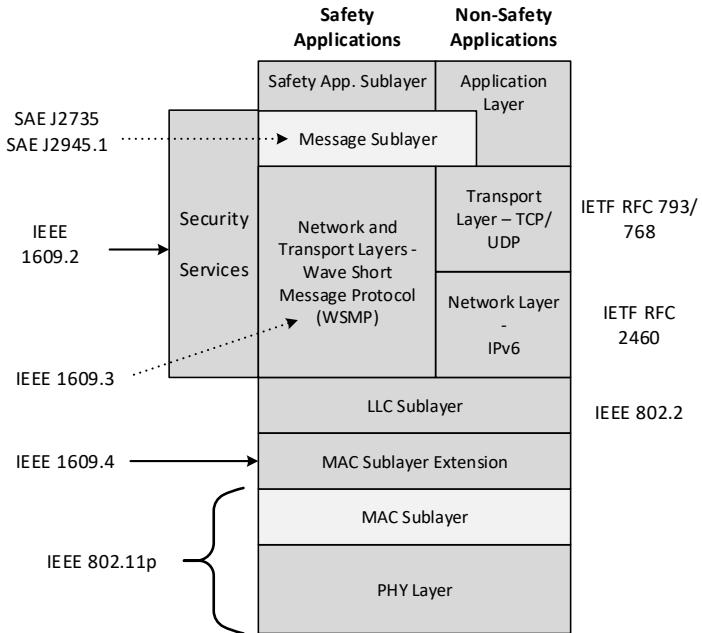


Abbildung 2.2 – Protokollstack bei WAVE [16]

Wenn der Kanal frei ist, konkurrieren die enthaltenen Nachrichten um den Kanalzugriff. Nachrichten höherer Access Category (AC) werden dabei durch Festlegung geringerer minimaler und maximaler Werte für das Contention Window (CW) sowie der Arbitration Interframe Spacing Number (AIFSN) bevorzugt. Die genaue Funktionsweise ist in [27] und [28] erläutert. Relevant für diese Arbeit sind die Parameter, die für die jeweilige Queue zu verwenden sind. In Tabelle 2.1 sind die EDCA-Parameter für WAVE aufgeführt, die für BSMs beim Versand auf dem Kanal mit Nummer 172 (SCH) zu verwenden sind [17].

Der Protokollstack in Abbildung 2.2 verdeutlicht auch klar die Trennung von Nachrichten von Safety Applications (z.B. Unfallwarnung, weitere Beispiele folgen in Tabelle 2.2) und Non-Safety Applications. Im Rahmen dieser Arbeit werden BSMs nur im Kontext einer Safety Application mit dem WAVE Short Message Protocol (WSMP) versendet. Da innerhalb der Implementierungsumgebung das Versenden von BSMs per WSMP ermöglicht wird, erfolgt keine nähere Betrachtung des WSMPs. Bei Non-Safety Applications werden auf Transport- und Netzwerkschicht die von der Internet Engineering Task Force (IETF) standardisierten Protokolle Transport Control Protocol (TCP)/User Datagram Protocol (UDP) und Internet Protocol Version 6 (IPv6) verwendet, worauf nicht näher eingegangen wird.

WAVE Nachrichten	AC	CWmin	CWmax	AIFSN
BSM mit Critical Event Flag	AC_VO	3	7	2
BSM ohne Critical Event Flag	AC_VI	15	1023	4
	AC_BE	15	1023	6
	AC_BK	15	1023	9

Tabelle 2.1 – EDCA-Parameter bei WAVE [17]

Anhand von Tabelle 2.1 ist ersichtlich, dass BSMs ein sogenanntes Critical Flag führen können, wodurch diese die höchste Priorisierung erhalten. Dadurch ist es möglich kritische Gefahrensituationen besonders zu kommunizieren. Ohne dieses Flag handelt es sich um eine gewöhnliche periodische BSM (siehe [17], Kapitel 3). Von weiterer Relevanz aus dem Protokollstack (Abbildung 2.2) ist die dort als *Message Sublayer* bezeichnete Schicht. Darunter fällt neben der allgemeinen Beschreibung einer BSM auch deren Generierung in Verbindung mit einer Congestion Control (siehe [29] und [17]). In den folgenden Unterkapiteln werden diese ausführlich erläutert, da sich hierauf ein Großteil der Implementierung in Kapitel 3 bezieht.

### 2.1.3 Basic Safety Message (BSM)

BSMs sind von der SAE Intern. spezifiziert und für eine Vielzahl von Anwendungsfällen angedacht, hauptsächlich sollen damit sicherheitsrelevante Daten des Fahrzeugzustandes periodisch übertragen und Vehicle-to-Vehicle (V2V) Safety Systems umgesetzt werden können. Grundsätzlich muss eine BSM Mindestbestandteile (Part I) beinhalten, die bei jedem Versenden vorhanden sein müssen (siehe [29], Kapitel 5.2, und [17]). Darüber hinaus können optionale Daten involviert werden (Part II). Ein Überblick über die entsprechenden Komponenten ist in Abbildung 2.3 dargestellt.

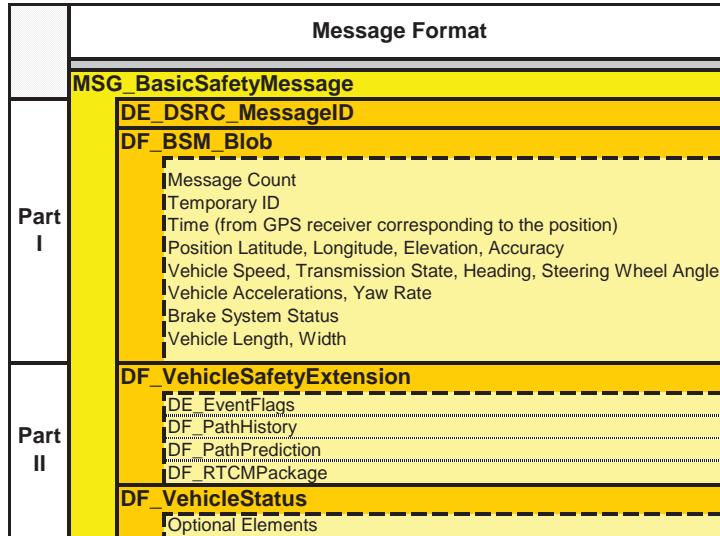


Abbildung 2.3 – BSM-Format [29] [30]

Laut [17] ist Kanal 172 für Safety Applications und für die Versendung von BSMs vorgesehen. Dies liegt daran, dass die FCC diesen (DSRC)-Kanal für sicherheitsrelevante Übertragungen festgelegt hat. In [30] und [31] werden hierzu verschiedene Aspekte betrachtet, welcher Kanal am Besten in Abhängigkeit eines Single- oder Dual-Transceivers ist. Im Rahmen dieser Arbeit wurden, wie schon in Abschnitt 2.1.1 erwähnt, Dual-Transceiver betrachtet, in dem der Kanal nicht gewechselt wird und keine anderen Nachrichten versendet oder empfangen werden.

BSMs sind das wichtigste Datenformat für bereits erwähnte V2V Safety Systems, dies soll an dieser Stelle verdeutlicht werden. Grundgedanke ist hierbei, dass das System durch Datenfusion Gefahren erkennt, den Fahrer eines Fahrzeuges daraufhin warnt und dadurch Unfälle vermieden werden können. Mit Datenfusion ist gemeint, dass die Inhalte der von anderen Fahrzeugen erhaltenen BSMs kontinuierlich ausgewertet und in die Gefahrenbewertung einbezogen werden. Es gibt verschiedene Unfallszenarien (beschrieben in [17], Kapitel 4), die hierbei berücksichtigt werden. Ebenso sind verschiedene Safety Applications vorgesehen, die in dem jeweiligen Szenario Anwendung finden. Eine Übersicht hierzu bietet Tabelle 2.2. Für die Funktionstüchtigkeit solcher Systeme ist eine hohe Übertragungs- und Empfangsrate von BSMs immens wichtig.

Gefahrensituation	Safety Application
Fahrzeug voraus stoppt	Forward Crash Warning (FCW)
Kontrollverlust ohne vorherige Fahrzeugaktion	Control Loss Warning (CLW)
Fahrzeuge biegen an Kreuzungen ohne Ampelführung ab	Intersection Movement Assist (IMA), Left Turn Assist (LTA)
Fahrzeug voraus verliert Geschwindigkeit	Emergency Electronic Brake Lights (EEBL), FCW
Fahrspurwechsel in gleicher Fahrtrichtung	Blind Spot Warning (BSW)/Lane Change Warning (LCW)

Tabelle 2.2 – Beispiele von WAVE Safety Applications [17]

### 2.1.4 Generierung und Congestion Control einer Basic Safety Message

Im folgenden Abschnitt wird die Generierung von BSMs nach [17], Kapitel 6.3, und die darin enthaltene Congestion Control genau beschrieben. Bei der Generierung sind eine Fülle von Parametern notwendig, welche periodisch berechnet werden müssen. Alle folgenden Berechnungsvorschriften basieren auf diesem Standard. Eine Übersicht dieser Parameter mit deren Abhängigkeiten ist in Abbildung 2.4 aufgezeigt. In genanntem Standard wurden die Bezeichnungen *Generating* mit dem *Generieren und Versenden einer BSM* und *Scheduling* mit dem *Festlegen des nächsten Zeitpunktes, an dem eine BSM generiert und versendet werden soll*, gleichgesetzt.

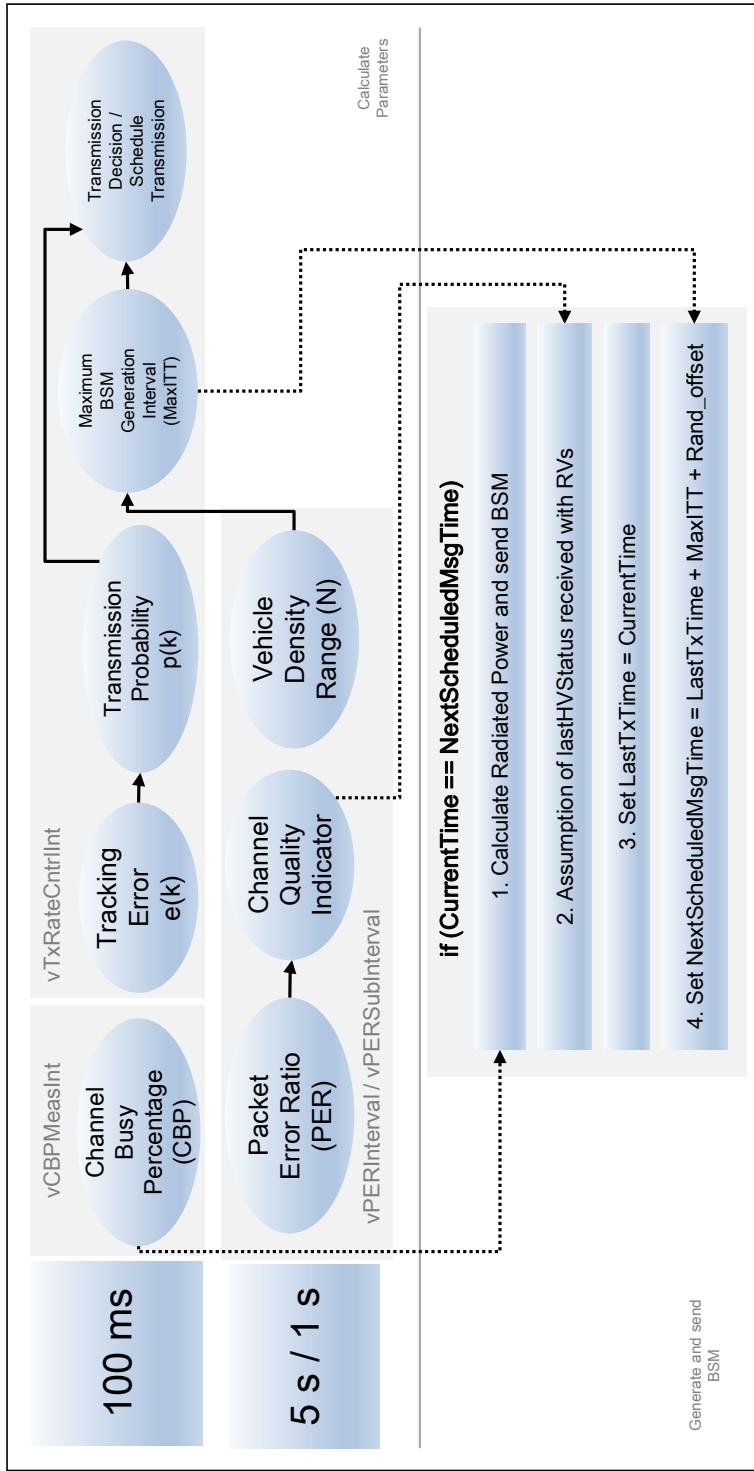


Abbildung 2.4 – Übersicht über die Generierung einer BSM (eigene Darstellung nach [17])

Die Berechnungen werden immer am Ende des jeweiligen Intervalls ( $vCBPMeasInt$  (100 ms),  $VPERSubInterval$  (1 s),  $vPERInterval$  (5 s) sowie  $vTxRateCntrlInt$  (100 ms)) durchgeführt. Mit  $k$  wird die entsprechende Instanz dieses Intervalls angesprochen. Die Input-Parameter werden hierbei als Funktion von  $k$  aufgeführt.  $F$  repräsentiert zum Beispiel eine bestimmte Funktion.  $F_{(k)}$  verwendet bei der Berechnung die Parameter, die während der  $k$ -ten Instanz des Intervalls abgerufen wurden.

In allen folgenden Berechnungsformeln werden die in [17], Kapitel 7, Tabelle 21, festgelegten konstanten Werte direkt eingesetzt. Der Parameter  $vCBPMeasInt$  wird z.B. durch den festgelegten Wert von 100 ms in allen Bemessungen ersetzt. Dies wurde mit allen in genannter Tabelle definierter Werten analog gehandhabt.

Aus Abbildung 2.4 soll neben den zu ermittelnden Parametern im jeweiligen Intervall auch ersichtlich werden, wie diese die eigentliche Generierung und Versendung einer BSM beeinflussen. Es fällt bereits auf, dass nur drei Parameter (*Channel Busy Percentage (CBP)*, *Channel Quality Indicator* sowie *Maximum BSM Generation Interval (MaxTT)*) direkt Einfluss auf diesen Generierungsprozess nehmen. Die in Abbildung 2.4 dargestellten Parameter in der oberen Hälfte der Abbildung sind in der aufgeführten Reihenfolge von links nach rechts zum jeweiligen Intervallsende zu berechnen. Alle aufgeführten Kennzahlen werden nun genau vorgestellt.

#### 2.1.4.1 Channel Busy Percentage (CBP)

Zunächst ist es definiert, die  $RawCBP_{(k)}$  zu berechnen. Hierbei wird der prozentuale Anteil berechnet, in der der Übertragungskanal die letzten 100 ms belegt war. Die Berechnung erfolgt alle 100 ms immer zum Intervallsende ( $vCBPMeasInt$ ), wie in Gleichung (2.1) aufgezeigt.

$$rawCBP_{(k)} = \frac{100 \cdot \text{Time that Channel is indicated as busy}}{100 \text{ ms}} \quad (2.1)$$

Anschließend ist dieser berechnete Wert zu glätten, was, wie in Gleichung (2.2) aufgezeigt, die eigentliche *Channel Busy Percentage (CBP)* festlegt.

$$CBP_{(k)} = 0.5 \cdot rawCBP_{(k)} + (1 - 0.5) \cdot CBP_{(k-1)} \quad (2.2)$$

#### 2.1.4.2 Packet Error Ratio (PER), Channel Quality Indicator und Vehicle Density in Range

*Packet Error Ratio (PER)*, *Channel Quality Indicator* und *Vehicle Density in Range* werden alle jeweils sekündlich ermittelt. Die beiden erstgenannten Parameter sind für eine Aussage über die verlorenen Pakete zwischen Fahrzeugpaaren und daraus

folgend für die Kanalqualität zu berechnen. Letztgenannter Parameter ist ausschlaggebend für die Ermittlung der *MaxITT* (siehe Abschnitt 2.1.4.5).

### Packet Error Ratio (PER)

$\delta_k$  ist die  $k$ -te Instanz von *vPERInterval* (5 s) und  $w_k$  die  $k$ -te Instanz von *vPERSubInterval* (1 s). Der Zusammenhang ist in Abbildung 2.5 dargestellt.

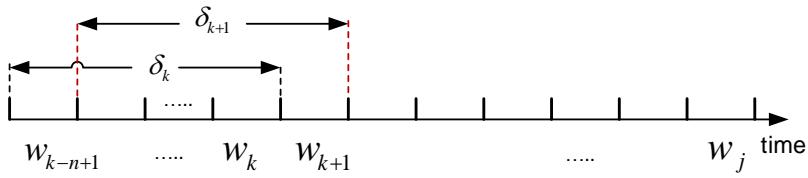


Abbildung 2.5 – PER Berechnungsintervalle (entnommen aus [17])

Die *PER* errechnet sich innerhalb dieser Intervalle von einem Host Vehicle (HV) für alle Nachbarfahrzeuge (Remote Vehicle (RV)), von denen BSMs erhalten wurden, wie in Gleichung (2.3) dargestellt. Dabei ist der in jeder BSM enthaltene Parameter *DE\_MsgCount* relevant. Es handelt sich um eine Art Sequenznummer, die jedes HV beim Versenden einer BSM fortlaufend hochzählt und einfügt. Bei jeder Berechnung der *PER* zum Ende von  $w_k$  (entspricht 1 s) wird auch immer  $\delta_k$  involviert und angepasst. Dieses umfasst 5 Sekunden. Da sich dieses 5 Sekunden-Fenster jede Sekunde um eine Sekunde "verschiebt", wird in diesem Zusammenhang auch von einem *Sliding Window* gesprochen.

$$PER_i(k) = \frac{\text{number of missed BSM from } RV_i \text{ during } [w_{k-n+1}, w_k]}{\text{total expected BSMs from } RV_i \text{ during } [w_{k-n+1}, w_k]}, \text{ where } k \geq n \quad (2.3)$$

Die Anzahl der zu erwartenden BSMs und die Anzahl der verlorenen Pakete aus Gleichung (2.3) berechnet sich wie in Gleichungen (2.4) bis (2.5).

$$\begin{aligned} \text{number of expected BSMs from } RV_i = & \\ 1 + & \\ (DE\_MsgCount \text{ of last BSM from } RV_i - & \\ DE\_MsgCount \text{ of first BSM from } RV_i) & \end{aligned} \quad (2.4)$$

$$\begin{aligned}
 & \text{number of missed BSM from } RV_i = \\
 & \quad \text{number of expected BSMs from } RV_i - \\
 & \quad \text{number of received BSMs from } RV_i
 \end{aligned} \tag{2.5}$$

Da  $k \geq n$  erfüllt sein muss, und  $n = 5$  laut [17] ist, kann eine Ermittlung der PER erst nach 5 s erfolgen, da  $k \geq 5$  sein muss. Davor ist diese undefiniert bzw. ist anhand des Standards nicht genau ableitbar, ob für  $k < 5$  die jeweiligen PERs berechnet werden dürfen / müssen. Bei der Berechnung der PER am Ende der  $k$ -ten Instanz von  $w_k$  müssen auch folgende Voraussetzungen vorliegen, damit die jeweilige PER in weitere Berechnungen einbezogen werden darf:

- Innerhalb von  $\delta_k$  (5 s) müssen mindestens zwei BSMs vom gleichen RV empfangen worden sein.
- Die letzte erhaltene BSM von einem RV muss während  $w_k$  (1 s) erhalten worden sein.
- Treffen die beiden vorgenannten Voraussetzungen zu, muss noch geprüft werden, ob sich die 2D-Position des zugehörigen RV innerhalb einer Entfernung von 100 m zum HV befindet.

Treffen diese Voraussetzungen zum Ende von  $w_k$  zu, kann die berechnete  $PER_i(k)$  zwischen HV und  $RV_i$  für die Berechnung des *Channel Quality Indicators* verwendet werden. Für alle Fahrzeugpaare ist dies entsprechend zu prüfen und zu ermitteln.

### Channel Quality Indicator

Der *Channel Quality Indicator* wird auch zum Ende von  $w_k$  nach Bemessung aller  $PER_i(k)$  ermittelt. Hierbei wird der Durchschnitt aller gültigen  $PER_i(k)$  (wurde in Abschnitt 2.1.4.2 definiert) gebildet, ein Wert von 0,3 darf allerdings nicht überschritten werden. Algorithmus 2.1 zeigt die Berechnungsschritte.

### Vehicle Density in Range

Diese Variable wird folgend mit  $N_{(k)}$  bezeichnet und repräsentiert die Anzahl an individuellen Fahrzeugen zum Ende von  $w_k$ , die sich in einem Umkreis von 100 m befinden. Der Standard ist hier leider ungenau, denn wie in Abschnitt 2.1.4.2 schon festgestellt wurde, muss  $k \geq 5$  sein. Es ist deshalb unklar, ob die Berechnung dieses Parameters schon vorher erfolgen muss. Aufgrund dessen wurden bei der Ausführung aller Simulationen (Abschnitt 4.3) die erfassten Statistikwerte erst nach 6 s aufgezeichnet.

---

**Require:** Gleichung (2.3) for all  $RV_i$   
**Ensure:**  $0 \leq channelQualityIndicator \leq 0.3$

```

1:  $sum \leftarrow 0$ 
2:  $count \leftarrow 0$ 
3: for all  $i$  such that  $i \geq 1$  do
4:    $sum \leftarrow sum + PER_i(k)$ 
5:    $count \leftarrow count + 1$ 
6: end for
7:  $channelQualityIndicator \leftarrow sum \div count$ 
8: if ( $channelQualityIndicator > 0.3$ ) then
9:    $channelQualityIndicator \leftarrow 0.3$ 
10: end if

```

---

Algorithmus 2.1 – Berechnung des *Channel Quality Indicators* [17]

#### 2.1.4.3 Tracking Error

Dieser Parameter repräsentiert einen Kennwert für Verzögerungen und Paketverluste zwischen dem tatsächlichen Zustand eines HVs und den bei einem RV empfangenen Fahrzeugdaten (in Form einer BSM). Auch die lokale Ungenauigkeit des GPS-Systems und Zeitverzögerungen werden berücksichtigt. Folgend wird dieser Parameter mit  $e_{(k)}$  bezeichnet. Der Grundgedanke ist der Vergleich der aktuell gültigen Positionsdaten und dessen Messzeitpunkt (*TimePositionChangedLocal, PositionLocal*) eines HVs und der Positionsdaten, von denen angenommen wird, dass diese zuletzt erfolgreich an  $RV_i$  übertragen wurden (*TimePositionChangedRemote, PositionRemote*). Die Berechnung erfolgt immer am Ende von *vTxRateCntrlInt* (alle 100 ms) nach Algorithmus 2.2.

Der *Tracking Error* wird ausschließlich für die Berechnung der *Transmission Probability* benötigt, welche folgend erläutert wird.

#### 2.1.4.4 Transmission Probability

Die *Transmission Probability*, folgend mit  $p_{(k)}$  bezeichnet, wird anhand des zuvor errechneten *Tracking Errors*  $e_{(k)}$  auch zum Ende von *vTxRateCntrlInt* (100 ms) berechnet. Dieser Parameter wird nur bei der noch folgenden *Transmission Decision* verwendet. Die Ermittlung erfolgt nach Gleichung (2.6).

$$p_{(k)} = \begin{cases} 1 - \exp(-75 \cdot |e_{(k)} - 0.2|^2) & \text{if } 0.2 \leq e_{(k)} < 0.5, \\ 1 & \text{if } e_{(k)} \geq 0.5, \\ 0 & \text{otherwise} \end{cases} \quad (2.6)$$

---

**Require:** *current TimePositionChangedLocal* and *current TimePositionChangedRemote*

**Require:** *current PositionLocal* and *current PositionRemote*

**Ensure:**  $e_{(k)} \geq 0$

```

1: currentEstimatedPositionLocal  $\leftarrow 0$ 
2: currentEstimatedPositionRemote  $\leftarrow 0$ 
3:  $\Delta TimeLocal = currentTime - TimePositionChangedLocal$ 
4:  $\Delta TimeRemote = currentTime - TimePositionChangedRemote$ 
5: if ( $\Delta TimeLocal > 0.15$  s) then
6:    $e_{(k)} = 0$ 
7:   return
8: end if
9: if ( $\Delta TimeRemote > 3$  s) then
10:   $e_{(k)} = 0$ 
11:  return
12: end if
13: if ( $\Delta TimeLocal < 0.05$  s) then
14:   currentEstimatedPositionLocal = PositionLocal
15: else if ( $0.05$  s  $\leq \Delta TimeLocal \leq 0.15$  s) then
16:   estimate currentEstimatedPositionLocal with  $\Delta TimeLocal$  and
      PositionLocal
17: end if
18: if ( $\Delta TimeRemote < 0.05$  s) then
19:   currentEstimatedPositionRemote = PositionRemote
20: else if ( $0.05$  s  $\leq \Delta TimeRemote \leq 3$  s) then
21:   estimate currentEstimatedPositionRemote with  $\Delta TimeRemote$  and
      PositionRemote
22: end if
23:  $e_{(k)} \leftarrow$  distance between currentEstimatedPositionRemote and currentEstimatedPositionLocal

```

---

Algorithmus 2.2 – Berechnung des *Tracking Errors* [17]

#### 2.1.4.5 Maximum BSM Generation Interval MaxITT

Auch dieser Parameter ist am Ende von *vTxRateCntrlInt* zu berechnen und ist entscheidend für die Festlegung der Zeit, wann die nächste BSM zu generieren und zu versenden ist. Von großer Bedeutung hierfür ist der in Abschnitt 2.1.4.2 beschriebene Parameter *Vehicle Density in Range* ( $N_{(k)}$ ). Dieser wird zunächst anhand der Gleichung (2.7) geglättet und mit  $N_s(k)$  bezeichnet.

$$N_s(k) = 0.05 \cdot N_{(k)} + (1 - 0.05) \cdot N_s(k - 1) \quad (2.7)$$

Anschließend erfolgt die Berechnung von *MaxITT* nach Gleichung (2.8). Der Wert von *MaxITT* bewegt sich demnach in einem Spektrum zwischen 100 ms und 600 ms.

$$MaxITT_{(k)} = \begin{cases} 100 & \text{if } N_s(k) \leq 25, \\ 100 \cdot \frac{N_s(k)}{25} & \text{if } 25 < N_s(k) < 150, \\ 600 & \text{if } 150 \leq N_s(k) \end{cases} \quad (2.8)$$

#### 2.1.4.6 Transmission Decision

Im Anschluss an die Bemessung der *MaxITT* (siehe Gleichung (2.8)) wird basierend auf den berechneten Werten für *MaxITT* und der *Transmission Probability*  $p_k$  die *Transmission Decision* auch zum Ende von *vTxRateCntrlInt* durchgeführt. Hierbei werden, wie in Algorithmus 2.3 beschrieben (die dort enthaltene Bernoulli-Funktion gibt bei Ausführung den Zufallswert 0 oder 1 zurück), drei verschiedene Flags gesetzt, die im Anschluss bei der *Schedule Transmission* in Abschnitt 2.1.4.7 benötigt werden.

---

**Require:** current  $p_{(k)}$ , current  $MaxITT(k)$

```

1:  $rand \leftarrow bernoulli(0.5)$ 
2:  $TxDcisionCriticalEvent \leftarrow \text{false}$ 
3:  $TxDcisionDynamics \leftarrow \text{false}$ 
4:  $TxDcisionMaxITT \leftarrow \text{false}$ 
5: if (CriticalEventCondition occurs) then
6:    $TxDcisionCriticalEvent \leftarrow \text{true}$ 
7: end if
8: if ( $rand \leq p_{(k)}$ ) and (NextScheduledMsgTime - CurrentTime)  $\geq 0.025s$  then
9:    $TxDcisionDynamics \leftarrow \text{true}$ 
10: end if
11: if (NextScheduledMsgTime - (LastTxTime +  $MaxITT_{(k)}$ ))  $\geq 0.025s$  then
12:    $TxDcisionMaxITT \leftarrow \text{true}$ 
13: end if
```

---

Algorithmus 2.3 – *Transmission Decision* [17]

Bei dem Parameter *NextScheduledMsgTime* wird von dem Zeitpunkt ausgegangen, an dem die nächste BSM zu generieren ist, wie in Abschnitt 2.1.4.9 beschrieben. Was unter einer *CriticalEventCondition* zu verstehen ist, wird in [17] erläutert. Im Rahmen dieser Arbeit wurde diese Condition allerdings nicht in die Implementierung einbezogen, weshalb von einer Beschreibung abgesehen wird.

#### 2.1.4.7 Schedule Transmission

Unmittelbar im Anschluss an die in Algorithmus 2.3 vorgestellte *Transmission Decision* on ist *Schedule Transmission* auszuführen (Algorithmus 2.4).

---

**Require:** Algorithmus 2.3

```

1: if (TxDecisionCriticalEvent == true) or (TxDecisionDynamics == true) then
2:   cancel current scheduled transmission
3:   NextScheduledMsgTime = currentTime
4: else
5:   if (TxDecisionMaxITT == true) then
6:     cancel current scheduled transmission
7:     NextScheduledMsgTime ← max(CurrentTime, LastTxTime + MaxITT(k))
8:   end if
9: end if

```

---

**Algorithmus 2.4 – Schedule Transmission [17]**

#### 2.1.4.8 Radiated Power Calculation

Für jede zu übertragende BSM wird die *Radiated Power* (RP) separat errechnet. In Algorithmus 2.5 wird der entsprechende Algorithmus aufgezeigt. Davon leitet sich die *Transmit Power* ab. Die Berechnung der *Transmit Power* ist in [17], Kapitel 6.4.1, beschrieben. Von der ermittelten *Radiated Power* werden hier noch Störfaktoren sowie der Antennengewinn einbezogen. Da im Rahmen dieser Arbeit von einer verlustfreien Antenne ausgegangen wird, wurde diese Berechnung nicht berücksichtigt und immer der errechnete Wert für RP mit der *Transmit Power* gleichgesetzt.

---

**Require:** current TxDecisionCriticalEvent, current TxDecisionDynamics

**Require:** current BaseRP, current PreviousRP, current RP

```

1: if (TxDecisionCriticalEvent ==true) or (TxDecisionDynamics == true) then
2:   RP = 20 dBm
3: else
4:   f(CBP) ← Gleichung (2.9)
5:   BaseRP ← PreviousRP + 0.5 · (f(CBP) − PreviousRP)
6:   RP ← BaseRP
7:   PreviousRP = BaseRP
8: end if

```

---

**Algorithmus 2.5 – Berechnung der Radiated Power [17]**

$$f(CBP) = \begin{cases} 20 & \text{if } CBP_{(k)} \leq 50, \\ 20 - \frac{1}{3} \cdot (CBP_{(k)} - 50) & \text{if } 50 < CBP_{(k)} < 80, \\ 10 & \text{if } 80 \leq CBP_{(k)} \end{cases} \quad (2.9)$$

#### 2.1.4.9 Generierung einer BSM und Festlegung der darauffolgenden BSM Generierung

Hier handelt es sich um die auszuführenden Schritte, wenn  $currentTime == NextScheduledMsgTime$  ist. Dieser Abschnitt ist demnach losgelöst von den zuvor berechneten Parametern und deren gleichbleibender Berechnungsintervalle. Bei Ausführung werden die jeweils zuletzt berechneten zuvor beschriebenen Parameter herangezogen. Nur die Radiated Power wird separat für jede BSM berechnet. In Algorithmus 2.6 sind die durchzuführenden Schritte beschrieben. Die dort enthaltene Bernoulli-Funktion gibt bei Ausführung den Zufallswert 0 oder 1 zurück.

---

**Require:**  $currentTime == NextScheduledMsgTime$ , current  $channelQualityIndicator$   
**Require:**  $txFailed \geq 0$  **and**  $txFailed \leq 3$ , current  $latestHVState$   
**Require:** current  $MaxITT$

- 1:  $rand \leftarrow uniform(-0.005 \text{ s}, 0.005 \text{ s})$
- 2: generate BSM, calculate RP as in Algorithmus 2.5 **and** send BSM  
{update  $txFailed$  after sending BSM}
- 3:  $txFailedRand \leftarrow bernoulli(0.5)$
- 4: **if** ( $txFailedRand < channelQualityIndicator$ ) **then**
- 5:    $txFailed \leftarrow txFailed + 1$
- 6: **else**
- 7:    $txFailed \leftarrow 0$
- 8: **end if**  
{the following is necessary to save the last HVStatus, it is used to calculate the trackingError as in Algorithmus 2.2, if  $txFailed$  is 0 we assume, that the last BSM was reveived from RVs, else not}
- 9: **if** (**not** ( $txFailed > 0$  **and**  $txFailed \leq 3$ )) **then**
- 10:    $txFailed \leftarrow 0$
- 11:    $latestHVState$  is  $lastSentBSM$
- 12: **end if**
- 13:  $LastTxTime \leftarrow currentTime$
- 14:  $NextScheduledMsgTime \leftarrow LastTxTime + MaxITT + rand$

---

## 2.2 ETSI ITS-G5

Folgend werden die relevanten Einzelheiten zu ETSI ITS-G5 erläutert. Zunächst werden hier analog zu WAVE die Kanalallokierung und der Protokollstack genauer vorgestellt. Anschließend wird das Beacon-Format CAM veranschaulicht. Den Abschluss bildet eine Zusammenfassung der DCC und deren Auswirkungen auf die Generierung von CAMs.

### 2.2.1 Frequenzbandallokierung und Kanalübersicht

Wie bei WAVE wurde auch der selbe Frequenzbereich von 5,855 MHz bis 5,925 MHz allokiert (siehe [28], [32]). Ein Unterschied zu WAVE ist allerdings, dass nicht alle verfügbaren Kanäle für sicherheitsrelevante Applikationen herangezogen werden, sondern eine Aufteilung in Safety Applications und Non-Safety Applications erfolgt ist. In Abbildung 2.6 wird dies entsprechend dargestellt.

	5855	5865	5875	5885	5895	5905	5915	5925
	ITS-G5B		ITS-G5A			ITS-G5D		
Kanalnummer	172	174	176	178	180	182	184	
Kanalbezeichnung	G5-SCH4	G5-SCH3	G5-SCH1	G5-SCH2	G5-CCH	G5-SCH5	G5-SCH6	
Vorgesehene Nutzung	non-safety applications		safety applications			ITS applications		

Abbildung 2.6 – Kanalallokierung bei ETSI [28] [32]

Im Vergleich zu WAVE in Abbildung 2.1 ist erkennbar, dass der Fokus nicht aller Kanäle auf Safety Applications liegt. Auch hier wurde bei der späteren Implementierung davon ausgegangen, dass ein Dual-Radio verwendet wird, welcher kontinuierlich auf einem Kanal verweilt und kein Kanalswitch erfolgt. In [33], Kapitel 5.3, wird hierauf Bezug genommen.

### 2.2.2 Überblick über den Protokollstack

Der Protokollstack bei ETIS ITS-G5 basiert grundsätzlich auch auf WLAN 802.11p. Allerdings wurden hierbei eigene Standards durch ETSI definiert. In Abbildung 2.7 ist der Protokollstack mit relevanten Standards abgebildet.

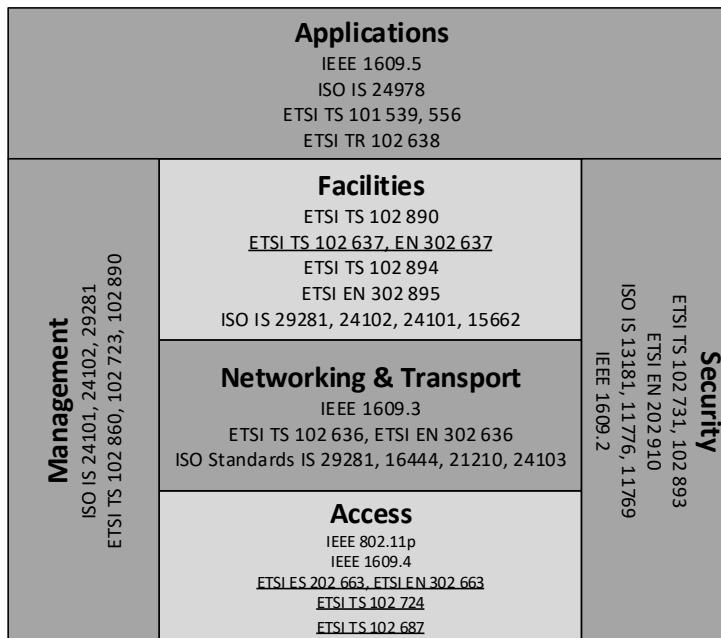


Abbildung 2.7 – Protokollstack bei ETSI [34] [35]

Die relevanten Schichten wurden farblich hervorgehoben, die für diese Arbeit wichtigen Standards sind unterstrichen. Auffällig ist hier im Vergleich zu Abbildung 2.2, der deutlich höhere Umfang an involvierten Standards, sowie die zusätzliche *Facilities*-Schicht. Auf dieser Ebene werden die CAMs generiert, wofür ein *CAM Basic Service* vorhanden ist, der im folgenden Abschnitt 2.2.3 vorgestellt wird.

Die Access-Schicht ist für die CAM-Generierung auch von großer Bedeutung. Es wird hier auch auf das von WLAN 802.11p und wie bei WAVE übernommene QoS-Verfahren EDCA verwendet. Die entsprechende Parametrisierung ist in Tabelle 2.3 abgebildet. Alle aufgeführten Nachrichtentypen sind für den CCH vorgesehen [36]. Dabei ist auffällig, dass es neben einer CAM noch ein weiteres Nachrichtenformat, das der Decentralized Environmental Notification Message (DENM), gibt. Ein Vergleich beider Formate erfolgt im nächsten Abschnitt.

Darüber hinaus befindet sich auf der Access-Schicht der wichtigste Bestandteil der DCC. Wie diese im Kontext der CAM-Generierung vorgesehen ist, wird in dem Abschnitt 2.2.4 erläutert.

Nachrichtenformat	AC	CWmin	CWmax	AIFSN
Hoch-priorisierte DENM	AC_VO	3	7	2
DENM	AC_VI	7	15	3
CAM	AC_BE	15	1023	6
Multi-Hop DENM, sonstige Daten	AC_BK	15	1023	9

Tabelle 2.3 – EDCA-Parameter bei ETSI [33] [36]

### 2.2.3 Cooperative Awareness Message (CAM)

#### 2.2.3.1 CAM Basic Service

Für die Generierung/Versendung, das Empfangsmanagement sowie das Encoden und Decoden von CAMs ist der *CAM Basic Service* auf *Facilities*-Ebene zuständig. Die Einordnung in den Protokollstack ist in Abbildung 2.8 grafisch veranschaulicht. Drei Komponenten sind innerhalb des *CAM Basic Services* wichtig:

- Vehicle Data Provider (VDP): Verwaltet die Statusinformationen des Fahrzeuges.
- Position and Time Management (POTI): Stellt die Positionsdaten und Zeitinformationen zur Verfügung.
- Local Dynamic Map (LDM): Die Daten empfangener CAMs werden innerhalb dieser Datenbank verwaltet [37].

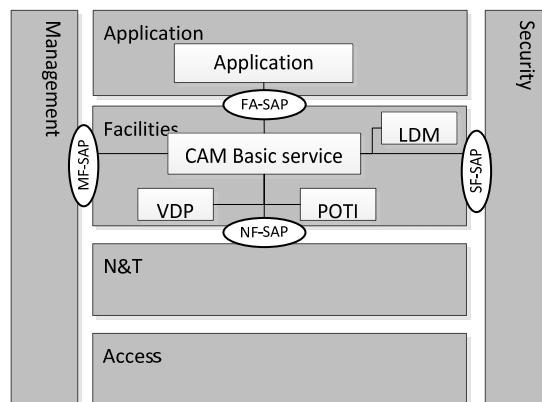


Abbildung 2.8 – CAM Basic Service [38]

### 2.2.3.2 Das CAM-Format

Wie bei der BSM bei WAVE (Abbildung 2.3) besteht eine CAM aus Mindestbestandteilen und optionalen Containern. In Abbildung 2.9 sind alle möglichen Komponenten aufgeführt.

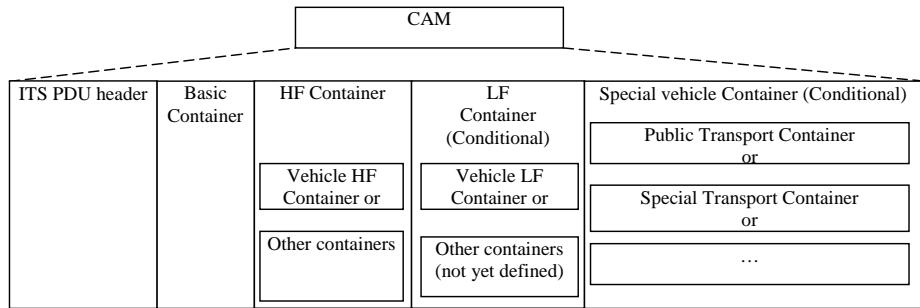


Abbildung 2.9 – CAM Format [38]

Essentiell und jeder CAM beizufügen sind hiervon folgende (nach [38]):

- ITS-S PDU Header: Beinhaltet die Datenelemente *ProtocolVersion*, *MessageID* und *StationID* (spezifiziert in [39]).
- Basic Container: Enthält Informationen über den Sendertyp sowie dessen letzte Position.
- High Frequency (HF) Container: Beinhaltet oft ändernde Fahrzeugdaten (z.B. Geschwindigkeit).

Weitere Container sind optional.

#### Abgrenzung zu DENM

Safety Applications bei WAVE in Tabelle 2.2 sind bei ETSI ITS-G5 auch vorgesehen und in [40], Tabelle 1, spezifiziert. Für die ereignisgesteuerte Benachrichtigung anderer Fahrzeuge wird hier allerdings eine DENM versendet. Es zeigt sich ein deutlicher Unterschied zu WAVE. Dort wird für das periodische Versenden von Nachrichten (Beaconing) sowie für hochpriorisierte ereignisgesteuerte Warnmeldungen das gleiche Format in Form von BSMs verwendet. ETSI ITS-G5 teilt diese Funktionsbereiche auf zwei Nachrichtenformate auf: DENM und CAM. Eine Abgrenzung dieser beiden Formate ist in Tabelle 2.4 aufgezeigt. Empfangene CAMs sind nach [38], Kapitel 6, nicht an weitere Fahrzeuge weiterzuleiten und sind demnach reine Single-Hop-Beacons. DENMs unterscheiden sich hiervon und können auch im Multi-Hop-Verfahren von Fahrzeug zu Fahrzeug weitergeleitet werden. Da im Rahmen dieser Arbeit nur Single-Hop-Beaconing betrachtet wird, wurden DENMs bei der Implementierung nicht

weiter einbezogen.

	CAM	DENM
Sendemodus	periodisch (1 Hz - 10 Hz)	ereignisgesteuert
Inhalt	Bewegungszustand des Fahrzeugs (Zeit, Position, Geschwindigkeit, etc.)	spezifische Daten (Hindernistyp, Zeit, Gültigkeitsdauer, etc.)
senden	Abfrage interner Sensorwerte über Bussysteme	aufwendige Datenfusion von Sensordaten
empfangen	aufwendige Verwaltung/Aktualisierung in LDM	einfache Relevanzprüfung (liegt Gefahr auf aktueller Route?)
Beaconing Variante	Single-Hop	Single-/Multi-Hop
Kanal	CCH	CCH

Tabelle 2.4 – Vergleich von CAM und DENM [3] [38] [36]

#### 2.2.4 Decentralized Congestion Control (DCC)

Die DCC ist bei ETSI ITS-G5 ein sehr wichtiger Bestandteil, der sich auf den gesamten Datenverkehr auswirkt. Grundgedanke ist eine angepasste Auswahl verschiedener Parameter, um eine erkannte Kanallast zu verringern. In Abschnitt 1.3 wurden bereits Arbeiten genannt, die sich intensiv mit der Bewertung dieser (Standard)Parametrisierung auseinander gesetzt haben. Insbesondere deshalb lag der Fokus im Rahmen dieser Arbeit nicht nur auf der Standardvariante (beschrieben in [10]).

Zunächst wird die Architektur der DCC vorgestellt. Der Fokus liegt auf der Access-Ebene, da dort die von der DCC gesetzten Parameter verwendet werden. Dafür zuständig ist eine Zustandsmaschine, die in Abhängigkeit von der gemessenen Kanallast in einem bestimmten Intervall die Parameter festlegt. Ansätze einer individuellen Parametrisierung werden an dieser Stelle aufgegriffen (Abschnitt 2.2.4.1).

Die Generierung von CAMs ist ein hochdynamischer Prozess. Die von der DCC festge-

legten Parameter haben direkten Einfluss auf die Generierungsrate und sind deshalb von besonderem Interesse für diese Arbeit (Abschnitt 2.2.5) .

#### 2.2.4.1 Architektur

Die Besonderheit von ETSI ITS-G5 ist der über (fast) alle Protokollsichten greifende Ansatz der DCC. In Abbildung 2.10 ist der Cross-Layer-Ansatz grafisch dargestellt.

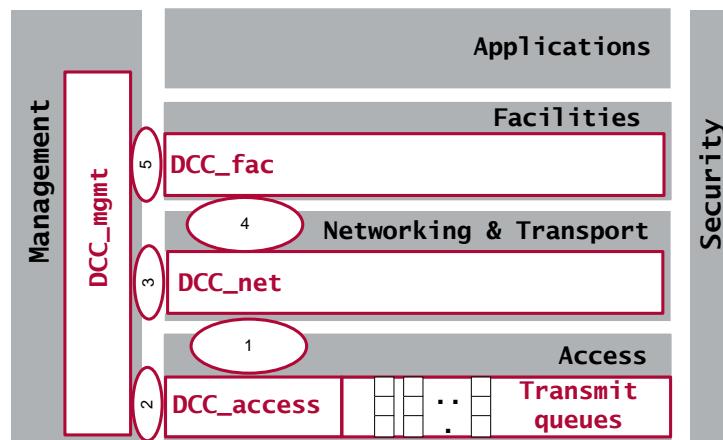


Abbildung 2.10 – DCC Architektur [10] [41]

Entscheidend für die in Abschnitt 2.2.5 erläuterte Generierung von CAMs sind insbesondere die Komponenten *DCC\_mgmt* und *DCC\_access*. *DCC\_fac* umfasst den CAM Basic Service, welcher in Abschnitt 2.2.3.1 schon beschrieben wurde.

#### DCC Management

Innerhalb dieser Komponenten werden alle Parameter und Grenzwerte (Network Design Limits (NDL)), die für Berechnungen benötigt werden, zentral über alle involvierten Schichten verwaltet [10]. Die erwähnten Grenzwerte werden in einer NDL Database gespeichert.

Hierzu gehören z.B.:

- Minimum und Maximum von Kontrollparametern
- Default-Werte, z.B. für Transmit Power
- Zeitkonstanten
- gemessene Kanallastwerte

### 2.2.4.2 DCC Access

Das Herz der DCC repräsentiert die *DCC Access*. Hier erfolgt eine zustandsabhängige Steuerung von Parametern für jedes Paket individuell [10]. Abbildung 2.11 veranschaulicht dieses Prinzip.

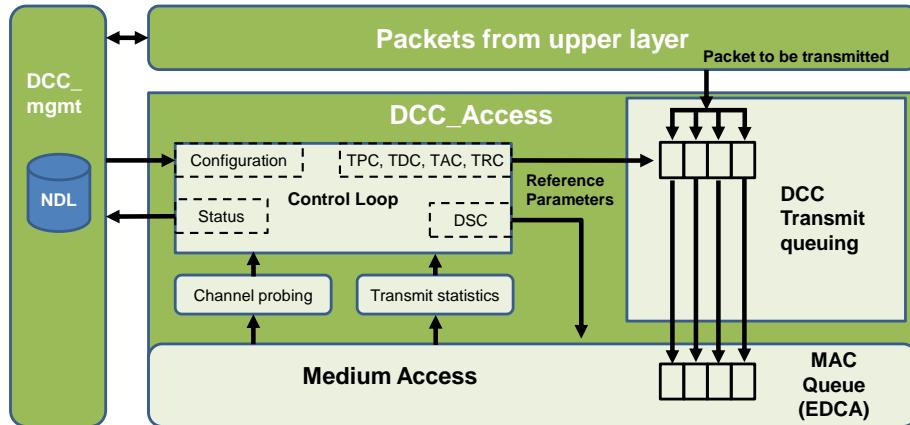


Abbildung 2.11 – DCC Access [10]

Trifft ein Paket von einer höheren Schicht bei *DCC Access* ein, erfolgt zunächst eine Einreihung in sogenannte *DCC Transmit Queues* (siehe [42], Kapitel 5.3.3). Diese stellen ein Ebenbild der gewöhnlichen EDCA-Queues mit den gleichen Prioritätsklassen dar. Die aufgezeigte Control Loop beinhaltet eine Zustandsmaschine, die die Paketparametrisierung abhängig vom aktiven Zustand vorgibt, z.B. wann das nächste Paket aus der *DCC Transmit Queue* in die EDCA-Queue eingereiht werden darf. In Abbildung 2.11 sind die Parameter, die innerhalb der Control Loop in Bezug auf *DCC Transmit Queues* gesetzt werden, rechts oben aufgeführt (in dem Kästchen "Control Loop"). Als Parameter werden die Transmit Power, Datarate, Packetinterval sowie die Carrier Sense zustandsabhängig geändert.

Folgende *DCC Access* Mechanismen sind von besonderer Bedeutung (beschrieben in [10]):

- Transmit Power Control (TPC): Legt den Referenzwert der *Transmit Power* fest. Dieser Wert wird bereits auf der höheren Netzwerkschicht gesetzt.
- Transmit Datarate Control (TDC): Legt die Datenrate fest. Auch dieser Wert wird bereits auf Netzwerkschicht gesetzt.
- Transmit Rate Control (TRC): Legt das Paketintervall fest. Damit ist die Zeitperiode definiert, die zwischen zwei Paketen liegen muss, bevor das nächste

Paket aus der DCC Transmit Queue entnommen und in die EDCA-Queue eingereiht werden darf (wird auch Gatekeeping genannt [43]). Es wird bei der Generierung von CAMs (Abschnitt 2.2.5) als  $T_{GenCam\_DCC}$  verwendet.

- DCC Sensitivity Control (DSC): Legt den Referenzwert für Clear Channel Assessment (CCA) fest. Dadurch wird der Referenzwert festgelegt, mit dem entschieden wird, ob der Kanal frei oder belegt ist.

In [10] sind noch weitere Parameter beschrieben, die von der DCC verwendet werden können. Im Rahmen der Arbeit wurde zusätzlich zu den oben genannten Parametern die maximale Länge einer *DCC Transmit Queue* (diese beträgt 2 für den CCH) berücksichtigt. Auch wurde die in [42] erwähnte Lifetime eines Paketes beachtet. Nach [38], Tabelle 2, beträgt diese 1 s. Beim Entnehmen eines Paketes aus der *DCC Transmit Queue* erfolgt demnach eine Prüfung, ob das Paket seine Lifetime schon überschritten hat oder nicht. Bei einer Überschreitung wird das Paket verworfen, ansonsten an die EDCA-Queue übergeben. In [12] wurde dafür ein anderes Konzept angewendet. Hier wurde innerhalb der DCC Transmit Queue immer nur die aktuellste CAM eingereiht. Im Rahmen dieser Arbeit fand dieses Konzept allerdings keine Anwendung.

### Zustandsmaschine zur Parametrisierung

Für die Festlegung des aktuellen Zustandes ist die gemessene Kanallast relevant. Als Referenzwert kann hierfür die *Channel Busy Ratio (CBR)*, definiert in [32], Kapitel 4.2.10, herangezogen werden. Diese ist wie in Gleichung (2.2) festgelegt.

$$CBR = \frac{T_{busy}}{T_{CBR}} \quad (2.10)$$

$T_{busy}$  ist die Zeit in Millisekunden (ms), in der innerhalb einer Periode von  $T_{CBR}$  die Empfangssignalstärke einen Wert von -85 dBm überschreitet und dadurch der Kanal als belegt angesehen wird.

$T_{CBR}$  beträgt hierbei 100 ms.

Die Zustandsmaschine kann zwischen drei Zuständen wechseln: RELAXED, ACTIVE und RESTRICTIVE. Auf dem SCH kann der ACTIVE-Zustand in mehrere Unterzustände eingeteilt werden, auf dem CCH ist dies nicht vorgesehen. In Tabelle 2.4 wurde bereits festgehalten, dass CAMs auf dem CCH gesendet werden. Von einer genaueren Betrachtung der Zustandsübergänge für den SCH wird deshalb abgesehen, da im Rahmen dieser Arbeit nur das Versenden von CAMs über den CCH betrachtet wird [10]. Abbildung 2.12 beschreibt die Zustandsübergänge für den CCH.

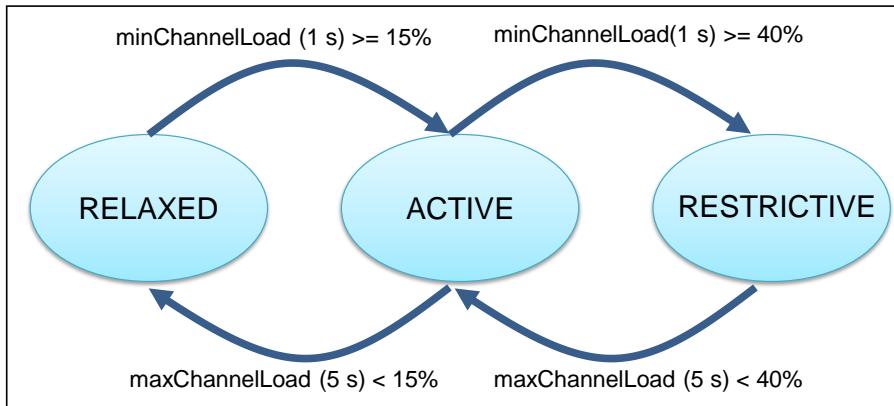


Abbildung 2.12 – DCC Zustandsmaschine für den CCH [10]

*minChannelLoad* und *maxChannelLoad* stellen hierbei Funktionen dar, die den minimal bzw. maximal gemessenen Wert der Kanallast innerhalb der letzten Sekunde ermittelt bzw. in den letzten fünf Sekunden. Hierzu muss bedacht werden, dass eine Messung der Kanallast alle 100 ms stattfindet. Ein Zustandsübergang von *RELAXED* zu *ACTIVE* erfolgt somit, wenn innerhalb der letzten Sekunde alle nach Gleichung (2.10) berechneten *CBR* mindestens 15% oder mehr betragen. Eine erneute Messung beginnt eine Sekunde später. Dieses Prinzip gilt für alle weiteren Zustandsübergänge analog wie in Abbildung 2.12 aufgeführt.

#### Standardparameter der DCC

Für die einzelnen Zustände sind durch die DCC die in Tabelle 2.5 aufgeführten Werte zu setzen [10]. Für CAMs sind hierbei nach [36], Tabelle 5, andere Werte vorgesehen. Hier ist festgelegt, dass die *Transmit Power* maximal 23 dBm betragen darf und CAMs mit der EDCA-Priorität *AC\_BE* zu behandeln sind. In Tabelle 2.6 wurden die entsprechenden Werte geändert und hervorgehoben. Die Bezeichnung *ref* in genannten Tabellen bedeutet, dass hier keine Änderung des vorherigen Wertes vorgenommen wird. In den Zuständen *RELAXED* und *RESTRICTIVE* erfolgt auch keine Unterscheidung nach EDCA ACs.

	Zustand					
	RELAXED	ACTIVE			RESTRICTIVE	
EDCA AC		AC_VO	AC_VI	AC_BE	AC_BK	
TPC	33 dBm	ref	25 dBm	20 dBm	15 dBm	-10 dBm
TRC	0,04 s	ref	ref	ref	ref	1 s
TDC	3 Mbit/s	ref	ref	ref	ref	12 Mbit/s
DSC	-95 dBm	ref	ref	ref	ref	-65 dBm

Tabelle 2.5 – Standardparameter der DCC für den CCH [10]

	Zustand		
	RELAXED	ACTIVE	RESTRICTIVE
EDCA AC		AC_BE	
TPC	23 dBm	20 dBm	-10 dBm
TRC	0,04 s	ref	1 s
TDC	3 Mbit/s	ref	12 Mbit/s
DSC	-95 dBm	ref	-65 dBm

Tabelle 2.6 – Für CAMs relevante DCC Standardparameter [10] [36]

### DCC Profile

In [41] wurden die Standardparameter durch sogenannte *DCC Profiles* in Abhängigkeit des verwendeten Kanals verändert. Insbesondere werden in [41] Multi-Channel Szenarien beschrieben. Dies bedeutet, dass bei einer Überlastung des CCHs das Paket auf einen SCH umgeleitet werden kann. Im Rahmen dieser Arbeit wurde allerdings nur auf dem CCH verblieben. Dabei ist von Interesse, dass CAMs nach [41], Kapitel C.2, dem DCC Profile 2 zuzuordnen sind. Hier sind in [41], Tabelle 1, Werte für das Paketintervall (TRC), wie in Tabelle 2.7 beschrieben, definiert. Diese wurden bei der Implementierung berücksichtigt und zusätzlich betrachtet.

	Zustand		
	RELAXED	ACTIVE	RESTRICTIVE
EDCA AC		AC_BE	
TPC	23 dBm	20 dBm	-10 dBm
TRC	0,095 s	0,190 s	0,250 s
TDC	3 Mbit/s	ref	12 Mbit/s
DSC	-95 dBm	ref	-65 dBm

Tabelle 2.7 – Für CAMs relevante DCC Parameter nach DCC Profile 2 [41]

### Gatekeeping durch LIMERIC

In [43] ist noch eine weitere Variante beschrieben, wie Werte für das Paketintervall ermittelt werden können. Dies wurde auch in [12] untersucht. Der Berechnungsalgorithmus basiert auf dem sogenannten LIMERIC-Algorithmus [44]. Hierbei handelt es sich um eine dynamische Berechnung des Parameters  $T_{GenPacket\_DCC}$ , welcher die Zeit vorschreibt, die zwischen dem Einreihen zweier Pakete in die EDCA-Queue verstrichen sein muss. Bis zum Erreichen der Zeit verweilen die Pakete in der  $DCC$  *Transmit Queue* und werden nach Priorität und der Reihenfolge (einzelnen) entnommen.

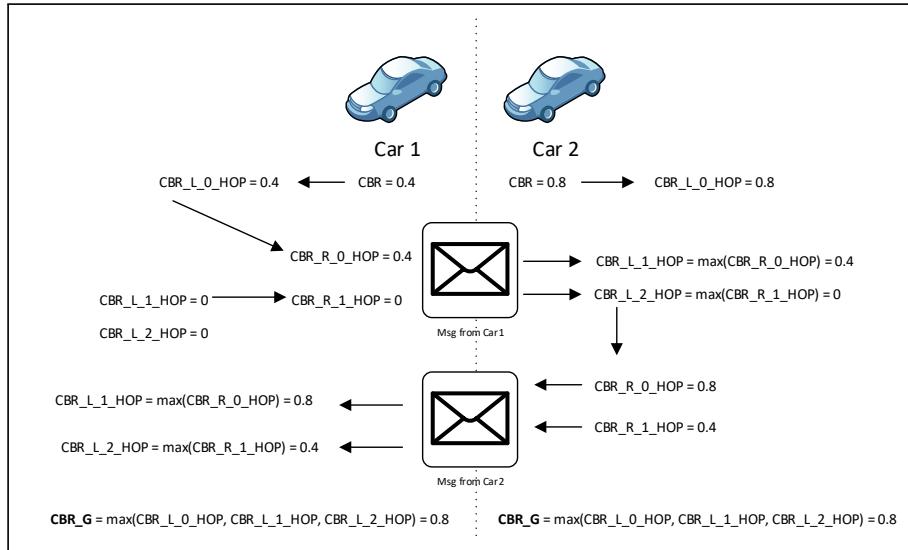
Die Ermittlung setzt hierbei im gleichen Intervall ein, wie die Bemessung der CBR, alle 100 ms. Dabei soll diese für jeden Kanal individuell ausgeführt werden [43]. Einzuberechnen sind hierfür Parameter, die für das Geonetworking Protokoll (siehe [45] und [36]) vorgesehen sind. Hierbei handelt es sich um verschiedene Kanallastwerte, die nicht nur lokal gemessen wurden. Auch die errechneten Werte der Nachbarfahrzeuge werden involviert und an Nachbarfahrzeuge versendet. Das Geonetworking Protokoll hat aus allen Werten die globale CBR,  $CBR_G$ , bereitzustellen. Eine Übersicht der verschiedenen CBRs ist in Tabelle 2.8 aufgelistet. Die  $CBR_G$  ermittelt sich nach Gleichung (2.11). Wie die Werte zu verstehen sind, veranschaulicht Abbildung 2.13.

$$CBR_G = \max(CBR\_L\_0\_HOP, CBR\_L\_1\_HOP, CBR\_L\_2\_HOP) \quad (2.11)$$

Die Kalkulation von  $T_{GenPacket\_DCC}$  berechnet sich bei jedem Fahrzeug wie folgt. Zunächst ist die  $CBR\_DCC$  nach Gleichung (2.12) zu bestimmen.

Parameter	Beschreibung
$CBR\_L\_0\_HOP$	lokale CBR (siehe Gleichung (2.10)) → wird als $CBR\_R\_0\_HOP$ an Nachbarfahrzeuge versendet
$CBR\_L\_1\_HOP$	$\max(CBR\_R\_0\_HOP)$ → maximale Kanallast der Nachbarn, wird als $CBR\_R\_1\_HOP$ an alle Nachbarfahrzeuge versendet
$CBR\_L\_2\_HOP$	$\max(CBR\_R\_1\_HOP)$
$CBR\_R\_0\_HOP$	lokale CBR → wird an Nachbarfahrzeuge versendet
$CBR\_R\_1\_HOP$	$CBR\_L\_1\_HOP$ → wird an Nachbarfahrzeuge versendet

Tabelle 2.8 – CBR Werte des Geonetworking Protokolls [43] [36]



**Abbildung 2.13** – Beispiel der verschiedenen Kanallastwerte des Geonetworking Protokolls (eigene Darstellung, nach [36])

$$CBR_{DCC} = \max(CBR_{L\_0\_HOP}, CBR_G) \quad (2.12)$$

Folgend basiert der Algorithmus nicht direkt auf CBR-Werten. Diese werden in eine Rate von Nachrichten pro Sekunde umgerechnet. Die Konvertierung dieser Rate  $r$  und der CBR wird nach [43], Kapitel 5.3, wie in Gleichung (2.13) definiert, abgeleitet.

$$r = CBR_{DCC} \cdot 2000 \quad (2.13)$$

$CBR_{DCC}$  wird hierbei wie in Gleichung (2.12) berechnet.

Der eigentliche Algorithmus, der für die Berechnung von  $T_{GenPacket\_DCC}$  relevant ist, wurde in [43] aus [44] übernommen und lautet wie in Gleichung (2.14).

$$r_j(t) = (1 - \alpha) \cdot r_j(t - 1) + sign(r_g - r(t - 1)) \cdot min[X, \beta \cdot |r_g - r(t - 1)|] \quad (2.14)$$

$r_j$  ist hierbei die sogenannte  $ego\_target\_rate$ ,  $r_g$  ist die  $target\_rate$ ,  $X$  entspricht 1,  $\alpha$  entspricht 0,1 und  $\beta$  entspricht 1/150. Der Wert von  $r_g$  wurde analog zu [46], Tabelle 14, mit einem konstanten CBR-Wert (in besagter Quelle als  $CBP_{Target}$  bezeichnet)

von 80% belegt (dadurch  $r_g = 0.8 \cdot 2000$ ). Nachdem  $r_j$  berechnet wurde, kann nach Gleichung (2.15)  $T_{GenPacket\_DCC}$  ermittelt werden [43].

$$T_{GenPacket\_DCC(t)} = \frac{1}{r_j(t)} \quad (2.15)$$

Diese Berechnung muss nach Ermittlung der lokalen CBR erfolgen (somit auch alle 100 ms).  $T_{GenPacket\_DCC}$  ist nach Berechnung das aktuell gültige Paketintervall, das vorgibt, wann das nächste Paket aus der DCC Transmit Queue entnommen werden darf. Bei der Generierung von CAMs wird es als  $T_{GenCam\_DCC}$  verwendet (Abschnitt 2.2.5).

### 2.2.5 Generierung von CAMs

Die Generierung von CAMs ist ein hochdynamischer Prozess und wird abhängig von der Veränderung der aktuellen Fahrzeugdynamik im Vergleich zu den in der letzten versendeten CAM enthaltenen Werten durchgeführt (siehe [38], Kapitel 6.1.3). Die Prüfung erfolgt alle 100 ms ( $T_{CheckCamGen}$ ). Es können auch kürzere Zeitperioden hierfür festgelegt werden. Dabei wird geprüft, ob seit der letzten CAM-Generierung mindestens die Zeit  $T_{GenCam\_DCC}$  verstrichen ist, und:

- der Betrag der Differenz der aktuellen Geschwindigkeit sowie der Geschwindigkeit, die in der letzten versendeten CAM enthalten ist, größer als 0,5 m/s ist, oder
- die Distanz der aktuellen Position sowie der Position, die in der letzten versendeten CAM enthalten ist, größer als 4 m ist, oder
- der Betrag der Differenz des aktuellen Lenkeinschlages (Heading) sowie des Lenkeinschlages, der in der letzten versendeten CAM enthalten ist, größer als 4° ist.

Trifft dies zu, wird eine CAM versendet und  $T_{GenCam}$  auf die Zeit gesetzt, die seit der letzten CAM-Generierung verstrichen ist. Werden in dem gleichen  $T_{GenCam}$  Intervall zwei weitere CAMs versendet, ohne dass eine der dynamischen Änderungen eingetreten ist, wird  $T_{GenCam}$  auf  $T_{GenCamMax}$  gesetzt.

Von Seiten der DCC ist der Parameter  $T_{GenCam\_DCC}$  bereitzustellen. Hierbei handelt es sich um das Paketintervall, das auf *DCC Access* Ebene den Paketfluss der *DCC Transmit Queues* regelt (wie das Paketintervall festgelegt wird, wurde in Abschnitt 2.2.4.2 erläutert). Bei der CAM-Generierung wird damit das Generierungsintervall festgelegt. Auffallend ist, dass dieser Wert zunächst auf 1 s festgesetzt wird.

Nur bei Änderungen der Fahrzeugdynamik werden CAMs in kürzeren Intervallen generiert. Der genaue CAM-Generierungsalgorithmus ist in Algorithmus 2.7 beschrieben.

---

```

Require: current lastTimeCAMGenerated, current T_GenCam_DCC
Require: current counterNGenCam
Require: T_GenCamMin ← 0.1 s, T_GenCamMax ← 1 s
Require: T_GenCamMin ≤ T_GenCam_DCC ≤ T_GenCamMax
Ensure: T_GenCamMin ≤ T_GenCam ≤ T_GenCamMax
Ensure: repeat Algorithmus 2.7 after T_CheckCamGen

1: T_CheckCamGen ← T_CenCamMin
2: N_GenCam ← 3
3: timeElapsedSinceLastCAM ← CurrentTime - lastTimeCAMGenerated
4: if (first time called) then
5:   T_GenCam ← T_GenCamMax
6: end if
   {CONDITION 1 follows}
7: if (timeElapsedSinceLastCAM ≥ T_GenCam_DCC) then
8:   difSpeed ← absolute difference between current speed and speed included in
   previously transmitted CAM
9:   difPos ← distance between current position and position included in previously
   transmitted CAM
10:  difHeading ← absolute difference between current heading and heading in-
   cluded in previously transmitted CAM
11:  if (difSpeed > 0.5 m/s or difPos > 4 m or difHeading > 4°) then
12:    generate CAM immediately
13:    lastTimeCAMGenerated ← currentTime
14:    counterNGenCam ← 1
15:    T_GenCam ← timeElapsedSinceLastCAM
    {end of CONDITION 1, CONDITION 2 follows}
16:  else if (timeElapsedSinceLastCAM ≥ T_GenCam_DCC and timeElapsedSince-
   LastCAM ≥ T_GenCam) then
17:    generate CAM immediately
18:    lastTimeCAMGenerated ← currentTime
19:    counterNGenCam ← counterNGenCam + 1
20:    if (counterNGenCam == N_GenCam) then
21:      T_GenCam ← T_GenCamMax
22:    end if
23:  end if
24: end if

```

---

**Algorithmus 2.7 – Generierung von CAMs [38]**

---

## Kapitel 3

---

# Implementierung in VEINS

---

In diesem Kapitel werden zunächst die verwendeten Tools vorgestellt, die bei der Implementierung Verwendung fanden. Anschließend kommt eine Erläuterung der grundlegenden Herangehensweise und allgemeiner Festlegungen bezüglich der Implementierung beider Mechanismen. Der Fokus liegt bei der Veranschaulichung, wie die in Kapitel 2 vorgestellten Mechanismen in die Simulationsumgebung integriert wurden. Hierbei wird auf die jeweiligen C++-Klassen oder NED-Dateien verwiesen. Auf Abbildung von Codelistings wird größtenteils verzichtet, da der gesamte implementierte Code auf dem beigefügten Datenträger einsehbar ist (siehe Anhang C). Der Abschluss dieses Kapitels erläutert, wie Tests der Implementierung erfolgen.

Zu beachten ist, dass im Folgenden die Verwendung der Begriffe WAVE und ETSI als Synonym für die in Kapitel 2 vorgestellten Generierungsmechanismen verwendet werden.

### 3.1 Verwendete Tool-Chain

#### 3.1.1 Omnet++

Omnet++ ist eine Open-Source Simulationsumgebung für die Entwicklung von Netzwerksimulationen, die bereits seit 1992 in Aufbau ist. Die Nutzung ist allerdings nur für akademische Zwecke kostenlos (siehe [47] und [7]). Es werden die meisten aktuellen Betriebssysteme unterstützt. Omnet++ basiert auf einer angepassten Version der weit verbreiteten IDE *Eclipse* und erleichtert dadurch den Einstieg. Bei Omnet++ handelt sich um eine diskrete Simulationsumgebung, d.h. Simulationen laufen nicht in Echtzeit ab, sondern diese wird auch simuliert. Die Ausführung der Simulationen wird durch eine GUI unterstützt, kann allerdings auch per Kommandozeilenbefehle ausgeführt werden. Omnet++ basiert auf einer hierarchischen

Struktur mit sogenannten Komponenten als Hauptbestandteil, die das Verhalten von beispielsweise bestimmten Netzwerkschichten festlegen. Komponenten sind in C++ zu programmieren. Um diese zu hierarchischen Netzwerkstrukturen zu verbinden, ist eine Beschreibung in einer eigenen Beschreibungssprache, Network Description (NED) genannt, notwendig. Der große Vorteil ist die einfache Wiederverwendbarkeit von Komponenten in verschiedenen Netzwerktopologien. Aufgrund der langen Entwicklungszeit, stetiger Updates und einer sehr großen Usercommunity ist Omnet++ ein sehr leistungsfähiges Tool für die Simulation von Netzwerken. Darüber hinaus gibt es bereits sehr viele Libraries und Frameworks, die bestimmte Domänen abbilden, wie z.B. das *INET Framework*, das eine Simulation des kompletten Protokollstacks der Internetprotokolle ermöglicht oder das *VEINS-Framework*, das für Car2X-Simulationen eingesetzt werden kann. Im Rahmen dieser Arbeit wurde von Omnet++ Version 5.0 verwendet.

### 3.1.2 VEINS

Dies ist ein Open-Source Framework für die Simulation von Car2X-Szenarien (siehe [6] und [47]). Die Architektur von VEINS ist in Abbildung 3.1 dargestellt. Innerhalb von Omnet++, das die Simulationsausführung und das Sammeln von statistischen Daten übernimmt, stellt VEINS Komponenten zur Verfügung, die auf dem IEEE-Protokollstack (WLAN 802.11p, IEEE 1609.x) basieren und die PHY-Schicht und MAC-Schicht, sowie die Mobilität von Fahrzeugen abbilden. Es gibt enorm viele Erweiterungen, da VEINS bei bereits über 100 Forschungsarbeiten eingesetzt wurde. Es befindet sich zum aktuellen Zeitpunkt in der ständigen Weiterentwicklung. VEINS ist bidirektional zu dem Verkehrssimulationstool SUMO per TCP Socket gekoppelt (hierfür wurde das Traffic Control Interface (TraCI) Protokoll verwendet) und ermöglicht dadurch die statistische Erfassung der Verkehrsdaten aus SUMO durch Omnet++. Im Rahmen dieser Arbeit wurde Version 4.4 von VEINS benutzt.

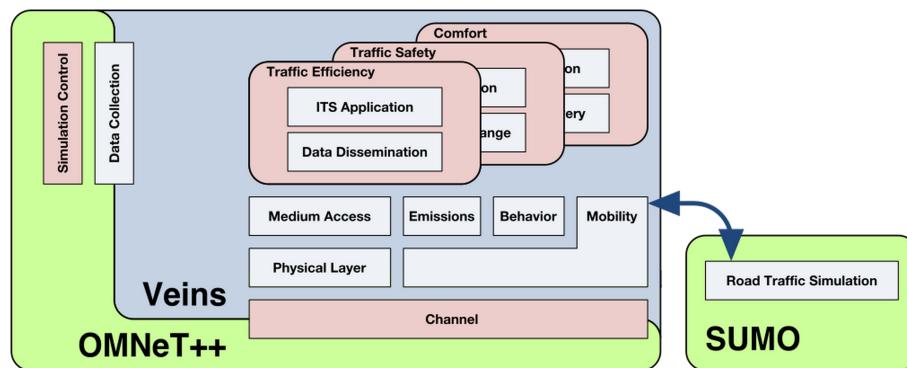
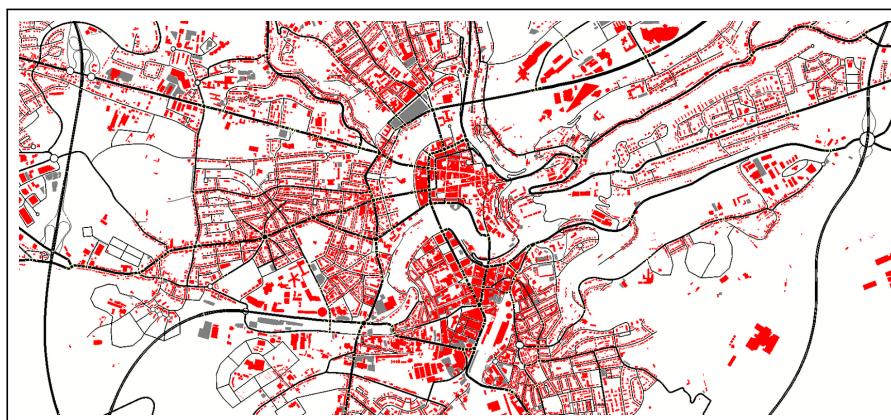


Abbildung 3.1 – Architektur des VEINS-Framework [6]

### 3.1.3 SUMO

SUMO ist ein Open-Source Verkehrssimulator, der die Abbildung komplexer Verkehrsszenarien ermöglicht (siehe [8] und [47]). Bereits im Jahre 2000 begann die Entwicklung und es unterstützt eine Vielzahl von Funktionen, z.B. das Modellieren von Passanten und der Einbezug von Ampelsystemen und -phasen. Einsatz fand der Simulator für Verkehrsvorhersagen bei Massenveranstaltungen. Die Größe des Netzwerkes oder die Anzahl der Fahrzeuge ist nicht festgelegt und kann variiert werden. Dadurch können sehr einfach Erkenntnisse in Abhängigkeit der Verkehrsichte ermittelt werden. Anhand des schon unter Abschnitt 3.1.2 genannten TraCI-Protokolls ist SUMO mit Omnet++ bzw. VEINS bidirektional verbunden. Es ist möglich, SUMO mit Hilfe einer GUI oder per Kommandozeile auszuführen. In Abbildung 3.2 ist ein Screenshot der GUI dargestellt. Im Rahmen dieser Arbeit wurde Version 0.25.0 verwendet.



**Abbildung 3.2 –** SUMO Screenshot während der Ausführung des Szenarios Luxemburg mit farblicher Unterscheidung von Straßen (schwarz), Gebäuden (rot) und Parkplätzen (grau)

## 3.2 Herangehensweise und Festlegungen

Auf beigefügtem Datenträger ist der gesamte implementierte Code einsehbar. Die folgenden Ausführungen beziehen sich rein auf Bezeichnungen einzelner Dateien, das entsprechende Verzeichnis der jeweiligen Datei zeigt Anhang C.

Unter VEINS wurden hauptsächlich die Klassen `BaseWaveApplLayer.cc`, `Mac1609_4.cc`, `TraCIDemo11p.cc` sowie `TraCIDemoRSU11p.cc` erweitert. Von Beginn an lag der Fokus bei der Implementierung auf einem Vergleich der in Kapitel 2 vorgestellten Generierungsmechanismen von BSMs und CAMs. VEINS beinhaltet den

Protokollstack nach WAVE (IEEE 802.11p, IEEE 1609.x). Ziel der Implementierung war keine 1:1-Nachbildung des ETSI-Protokollstacks (Abbildung 2.7). Hierfür gibt es bereits eine Implementierung mit Namen Artery [48]. Eine Verwendung von Artery war kurzzeitig angedacht, wurde allerdings nicht weiter verfolgt. Es sollten nur die relevanten Komponenten in VEINS integriert werden, die eine möglichst objektive Bewertung der Algorithmen (unabhängig von der zum Teil unterschiedlichen Struktur bei WAVE und ETSI hinsichtlich des Protokollstacks) ermöglichen. Auf Seiten von ETSI wurden die Klassen `BaseWaveApplLayer.cc`, `TraCIDemo11p.cc` sowie `TraCIDemoRSU11p.cc` mit der Network & Transport sowie Facilities-Schicht gleichgesetzt. Bei WAVE erfolgte die Implementierung des BSM-Generierungsmechanismus in der Klasse `BaseWaveApplLayer.cc`.

---

```

1 ...
2 *.node[*].beaconETSI = true
3 *.node[*].beaconWAVE = false
4 ...
5 *.node[*].appl.vCBPMeasInt = 0.1s
6 ...
7 *.*.*.nic.mac1609_4.NDL_defTxPower = 23dBm
8 ...

```

---

**Listing 3.1** – Auszug aus `omnetpp.ini`

Die Ausführung von Szenarien erfolgt anhand einer `omnetpp.ini`-Datei. Diese befindet sich im jeweiligen Unterverzeichnis des Verzeichnisses `examples`. In einer `omnetpp.ini`-Datei werden Startparameter, wie z.B. Dauer der Simulationszeit, sowie Parameter festgelegt, die in den C++-Klassen bei der Ausführung übernommen werden. Das genaue Prozedere ist in der Simulation Manual zu Omnet++ unter [7] einsehbar. Beide zu implementierenden Mechanismen verwenden eine Fülle von konstanten Parametern, die alle über die jeweilige `omnetpp.ini`-Datei gesetzt werden. Auf Seite von WAVE ist z.B. der Wert für `vCBPMeasInt` in [17], Tabelle 21, definiert. Bei ETSI sind alle Werte der NDL in [10], Kapitel A.4, festgelegt. Diese konstanten Werte wurden per `omnetpp.ini`-Datei gesetzt, um etwa zukünftige Änderungen schnell einzuflegen zu können bzw., um unterschiedliche Werte auswerten zu können.

Per `omnetpp.ini`-Datei ist es auch möglich festzulegen, ob der WAVE- oder ETSI-Mechanismus ausgeführt werden soll. Durch Setzen des entsprechenden Flags wird dies gewährleistet. In Listing 3.1 ist ein Auszug aus der `omnetpp.ini`-Datei mit entsprechenden Flags sowie o.g. Beispielwerten aufgeführt (node entspricht hier einem Fahrzeug). Von Vorteil hierbei ist, dass das gleiche Szenario nur durch Tauschen der Flag-Werte mit dem anderen Mechanismus ausgeführt werden kann. Nachteilig ist hierbei, dass der Code für WAVE und ETSI in den jeweiligen gleichen Klassen

enthalten ist, was allerdings aufgrund des Vorteils der einfacheren Ausführung von Szenarien in Kauf genommen wurde.

Folgend werden die erfolgten Erweiterungen von VEINS und die Umsetzung der Generierungsmechanismen beschrieben.

### 3.3 Integration der WAVE-BSM-Generierung in VEINS

#### 3.3.1 Erweiterung der MAC-Schicht

In der Klasse `Mac1609_4.cc` waren auf Seiten der WAVE-Implementierung wenige Anpassungen notwendig. Ab Zeile 89 folgt zunächst die Kanalallokierung wie in Abbildung 2.1 aufgezeigt. Angemerkt sei, dass der WAVE-Mechanismus nach [17] Kanal 172 für das Versenden von BSMs verwendet. VEINS ist grundsätzlich auf das Versenden von Beacons auf dem CCH ausgelegt. Dies wurde beibehalten. BSMs werden demnach auf dem CCH versendet, die Frequenz allerdings auf die von Kanal 172 gesetzt.

Im Anschluss werden die EDCA-Parameter wie in Tabelle 2.1 festgelegt. Von großer Bedeutung ist die Klasse `ChannelBusyListenerWave`, die wie in Abschnitt 3.3.2.1 beschrieben, die Zeiten erfasst, in denen der Kanal belegt ist. Eine Beschreibung erfolgt in Abschnitt 3.3.2.1. In der Methode `handleSelfMsg(...)` wird das Entnehmen der nächsten Nachricht aus der EDCA-Queue anhand des Timers `nextMacEvent` gesteuert. Es handelt sich um eine Standardfunktion von VEINS. Hinsichtlich der `txPower` erging hier eine Änderung, welche aus der jeweiligen Nachricht entnommen wird (Zeilen 658 bis 663).

#### 3.3.2 Erweiterung der Applikationsschicht

Hierfür erfolgte eine Erweiterung der drei Klassen `BaseWaveAppLayer.cc`, `TraCI-Demo11p.cc` und `TraCIDemoRSU11p.cc`. Auf Seiten von WAVE hat in diesen der Großteil der Implementierung stattgefunden. Innerhalb der `initialize(...)`-Methode wird der Parameter `beaconWAVE` aus der `omnetpp.ini`-Datei gelesen und dadurch alle relevanten Bestandteile für die BSM-Generierung ausgeführt. Nach der Initialisierung aller konstanten Werte wird die Startzeit berechnet, zu der ein Fahrzeug mit der Versendung der ersten BSM beginnt. Um möglichst Kollisionen von Nachrichten zu vermeiden, wurde hier eine zufällige Zeit zwischen 0.005 s und 0.095 s ermittelt und die aktuelle Zeit aufaddiert. Alle relevanten Timer für die Berechnung der benötigten Parameter (siehe Abbildung 2.4) werden anhand dieser Startzeit zum ersten Mal ausgeführt, beginnend mit dem initialen Versenden der BSM. Mit dem Begriff Timer sind im Folgenden SelfMessages gemeint, die durch Aufruf der Methode `scheduleAt(...)` zu einer bestimmten Simulationszeit innerhalb

eines Fahrzeuges "an sich selbst" versendet werden. Damit wird eine Timer-Methodik ermöglicht.

Innerhalb der `handleSelfMsg(...)`-Methode werden die Berechnungen ausgeführt, wenn der jeweilige Timer abgelaufen ist. Gemeint ist damit, dass die entsprechende SelfMessage zu diesem Zeitpunkt eintrifft. Die Unterscheidung der Timer geschieht per enum-Werten (anhand `msg->getKind()` abrufbar), welche folgend beschrieben werden. Die Reihenfolge ist dabei analog zur Ausführungszeit zu verstehen. *CBP\_MEASURE\_INTERVAL\_WAVE* wird zum Zeitpunkt des Timers als erstes ausgeführt, auch wenn andere Timer zur selben Zeit eingeplant sind. Zum besseren Verständnis ist es ratsam Abbildung 2.4 heranzuziehen.

***CBP\_MEASURE\_INTERVAL\_WAVE:*** Durchgeführt wird die Berechnung der *CBP* und die entsprechende Variable aktualisiert (Abschnitt 3.3.2.1).

***PER\_CALCULATION\_SUBINTERVAL\_WAVE:*** Die Berechnung der *PER*, des *Channel Quality Indicators* sowie der *Vehicle Density in Range* werden ermittelt (siehe hierzu analog *PER\_CALCULATION\_INTERVAL\_WAVE*). In Abschnitt 2.1.4.2 ist bereits angemerkt, dass es aus dem relevanten Standard nicht genau abzuleiten ist, ob die Bemessung dieser Werte erst nach 5 s erstmalig erfolgt oder bereits nach 1 s geschieht. Allgemein wurde erst nach 5 s die Berechnung der *PER* gestartet. Innerhalb der NED-Datei `BaseWaveApplLayer.ned` kann allerdings ein Flag gesetzt werden (`calcVeh`), womit zumindest die Berechnung der *Vehicle Density in Range* nach 1 s beginnt. Eine Erläuterung der durchgeföhrten Schritte folgt in Abschnitt 3.3.2.2. Damit die während einer Simulation erfasssten Werte möglichst objektiv betrachtet werden können, beginnt aufgrund der Unklarheiten bei diesem Berechnungsparameter die Aufzeichnung der Werte bei jedem Fahrzeug erst nach einer Zeit von 6 s.

***TX\_RATE\_CNTRL\_INTERVAL\_WAVE:*** Bei Ablauf dieses Timers werden *Tracking Error*, *Transmission Probability* und *MaxITT* berechnet sowie die *Transmission Decision* und *Schedule Transmission* ausgeführt (Abschnitt 3.3.2.3).

***SEND\_BEACON\_BSM:*** Es handelt sich um die Ausführung der Punkte 1.-4. aus Abbildung 2.4, wenn die aktuelle Simulationszeit mit der `NextScheduledMsgTime` übereinstimmt. Abschnitt 3.3.2.4 erläutert die hier erledigten Schritte.

### 3.3.2.1 Integrierung des Intervalls *vCBPMeasInt*

Der Timer *CBP\_MEASURE\_INTERVAL\_WAVE* Timer wird alle 100 ms wiederholt ausgeführt. Die Berechnung der *CBP* erfolgt mit Hilfe der Klasse `Mac1609_4.cc`. Diese nutzt einen in VEINS von Grund auf integrierten Signal-Mechanismus um alle Zeiten zu erfassen, in denen der Übertragungskanal belegt (busy) ist. Der Signal-Mechanismus ist in der Simulation Manual zu Omnet++ unter [7] genau beschrieben.

Um alle busy-Zeiten zu erfassen, verwendet die Klasse Mac1609\_4.cc ein Objekt der Klasse ChannelBusyListenerWave. Immer wenn auf MAC-Ebene festgestellt wird, dass der Kanal busy bzw. frei (idle) ist, wird die darin befindliche `receiveSignal(...)`-Methode aufgerufen und mithilfe der Datenstruktur BusyTime erfasst. Ein Vector verwaltet alle BusyTime-Objekte. Dieser Vector bewahrt alle Zeiten der letzten 6 s auf. Im Rahmen dieses Timers wird zur Ermittlung der CBP auf Ebene der Klasse BaseWaveAppLayer auf das auf MAC-Ebene befindliche Objekt der ChannelBusyListenerWave-Klasse zugegriffen und per `calculateChannelBusyPercentage(...)`-Methode der für dieses Intervall gültige Wert für CBP berechnet. Bei der Berechnung werden hierbei Zeiten, in denen der Kanal busy war, die innerhalb des jeweils relevanten Intervalls lagen, aufaddiert und analog zu Gleichung (2.2) berechnet. Auch Sonderfälle, in denen z.B. zum Zeitpunkt der Berechnung der Kanal busy ist, finden ebenso Berücksichtigung.

### 3.3.2.2 Integrierung der Intervalle `vPERInterval` und `vPERSubInterval`

Anhand der Timer `PER_CALCULATION_SUBINTERVAL_WAVE` und `PER_CALCULATION_SUBINTERVAL_WAVE` werden diese Berechnungsintervalle umgesetzt. So erfolgt die Berechnung der *PER*, des *Channel Quality Indicators* sowie der *Vehicle Density in Range*.

#### PER

Die Ermittlung der *PER*, des *Channel Quality Indicators* sowie der *Vehicle Density in Range* wird in eine eigene Klasse (`PERAndChannelQualityIndicatorCalculator`) ausgelagert (auf die Problematik, ob eine Berechnung schon nach 1 s oder erst nach 5 s erfolgt, wird an dieser Stelle nicht weiter eingegangen). Immer nach Ablauf dieses Timers wird zunächst die `calculatePER(...)`-Methode aufgerufen, welche alle aktuell gültigen *PERs* berechnet. Entscheidend für die Funktionalität der Methode ist, dass beim Eintreffen der BSMs anderer Fahrzeuge deren Sequenznummer erfasst wird (Abschnitt 2.1.4.2). In den Klassen `TraCIDemo11p` sowie `TraCIDemoRSU11p` geschieht dies in der Methode `onBeacon(...)`. Bei Eintreffen einer BSM wird die Methode `receivedBSM(...)` oben genannter Klasse aufgerufen.

Dort werden alle relevanten Werte aus der empfangenen BSM extrahiert und mithilfe einer eigenen Datenstruktur `carinfo` innerhalb einer Map verwaltet. Die schon erwähnte `calculatePER(...)`-Methode verwendet bei Aufruf diese Map und prüft zunächst, ob darin Werte enthalten sind, die älter als 5 s sind, und entfernt diese. Anschließend wird geprüft, ob von dem Fahrzeug, das die BSM versendete, nur eine Nachricht ankam. Falls dies der Fall ist, wird diese BSM gekennzeichnet, was zu einem (vorläufigen) Ausschluss aus der *PER*-Berechnung führt. Darauffolgend wird für jede in der Map vorhandene und noch nicht durch vorrangige Prüfungen

ausgeschlossene BSM geprüft, ob diese innerhalb der letzten Sekunde eintraf, was bei Verneinung auch zum Ausschluss aus der Berechnung führt. Zuletzt erfolgt noch eine Überprüfung der Distanz zwischen der Empfängerposition und der in jeder BSM enthaltenen Senderposition, was auch zum Ausschluss führt, falls eine Distanz von 100 m überschritten wird. Letztlich folgt für alle als gültig markierten BSMs eine Berechnung der jeweiligen PER. In Listing 3.2 ist dieser letzte Schritt veranschaulicht (vgl. hierzu Gleichung (2.3)).

---

```

1 for (iter = manageMap.begin(); iter != manageMap.end(); ↓
      iter++) {
2 if (iter->second.useInCalc) {
3     double numberOfWorkingBSM = 1 + (iter->second.lastmsgcount - ↓
                                         iter->second.msgcountFromFirstRecMsg);
4     double numberOfWorkedBSM = numberOfWorkingBSM - ↓
                                         iter->second.receivedMsgs;
5     double per = numberOfWorkedBSM / numberOfWorkingBSM;
6 ...
7     iter->second.perCalculated = per;
8 }
9 }
```

---

**Listing 3.2 – Berechnung der PER**

### Channel Quality Indicator

Unmittelbar nach der Berechnung aller *PERs* erfolgt innerhalb dieses Timers der Aufruf der Methode zur Ermittlung des *Channel Quality Indicators*, wozu von allen gültigen *PER*-Werten der Durchschnitt errechnet wird. Dabei wird überprüft, dass ein Wert von 0,3 (*vPERMax*) nicht überschritten wird.

### Vehicle Density in Range

Nach Berechnung des *Channel Quality Indicators* wird auch der Wert der *Vehicle Density in Range* ermittelt. Der zugehörige Standard [17] ist in diesem Zusammenhang nicht präzise. Die Umsetzung erfolgte, dass die Map mit den *PER*-Werten herangezogen wird. Alle darin befindlichen Fahrzeuge, deren letzte BSM in der letzten Sekunde eintraf und sich innerhalb einer Distanz von 100 m befinden, werden gezählt. Vorher ausgeschlossene BSM sind bei dem wiederholten Ablauf dieses Timers neu zu prüfen.

### 3.3.2.3 Integrierung des Intervalls *vTxRateCntrlInt*

Der Timer *TX\_RATE\_CNTRL\_INTERVAL\_WAVE* wird alle 100 ms ausgeführt und berechnet in fester Reihenfolge die Parameter *Tracking Error*, *Transmission Probability*, *MaxITT*, *Transmission Decision* und *Schedule Transmission*.

#### Tracking Error

Die Ermittlung des *Tracking Errors* wurde in eine eigene Klasse ausgelagert. Von einem Objekt der Klasse *TrackingErrorCalculator* wird hierfür die Methode *calculateTrackingError* aufgerufen. Grundsätzlich ist der in Algorithmus 2.2 veranschaulichte Algorithmus analog auszuführen. Zu erwähnen ist, dass bei jedem Senden einer BSM geprüft wird, ob der *lastHVStatus* anzupassen ist. Damit ist die BSM gemeint, von der man ausgeht, dass diese bei umliegenden Fahrzeugen (RV) angekommen ist. Hierauf wird in Abschnitt 3.3.2.4 noch genau eingegangen. Bei der Distanzberechnung wird eine Methode der *Coord*-Klasse verwendet, die von Grund auf in VEINS vorhanden ist. Im Rahmen der Berechnung des *Tracking Errors* muss auch in Abhängigkeit der zeitlichen Differenz zwischen der Zeit, an dem die letzte Position gemessen wurde und der aktuellen Zeit ermittelt werden, welche Position zum jetzigen Zeitpunkt schon erreicht ist bzw. sein müsste. Dafür wurde auf die Methode *getPositionAt(...)* der Klasse *Move* zurückgegriffen, welche auch standardmäßig in VEINS vorhanden ist. Damit kann durch Übergabe der zeitlichen Differenz, addiert mit der Zeit der letzten Positionsmeßung, die Position ermittelt werden, die in Abhängigkeit von Geschwindigkeit und Fahrtrichtung zum entsprechenden Zeitpunkt erreicht sein müsste. Die in [17], Kapitel A.2, beschriebenen Methoden fanden deshalb keine Anwendung.

#### Transmission Probability

Für die Berechnung der *Transmission Probability* wurde die Klasse *TransmissionProbabilityCalculator* implementiert. Im Anschluss an Ermittlung des *Tracking Errors* erfolgt ein Aufruf der Methode *calculateTransmissionProbability(...)*. Darin geschieht die Kalkulation analog zu Gleichung (2.6).

#### MaxITT

Analog zu den Berechnungen zuvor ist dafür eine eigene Klasse erstellt worden. Die Klasse *MaximumInterTransmitTimeCalculator* errechnet durch Aufruf der Methode *calculateMaximumInterTransmission(...)* den Wert in ms wie in Gleichung (2.8) definiert. Innerhalb genannter Methode wird für die Bemessung der *Smooth Vehicle Density in Range* immer der zuletzt berechnete Wert der *Vehicle*

*Density in Range* (siehe Abschnitt 3.3.2.2) verwendet, wovon der errechnete Wert für *MaxITT* letztlich abhängt.

### Transmission Decision

Für die *Transmission Decision* ist die `performTransmissionDecision(...)`-Methode vorhanden. Zunächst werden bei jedem Aufruf die drei `bool`-Variablen `txDecision_Dynamics`, `txDecision_Critical_Event` und `txDecision_Max_ITT` auf `false` gesetzt. Für die Bestimmung von `txDecision_Dynamics` ermittelt ge nannte Methode zunächst einen Zufallswert anhand einer Bernoulli-Funktion. Bei dieser wird als Wert entweder 0 oder 1 zurückgegeben. Die Flags erhalten anschlie ßend Werte wie in Algorithmus 2.3 beschrieben. Als `NextScheduledMsgTime` wird die nächste Ausführungszeit des Timers *SEND\_BEACON\_BSM* verwendet.

### Schedule Transmission

Zum Abschluss des Timers *TX\_RATE\_CNTRL\_INTERVAL\_WAVE* erfolgt die *Schedule Transmission* (Algorithmus 2.4). Hierbei wird geprüft, ob eines der unmittelbar zuvor gesetzten Flags der *Transmission Decision* den Wert `true` hat. Falls dies zutrifft, geschieht die Zurücksetzung und sofortige Ausführung des Timers *SEND\_BEACON\_BSM*. Dies führt zum sofortigen Versand einer BSM (Abschnitt 3.3.2.4). Sind alle drei Flags `false`, geschieht keine Änderung.

#### 3.3.2.4 Generierung und Versendung einer BSM

Der Timer *SEND\_BEACON\_BSM* wird immer ausgeführt, wenn die aktuelle Simulationszeit gleich der `NextScheduledMsgTime` ist. Dabei werden die Punkte 1. bis 4. aus Abbildung 2.4 der Reihe nach erledigt.

### Radiated Power

Zu Beginn erfolgt die Berechnung der *Radiated Power* für die zu versendende BSM. Dies geschieht in der Methode `calculateRadiatedPower()`. Zunächst wird geprüft, ob eines der Flags `txDecision_Dynamics` oder `txDecision_Critical_Event` den Wert `true` hat. Dies kann nur auftreten, wenn in der *Schedule Transmission* (Abschnitt 3.3.2.3) eines dieser Flags schon den Wert `true` hatte. Wenn es zutrifft, wird die Variable `rp` auf den maximalen Wert von 20 dBm gesetzt. Ist dies unzutreffend, erfolgt die Berechnung von `rp` analog zu Algorithmus 2.5. Bei der Implementierung wurde von einer verlustfreien Antenne ausgegangen, wie in Abschnitt 2.1.4.8 bereits thematisiert. Mit Blick auf Abbildung 2.4 wurde hier der im unteren Teil aufgezeigte 1. Punkt implementiert.

### Letzter HVStatus und Versendung der BSM

Innerhalb der Methode `assumptionOfLatestHVStatusAndSendBSM()` erfolgt der in Abbildung 2.4 im unteren Teil aufgeführte 2. und 3. Punkt. Sollte noch keine BSM versendet worden sein, wird unmittelbar die Methode `sendBeaconBSM(true)` aufgerufen. Der Parameter `true` führt hierzu, dass die für die Berechnung des *Tracking Errors* (Abschnitt 3.3.2.3) relevanten Daten der BSM als letzter HVStatus zwischengespeichert werden. In der `sendBeaconBSM(...)`-Methode erfolgt nach der Versendung auch die Zuweisung der aktuellen Zeit der Variable `lastTxTime`. Ebenso wird die Variable `txFailed` aktualisiert. Dies geschieht in der Methode `updateTxFailed`, in der anhand eines Zufallswertes (mit Hilfe einer Bernoulli-Funktion ein Wert von 0 oder 1) und des *Channel Quality Indicators* eine Aktualisierung erfolgt (wie in Algorithmus 2.6, Zeilen 3 bis 8). Liegt der Wert von `txFailed` über 0 und ist maximal 3, wird davon ausgegangen, dass die zu versendende BSM bei umliegenden RVs nicht eintreffen wird, weshalb die `sendBeaconBSM(false)`-Methode mit `false` als Parameter aufgerufen wird, andernfalls geschieht der Aufruf mit `true`. In beiden Fällen folgt die Generierung und Versendung einer BSM. Die zuvor berechnete *Radiated Power* wird hierbei der BSM beigefügt. Nur bei einem Aufruf mit `true` als Parameter führt die Methode eine Zwischenspeicherung der generierten BSM durch.

### Berechnung der NextScheduledMsgTime

Wie in Abbildung 2.4 als 4. Punkt aufgezeigt, geschieht zuletzt die Berechnung der `NextScheduledMsgTime`. Diese ist gleichbedeutend mit der Zeit, wann dieser Timer wiederholt auszuführen ist. Die Ermittlung erfolgt analog zu Algorithmus 2.6, Zeile 14.

## 3.4 Integration der ETSI-CAM-Generierung inklusive DCC in VEINS

Analog Abschnitt 3.3 folgen nun die im Rahmen der Implementierung vorgenommenen Erweiterungen von VEINS-Dateien und eigens erstellten Dateien. Auch hier wird sich an den verwendeten Timern orientiert, damit der Kontrollfluss nachvollziehbar ist. Eine Unterteilung erfolgt allerdings anhand der drei Varianten, die in Abschnitt 2.2.4.2 beschrieben wurden: DCC mit Standardparametern, DCC Profile sowie der Gatekeepingansatz mit LIMERIC. Bei den beiden letztgenannten werden im Unterschied zur Standardvariante die Paketintervalle in den jeweiligen Zuständen anders festgelegt. Bei der Vorstellung der Implementierung dieser Varianten wird nur auf die Umsetzung der Unterschiede zur Standardvariante eingegangen. Die Ausführungsvariante eines Szenarios kann durch die Parameter `useDCCProfileCAM`

(DCC Profile) bzw. `useGateKeeping` (LIMERIC) per `omnetpp.ini` oder innerhalb von `Mac1609_4.ned` gesteuert werden.

### 3.4.1 Erweiterung der MAC-Schicht

Innerhalb der Klasse `Mac1609_4.cc` wurde die DCC installiert. Ab Zeile 150 beginnt der Kontrollfluss mit der Festlegung der Frequenzen aller Kanäle, wie in Abschnitt 2.2.1. Darauffolgend werden alle NDL-Parameter aus der `omnetpp.ini` gelesen. Anschließend werden in Zeilen 222 bis 244 die Parameterwerte für die jeweiligen Zustände festgelegt. Es erfolgt hier eine Vereinfachung: Nur die Werte finden Berücksichtigung, die für CAMs relevant sind (siehe Tabelle 2.6). Für die Verwaltung der Werte wurde eine eigene Datenstruktur `State` verwendet. Die Handhabung erläutern die folgenden Abschnitte. Ab Zeilen 247 bis 261 erfolgt die Übernahme der EDCA-Parameter wie in Tabelle 2.3 aufgezeigt. Für die Umsetzung der Zustandsmaschine der DCC werden folgende Timer verwendet (folgend werden die Variablen-Namen verwendet):

**timer\_NDL\_channelLoad:** Dieser Timer wird alle 100 ms ausgeführt und berechnet die CBR der letzten 100 ms (Abschnitt 3.4.1.1).

**timer\_NDL\_timeUp:** In den Zuständen RELAXED und ACTIVE ist dieser Timer notwendig, um nach jeder Sekunde zu prüfen, ob ein Wechsel von RELAXED zu ACTIVE oder von ACTIVE zu RESTRICTIVE erfolgen muss (siehe Abschnitt 3.4.1.1).

**timer\_NDL\_timeDown:** In den Zuständen ACTIVE und RESTRICTIVE muss nach 5 s geprüft werden, ob ein Wechsel von ACTIVE zu RELAXED bzw. von RESTRICTIVE zu ACTIVE erfolgen kann. Die genaue Funktionsweise ist in Abschnitt 3.4.1.1 beschrieben.

**timer\_DCC\_Toff:** Bei Ankunft eines Paketes einer höheren Schicht auf MAC-Ebene erfolgt zunächst eine Einreichung des Paketes in die DCC Transmit Queue. Dabei sind die maximale Länge der Queue, das Paketintervall sowie die Lifetime eines Paketes von Bedeutung. Diese regeln, wann ein Paket in die eigentliche EDCA-Queue eingereiht werden darf (Abschnitt 2.2.4.2). Die Implementierung hierzu ist in Abschnitt 3.4.1.1 erläutert.

Beim Eintreffen des Timers erfolgt die Bearbeitung in der `handleSelfMsg`-Methode ab Zeilen 567 bis 876.

#### 3.4.1.1 DCC mit Standardparamter

In Zeile 291 geschieht die Initialisierung der Zustandsmaschine durch die Methode `setDCCState(...)`. Abhängig vom jeweiligen aktuellen Zustand erfolgt die Fest-

legung der Parameter und ein eventueller Zustandswechsel. Dies wird durch die Methode `setStateParams(State & state)` durchgeführt. Bei State handelt es sich um eine Datenstruktur, die die in Tabelle 2.6 aufgeführten konstanten Werte verwaltet. Immer wenn ein Zustandswechsel erfolgt, kommt diese Methode zum Einsatz. In Listing 3.3 ist der entsprechende Code dieser Methode abgebildet.

---

```

1 void Mac1609_4::setStateParams(State & state) {
2     currentState = &state;
3     this->setCCAThreshold(currentState->carrierSense);
4     this->setTxPower(FWMath::dBm2mW( currentState->txPower));
5     bitrate = currentState->datarate;
6     NDL_refPacketInterval = currentState->packetInterval;
7     ...
8     BaseWaveApplLayer * appLayer = ↓
9         FindModule<BaseWaveApplLayer*>::↓
10        findSubModule(getParentModule()-> getParentModule());
11        //set T_GenCam_DCC in Appl-Layer
12        appLayer->setPaketInterval( NDL_refPacketInterval , false);
13 }
```

---

**Listing 3.3 – Methode `setStateParams`**

Hier werden die entsprechenden Parameter innerhalb der Klasse `Mac1609_4` gesetzt. Bei der Generierung von CAMs in der Klasse `BaseWaveApplLayer` werden von dieser Stelle die `txPower` und `bitrate` abgerufen und jeder CAM beigefügt. Das Verfahren beschreibt Abschnitt 3.4.2.

#### Berechnung der Kanallast (CBR) mit dem Timer `timer_NDL_channelLoad`

Alle 100 ms wird dieser Timer ausgeführt und innerhalb der `handleSelfMsg(...)`-Methode ab Zeile 739 abgearbeitet. Anhand eines Objektes der Klasse `ChannelLoadListenerEtsi` und dessen Methode `calculateChannelLoad(...)` wird die Kanallast der letzten 100 ms berechnet. Es erfolgt eine Verwaltung aller gemessenen Zeitintervalle, in denen der Kanal belegt war. In der Map `addedTimes` geschieht dies, welche im Rahmen der Timer `timer_NDL_timeUp` und `timer_NDL_timeDown` für die Erkennung von Zustandsübergängen Anwendung findet. Falls die Gatekeepervariante mit LIMERIC ausgeführt wird, erfolgt in diesem Timer auch die Aktualisierung des Parameters `T_GenCam_DCC` in der Klasse `BaseWaveApplLayer`, was im Rahmen der CAM-Generierung in Abschnitt 3.4.2 eine Rolle spielt.

#### Prüfung eines Zustandswechsels mit dem Timer `timer_NDL_timeUp`

In den Zuständen `RELAXED` und `ACTIVE` ist jeweils nach einer Sekunde zu prüfen, ob innerhalb der letzten Sekunde eine Kanallast von mindestens 15% gemessen

wurde. Dafür ist es notwendig, die in Abschnitt 3.4.1.1 erklärte Klasse heranzuziehen. Es müssen alle gemessenen CBR-Werte entsprechend geprüft werden. Dafür wird die Methode `getMinChannelLoad(...)` genutzt. Hierin werden anhand der Map `addedTimes` die aufgezeichneten Zeiten durchsucht und der minimale Wert der CBR der letzten Sekunde zurückgegeben.

Anschließend folgt ein Aufruf der Methode `setDCCState(channelLoad, false)`. Der Übergabeparameter `channelLoad` ist der minimal gemessene Wert der CBR der letzten Sekunde. Der zweite Übergabeparameter ist dazu da, innerhalb der Methode zu erkennen, ob der Aufruf durch den Timer `timer_NDL_timeUp` (Parameter ist `false`) oder den Timer `timer_NDL_timeDown` (Parameter ist `true`) erfolgt ist. Innerhalb besagter Methode ist zu prüfen, welcher Zustand aktuell gesetzt ist. Danach erfolgt die Überprüfung des Parameters `channelLoad`. Befindet man sich z.B. aktuell im Zustand ACTIVE und diese Methode wird im Rahmen von `timer_NDL_timeUp` aufgerufen, ist zu kontrollieren, ob `channelLoad` größer oder gleich 40% ist. Diese Prüfung geschieht in den Zeilen 477 bis 526. Ist dies der Fall, wird der Zweig in Zeile 492 eingeschlagen und es folgt ein Zustandsübergang in den Zustand RESTRICTIVE. Alle weiteren Zustandsübergänge, wie in Abbildung 2.12 beschrieben, werden analog geprüft.

#### Prüfung eines Zustandswechsels mit dem Timer `timer_NDL_timeDown`

Dieser Timer ist nur in den Zuständen ACTIVE und RESTRICTIVE aktiv. Es erfolgt hier der gleiche Ablauf, wie bei Timer `timer_NDL_timeUp`. Nur ist es hier relevant die Methode `getMaxChannelLoad(...)` der Klasse `ChannelLoadListenerEtsi` zu verwenden, um die maximale CBR der letzten fünf Sekunden ermitteln zu können. Die Prüfung in der Methode `setDCCState(...)` ist ebenso gleich. Ist beispielsweise der aktuelle Zustand RESTRICTIVE und nach fünf Sekunden wird eine maximale CBR von 30% festgestellt, erfolgt in den Zeilen 528 bis 554 ein Zustandswechsel in den ACTIVE-Zustand.

#### Entnahme eines Paketes aus der DCC Transmit Queue mit dem Timer `timer_DCC_Toff`

Verwendet wird dieser Timer, wenn eine CAM (zum ersten Mal) versendet werden soll. Zunächst ist hier die Methode `handleUpperMsg` (von Grund auf in VEINS vorhanden) wichtig. Trifft von der Applikationsschicht (`BaseWaveApplLayer`) eine CAM ein, erfolgt zunächst keine Einsortierung in die entsprechende EDCA-Queue. Hier wurde eine DCC-Queue angelegt, die die in Abschnitt 2.2.4.2 genannte DCC Transmit Queue darstellt. Zur Vereinfachung wurde nur eine einzelne Queue genutzt, da im Rahmen der Arbeit nur CAMs versendet werden und somit nur eine DCC Transmit Queue für die AC\_BE benötigt wird. In Zeilen 967 bis 980 tritt nun eine

Einsortierung in die DCC Transmit Queue ein. Allerdings wird die in [10], Tabelle 9, definierte QueueLength von zwei berücksichtigt. Sollte diese Maximallänge bereits erreicht sein, findet keine weitere Einreihung statt und das Paket wird verworfen. Ist dies nicht der Fall, wird das Paket eingereiht und der Timer `timer_DCC_Toff` unmittelbar danach gestartet.

Der betreffende Timer prüft zu Beginn das Paketintervall, ob es sich im gültigen Bereich zwischen 100 ms und 1 s befindet. Innerhalb dieses Intervalls darf nur ein Paket aus der DCC Transmit Queue entnommen und in die eigentliche EDCA-Queue eingereiht werden. Diese Prüfung geschieht in Zeile 764. Danach wird das erste Paket aus der Queue auf dessen Lebenszeit überprüft. Die gültige Lebenszeit beträgt eine Sekunde. Falls diese überschritten wurde, wird das Paket verworfen und das nächste Paket entnommen. Diese Prozedur wird solange fortgeführt, bis die Queue leer ist oder ein Paket seine Lebenszeit nicht überschritten hat und somit in die EDCA-Queue eingereiht werden darf, was in Zeilen 764 bis 860 erfolgt. Dort wird auch der Timer `nextMacEvent` zur entsprechenden Zeit festgelegt, wann ein Paket aus der EDCA-Queue versendet wird. Dies ist von Grund auf in VEINS vorhanden und wurde insoweit angepasst, indem die `txPower` und `bitrate` aus der CAM vor der Weiterleitung an die PHY-Schicht gelesen werden.

#### 3.4.1.2 DCC Profile

Bei dieser Variante stand eine andere Belegung der Werte für das Paketintervall an. In Tabelle 2.7 sind diese beschrieben. Ansonsten gibt es zu der in Abschnitt 3.4.1.1 beschriebenen Implementierung inklusive aller aufgeführten Timer keine Unterschiede. Nur in den Zeilen 233 bis 244 wird das Paketintervall entsprechend angepasst.

#### 3.4.1.3 LIMERIC

Auch hier ist das Paketintervall mit anderen Werten zu belegen. Allerdings wird das jeweils gültige Paketintervall, wie in Abschnitt 2.2.4.2 beschrieben, dynamisch in Abhängigkeit verschiedener Kanallastwerte immer im Anschluss an die Berechnung der aktuellen CBR berechnet. Relevant hierfür ist der Timer `timer_NDL_channelLoad`, der wie in Abschnitt 3.4.1.1 erläutert, alle 100 ms die CBR berechnet.

Innerhalb der Methode `calculateChannelLoad` der Klasse `ChannelLoadListener-Etsi` wird in den Zeilen 211 bis 215 die Kalkulation der `Ego_Target_Rate` durchgeführt, womit das aktuell gültige Paketintervall (`T_GenPacket_DCC`) anhand Gleichung (2.15) berechnet wird. Innerhalb der Methode `calculateTargetRate` erfolgt die Ermittlung analog zu Gleichung (2.14). Alle benötigten Kanallastwerte werden in der Datenstruktur `lastCbrValues` bzw. `CBRValues` vom Typ `cbrValues` verwaltet. Der Unterschied zwischen diesen beiden Datenstrukturen ist folgender. In `lastCbrValues` sind die zuletzt am Ende des Timers `timer_NDL_channelLoad` be-

rechneten Werte enthalten. Diese sind relevant beim Versenden einer CAM. Zwischen den Berechnungsintervallen werden die maximalen Werte mit `cbrValues` verwaltet. Bei eintreffenden CAMs muss immer ein Vergleich stattfinden zwischen dem schon aktuell gesetzten Wert und dem empfangenen Wert, der maximale ist entsprechend zu übernehmen.

Beim Versenden einer CAM in der Klasse `BaseWaveApplLayer` werden in der Methode `sendBeaconCAM` die zuletzt berechneten Werte der `CBR_L_0_Hop` und `CBR_L_1_Hop` anhand `lastCbrValues` abgerufen und der CAM beigefügt. Bei Empfang einer CAM wird in den Klassen `TraCIDemo11p` bzw. `TraCIDemoRSU11p` die Methode `onBeacon` aufgerufen. Dort wird aus der jeweiligen CAM der Wert für `CBR_R_0_Hop` und `CBR_R_1_Hop` ausgelesen und immer der maximale Wert `CBR_L_1_Hop` bzw. `CBR_L_2_Hop` innerhalb der Datenstruktur `cbrValues` zugewiesen (siehe hierzu Tabelle 2.8). Aus den genannten Werten erfolgt nach der Berechnung der lokalen CBR die Ermittlung der `CBR_G` und unmittelbar danach die Berechnung der `Ego_Target_Rate`. Aus letzterer kann das eigentliche Paketintervall `T_GenPacket_DCC` abgeleitet werden (siehe Zeilen 261 bis 270 in der Klasse `ChannelLoadListener-Etsi`).

Letztlich wird innerhalb des Timers `timer_NDL_channelLoad` das zuvor berechnete Paketintervall an die Klasse `BaseWaveApplLayer` weitergeleitet, wenn das `useGateKeeping`-Flag gesetzt ist. Dort beeinflusst es als `T_GenCam_DCC` die CAM-Generierung. Zusätzlich wird es auch analog, wie in Abschnitt 3.4.1.1 beschrieben, als Gatekeeper für die Paketentnahme aus der DCC Transmit Queue verwendet.

### 3.4.2 Erweiterung der Applikationsschicht

In der Klasse `BaseWaveApplLayer.cc` geschahen individuelle Anpassungen. Alle notwendigen Parameter werden wie in den zuvor beschriebenen Klassen aus der `omnetpp.ini` gelesen und zu Beginn initialisiert. Die Zeit, zu der das Versenden des ersten Beacons startet, wird analog zu dem WAVE-Mechanismus festgelegt (siehe Abschnitt 3.3).

Diese Klasse ist auf Seiten von ETSI für die CAM-Generierung zuständig, wofür wie gehabt Timer benutzt werden. Die beiden relevanten Timer werden folgend vorgestellt (Benennung erfolgt hier wie bei WAVE anhand der `msg->getKind()`-Methode, Zeilen 233 bis 250).

#### 3.4.2.1 Prüfung der Änderung von dynamischen Fahreigenschaften

Die Ausführung des Timers `CHECK_CAM_PARAMS` verläuft alle 100 ms. In der Methode `performCAMGenerating` wird analog zu Algorithmus 2.7 vorgegangen. In den Zeilen 184 bis 228 wird zunächst geprüft, ob die seit der letzten Versendung einer CAM vergangene Zeit mindestens `T_GenCam_DCC` ist. Dabei handelt es sich um das

auf MAC-Ebene festgelegte Paketintervall, welches bei Zustandswechseln immer aktualisiert wird. Es findet auch immer eine Prüfung statt, ob es größer oder gleich 100 ms und kleiner oder gleich 1 s ist. Trifft es zu, dass *T\_GenCam\_DCC* mindestens der vergangenen Zeit seit der letzten versendeten CAM entspricht, geschieht die Prüfung der dynamischen Fahreigenschaften (Geschwindigkeit, Lenkeinschlag, Position) wie in Abschnitt 2.2.5 beschrieben. Es wird bei der Berechnung auf Methoden der Klassen Coord und Move zurückgegriffen, welche von Grund auf in VEINS vorhanden sind. In Zeilen 207 bis 225 wird die zweite Condition nach [38], Kapitel 6.1.3, geprüft. Werden zwei weitere CAMs im gleichen *T\_GenCam*-Intervall versendet, ohne dass zuvor das Eintreten einer der dynamischen Eigenschaften festgestellt wurde, wird *T\_GenCam* auf 1 s gesetzt und somit die CAM-Generierung gedrosselt.

#### 3.4.2.2 Generierung und Versenden einer CAM

Die erste CAM wird zur *startBeaconingTime* anhand des Timers *SEND\_BEACON\_CAM* versendet. Dies ist notwendig, da bei der CAM-Generierung anhand der zuletzt versendeten CAM geprüft wird, ob sich Änderungen in Geschwindigkeit, Lenkeinschlag oder Position ergeben haben. Ansonsten wird dieser Timer ausschließlich aus der Methode *performCAMGenerating* während des Timers *CHECK\_CAM\_PARAMS* festgelegt. Nur wenn eine der beiden Conditions (wie in Abschnitt 3.4.2.1 beschrieben) zutrifft, wird eine CAM mit Hilfe dieses Timers versendet. Wird dieser Timer ausgeführt, erfolgt die Versendung einer CAM und deren Zwischenspeicherung. Zusätzlich wird der *lastTxTime* die aktuelle Zeit zugewiesen.

## 3.5 Testen der Implementierung

Für die Verifikation der Implementierung fanden nach allen Implementierungsschritten regelmäßig Testläufe anhand eines Szenarios statt. Damit die Testumgebung soweit wie möglich der echten Simulationsumgebung entspricht, galt folgendes Konzept.

Der komplette Sourcecode wurde gespiegelt. Auf beigefügtem Datenträger (Anhang C) ist das Verzeichnis *testVeins* zu betrachten. Es handelt sich um den gleichen Code, der für die Ausführung der Szenarien verwendet wurde. Allerdings ist auf Seiten von WAVE und ETSI in dem Verzeichnis *examples* je ein Testszenario vorhanden und der Sourcecode wurde so angepasst, dass anhand der *omnetpp.ini* bestimmte Testfälle innerhalb der Original-Simulationsumgebung ausgeführt werden können. Eine Übersicht aller hierfür erstellter Testfälle ist in Anhang A aufgeführt. Innerhalb der *omnetpp.ini* kann z.B. durch Angabe von *\*.\*.\*.testCaseNumber = 1.1* der Testfall mit der Nummer 1.1 ausgeführt werden. Sollte ein Testfall fehlschlagen, wird eine Exception geworfen und die Simulation

wird vorzeitig abgebrochen. Ansonsten läuft die Simulation ohne Fehler bis zu dem vorgesehenen Ende nach einer Simulationszeit von 100 s.

Mit dieser Herangehensweise war es möglich auf Seiten von WAVE alle notwendigen Gleichungen zur Berechnung der Parameter zu testen. Auf Seiten von ETSI wurde insbesondere das Verhalten der Zustandsmaschine sowie die Berechnungen der dynamischen Fahreigenschaften (Unterschiede in Geschwindigkeit, Position bzw. Lenkeinschlag) geprüft.

Darüber hinaus wurden alle berechneten Parameter innerhalb der Simulation aufgezeichnet (mit Omnet++ Vektoren und Skalaren, siehe Simulation Manual unter [7]). Beispielsweise werden bei der Berechnung der CBR bzw. CBP alle Zeiten erfasst, in denen der Kanal belegt war, und aufaddiert. Dies geschieht auf Seiten von ETSI in der Klasse Mac1609\_4 und der entsprechende Wert kann in der Skalar-Statistik mit Namen TotalChannelLoadTime eingesehen werden. Auf Seite von WAVE wird dieser Wert in der Klasse BaseWaveApplLayer aufaddiert und dort als Skalar-Wert mit Namen TotalBusyTime geführt. Von Grund auf wird in VEINS dieser Wert in der Klasse Mac1609\_4 unter dem Skalar-Namen totalBusyTime aufgezeichnet. Der Vergleich von Stichproben bei Testläufen mit der Parametereinstellung `*.**.testCaseNumber = 0` hat gezeigt, dass diese Skalar-Werte identisch waren. Aufgrund dieser Erkenntnisse wird von der korrekten Berechnung der Kanallastzeiten ausgegangen.

---

## Kapitel 4

# Leistungsbewertung beider Mechanismen

---

In diesem Kapitel erfolgt eine ausführliche Evaluierung der implementierten Mechanismen durch Simulation verschiedener Szenarien. Zunächst werden die allgemeinen Erkenntnisse, die bereits nach der Implementierung und unabhängig von Simulationsergebnissen auffielen, zusammengefasst. Im Anschluss erfolgt eine Beschreibung der verwendeten Szenarien sowie die für eine möglichst objektive Bewertung beider Mechanismen festgelegten Kennwerte, die anhand von Boxplots miteinander verglichen werden. Danach werden individuelle Auffälligkeiten aufgezeigt und diskutiert.

### 4.1 Erste Erkenntnisse nach der Implementierung

Unabhängig von der Betrachtung der statistischen Auswertung der Szenarien waren nach erfolgter Implementierung bereits mehrere - zum Teil gravierende - Unterschiede zwischen WAVE und ETSI aufgefallen. Ein Vergleich erfolgte insbesondere anhand der Parameter, die auf Seiten von ETSI bei der DCC verwendet werden (Abschnitt 2.2.4.2). Ein Vergleich der Parameter ist in Tabelle 4.1 aufgezeigt.

Zusätzlich ist auffallend, dass auf Seiten von WAVE der Wert für die Kanallast CBP nur für die Berechnung der *Radiated Power* herangezogen wird und nicht etwa für eine Regulierung weiterer Parameter, wie auf Seiten von ETSI. Dort stellt die Kanallast in Form der CBR den entscheidenden Messwert dar, an dem die weitere Festlegung der Parameter durch die DCC gebunden ist.

Parameter	WAVE	ETSI
TxPower	10 dBm bis 20 dBm	-10 dBm, 20 dBm oder 23 dBm
Datenrate	6 Mbit/s	3 Mbit/s oder 12 Mbit/s
Empfangsempfindlichkeit	-92 dBm	-65 dBm, -85 dBm oder -95 dBm
Generierungsintervall	100 ms bis 600 ms	100 ms bis 1000 ms
Default-Generierungsintervall	100 ms	1000 ms

**Tabelle 4.1 – Allgemeiner Vergleich beider Mechanismen [10] [17]**

## 4.2 Vorgehensweise

Ausgangspunkt für die Evaluierung war das Ziel einer möglichst objektiven Gegenüberstellung beider Mechanismen. Individuelle Eigenschaften der Mechanismen - z.B. das Zustandsverhalten der DCC bei ETSI im Detail oder die genaue Betrachtung der berechneten Parameter bei WAVE - sind für einen objektiven Vergleich ungünstig. Deshalb wurden die folgenden Festlegungen getroffen.

### Auswahl der Szenarien

Der Fokus bei der Auswahl der Szenarien lag somit vorrangig nicht auf individuellen Eigenschaften der jeweiligen Mechanismen. Von Interesse war das Verhalten nicht nur innerhalb eines bestimmten Szenarios. Insbesondere, weil bei ETSI die CAM-Generierung von sich dynamisch ändernden Parametern wie Geschwindigkeit sehr stark beeinflusst wird, wurden drei Szenarien betrachtet:

- Autobahnkreuz: Charakteristisch für dieses Szenario sind hohe Geschwindigkeiten und geringer Lenkeinschlag. Eine genaue Beschreibung folgt in Abschnitt 4.3.1.
- Stadt: Niedrige Geschwindigkeiten, Standzeiten, sowie häufige Änderungen des Lenkeinschlages sind repräsentativ für dieses Szenario. Eine genaue Beschreibung folgt in Abschnitt 4.3.2.
- Stau: Wie sich die beiden Mechanismen in einem Szenario verhalten, in dem sich alle Fahrzeuge nicht bewegen, wird mit diesem Szenario bewertet. Eine genaue Beschreibung folgt in Abschnitt 4.3.3.

Innerhalb dieser Szenarien war es von Interesse bestimmte Kennwerte während der Simulation aufzuzeichnen und gegenüberzustellen. Dabei sollte es einerseits möglich sein, Rückschlüsse auf das Verhalten bei Kanallast zu ziehen, andererseits sollte auch eine allgemeine Bewertung des Beaconing abgeleitet werden können.

#### Festlegung der zu vergleichenden Kennwerte

Um die zuvor genannten Punkte zu erfassen, wurden insgesamt fünf Kennwerte in allen Szenarien aufgezeichnet und näher betrachtet. Eine Gegenüberstellung dieser Werte erfolgt innerhalb der jeweiligen Szenarien (Abschnitt 4.3) anhand von Boxplot-Grafiken. Alle in Anhang B veranschaulichten Boxplot-Grafiken sind auch auf dem beigefügten Datenträger einsehbar (Anhang C). Als relevante Kennwerte wurden folgende betrachtet:

**Generierungsintervall:** Hiermit ist die Periode zwischen dem Versenden von zwei aufeinanderfolgenden Beacons gemeint. Auf Seiten von ETSI wird es durch die Kalkulation der aktuellen Zeit beim Versenden eines Beacons abzüglich der letzten Zeit, zu der ein Beacon versendet wurde, berechnet. Bei WAVE wird die nach jedem versendeten Beacon neu berechnete *NextScheduledMsgTime* hierfür herangezogen.

**gesendete Beacons:** Hierbei werden die insgesamt versendeten Beacons innerhalb einer bestimmten Simulationsausführung angesprochen.

**empfangene Beacons:** Hier werden alle empfangenen Beacons während einer Simulation adressiert. Für sicherheitsrelevante Applikationen ist ein hoher Wert von Vorteil, was allerdings beim Empfänger auch zu einer hohen Datenverarbeitung der empfangenen Daten führen kann.

**Kanallast (CBR):** Anhand dieses Wertes soll eine Aussage über die jeweilige Congestion Control ermöglicht werden. Bei beiden Mechanismen wird die CBR nach Gleichung (2.10) ermittelt und kann somit direkt gegenübergestellt werden. Konstant hohe Werte in den Grafiken in Anhang B deuten auf eine dauerhafte Überlastung des Kanals hin.

**verlorene Pakete::** Standardmäßig wird in VEINS auf MAC-Ebene die Anzahl an verlorenen Paketen gezählt. Darunter fallen einerseits Pakete, die verworfen werden, weil während des Empfangs eines Paketes gerade gesendet wird. Zusätzlich fallen in diese Kategorie auch verlorene Pakete wegen Bitfehlern bzw. Paketkollisionen. Hohe Paketverluste können zu Fehlinformationen (weil die zuletzt erhaltenen Daten veraltet sind) und Datenfehlern führen.

### Simulationsablauf und -parameter

Nachdem die Kennwerte festgelegt wurden, war die Fragestellung, wie die verschiedenen Szenarien ausgeführt werden sollen. In Abhängigkeit der Fahrzeuganzahl nehmen auch Simulationen mit einer Simulationszeit von 100 s in realer Zeit eine Dauer von mehreren Stunden in Anspruch. Um möglichst viele Daten zu erhalten, war es somit zunächst unvorteilhaft, eine solch lang andauernde Simulationszeit zu wählen. Es wurde deshalb festgelegt, dass alle Szenarien für eine Dauer von 10 s die Kennwerte aufzeichnen. Um ein "Einpendeln" der zu berechnenden Parameter zu gewährleisten, wurden die ersten 6 s der Simulation nicht statistisch erfasst. Zeitgleich wurde jedes Szenario jeweils sechsmal parallel ausgeführt. Es wird innerhalb der Simulation dafür gesorgt, dass Fahrzeuge unterschiedliche und bei jeder Ausführung zufällig festgelegte Routen abfahren. Dadurch war es möglich in relativ kurzer Zeit mehrere hundert Kennwerte erfassen zu können. Die Aufzeichnung erfolgte mit Omnet++ Vektoren und Skalaren. Für die statistische Auswertung wurden bei Vektor-Werten die gemessenen Durchschnittswerte herangezogen (innerhalb von 10 s), bei Skalar-Werten wurden die innerhalb von 10 s gemessenen Werte verwendet. Durch diese Herangehensweise wurde es möglich Trends zu erkennen. Anhand dieser ersten Erkenntnisse konnten anschließend auffällige Szenarien durch eine längere Simulationszeit ausgeführt werden, um eine genauere Untersuchung zu bewerkstelligen. In Tabelle 4.2 sind alle Parameter aufgeführt, die bei Ausführung der Szenarien einbezogen und auf alle Szenarien angewandt wurden.

Parameter	Wert
Simulationszeit	6 Ausführungen parallel für je 10 s, 1 Ausführung für 200 s
Anzahl Fahrzeuge	100, 200
Loss Modell	Simple path loss Modell ( $k=2$ )
EDCA-Priority	AC_VO (BSMs), AC_BE (CAMs)
Beacon-Größe	300 Bytes, 600 Bytes
Frequenzband	5.6 GHz (BSMs), 5.9 GHz (CAMs)

Tabelle 4.2 – Simulationsparameter

Weitere Parameter werden individuell für beide Mechanismen per `omnetpp.ini` festgelegt. Bei der Festlegung des Frequenzbandes wurde bei ETSI der CCH wie vorgesehen verwendet. Bei WAVE ist der BSM-Mechanismus in [17] für Kanal 172 beschrieben. Zur Vereinfachung wurde hierfür innerhalb von VEINS auch der CCH verwendet, das Frequenzband wurde allerdings auf das des Kanals 172 (siehe Abbildung 2.1) angepasst. Bei der Beacon-Größe wurde sich an [43] orientiert. Dort wird

von einer CAM-Größe von 400 Bytes ausgegangen. Als relevante Größen wurden daraufhin alle Szenarien mit 300 bzw. 600 Bytes ausgeführt. Durch das parallele Ausführen der Szenarien ergeben sich bei 200 Fahrzeugen jeweils 1200 gemessene Kennwerte, was als ausreichend für das Erkennen von Trends angesehen wurde.

### 4.3 Evaluierung anhand von Szenarien

Folgend werden die drei herangezogenen Szenarien vorgestellt, ihre individuellen Eigenschaften aufgeführt und die gemessenen Kennwerte anschließend anhand von Boxplot-Grafiken veranschaulicht. Für eine übersichtlichere Darstellung sind alle herangezogenen Boxplots, die den WAVE- und alle ETSI-Mechanismen gegenüberstellen, in Anhang B in der Reihenfolge der folgenden Unterkapitel aufgeführt. Eine Unterscheidung passiert hierbei durch Byte-Größe eines Beacons sowie Anzahl der Fahrzeuge. Bei der Gegenüberstellung der Kennwerte werden auftretende Erkenntnisse diskutiert und analysiert. Es ist zu beachten, dass die Werte der Y-Achse der Boxplots individuell für jedes Szenario gewählt wurden. Dabei wurde der Höchstwert des jeweiligen Kennwertes für alle vier Varianten (300 Bytes bzw. 600 Bytes und 100 Fahrzeuge bzw. 200 Fahrzeuge) verwendet. Um alle Boxplots szenarioübergreifend miteinander vergleichen zu können, sind auf beigefügtem Datenträger (Anhang C) alle Boxplots mit einheitlichen Y-Achsenwerten enthalten. Detailliertere Vergleiche der Kennwerte geschehen anhand der Median-Werte. Da die Kennwerte zum Teil sehr streuen sind Vergleiche der Median-Werte am aussagekräftigsten.

#### 4.3.1 Szenario Autobahnkreuz

Dieses Szenario repräsentiert ein typisches Autobahnkreuz. Abbildung 4.1 zeigt eine Darstellung des Szenarios per SUMO-GUI.

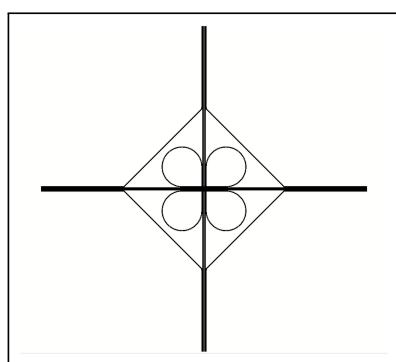


Abbildung 4.1 – SUMO-Gui-Screenshot des Szenarios *Autobahnkreuz*

Die Simulation beginnt mit der Ausführung bei Sekunde 40. Dadurch wird gewährleistet, dass das Zentrum des Autobahnkreuzes bereits komplett mit Fahrzeugen gefüllt ist. Neue Fahrzeuge werden jeweils von den Seiten hinzugefügt und fahren auf das Zentrum zu. Manche biegen bei der Fahrt in das Zentrum ab. Aufgrund sehr hoher Geschwindigkeiten sind die meisten Fahrzeuge nur etwa 50 s unterwegs. Mit Angabe der Fahrzeuganzahl von 100 oder 200 Fahrzeugen ist in diesem Szenario gemeint, dass sich während der Ausführungszeit diese Anzahl an Fahrzeugen in der Mitte des Autobahnkreuzes befinden. In Tabelle 4.3 sind weitere Eigenschaften dieses Szenarios aufgelistet.

Eigenschaft	Wert
Größe	2000 m x 2000 m
Geschwindigkeit	zwischen 100 km/h und 180 km/h
Besonderheiten	3-spurige Fahrbahn, im Zentrum 5-spurig

Tabelle 4.3 – Individuelle Eigenschaften von Szenario *Autobahnkreuz*

#### 4.3.1.1 Gegenüberstellung der Kennwerte

Auf Unterschiede, die in Abhängigkeit von Beacon-Größe und Fahrzeuganzahl auftreten, wird dabei vorerst nur auf Seiten von WAVE eingegangen. Alle drei ETSI-Varianten werden hier zusammengefasst betrachtet. Eine individuelle Betrachtung dieser erfolgt in einem späteren Kapitel (Abschnitt 4.4.2.2).

##### Generierungsintervall

In Abbildungen B.1 bis B.2 sind alle erfassten Werte der Generierungsintervalle gegenübergestellt. Auffällig ist, dass auf Seiten von WAVE in allen betrachteten Konstellationen das Generierungsintervall praktisch gleich bleibt. Das Generierungsintervall beträgt hier fast ausschließlich 100 ms.

Auf Seiten von ETSI sind deutliche Unterschiede erkennbar. In Abbildung B.1a bei 300 Bytes und 100 Fahrzeugen ist bei allen drei Varianten das Intervall zwischen 100 ms und 200 ms. Bei einem Vergleich mit Abbildung B.2a bei 600 Bytes und 100 Fahrzeugen fallen insbesondere die erhöhten Werte bei der Standardvariante auf. Durch die doppelte Beacon-Größe nimmt die Kanallast zu, was zu einem häufigeren Wechsel in den RESTRICTIVE-Zustand führt, wodurch das Generierungsintervall beeinflusst wird. Auffällig ist, dass die ETSI-LIMERIC Variante in allen vier Varianten um 200 ms die häufigsten Werte aufweist und sich am konstantesten verhält.

### Kanallast (CBR)

In Abbildung B.3a bei 300 Bytes und 100 Fahrzeugen ist zunächst kein großer Unterschied bei den Kanallasten erkennbar. Diese liegt bei durchschnittlich 30% (Median-Wert). Bei 200 Fahrzeugen ist in Abbildung B.3b auf Seiten von WAVE eine deutlich höhere Kanallast von einem Median-Wert von 47% im Vergleich zu den ETSI-Varianten zu erkennen, welche bei 25% (Standardvariante) bzw. 30% liegen. Auffällig hierbei ist, dass die Standardvariante bei ETSI die niedrigsten Werte aufweist.

Bei einer Beacon-Größe von 600 Bytes in Abbildung B.4 sind alle ETSI-Varianten klar vor WAVE (haben eine geringere Kanallast). Bei 100 Fahrzeugen liegt die Kanallast bei WAVE bei ca. 50%, bei allen drei ETSI-Varianten dagegen zwischen durchschnittlich 25% und 30%. Insbesondere bei 200 Fahrzeugen scheint hier der Congestion Control Mechanismus bei WAVE nicht ausreichend zu greifen (Abbildung B.4b), denn hier steigt die Kanallast auf 63% (Median-Wert), bei ETSI hingegen bleibt diese bei etwas unter 30%.

### Gesendete Beacons

Gesendete Beacons sind im Kontext dieses Szenarios von großem Interesse. Geht es um die Umsetzung von Safety Applications, ist aufgrund der hohen Geschwindigkeiten auf einer Autobahn eine hohe Anzahl an gesendeten Beacons wünschenswert, wenn nicht sogar notwendig, um Gefahrensituationen rechtzeitig anderen Fahrzeugen mitteilen zu können. Unabhängig von der Anzahl der Fahrzeuge und der Beacon-Größe sind bei WAVE höhere Werte zu verzeichnen (Abbildungen B.5 bis B.6). Der Median-Wert liegt hier sogar konstant bei 106. Dies hängt unmittelbar mit den zuvor festgestellten kurzen Generierungsintervallen zusammen.

Auf Seiten von ETSI ist insbesondere in Abbildung B.6b deutlich erkennbar, dass die Standardvariante die geringste Anzahl gesendeter Beacons zulässt, trotz der sehr hohen Geschwindigkeiten. Hier werden in etwa 20 Beacons (innerhalb von 10 s) versendet. DCC Profile und LIMERIC liefern hier bessere Werte. In allen Varianten liegen deren Werte um 40 gesendete Beacons. Nur bei 300 Bytes und 100 Fahrzeugen liegt die Standardvariante etwas vor den anderen (Abbildung B.5a) mit Werten um 50 (Median-Wert) gesendete Beacons.

### Empfangene Beacons

Wie zuvor bereits festgestellt, wurden auf Seiten von WAVE mehr Beacons gesendet, was die Empfangsrate unabhängig von Beacon-Größe oder Fahrzeuganzahl höher als bei ETSI (Abbildungen B.7 bis B.8) werden lässt. Bei 300 Bytes und 200 Fahrzeugen sind die Unterschiede am deutlichsten (Abbildung B.7b). Hier erreichen alle ETSI-

Varianten maximal ca. 4600 empfangene Beacons, WAVE dagegen die fast doppelte Anzahl von ca. 8600. Bei 600 Bytes bleibt dieses Bild erhalten, allerdings sind es bei allen ETSI-Varianten maximal 2400 und bei WAVE 4900 (Median-Werte). Auffällig ist auch, dass die Anzahl bei WAVE bei 600 Bytes unabhängig von der Fahrzeuganzahl gleich bleibt.

Auffällig auf Seite von ETSI ist, dass die Beacon-Größe eine Rolle zu spielen scheint. In Abbildung B.7a bei 300 Bytes und 100 Fahrzeugen werden bis zu 4000 Beacons empfangen, in Abbildung B.8b bei 600 Bytes und 200 Fahrzeugen dagegen ist selbst der Maximalwert unterhalb von 2000.

### Paketverluste

Bei 300 Bytes und 100 Fahrzeugen sind schon Unterschiede zwischen WAVE und allen ETSI-Varianten erkennbar (Abbildung B.9a). Die Median-Werte liegen hier mit 950 bei WAVE und 690 bei ETSI mit Standardvariante noch relativ nahe beieinander. Durch die sehr hohe Senderate bei WAVE ist es nicht verwunderlich, dass bei 200 Fahrzeugen noch sehr viel mehr Paketverluste auftreten, als bei allen ETSI-Varianten (Abbildung B.10). Bei 600 Bytes und 200 Fahrzeugen erreichen diese mitunter Werte von über 4600 (Median-Wert).

Auf Seiten von ETSI bleiben die erfassten Werte bei allen vier Varianten bei ca. 200 bis 900 verlorenen Paketen (Median-Werte), was deutlich von WAVE differiert (Abbildungen B.9 bis B.10).

#### 4.3.1.2 Zusammenfassung der erkannten Tendenzen

Die in Anhang B grafisch dargestellten Unterschiede werden zur Verdeutlichung in Zahlen zusammengefasst und bewertet. Es werden dabei minimale und maximale Werte der beobachteten Median-Werte als Grundlage herangezogen. Auf Unterschiede, die in Abhängigkeit von Beacon-Größe und Fahrzeuganzahl auffallen, wird hierbei vorerst nur auf Seiten von WAVE eingegangen. Alle drei ETSI-Varianten werden zunächst als ein Mechanismus betrachtet. Eine individuelle Betrachtung dieser erfolgt in einem späteren Kapitel (Abschnitt 4.4.2.2). In Tabelle 4.4 sind die Kennwerte aufgeführt. Insbesondere das durchgehend sehr kurze Generierungsintervall bei WAVE ist auffällig. Es ist ersichtlich, dass WAVE bei den Kennwerten gesendete und empfangene Beacons deutlich bessere Werte aufweist als alle drei ETSI-Varianten, was aufgrund der hohen Geschwindigkeiten wünschenswert ist. Allerdings stellt sich die Frage, ob die wesentlich höhere Anzahl an empfangener Beacons auch einen wirklichen Nutzen bringt. In Kauf genommen wird dadurch auf jeden Fall eine wesentlich höhere Kanallast.

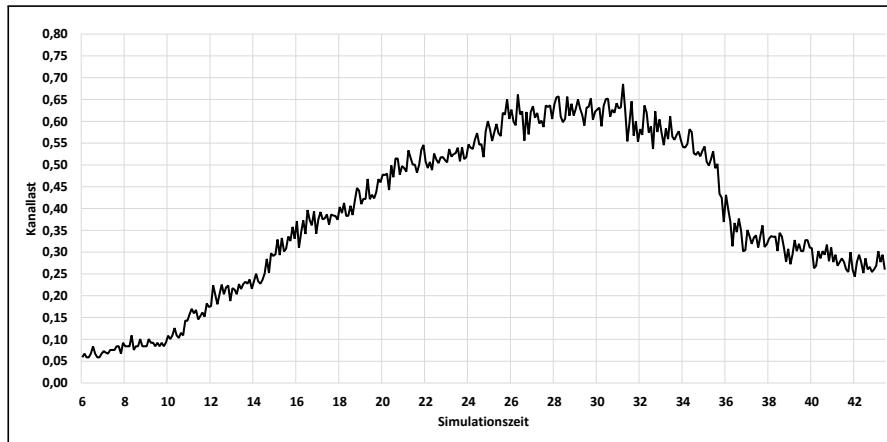
Kennwert	WAVE	ETSI
Generierungsintervall	100 ms	160 ms bis 350 ms
Kanallast	30% bis 63%	27% bis 33%
gesendete Beacons	106	19 bis 55
empfangene Beacons	4599 bis 8643	1237 bis 2957
Paketkollisionen	953 bis 4638	220 bis 882

**Tabelle 4.4** – Vergleich der Kennwerte des Szenarios *Autobahnkreuz* anhand der beobachteten Median-Werte

#### 4.3.1.3 Erkenntnisse bei längerer Simulationszeit

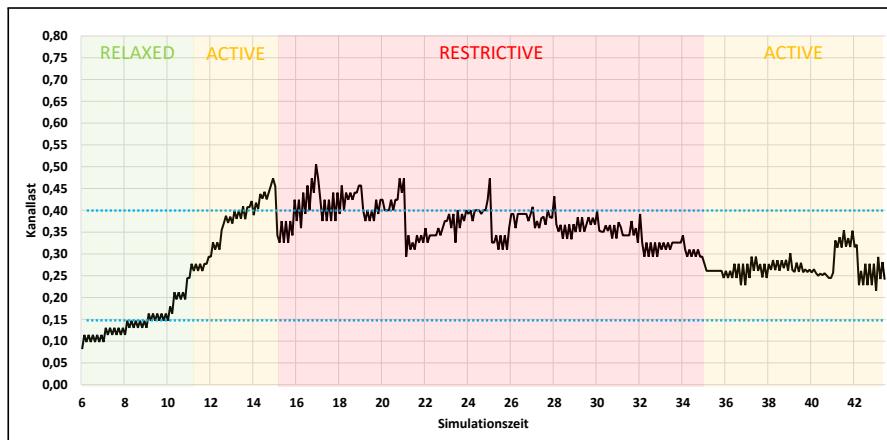
Die zuvor festgestellten Erkenntnisse sollen mittels einer längeren Ausführungszeit der Simulation von 200 s noch näher betrachtet werden. Hierbei wurde nur die Konstellation 600 Bytes und 200 Fahrzeuge umgesetzt.

Bei WAVE fiel zuvor die zum Teil extrem hohe Kanallast auf. Da es sich bei den bisher verglichenen Werten um Median-Werte aller Fahrzeuge handelt, soll nun zunächst ein Vergleich an einem zufällig ausgewählten Fahrzeug erfolgen. Dieses Verkehrsmittel fährt bei WAVE und ETSI die selbe Strecke und Zeit. Hier wurde ein Fahrzeug gleich zu Beginn der Simulation gewählt. Dieses fährt von oben mit einer Durchschnittsgeschwindigkeit von 160 km/h gerade nach unten während der Simulationszeit von Sekunde 0 bis Sekunde 44 (Abbildung 4.1). Nach ca. Sekunde 20 befindet es sich in der Mitte des Autobahnkreuzes. Zu diesem Zeitpunkt sind aus allen anderen Richtungen analog Fahrzeuge auf die Mitte zugefahren. Die Messung der Kanallast beginnt erst nach Sekunde 6. Anhand dieses Fahrzeugs wird der Verlauf der Kanallast bei WAVE und ETSI verglichen. In Abbildung 4.2 ist zunächst der Verlauf bei WAVE aufgeführt. Von Sekunde 10 bis Sekunde 32 ist ein Anstieg der Kanallast auf fast 70% zu verzeichnen. Der rapide Abfall der Kanallast ab Sekunde 36 liegt am Verlassen der Mitte des Autobahnkreuzes, wodurch weniger Fahrzeuge in Reichweite sind. Durchschnittlich beträgt die Kanallast für dieses Fahrzeug über die gesamte Dauer rund 38%.



**Abbildung 4.2** – Kanallastverlauf des Fahrzeugs *Node2* bei WAVE während des Szenarios *Autobahnkreuz*

In Abbildung 4.3 wurde analog für ETSI mit Standardparametern der Kanallastverlauf für das gleiche Fahrzeug aufgezeichnet.



**Abbildung 4.3** – Kanallastverlauf des Fahrzeugs *Node2* bei ETSI (Standardvariante) während des Szenarios *Autobahnkreuz*

In besagter Abbildung sind zusätzlich die Zustände der DCC aufgeführt. Bei den Kanallastwerten von 15% und 40% wurde je eine horizontale gestrichelte Linie eingefügt, um die für die Zustandsmaschine relevanten Kanallastgrenzen zu markieren. Anfangs zeichnet sich hier ein ähnliches Bild wie bei WAVE ab. Allerdings liegt der gemessene Maximalwert bei Sekunde 17 etwas über 50%. Allgemein ist der Kanallastverlauf während des Zustandes RESTRICTIVE flacher als bei WAVE. Der Zustandswechsel zu ACTIVE bei Sekunde 36 ist dem Verlassen der Autobahnkreuzmitte zuzuschreiben. Die durchschnittliche Kanallast für dieses Fahrzeug beträgt ca.

30%.

Dieses punktuelle Beispiel festigt die zuvor gewonnenen Erkenntnisse der geringeren Kanallasten bei ETSI. Darüber hinaus wurden alle während der Simulationszeit von 200 Sekunden erfassten Werte als Boxplot-Grafiken ausgewertet. Diese sind in Abbildungen B.11 bis B.15 dargestellt. Auffällig waren auch die in Abbildung B.10b stark streuenden Paketverluste bei WAVE. Dieses Bild zeichnet sich auch bei einer längeren Simulationszeit ab.

Vergleicht man hier zunächst die Generierungsintervalle in Abbildung B.2b und Abbildung B.11 fällt auf, dass bei einer längeren Simulationszeit die Bandbreite der Werte geringer wird. Am auffälligsten sind hierbei die Median-Werte, die bei allen betrachteten vier Varianten gleich bleiben.

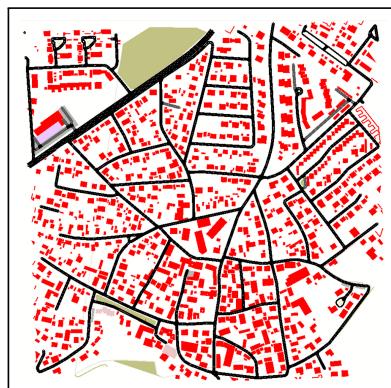
Analog dazu ist auch die Kanallast zu bewerten. In Abbildung B.4b und Abbildung B.12 ist auf Seite von WAVE der Median-Wert von 63% gleich. Auf Seite von ETSI werden bei einer längeren Ausführungszeit etwas höhere Median-Werte erkannt (in Abbildung B.4b liegen diese zwischen 27% und 29%, in Abbildung B.12 zwischen 32% und 38%).

Bei den anderen drei Kennwerten ist aufgrund der längeren Simulationszeit anhand der Median-Werte kein Vergleich möglich, da die Werte bei längerer Simulationszeit natürlich viel höher sind. Allerdings lohnt sich ein rein visueller Vergleich der Boxplots. In Abbildung B.6b und Abbildung B.13 zeichnet sich bei den drei ETSI-Varianten ein ähnliches stufenförmiges Bild ab. Auch hier ist der Median bei der Standardvariante am niedrigsten, gefolgt von DCC Profile und LIMERIC mit dem höchsten Wert. Auch in Abbildung B.8b in Bezug zu Abbildung B.14 sowie Abbildung B.10b mit Abbildung B.15 zeigen sich ähnliche Boxplot-Verläufe.

Mithilfe der genannten Boxplots ist auch erkennbar, dass eine längere Simulationszeit die zuvor schon festgestellten Erkenntnisse bestätigt.

### 4.3.2 Szenario Stadt

Die Simulationausführung beginnt bei der Simulationszeit mit Sekunde 10. Die Fahrzeuge sind zu diesem Zeitpunkt alle auf der Karte in Bewegung. Die Aufzeichnung der Kennwerte beginnt bei jedem Fahrzeug erst nach Sekunde 6. In Tabelle 4.5 sind weitere Eigenschaften dieses Szenarios aufgelistet, Abbildung 4.4 zeigt das Szenario per SUMO-GUI.



**Abbildung 4.4 – SUMO-Gui-Screenshot des Szenarios Stadt**

Fokus bei diesem Szenario ist weniger der Blick auf die Kanallast, sondern vielmehr auf die gesendeten und empfangenen Beacons ausgelegt. Interessant ist insbesondere wie sich die häufig ändernden dynamischen Fahreigenschaften durch Abbiegen und Anhalten auf die Mechanismen auswirken.

Eigenschaft	Wert
Größe	800 m x 800 m
Geschwindigkeit	zwischen 5 km/h und 50 km/h
Besonderheiten	Kreuzungen, häufiges Abbiegen, Einfluss von Gebäuden auf Funkverbindungen, Haltephasen

**Tabelle 4.5 – Individuelle Eigenschaften von Szenario Stadt**

#### 4.3.2.1 Gegenüberstellung der Kennwerte

Auf Unterschiede, die in Abhängigkeit von Beacon-Größe und Fahrzeuganzahl auftreten, wird hierbei vorerst nur auf Seiten von WAVE eingegangen. Alle drei ETSI-Varianten werden hier zusammengefasst betrachtet. Eine individuelle Betrachtung dieser erfolgt in einem späteren Kapitel (siehe Abschnitt 4.4.2.2).

##### Generierungsintervall

Wie bei dem Szenario zuvor zeichnet sich bei WAVE das gleiche Bild eines Generierungsintervalls von 100 ms ab (Abbildungen B.16 bis B.17). Auf der Seite von ETSI liegen alle drei Varianten bei ähnlichen Werten und kaum auseinander. Hier sind die häufigsten Median-Werte zwischen 230 ms und 280 ms zu verzeichnen. Auffallend bei allen drei ETSI-Varianten ist die hohe Streuung der Werte, was auf die

sehr verschiedenen Fahrsituationen und dadurch häufige Wechsel der dynamischen Fahreigenschaften innerhalb der Stadt zurückzuführen sind.

#### Kanallast (CBR)

Die Kanallastwerte bei diesem Szenario sind nach Abbildungen B.18 bis B.19 moderat, liegen die Median-Werte bei WAVE zwischen 4% bis 13%. In Ausnahmefällen wird ein Maximalwert (kein Median) bis zu 40% erreicht (bei einer Konstellation von 600 Bytes und 200 Fahrzeugen). Bei ETSI sind Ausnahmen bis zu 25% (siehe Abbildung B.19b) zu verzeichnen. Der Median liegt hier allerdings zwischen 3% bis 10%.

#### Gesendete Beacons

Für das Szenario *Stadt* ist die Anzahl gesendeter Beacons von besonderem Interesse, kann es hier vermehrt zu Gefahrensituationen kommen. In Abbildungen B.20 bis B.21 zeichnen sich deutliche Vorteile bei WAVE durch eine im Durchschnitt mehr als doppelt so hohe Anzahl gesendeter Beacons von 100 Beacons ab, als bei allen ETSI-Varianten von etwa 40 versendeten Beacons (Median-Wert). Auffällig ist, dass sich dieser Wert bei WAVE im Vergleich zu dem Szenario *Autobahnkreuz* nicht verändert (siehe Tabelle 4.4).

#### Empfangene Beacons

Aufgrund der zuvor festgestellten hohen Senderate bei WAVE sind die Werte für die empfangenen Beacons auf Seiten von WAVE auch sehr hoch. In Abbildungen B.22 bis B.23 sind bei WAVE Median-Werte zwischen 600 bis 1500, in Ausnahmefällen sogar bis zu 4000 empfangene Beacons zu verzeichnen (bei 600 Bytes und 200 Fahrzeugen). Alle drei ETSI-Varianten verzeichnen hier deutlich geringere Median-Werte von etwa 200 bis 400 empfangener Beacons (Maximalwert etwa 1500).

#### Paketverluste

In Abbildungen B.24 bis B.25 sind deutlich höhere Paketverluste bei WAVE erkennbar, was aufgrund der viel höheren Senderate als bei allen drei ETSI-Varianten auch nicht verwunderlich ist. Bei WAVE treten im Schnitt 50 bis 200 auf, vereinzelt konnten auch Werte bis zu 1000 erfasst werden. Auf Seiten von ETSI liegen unabhängig von der Beacon-Größe und Fahrzeuganzahl die Werte bei maximal 50 (Median-Werte).

##### 4.3.2.2 Zusammenfassung der erkannten Tendenzen

Die zuvor festgestellten Erkenntnisse werden in Tabelle 4.6 folgend zusammengefasst. Es werden dabei minimale und maximale Werte der beobachteten Median-Werte als

Grundlage herangezogen. Auf Unterschiede, die in Abhängigkeit von Beacon-Größe und Fahrzeuganzahl auftreten, wird vorerst nur auf Seiten von WAVE eingegangen. Alle drei ETSI-Varianten werden zusammengefasst betrachtet. Eine individuelle Anschauung dieser erfolgt in einem späteren Kapitel (Abschnitt 4.4.2.2). Auffallend bei diesem Szenario sind die moderaten Kanallastwerte auf beiden Seiten. Ebenso ist festzustellen, dass beide Mechanismen auch bei den anderen Kennwerten relativ nah beieinander liegen. Ins Auge sticht bei WAVE die gleichbleibende Anzahl an gesendeter Beacons wie bei dem Szenario *Autobahnkreuz*. Auch wenn die Unterschiede der Kennwerte zwischen WAVE und ETSI in diesem Szenario moderat sind, lassen sich ähnliche Tendenzen wie in Abschnitt 4.3.1.1 erkennen.

Kennwert	WAVE	ETSI
Generierungsintervall	100 ms	230 ms bis 280 ms
Kanallast	4% bis 13%	3% bis 10%
gesendete Beacons	106	34 bis 42
empfangene Beacons	678 bis 1366	204 bis 415
Paketkollisionen	15 bis 125	2 bis 15

**Tabelle 4.6** – Vergleich der Kennwerte des Szenarios *Stadt* mittels der beobachteten Median-Werte

#### 4.3.2.3 Erkenntnisse bei längerer Simulationszeit

Dieses Szenario wurde ebenso mit einer Simulationszeit von 200 s mit der Konstellation 600 Bytes als Beacon-Größe und einer Fahrzeuganzahl von 200 ausgeführt. In Abbildungen B.26 bis B.30 sind die Kennwerte hierzu grafisch dargestellt.

Das Generierungsintervall zeichnet hierbei kein anderes Bild hinsichtlich des Vergleichs zwischen WAVE und ETSI, als wie zuvor bei kürzerer Laufzeit. Auf Seiten von WAVE bleibt das Intervall fast ausschließlich bei 100 ms. Es ergibt sich keine Änderung zur kürzeren Laufzeit (Tabelle 4.6 und Abbildung B.17b). Die Median-Werte der ETSI-Varianten liegen alle drei in einem Bereich von 330 ms bis 360 ms und unterscheiden sich untereinander marginal, aber deutlich von WAVE. Im Vergleich zur kürzeren Variante sind hier die Intervalle um etwa 100 ms angestiegen.

Bei der Kanallast erhöht sich der Wert bei WAVE auf 21% im Mittel, alle drei ETSI-Varianten bleiben hier gleich bei etwa 10%. Bei WAVE ist nach Abbildung B.19b und Abbildung B.27 ein recht deutlicher Unterschied erkennbar.

Die Unterschiede bei gesendeten und empfangenen Beacons spiegeln das Bild der kürzeren Laufzeit identisch wieder. WAVE erreicht hier deutlich höhere Werte von 873 gesendeter (Abbildung B.28) bzw. 17205 empfangener Beacons (Abbildung B.29) im Vergleich zu 225 bis 233 gesendeter und 3622 bis 3966 empfangener Beacons

bei den drei ETSI-Varianten (Median-Werte).

Die Unterschiede bei den Paketverlusten sind ähnlich extrem. WAVE erreicht hier als Median-Wert 2416, die ETSI-Varianten liegen bei Werten von 148 bis 170.

Hier zeigen sich zum Teil deutlichere Unterschiede, als bei einer kürzeren Laufzeit. Insbesondere bei den gesendeten und empfangenen Beacons wird der Unterschied sehr deutlich. Hier ist wie bei Szenario *Autobahnkreuz* WAVE mit wesentlich höheren Werten vertreten, was allerdings zu höherer Kanallast und Paketverlusten führt.

### 4.3.3 Szenario Stau

Die Simulationausführung beginnt bei der Simulationszeit Sekunde 0 und endet bei Sekunde 16, die ersten 6 Sekunden werden dabei nicht statistisch aufgezeichnet. In Tabelle 4.7 sind weitere Eigenschaften dieses Szenarios aufgelistet.

Es soll verdeutlicht werden, dass sich alle Fahrzeuge in Empfangsreichweite befinden. Anzumerken sei, dass es sich hierbei um ein abstraktes Stauszenario und kein Realszenario handelt. Von Interesse hierbei, ist wie sich beide Mechanismen bei Fahrzeugstillstand verhalten. Fahrzeuge wurden deshalb auf einer fest definierten Fläche zufällig platziert und verweilen dort über die ganze Simulationsdauer. Dieses Szenario wurde insbesondere aufgrund des Wegfalls der dynamischen Fahreigenschaften, welche zumindest auf Seiten von ETSI einen sehr relevanten Aspekt bei der Beacon-Generierung darstellen, in die Bewertung aufgenommen.

Eigenschaft	Wert
Größe	Fahrzeuge werden auf einer Fläche von 200 m x 200 m zufällig platziert
Geschwindigkeit	0 km/h
Besonderheiten	alle Fahrzeuge befinden sich in Empfangsreichweite, für die gesamte Dauer bleiben die Fahrzeuge stehen

**Tabelle 4.7 – Individuelle Eigenschaften von Szenario *Stau***

#### 4.3.3.1 Gegenüberstellung der Kennwerte

Auf Unterschiede, die in Abhängigkeit von Beacon-Größe und Fahrzeuganzahl auftreten, wird hierbei vorerst nur auf Seiten von WAVE eingegangen. Alle drei ETSI-Varianten werden hier zusammengefasst betrachtet. Eine individuelle Betrachtung dieser erfolgt in einem späteren Kapitel (Abschnitt 4.4.2.2).

### Generierungsintervall

In Abbildungen B.31 bis B.32 fällt auf, dass bei allen Varianten auf Seiten von ETSI das Intervall konstant bei 1 s verweilt. Dadurch ist erkennbar, dass bei Fahrzeugstillstand keine Änderung einer dynamischen Fahreigenschaft auftritt, was zum Versenden von CAMs mit einer Rate von 1 Hz führt.

Umso verwunderlicher ist das Verhalten bei WAVE. Bei 100 Fahrzeugen (Abbildung B.31a sowie Abbildung B.32a) treten die häufigsten Werte um 200 ms auf. Erhöht sich die Fahrzeuganzahl auf 200, wird ein Intervall um 360 ms am meisten verwendet. Trotz des sich nicht ändernden Fahrzeugstillstandes werden sehr oft Beacons versendet. Vergleicht man diese mit den Erkenntnissen aus den beiden vorherigen Szenarien, stellt sich die Frage, warum das Generierungsintervall diesmal größer als 100 ms ist. Es hatte in den beiden vorherigen Szenarien eben jenen Wert konstant gehalten. Dies liegt daran, dass aufgrund des Stillstandes und der Entfernung der Fahrzeuge wesentlich mehr Fahrzeuge in einer Reichweite von 100 m erkannt werden. Diese Problematik wird in Abschnitt 4.4.1.1 ausführlich geschildert und diskutiert.

### Kanallast (CBR)

Deutliche Unterschiede sind bei der Kanallast festzustellen (Abbildungen B.33 bis B.34). Auf Seiten von ETSI wird in keiner Konstellation ein Median-Wert über 10% erreicht. Ganz anders verhält es sich hier bei WAVE. Bei 200 Fahrzeugen und 600 Bytes sind fast ausschließlich Werte um 75% zu verzeichnen (Abbildung B.34b). Über die gesamte Simulationszeit zeichnen sich hierbei auch keinerlei Schwankungen ab. Daraus lässt sich schließen, dass selbst bei sehr hoher gemessener Kanallast keine "Gegenmaßnahmen" zur Reduzierung der Last von Seiten des WAVE-Mechanismus eingeleitet werden.

### Gesendete Beacons

Durch das auf Seiten von WAVE bereits in Abschnitt 4.3.3.1 festgestellte kurze Generierungsintervall zeichnet sich auch beim Versenden von Beacons ein deutlich zurückhaltenderes Verhalten bei ETSI ab. In Abbildungen B.35 bis B.36 schwankt die Anzahl gesendeter Beacons bei WAVE zwischen 80 Beacons bei 100 Fahrzeugen und etwa 60 Beacons bei 200 Fahrzeugen. Unabhängig von Beacon-Größe und Fahrzeuganzahl werden bei ETSI konstant 10 Beacons versendet.

### Empfangene Beacons

Auch bei diesem Kennwert sind die Unterschiede zwischen beiden Mechanismen enorm (Abbildungen B.37 bis B.38). Auf Seiten von ETSI werden bei 300 Bytes

konstant um die 800 Beacons empfangen, bei 600 Bytes treten Schwankungen auf, was zu Median-Werten zwischen 300 bis 450 führt. Bei WAVE sind in allen Konstellationen deutlich mehr empfangene Beacons zu verzeichnen. In Abbildung B.7b wird ein fast konstanter Wert von 10000 bei einer Beacon-Größe von 300 Bytes und 200 Fahrzeugen erreicht. Dies ist durchaus kritisch zu betrachten, denn alle Fahrzeuge bewegen sich zu keinem Zeitpunkt der Simulation.

#### Paketverluste

Die zum Teil enormen Unterschiede der vorher gegenübergestellten Kennwerte sind bei den Paketverlusten nicht so drastisch zu verzeichnen (Abbildungen B.39 bis B.40). Nur bei 600 Bytes und 200 Fahrzeugen in Abbildung B.40b sind auf Seiten von WAVE doppelt so viele (ca. 4000) als bei ETSI (ca. 2000) zu verbuchen. Bei den anderen drei Konstellationen liegen beide Mechanismen etwa gleich bei 1000 bis 1800 Paketverlusten.

##### 4.3.3.2 Zusammenfassung der erkannten Tendenzen

Die zuvor festgestellten Erkenntnisse werden in Tabelle 4.8 folgend zusammengefasst. Dabei werden minimale und maximale Werte der beobachteten Median-Werte als Grundlage herangezogen. Der Vergleich der Kennwerte zeigte bei diesem Szenario deutliche - zum Teil drastische - Unterschiede. Da sich Fahrzeuge zu keiner Zeit bewegen, ist der WAVE-Mechanismus kritisch zu betrachten, denn dieser führt hier zu hohen konstanten Kanallasten und Paketverlusten.

Auch mit Fokus auf sicherheitsrelevante Applikationen stellt sich die Frage nach dem Nutzen von bis zu 10000 empfangenen Beacons. Aufgrund des Fahrzeugstillstandes können in diesem abstrakten Szenario keine Gefahrensituationen entstehen, denn Fahrzeuge bewegen sich zu keiner Zeit. Dadurch ist die Notwendigkeit solch enorm hoher Sende- und Empfangsraten nicht zu rechtfertigen. Sinnvoll wäre es sicherlich, dass die Fahrzeuge am Stauende eine höhere Sende- und Empfangsrate aufweisen, denn dies könnte die Gefahr von Auffahrunfällen am Stauende verringern.

Kennwert	WAVE	ETSI
Generierungsintervall	190 ms bis 360 ms	1000 ms
Kanallast	34% bis 76%	7% bis 10%
gesendete Beacons	57 bis 82	10
empfangene Beacons	6640 bis 10177	298 bis 820
Paketkollisionen	424 bis 3966	170 bis 1692

**Tabelle 4.8** – Vergleich der Kennwerte des Szenarios *Stau* mittels der beobachteten Median-Werte

#### 4.3.3.3 Erkenntnisse bei längerer Simulationszeit

Aufgrund der vorherigen Erkenntnisse dieses Szenarios war es von besonderem Interesse, wie sich der WAVE-Mechanismus auch bei einer längeren Simulationszeit verhält. Deshalb wurde dieses Szenario auch mit einer Zeit von 200 s ausgeführt. Es wurde wie bei den Szenarien zuvor auch hier die Konstellation von 600 Bytes und 200 Fahrzeugen herangezogen.

In Abbildung B.32b und Abbildung B.41 zeigt sich bei dem Generierungsintervall nur ein leichter Anstieg bei WAVE von 360 ms auf 390 ms des Median-Wertes. Die Kanallast im Vergleich (Abbildung B.4b und Abbildung B.12) zeigt bei beiden Boxplots Werte von ausschließlich 75%. Bei einer längeren Simulationszeit entspannt sich die Kanallast bei WAVE demnach nicht. Die anderen drei Werte sind nur bedingt vergleichbar, in Abbildungen B.43 bis B.45 zeichnen sich die wie in Abschnitt 4.3.3.1 schon festgestellten extremen Unterschiede zwischen WAVE und allen drei ETSI-Varianten ab. Ein Vergleich der drei ETSI-Varianten untereinander zeigt keinerlei Gegensätze auf.

### 4.4 Individuelle Betrachtung beider Mechanismen

In den vorherigen Abschnitten dieses Kapitels wurden anhand der drei Szenarien die Mechanismen zwischen WAVE und ETSI verglichen und zumeist gravierende Differenzen festgestellt. Folgend sollen beide Mechanismen individuell näher betrachtet und Auffälligkeiten diskutiert werden.

Aufgrund der zuvor gewonnenen Erkenntnisse wurde auf Seiten von WAVE genauer versucht herauszufinden, warum dieser Mechanismus dazu neigt, den Kanal zu überlasten und was es eventuell für Möglichkeiten gibt, dieses Verhalten zu verbessern. Dies wird in Abschnitt 4.4.1 diskutiert.

Insbesondere die drei ETSI-Varianten wurden zuvor nur allgemein betrachtet. Ein

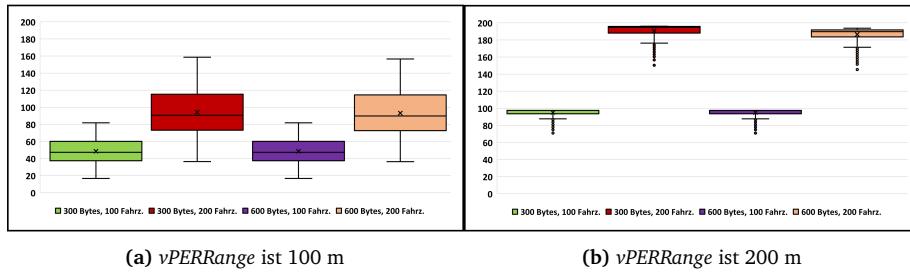
Vergleich dieser erfolgt mittels der drei Szenarien und Gegenüberstellung der durchschnittlich gemessenen Kennwerte (Abschnitt 4.4.2).

#### 4.4.1 WAVE

Auf Seiten von WAVE sollen hier zwei Punkte näher betrachtet werden, die sich durch die zuvor erhaltenen Erkenntnisse herauskristallisiert haben. Zum einen ist das im Vergleich zu ETSI sehr kurze Generierungsintervall Betrachtungsgegenstand. Zum Anderen wird die Rolle des TxDynamics-Flags, das Teil der *Transmission Decision* (Abschnitt 2.1.4.6) ist, untersucht.

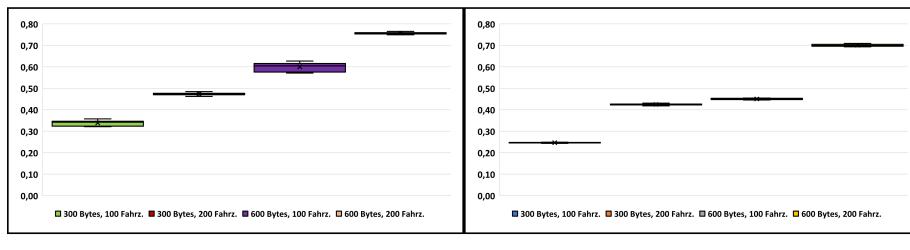
##### 4.4.1.1 Auffälligkeiten hinsichtlich des Generierungsintervalls

In Abschnitt 4.3 wurde im direkten Vergleich mit allen ETSI-Varianten festgestellt, dass der WAVE-Mechanismus aufgrund des oftmals sehr kurzen Generierungsintervalls dazu neigt den Kanal selbst bei einer bereits festgestellten sehr hohen Kanallast weiter zu belasten anstatt zu entlasten. Verwunderlich ist in diesem Zusammenhang, dass das Generierungsintervall sehr kurz ist, obwohl insbesondere bei dem Szenario *Stau* (Abschnitt 4.3.3), in dem über die ganze Simulationszeit sich kein Fahrzeug fortbewegt, bei einer Beacon-Größe von 600 Bytes und 200 Fahrzeugen (Abbildung B.32b) zwischen 300 ms und 450 ms variiert, was im Vergleich zu allen ETSI-Varianten (1 s) sehr kurz ist. Die Grundlage für dieses Verhalten liegt in der *MaxITT* (Abschnitt 2.1.4.5), welche zwischen 100 ms und 600 ms ist. Die Berechnung erfolgt alle 100 ms und wird für die Festlegung der *NextScheduledMsgTime* verwendet. Betrachtet man die Berechnung der *MaxITT* in Gleichung (2.8) genauer, stellt man fest, dass diese ausschließlich von den zuvor berechneten Parametern *Vehicle Density in Range* (Abschnitt 2.1.4.2) bzw. *Smooth Vehicle Density in Range* (siehe Gleichung (2.7)) abhängt. Der entscheidende Punkt hierbei ist, dass nur Fahrzeuge in einem Umkreis von 100 m (Parameter *vPERRange*, siehe [17], Tabelle 21) berücksichtigt werden, selbst wenn weiter entfernte Fahrzeuge in Funkreichweite sind und dementsprechend auch von diesen Beacons empfangen werden. Es stellte sich die Frage, ob sich durch Ausweitung des Parameters *vPERRange* von 100 m auf 200 m das Generierungsintervall verlängert und letztlich dadurch die Kanallast verringert wird. Interessant für diese Betrachtung ist das Szenario *Stau*, denn hier werden Fahrzeuge zufällig auf einer Fläche von 200 m x 200 m verteilt, was bei einer *vPERRange* von 100 m dazu führen müsste, dass nicht alle Fahrzeuge erkannt werden. In Abbildung 4.5 zeigt sich die Auswirkung auf den entscheidenden Parameter *Smooth Vehicle Density in Range* für die Berechnung von *MaxITT*.



**Abbildung 4.5 – Auswirkung der Änderung von  $vPERRange$  auf *Smooth Vehicle Density in Range* bei Szenario *Stau***

Wurden bei der Simulation 100 Fahrzeuge verwendet, liegt der Wert von *Smooth Vehicle Density in Range* bei einer  $vPERRange$  von 100 m im Durchschnitt bei 50 Fahrzeugen, bei 200 Fahrzeugen bei etwa 100 Fahrzeugen (Abbildung 4.5a). Wird  $vPERRange$  auf 200 m gesetzt, verdoppeln sich die zuvor genannten Werte nahezu (Abbildung 4.5b). In Abbildungen 4.6a bis 4.6b sind zum Teil deutliche Auswirkungen auf die Kanallast feststellbar. Bei 300 Bytes und 100 Fahrzeugen verringert sie sich von ca. 35% auf 25%. Ein ähnliches Bild ist bei 600 Bytes und 100 Fahrzeugen erkennbar. Hier verringert sich die Kanallast von ca. 60% auf 45%. Analog wie bei dem Szenario *Stau* wurde dies auch bei dem Szenario *Autobahnkreuz* untersucht. Die Erkenntnis ist ebenso eine Verringerung der Kanallast, allerdings nicht in diesem deutlichen Ausmaß. Die Boxplots dazu sind auf beigeigtem Datenträger einsehbar.



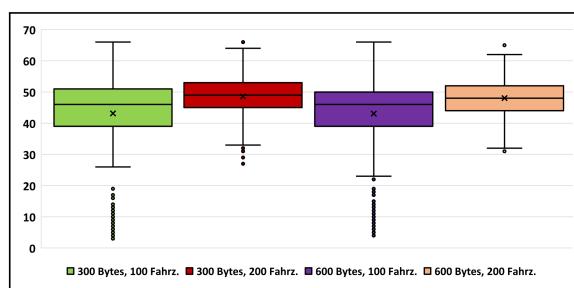
**Abbildung 4.6 – Auswirkung der Änderung von  $vPERRange$  auf die Kanallast bei Szenario *Stau***

#### 4.4.1.2 Einfluss des *TxDynamics*-Flags

Wie schon zuvor diskutiert, kann der für das Generierungsintervall entscheidende Wert *MaxITT* maximal 600 ms betragen. In Abschnitt 4.4.1.1 wurde bereits festgestellt, dass der Parameter  $vPERRange$  einen wichtigen Einfluss hat. Durch dessen Ausweitung auf 200 m konnte bei dem Szenario *Stau* bei 600 Bytes und 200 Fahrzeugen beobachtet werden, dass fast ausschließlich der Maximalwert für *MaxITT*

von 600 ms verwendet wurde. Überraschend war allerdings die nach wie vor noch relativ hohe Senderate von im Schnitt 50 Beacons. Zu erwarten wäre bei einem konstanten Generierungsintervall von 600 ms und einer Simulationszeit von 10 s ein Wert um 16 Beacons.

Entscheidend für dieses Verhalten ist das Flag *TxDynamics* während der *Transmission Decision* (Abschnitt 2.1.4.6), welche alle 100 ms ausgeführt wird. Es wird auf `true` gesetzt, wenn die zuvor berechnete *Transmission Probability* (Abschnitt 2.1.4.4) größer oder gleich dem durch eine Bernoulli-Funktion ermittelten Wert ist. Die Bernoulli-Funktion hat hier zufällig den Wert 0 oder 1 zurückzugeben. Zusätzlich muss die *NextScheduledMsgTime* mindestens 25 ms in der Zukunft liegen. In allen drei Szenarien hat sich unabhängig von der Beacon-Größe und Fahrzeuganzahl gezeigt, dass dieses Flag durchschnittlich ca. 50 mal im Rahmen einer Simulation von 10 s gesetzt wird. Unmittelbar nach der Festlegung des Flags erfolgt die Ausführung der *Schedule Transmission* (Abschnitt 2.1.4.7). Ist nun besagtes Flag zuvor auf `true` gesetzt worden, erfolgt hier ein sofortiges Versenden einer BSM und ein Stornieren der für die Zukunft geplanten BSM. Es kann somit trotz einer maximalen *MaxITT* von 600 ms und einer dementsprechend weit in der Zukunft liegenden *NextScheduledMsgTime* zu einem Versenden einer BSM nach 100 ms kommen, was auch häufig in den verwendeten Szenarien zu erkennen war. Dies führt zu einer höheren Senderate und letztlich zu höherer Kanallast. In Abbildung 4.7 wird anhand des Szenarios *Stau* veranschaulicht, wie oft dieses Flag bei einer Ausführung von 10 s gesetzt wurde. Die Median-Werte liegen hier unabhängig von Fahrzeuganzahl und Beacon-Größe bei 46 bis 49. Man muss bedenken, dass in einer Simulationszeit von 10 s dieses Flag maximal 100-mal auf `true` gesetzt werden könnte, da die Berechnung des Flags (Abschnitt 2.1.4.6) alle 100 ms ausgeführt wird.



**Abbildung 4.7 – Anzahl des *TxDynamics*-Flags mit Wert `true` bei dem Szenario *Stau***

#### 4.4.2 ETSI ITS-G5

Auf Seiten von ETSI soll zunächst auf die in Abschnitt 2.2.4.2 erläuterte Funktionsweise der DCC Transmit Queues in Zusammenhang mit den Erkenntnissen aus den Szenarien eingegangen werden.

Anschließend werden die drei ETSI-Varianten mittels der Szenarien (Abschnitt 4.3) und insbesondere der grafischen Gegenüberstellung in Anhang B genauer betrachtet.

##### 4.4.2.1 Auswirkung der DCC Transmit Queue

Bei allen betrachteten Szenarien wurde auch beobachtet, ob und inwieweit sich die Funktionsweise der DCC Transmit Queues (Abschnitt 2.2.4.2) auswirkt. Hiermit ist zum einen die maximale Queue-Länge von zwei sowie die Lifetime von 1 s gemeint. In keinem Szenario hat es sich gezeigt, dass eine CAM wegen dieser beiden Mechanismen verworfen wird. Das Paketintervall, das auf MAC-Ebene die Entnahme eines Paketes aus der DCC Transmit Queue und anschließende Einreihung in die EDCA-Queue regelt, wird auch bei der CAM-Generierung herangezogen und dient dort als die Zeitspanne, die mindestens verstrichen sein muss, damit die nächste CAM generiert werden darf. Auch führte dies dazu, dass die Lebenszeit eines Paketes nie überschritten wurde. Dieses Konzept lässt sich nach den Erkenntnissen aus den Szenarien als sehr sinnvoll und effektiv bewerten.

Allerdings ist an dieser Stelle zu erwähnen, dass in allen Szenarien ausschließlich CAMs versendet wurden und keine weiteren Nachrichten mit evtl. höherer AC-Priorität. Hier könnten sich andere Situationen ergeben, die dazu führen, dass die maximale Queue-Länge oder die Lebenszeit überschritten werden. Um in diesen Situationen gegenzusteuern, wäre die in [12] angewandte Vorgehensweise sinnvoll. Hier wurde immer nur die aktuellste CAM in die Queue eingereiht und eine evtl. schon vorhandene ältere CAM wird entnommen und verworfen. Beide Nachrichten tauschen sozusagen den Platz. Dadurch würde sich auch immer nur maximal eine CAM in der Queue befinden. Den betrachteten Standards zur DCC (siehe [10], [43] und [41]) war diese Vorgehensweise allerdings nicht zu entnehmen.

##### 4.4.2.2 Gegenüberstellung der ETSI-Varianten

Folgend ist der Fokus auf die drei ETSI-Varianten gerichtet, die in Abschnitt 4.3 nicht im Detail, sondern als ein großes Ganze mit WAVE verglichen wurden. Die Erkenntnisse werden hierbei in drei Tabellen je Szenario zusammengefasst. In diesen werden die minimalen und maximalen Median-Werte der Daten, die Grundlage der Grafiken in Anhang B sind, gegenübergestellt. Betrachtet werden einerseits alle vier Kombinationen von Beacon-Größe und Fahrzeuganzahl. Zusätzlich wird individuell

auf die drei ETSI-Varianten und die Beobachtungen in Abhängigkeit von Beacon-Größe und Fahrzeuganzahl eingegangen.

### Szenario Autobahnkreuz

In Tabelle 4.9 sind die entsprechenden Median-Werte dargestellt. Punktuell sind hier größere Unterschiede zu verzeichnen. Die Kennwerte in Abhängigkeit von Beacon-Größe und Fahrzeuganzahl werden im Detail diskutiert.

Bei allen drei Varianten befinden sich die kürzesten Intervallwerte bei der Kombination 300 Bytes und 100 Fahrzeuge mit 160 ms bei Standard und 200 ms bei beiden anderen. Die größten Unterschiede sind bei 600 Bytes und 200 Fahrzeugen zu verzeichnen. Dort wurden bei der Standardvariante 350 ms und bei LIMERIC 210 ms (Median-Werte) verwendet. Dieses Verhalten ist durch den RESTRICTIVE-Zustand zu erklären, der bei der Standardvariante 1 s als minimales Generierungsintervall vorschreibt. LIMERIC scheint hier der passende Ansatz zu sein, um eine höhere Senderate zu ermöglichen.

Bei der Kanallast verhalten sich alle drei Varianten relativ gleich in allen Kombinationen. Der größte Unterschied ist bei 300 Bytes und 100 Fahrzeugen festzustellen. Dort erreicht die Standardvariante Median-Werte von 33% und beide anderen Varianten von 27%.

Die Standardvariante hat bei den gesendeten Beacons die größten Modifizierungen bei Wechsel der Beacon-Größe und Fahrzeuganzahl zu verzeichnen. Hier sind bei 300 Bytes und 100 Fahrzeugen zunächst die höchsten Median-Werte mit 50 zu verzeichnen, welche bei 200 Fahrzeugen auf 28 abfallen und bei 600 Bytes und 200 Fahrzeugen nur noch 19 betragen. Die beiden anderen Varianten weisen hier keine so großen Unterschiede auf, welche bei DCC Profile bei von 40 auf 36 und bei LIMERIC von 49 auf 41 abfallen. Dieses Bild zeichnet sich analog bei den empfangenen Beacons ab.

Bei den Paketverlusten sind DCC Profile und LIMERIC in allen Kombinationen fast gleich auf. Durch die vorher festgestellten Erkenntnisse zeigt sich bei der Standardvariante, dass aufgrund der höchsten Anzahl gesendeter Beacons bei 300 Bytes und 100 Fahrzeugen auch die Paketverluste am größten sind.

Bei diesem Szenario sind Unterschiede zwischen der Standardvariante und den beiden anderen Varianten erkennbar. Insbesondere bei Zunahme der Beacon-Größe und der Fahrzeuganzahl sind die Varianten DCC Profile und LIMERIC besser zu bewerten, da eine höhere Anzahl gesendeter und empfangener Beacons mit moderaten Kanallasten zu verzeichnen sind.

Kennwert	Standard	DCC Profile	LIMERIC
Generierungs-intervall	160 ms bis 350 ms	200 ms bis 260 ms	200 ms bis 210 ms
Kanallast gesendete Beacons	24% bis 33%	26% bis 30%	24% bis 30%
empfangene Beacons	19 bis 50 1237 bis 3418	36 bis 42 1283 bis 2944	41 bis 50 1239 bis 2957
Paketkollisionen	343 bis 882	299 bis 771	220 bis 678

**Tabelle 4.9** – Gegenüberstellung der ETSI-Varianten anhand des Szenarios *Autobahnkreuz* bei 6 parallelen Ausführungen für je 10 s

Auch soll hier auf die Unterschiede bei Ausführung des Szenarios mit einer Simulationszeit von 200 s und 600 Bytes / 200 Fahrzeugen eingegangen werden (Abbildungen B.11 bis B.15).

Bei dem Generierungsintervall zeichnet sich ein ähnliches Bild wie zuvor festgestellt ab. Hier liegt der Median-Wert der Standardvariante bei 320 ms, gefolgt von 240 ms bei DCC Profile und 210 ms bei LIMERIC. Die Kanallast variiert bei DCC Profile am größten mit einem Median-Wert von 38%, gefolgt von der Standardvariante mit 34% und LIMERIC mit 32%. Auch bei einer längeren Simulationszeit erreicht LIMERIC die höchsten Median-Werte an gesendeten Beacons von 218, gefolgt von 211 bei DCC Profile und 155 bei der Standardvariante. Verwunderlich ist das Bild allerdings bei den Median-Werten der empfangenen Beacons. Hier performt DCC Profile mit 8913 am besten, gefolgt von der Standardvariante mit 8202 und LIMERIC bei 7534. Die meisten Paketverluste verzeichnet DCC Profile mit 3474, gefolgt von der Standardvariante mit 2635 und LIMERIC mit 2329.

Auch bei einer längeren Simulationszeit ist LIMERIC aufgrund des maximalen Median-Wertes gesendeter Beacons und der geringsten Kanallast als die optimalste Variante zu betrachten.

### Szenario Stadt

Studiert man in diesem Szenario die Unterschiede hinsichtlich des Paketintervalls sind keine Abweichungen zwischen der Standardvariante und DCC Profile festzustellen.

Dieses Bild zieht sich über alle weiteren Kennwerte. In Tabelle 4.10 ist dies deutlich zu erkennen. Die Kanallast steigt bei allen drei Varianten mit zunehmender Beacon-Größe und Fahrzeuganzahl nur moderat an. Auch bei den gesendeten Beacons ist nur eine geringfügige Abnahme bei steigender Beacon-Größe und Fahrzeuganzahl zu verzeichnen. Diese Erkenntnisse spiegeln sich auch bei den empfangenen Bea-

cons wieder. Sehr positiv sind die kaum vorhandenen Paketverluste bei allen drei Varianten.

Bei diesem Szenario zeigen sich die geringsten Unterschiede aller drei ETSI-Varianten, mit leichten Unterschieden bei LIMERIC und den beiden anderen Varianten.

Kennwert	Standard	DCC Profile	LIMERIC
Generierungs-intervall	230 ms bis 260 ms	230 ms bis 260 ms	260 ms bis 280 ms
Kanallast	3% bis 10%	3% bis 10%	3% bis 9%
gesendete Beacons	37 bis 42	37 bis 42	34 bis 37
empfangene Beacons	233 bis 415	233 bis 415	210 bis 371
Paketkollisionen	2 bis 15	2 bis 15	2 bis 15

**Tabelle 4.10 – Gegenüberstellung der ETSI-Varianten anhand des Szenarios *Stadt* bei 6 parallelen Ausführungen für je 10 s**

Auch hier wird auf die Unterschiede bei Ausführung des Szenarios mit einer Simulationszeit von 200 s und 600 Bytes / 200 Fahrzeugen eingegangen.

In Abbildung B.17b und Abbildung B.26 zeigt sich, dass die Median-Werte des Generierungsintervales bei allen drei ETSI-Varianten etwas ansteigen. Die größte Differenz ist hier bei DCC Profile von 260 ms auf 360 ms zu verzeichnen.

Auf die Kanallast wirkt sich diese Änderung nicht aus. Hier bleiben alle drei Varianten bei 10% (Abbildung B.19b und Abbildung B.27).

Die Unterschiede bei den gesendeten Beacons sind auch bei längerer Simulationszeit marginal. Hier erreicht LIMERIC den höchsten Wert von 233, gefolgt von DCC Profile mit 227 und 225 bei Standard (Abbildung B.28).

Bei den empfangenen Beacons liegt hier LIMERIC mit 3966 als Median-Wert vorne, die anderen beiden Varianten erreichen 3783 (DCC Profile) sowie 3622 (Standard) (Abbildung B.29).

Paketverluste sind bei allen drei Varianten sehr gering, wie schon bei einer kürzeren Laufzeit festgestellt. Die geringsten Median-Werte verzeichnet die Standardvariante mit 148, gefolgt von LIMERIC mit 164 und DCC Profile mit 170 (Abbildung B.30).

Die längere Ausführung dieses Szenarios zeigt, dass das Generierungsintervall bei allen drei Varianten um etwa 100 ms im Mittel größer wird. Die anderen Kennwerte differieren zwischen diesen dreien kaum.

### Szenario Stau

Viele Gemeinsamkeiten lassen sich auch bei diesem Szenario feststellen. Aus Tabelle 4.11 sind unabhängig von Beacon-Größe und Fahrzeuganzahl bei dem Generierungsintervall, der Kanallast und der gesendeten Beacons keine Unterschiede zu verzeichnen. Auffällig ist, dass die empfangenen Beacons bei 300 Bytes und 100 Fahrzeugen bei allen drei Varianten bei etwas über 800 liegen und mit zunehmender Fahrzeuganzahl und Beacon-Größe auf nur noch etwa 300 bei 600 Bytes und 200 Fahrzeugen abnehmen. Dies liegt an den im umgekehrten Verhältnis steigenden Paketverlusten, die zunächst bei etwa 170 liegen und bis zu etwa 1700 bei 600 Bytes und 200 Fahrzeugen ansteigen. Dies ist simulationsbedingt durch die gleichbleibenden Intervalle, die dazu führen, dass Beacons immer zur selben Simulationszeit versendet werden. Da die Fahrzeuge sich nicht bewegen, bleibt dieses Phänomen auch konstant erhalten und sollte nicht zu kritisch betrachtet werden.

Letztlich entscheidend ist, dass hier alle drei Varianten kaum voneinander zu unterscheiden sind.

Kennwert	Standard	DCC Profile	LIMERIC
Generierungsintervall	ausschließlich 1000 ms bei allen drei Varianten		
Kanallast	7% bis 10% bei allen drei Varianten		
gesendete Beacons	ausschließlich 10 bei allen drei Varianten		
empfangene Beacons	298 bis 820	298 bis 820	301 bis 816
Paketkollisionen	170 bis 1692	170 bis 1689	174 bis 1689

**Tabelle 4.11** – Gegenüberstellung der ETSI-Varianten anhand des Szenarios *Stau* bei 6 parallelen Ausführungen für je 10 s

Es wird folgend auf die Unterschiede bei Ausführung des Szenarios mit einer Simulationszeit von 200s und 600 Bytes / 200 Fahrzeugen eingegangen werden (Abbildungen B.41 bis B.45).

Bei den Kennwerten Generierungsintervall, Kanallast sowie gesendeter Beacons sind bei allen drei Varianten auch bei einer längeren Simulationszeit keine Unterschiede zu erkennen. Hier betragen die Median-Werte 1000 ms, 11% sowie 200 Beacons. Bei den empfangenen Beacons verzeichnet LIMERIC etwas niedrigere Werte von 6968 im Gegensatz zu 7135 bei beiden anderen Varianten. Paketverluste sind bei allen drei Varianten im Mittel bei 33550.

Die längere Simulationsaufführung zeigt keine Unterschiede zu den Erkenntnissen aus der sechsfachen parallelen Ausführung zu je 10 Sekunden.

---

## Kapitel 5

---

### Zusammenfassung

---

Im Rahmen dieser Arbeit wurden die für die Überlastkontrolle sowie für die Beacon-Generierung relevanten Mechanismen der Kommunikationsprotokolle IEEE WAVE und ETSI ITS-G5 gegenübergestellt. Durch Implementierung innerhalb der Simulationsumgebung Omnet++ unter Verwendung des Frameworks VEINS konnten die Mechanismen mit den Szenarien *Autobahnkreuz*, *Stadt* und *Stau* Kennwerte erfasst werden, die die Grundlage für eine Leistungsbewertung sind. Dabei ergeben sich zwischen beiden Protokollen erhebliche Unterschiede.

Bei WAVE zeigt sich bei allen drei Szenarien das gleiche Bild. Durch ein sehr kurzes Intervall zwischen zwei versendeten Beacons von fast ausschließlich 100 ms werden bei weitem mehr Beacons als bei ETSI ITS-G5 versendet. Dadurch ist die Anzahl empfangener Beacons auch wesentlich höher. Allerdings wird sehr häufig eine hohe Kanallast und viele Paketverluste (bedingt durch Kollisionen) verursacht bzw. in Kauf genommen. Auffällig und durchaus kritisch zu betrachten ist, dass bei WAVE die gemessene Kanallast (CBR) keine Änderungen hinsichtlich des Generierungsintervall es oder anderer Parameter, die die Kanallast senken könnten, wie z.B. die Datenrate, bewirken. Mit diesem Mechanismus erfolgt deshalb keine adäquate Gegensteuerung bei auftretenden Überlastsituationen.

Darüber hinaus wurde festgestellt, dass das Generierungsintervall, dass nach jedem versendeten Beacon neu berechnet wird, nur von der gemessenen Anzahl an Fahrzeugen im Umkreis von 100 m abhängig ist und maximal auf 600 ms gesetzt werden kann (bei 150 Fahrzeugen oder mehr). Diese Herangehensweise stellte sich als weitere Ursache für das ungenügende Verhalten bei Lastsituationen heraus. Eine Ausweitung auf 200 m zeigte auf, dass sich dadurch die Kanallast reduzieren ließe. Eine weitere Untersuchung des Generierungsintervall es ergab, dass es trotz einer längeren berechneten Periode zu einer Versendung nach 100 ms kommen kann. Dies war in Szenarien oftmals bei über der Hälfte der Simulationszeit der Fall.

Auf Seiten von ETSI ist der Generierungsmechanismus an dynamische Fahreigenschaften und an die DCC gekoppelt. Die DCC gibt in Abhängigkeit des aktuell festgelegten Zustandes mehrere Parameter vor, so z.B. das Paketintervall. Zustände werden anhand von Grenzwerten der minimal bzw. maximal gemessenen Kanallast (CBR) der letzten Sekunde bzw. letzten fünf Sekunden festgelegt. Das angesprochene Paketintervall ist für die Beacon-Generierung von besonderer Bedeutung, definiert es nämlich die minimale Zeitperiode, die zwischen zwei zu sendenden Beacons verstrichen sein muss. Liegt nach Verstreichen dieser Zeit zusätzlich eine Änderung der dynamischen Fahreigenschaften vor (z.B. Lenkeinschlag hat sich um 4° verändert), kann ein Beacon versendet werden.

Hierbei hat sich gezeigt, dass in allen drei Szenarien das Generierungsintervall situationsabhängig gesetzt wird, was im Vergleich zu WAVE zu längeren Generierungsintervallen führt. Dadurch werden weniger Beacons versendet, was in weniger empfangenen Beacons resultiert. Positiv ist dabei die in allen Szenarien geringere Kanallast sowie die enorm geringeren Paketverluste.

Da das Paketintervall eine entscheidende Rolle spielt, wurden auch noch zwei weitere ETSI-Varianten betrachtet. Die DCC Profile genannte Variante verwendet andere Zeitwerte für das Paketintervall in den Zuständen. Auch wurde das Paketintervall dynamisch in Abhängigkeit von globalen Kanallastwerten berechnet (LIMERIC). Es konnte festgestellt werden, dass in Szenarien mit hoher Kanallast die beiden letztgenannten Varianten zu höherer Anzahl von gesendeten (und dadurch auch empfangener Beacons) führen.

Abschließend sei nochmals erwähnt, dass der WAVE-Mechanismus durch kurze Generierungsintervalle eine hohe Anzahl gesendeter und empfangener Beacons ermöglicht, allerdings eine hohe Kanallast und Paketverluste in Kauf nimmt. Alle drei betrachteten ETSI-Varianten verhalten sich hier gegenteilig. Durch die an dynamische Fahreigenschaften gekoppelte Generierung von Beacons sind weniger gesendete und empfangene Beacons, dadurch allerdings auch eine viel geringere Kanallast und Paketverluste festgestellt worden.

Zukünftige Arbeiten könnten hier insbesondere bei dem WAVE-Mechanismus weitere Verbesserungsmöglichkeiten erforschen, denn die massiv hohen Kanallasten sind sehr kritisch zu betrachten. Aufgrund der festgestellten enormen Unterschiede beider Mechanismen stellt sich ebenso die Forschungsfrage, mit welcher Anzahl gesendeter und empfangener Beacons vorgesehene Safety Applications ihre Funktionalität gewährleisten können.

---

## Anhang A

---

# Übersicht aller Testfälle

---

Zur Verwaltung aller involvierten Testfälle wurde die Klasse `TestCaseHandler` im Verzeichnis `src.veins.masterthesis.util` erstellt. Alle dort festgelegten Testfälle können mit den im Verzeichnis `examples.masterthesis.Tests` enthaltenen Szenarien ausgeführt werden. Die Testfallnummer ist innerhalb der `omnetpp.ini` über den Parameter `*.**.testCaseNumber` zu setzen. Die Testfälle sind zum Teil nur für WAVE oder ETSI vorgesehen. Deswegen sind für beide Implementierungen je eine `omnetpp.ini` in dem entsprechenden Verzeichnis unter `examples.masterthesis.Tests` vorhanden. Folgend werden alle möglichen Testfälle erläutert.

- 0.0** Hiermit wird der normale Code ausgeführt und es passieren keine Parameterprüfungen.
- 1.1** ETSI-spezifischer Testfall setzt den Wert für CBR konstant auf 14,9%. Dadurch darf der Zustand RELAXED nicht gewechselt werden, was durch ASSERT-Methoden abgeprüft wird. Im WAVE-Modus führt diese Nummer zu einem Abbruch der Simulation.
- 1.2** Analog zu Nummer 1.1, nur wird CBR kontinuierlich auf 15% gesetzt. Dadurch geschieht nach einer Sekunde der Zustandsübergang auf ACTIVE. Dieser Zustand wird bis zum Ende der Simulation nicht verlassen.
- 1.3** Analog zu Nummer 1.2, nur wird CBR kontinuierlich auf 39,9% gesetzt. Dadurch erfolgt nach einer Sekunde der Zustandsübergang auf ACTIVE. Dieser Zustand wird bis zum Ende der Simulation nicht verlassen.
- 1.4** Analog zu Nummer 1.1, nur wird CBR kontinuierlich auf 40% gesetzt. Dadurch erfolgt nach einer Sekunde der Zustandsübergang auf ACTIVE und einer weiteren Sekunde der Wechsel zu RESTRICTIVE. Dieser Zustand wird bis zum Ende der Simulation nicht verlassen.

- 1.5 ETSI-spezifischer Testfall. Während der Simulationszeit von einschließlich 1 bis einschließlich 1.1 wird CBR auf 20% gesetzt, während der Simulationszeit von einschließlich 6 bis einschließlich 7.1 wird CBR auf 80% gesetzt. Das Verhalten der Zustandsmaschine wird über die Konsole ausgegeben. Es erfolgt ein Wechsel nach ACTIVE, dann zu RESTRICTIVE, anschließend werden die Echtwerte der CBR verwendet, die unter 15% liegen, was nach 5 Sekunden im RESTRICTIVE-Zustand zu einem Wechsel nach ACTIVE, und nach weiteren 5 Sekunden zu einem Wechsel zu RELAXED führt. Im WAVE-Modus führt diese Nummer zum Abbruch.
- 1.6 ETSI-spezifischer Testfall. Testet die Korrektheit der Zeitenerfassung im Rahmen der CBR-Berechnung in der Klasse ChannelLoadListenerEtsi. Zum Zeitpunkt der Berechnung wird ein Intervall betrachtet, dass alle Busy-Zeiten berücksichtigt, die beginnend von der aktuellen Zeit abzüglich von 100 ms bis zur aktuellen Zeit erfasst wurden. Dabei werden auch Zeiten berücksichtigt, die nur zum Teil in dieses Intervall fallen. Diese werden anteilig berücksichtigt. Bei WAVE führt diese Nummer zum Abbruch.
- 2.1 WAVE-spezifischer Testfall. Der Wert für CBP wird konstant auf 49,9% gesetzt. Fokus lag hierbei auf Beobachtungen der Auswirkungen bei diesem Wert. Bei ETSI führt dies zum Abbruch.
- 2.2 Analog zu 2.1 nur CBP wird konstant auf 50% gesetzt. Bei ETSI führt dies zum Abbruch.
- 2.3 Analog zu 2.1 nur CBP wird konstant auf 79,9% gesetzt. Bei ETSI kommt dies zum Abbruch.
- 2.4 Analog zu 2.1 nur CBP wird konstant auf 80% gesetzt. Bei ETSI führt dies zum Abbruch.
- 2.5 WAVE-spezifischer Testfall. Wie bei Nummer 1.6 wird die Zeiterfassung der Busy-Zeiten getestet, allerdings in Klasse ChannelBusyListenerWave.
- 3.1 WAVE-spezifischer Testfall. Innerhalb der Klasse PERAndChannelQualityIndicatorCalculator wird das Verhalten getestet, wenn keine BSM eines RV eintraf.
- 3.2 Analog zu 3.1, nur ist eine BSM eingetroffen, allerdings nicht innerhalb der letzten 5s. Hierbei wird [17], Kapitel A.8.3, Beispiel 1, nachempfunden.
- 3.3 Analog zu 3.2, BSM wurde innerhalb der letzten 5 s empfangen.
- 3.4 Analog zu 3.1, Nachbildung von [17], Kapitel A.8.3, Beispiel 2a.

- 3.5 Analog zu 3.4, Nachbildung von [17], Kapitel A.8.3, Beispiel 2b (Fahrzeug ist außerhalb von 100 m).
- 3.6 Analog zu 3.1, hier befinden sich zwei Fahrzeuge innerhalb von 100 m, beide Fahrzeuge haben PER von 0,75.
- 4.1 WAVE-spezifischer Testfall. Testet die Berechnung des Tracking Errors innerhalb der Klasse TrackingErrorCalculator. `deltaTimeHVLocalEstimate` wird auf 0,06 gesetzt, dadurch `trackingError` = 0.
- 4.2 WAVE-spezifischer Testfall. Analog zu 4.1, nur `deltaTimeHVRemoteEstimate` wird auf 3,1 gesetzt, dadurch `trackingError` = 0.
- 4.3 WAVE-spezifischer Testfall. Analog zu 4.1, nur `deltaTimeHVLocalEstimate` wird auf 0,04 und `deltaTimeHVRemoteEstimate` wird auf 0,04 gesetzt, `trackingError` wird mit bestimmter Koordinate berechnet und ergibt 1.
- 4.4 WAVE-spezifischer Testfall. Analog zu 4.3, nur `deltaTimeHVLocalEstimate` wird auf 0,06 und `deltaTimeHVRemoteEstimate` wird auf 0,06 gesetzt, `trackingError` wird mit bestimmter Koordinate berechnet.
- 5.1 WAVE-spezifischer Testfall. Testet die Berechnung der Transmission Probability in der Klasse BaseWaveApplLayer (Zeilen 1009 bis 1019). Tracking Error wird auf 0,19 gesetzt, Transmission Probability ist dadurch 0.
- 5.2 WAVE-spezifischer Testfall. Analog zu 5.1, nur Tracking Error wird auf 0,20 gesetzt, Transmission Probability ist dadurch 0.
- 5.3 WAVE-spezifischer Testfall. Analog zu 5.1, nur Tracking Error wird auf 0,49 gesetzt, Transmission Probability wird berechnet.
- 5.4 WAVE-spezifischer Testfall. Analog zu 5.1, nur Tracking Error wird auf 0,5 gesetzt, Transmission Probability ist dadurch 1.
- 5.5 WAVE-spezifischer Testfall. Analog zu 5.1, nur Tracking Error wird auf 0,8 gesetzt, Transmission Probability ist dadurch 1.
- 6.1 WAVE-spezifischer Testfall. Testet die MaxITT innerhalb der Klasse MaximumInterTransmitTimeCalculator. Vehicle Density in Range wird auf 0 gesetzt, dadurch Smoothed Vehicles Density in Range auch 0.
- 6.2 WAVE-spezifischer Testfall. Analog zu 6.1, nur Smoothed Vehicles Density in Range wird auf 25 gesetzt, dadurch ist MaxITT gleich 0,1.
- 6.3 WAVE-spezifischer Testfall. Analog zu 6.2, nur Smoothed Vehicles Density in Range wird auf 26 gesetzt, dadurch ist MaxITT gleich 0,104.

- 6.4** WAVE-spezifischer Testfall. Analog zu 6.3, nur Smoothed Vehicles Density in Range wird auf 149 gesetzt, dadurch ist MaxITT gleich 0,596.
- 6.5** WAVE-spezifischer Testfall. Analog zu 6.3, nur Smoothed Vehicles Density in Range wird auf 150 gesetzt, dadurch ist MaxITT gleich 0,6.
- 7.1** WAVE-spezifischer Testfall. Testet die berechnete Radiated Power innerhalb der Klasse BaseWaveApplLayer (Zeilen 1034 bis 1056). txDecision\_Dynamics wird auf true gesetzt, dadurch wird rp mit 20 belegt.
- 7.2** WAVE-spezifischer Testfall. Analog zu 7.1, nur txDecision\_Dynamics wird auf false, CBP auf 50% gesetzt. Dadurch muss fCBP den Wert 20 erhalten.
- 7.3** WAVE-spezifischer Testfall. Analog zu 7.2, nur CBP wird auf 51% gesetzt. Dadurch muss fCBP den Wert 19,6 erhalten.
- 7.4** WAVE-spezifischer Testfall. Analog zu 7.2, nur CBP wird auf 80% gesetzt. Dadurch muss fCBP den Wert 10 erhalten.
- 7.5** WAVE-spezifischer Testfall. Analog zu 7.4, nur CBP wird auf 81% gesetzt.
- 7.6** WAVE-spezifischer Testfall. Analog zu 7.4, nur CBP wird auf 100% gesetzt.
- 8.0** Testfall für WAVE und ETSI. Beim Versenden einer BSM oder einer CAM werden zwei Variablen mit einem bestimmten beigefügt. Bei allen Empfängern werden diese Variablen ausgelesen und auf Gleichheit geprüft.
- 9.1** ETSI-spezifischer Testfall. Testet die DCC Transmit Queue. Die Lifetime einer CAM ist älter als eine Sekunde, was zum Verwerfen der CAM führt. Führt im WAVE-Modus zum Abbruch.
- 9.2** ETSI-spezifischer Testfall. Die DCC Transmit Queue hat bereits 2 CAM eingereiht. Eine dritte CAM soll eingereiht werden, was zum Verwerfen dieser führt. Führt im WAVE-Modus zum Abbruch.

Anmerkung: Bei der Ausführung eines ETSI-Testfalles wird unabhängig von der Nummer bei Aufruf der Methode performCAMGenerating in der Klasse BaseWaveApplLayer ein Test der Berechnungsmethoden der dynamischen Fahreigenschaften ausgeführt (siehe Klasse CAMGenerationHelper, Methode performTests).

Wird eine andere TestCaseNumber als zuvor aufgeführt verwendet, führt dies zum Abbruch.

---

## Anhang B

---

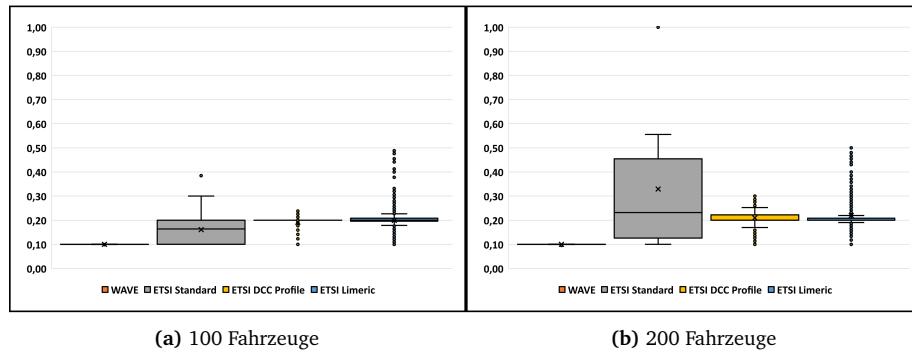
### Grafische Gegenüberstellung aller Evaluierungskennwerte

---

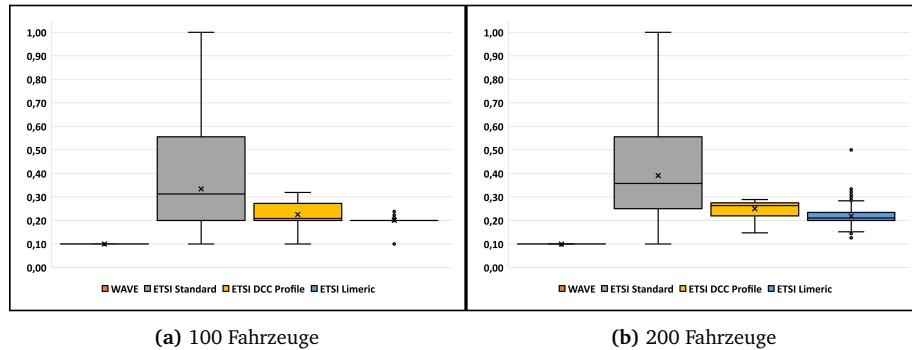
Alle nun folgenden Boxplot-Grafiken wurden im Rahmen von Abschnitt 4.3 für eine Leistungsbewertung zwischen beiden Mechanismen verwendet. In allen Boxplots ist die Reihenfolge der aufgelisteten Werte wie folgt zuzuordnen: WAVE (Farbe Orange), ETSI mit Standardparametern (Farbe Grau), ETSI mit DCC Profile (Farbe Gelb), ETSI mit dynamischer Paketintervallberechnung anhand von LIMERIC (Farbe Blau). Alle Daten wurden zunächst parallel mit sechs Ausführungen zu je 10 Sekunden Simulationszeit aufgezeichnet. Anschließend wurde für jedes Szenario eine Ausführung mit einer Simulationszeit von 200 Sekunden durchgeführt.

## B.1 Autobahnkreuz

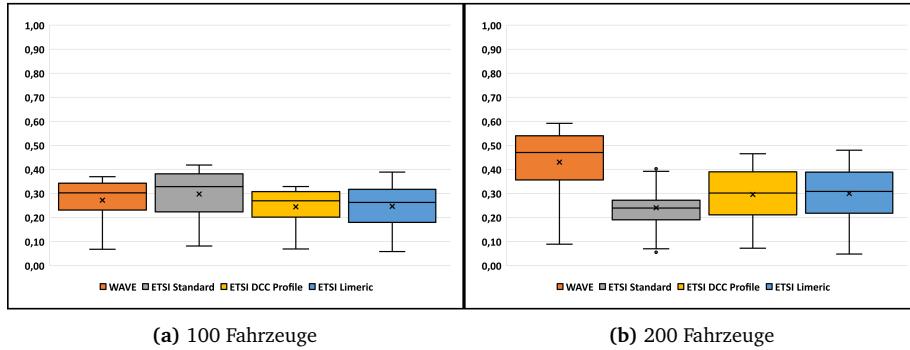
### B.1.1 Sechsfache parallele Ausführung mit einer Simulationszeit von je 10 Sekunden



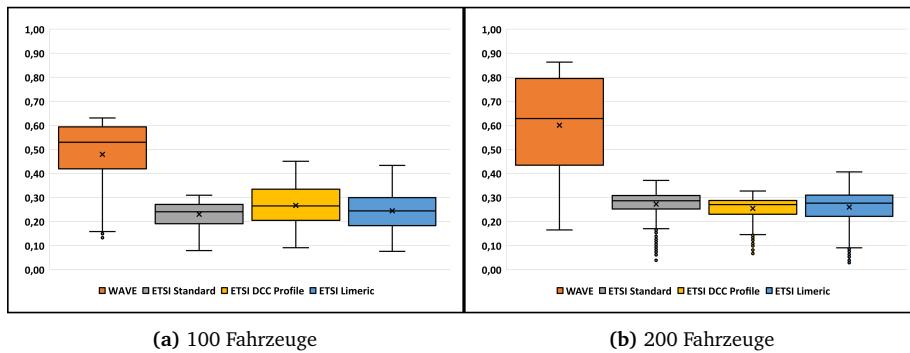
**Abbildung B.1** – Generierungsintervall in Sekunden bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden



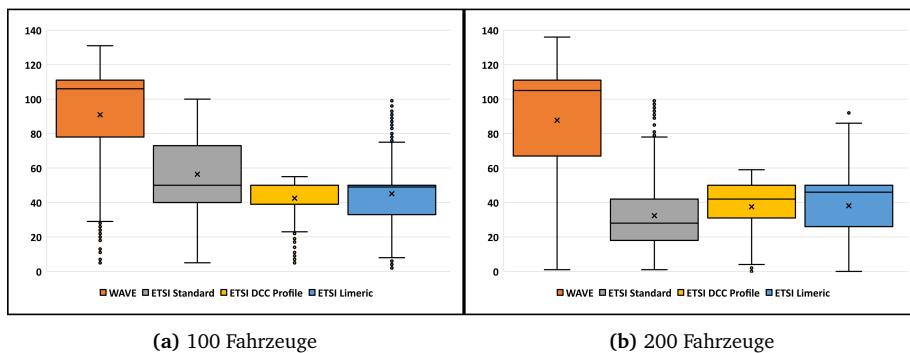
**Abbildung B.2** – Generierungsintervall in Sekunden bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden



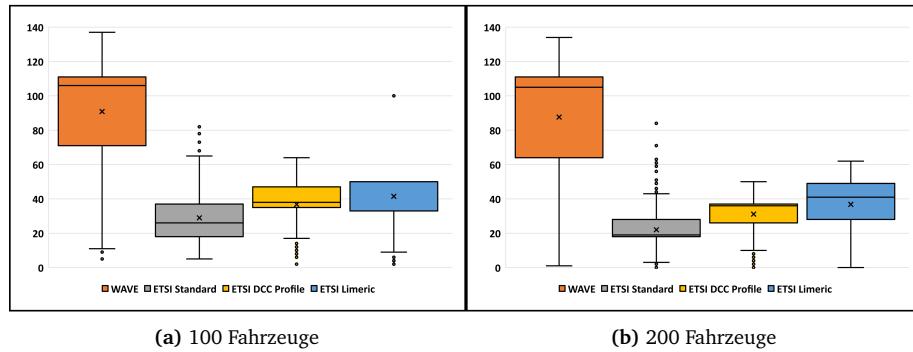
**Abbildung B.3** – Kanallast (CBR) bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden



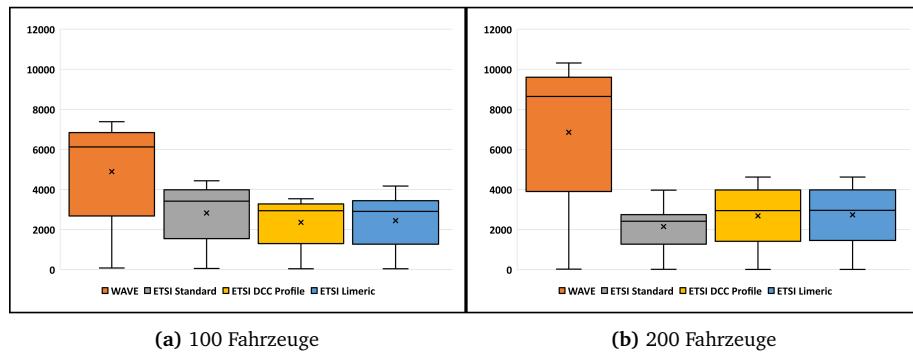
**Abbildung B.4** – Kanallast (CBR) bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden



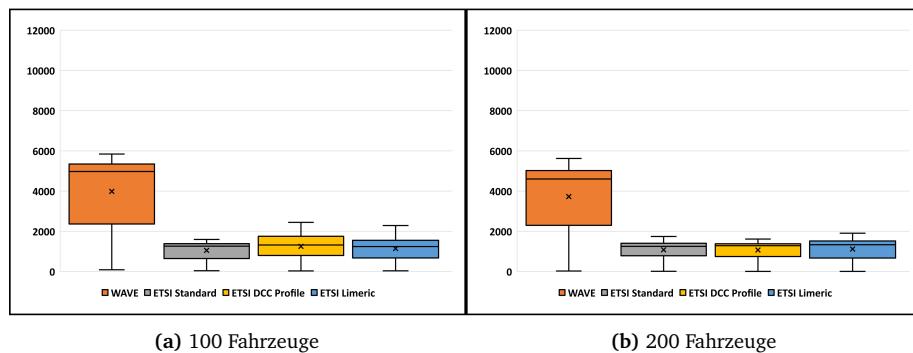
**Abbildung B.5** – Gesendete Beacons bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden



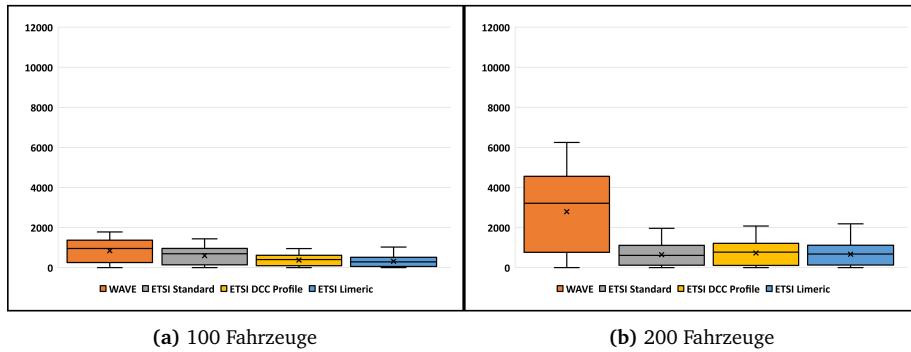
**Abbildung B.6** – Gesendete Beacons bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden



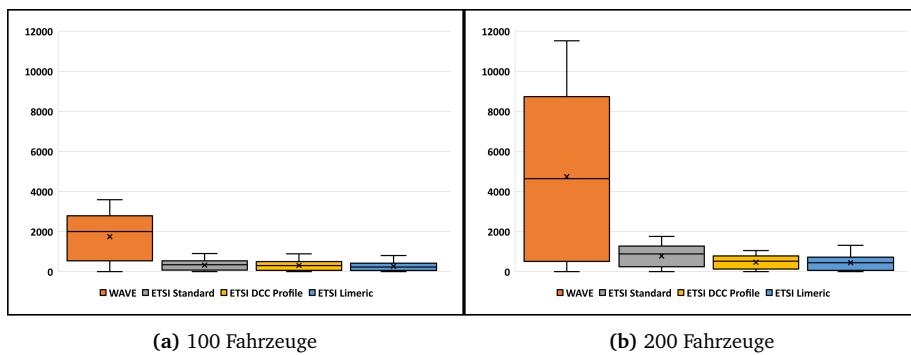
**Abbildung B.7** – Empfangene Beacons bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden



**Abbildung B.8** – Empfangene Beacons bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden

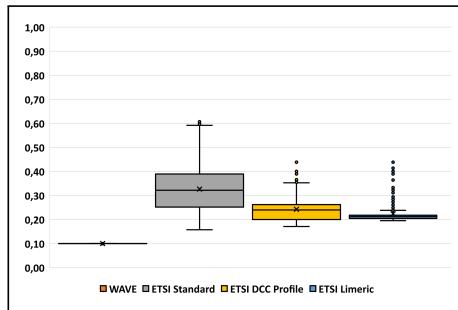


**Abbildung B.9** – Paketverluste bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden

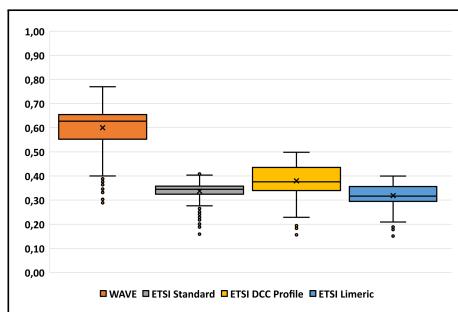


**Abbildung B.10** – Paketverluste bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden

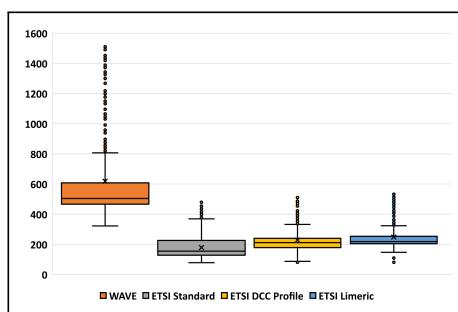
### B.1.2 Ausführung dieses Szenarios mit einer Simulationszeit von 200 Sekunden



**Abbildung B.11** – Generierungsintervall in Sekunden bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario Autobahnkreuz



**Abbildung B.12** – Kanallast (CBR) bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario Autobahnkreuz



**Abbildung B.13** – Gesendete Beacons bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario Autobahnkreuz

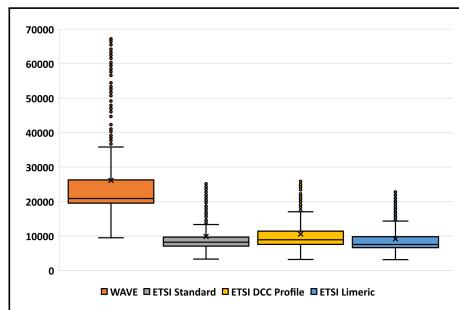


Abbildung B.14 – Empfangene Beacons bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Autobahnkreuz*

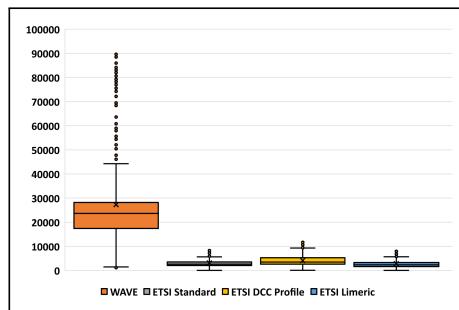
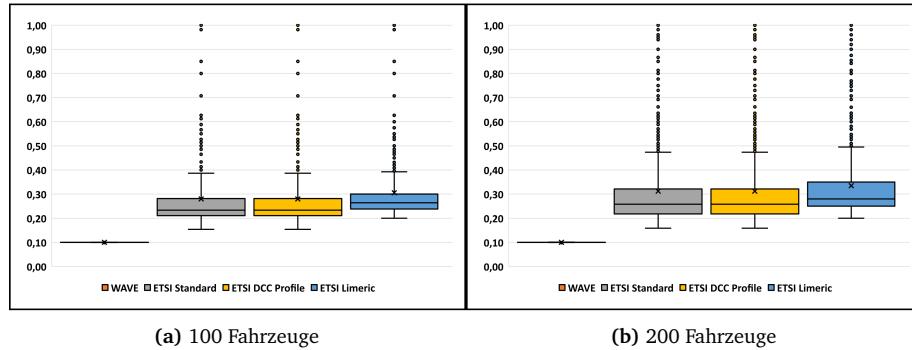


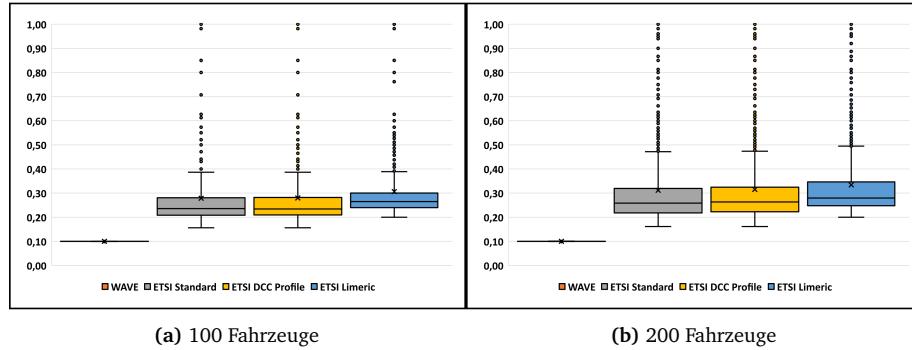
Abbildung B.15 – Paketverluste bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Autobahnkreuz*

## B.2 Stadt

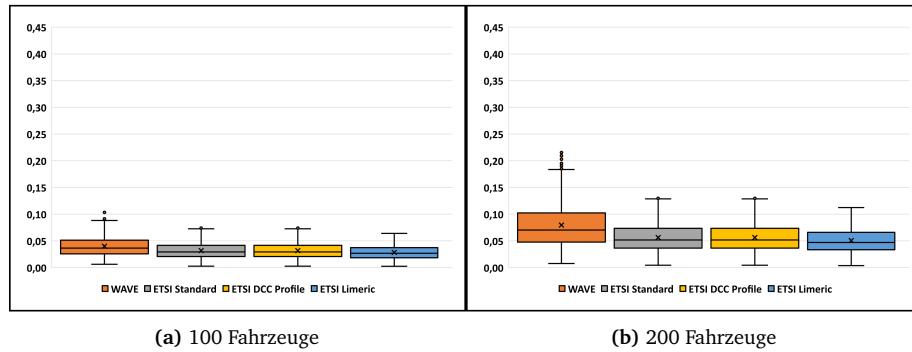
### B.2.1 Sechsfache parallele Ausführung dieses Szenarios mit einer Simulationszeit von je 10 Sekunden



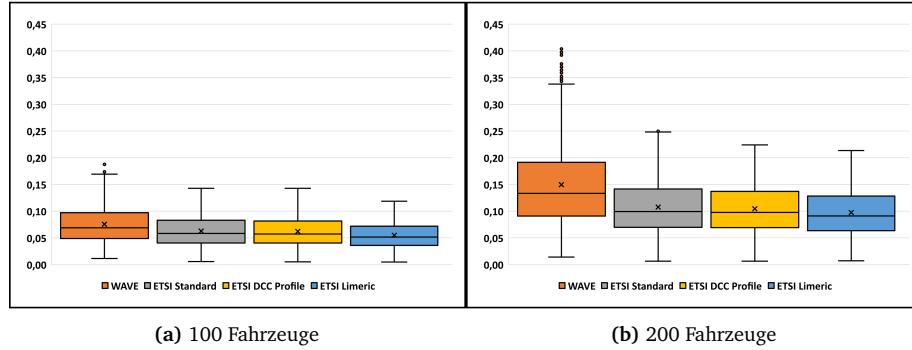
**Abbildung B.16** – Generierungsintervall in Sekunden bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stadt* mit einer Simulationszeit von je 10 Sekunden



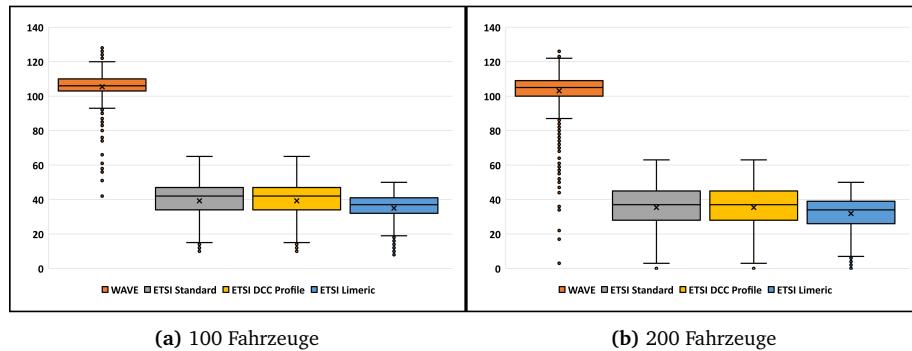
**Abbildung B.17** – Generierungsintervall in Sekunden bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stadt* mit einer Simulationszeit von je 10 Sekunden



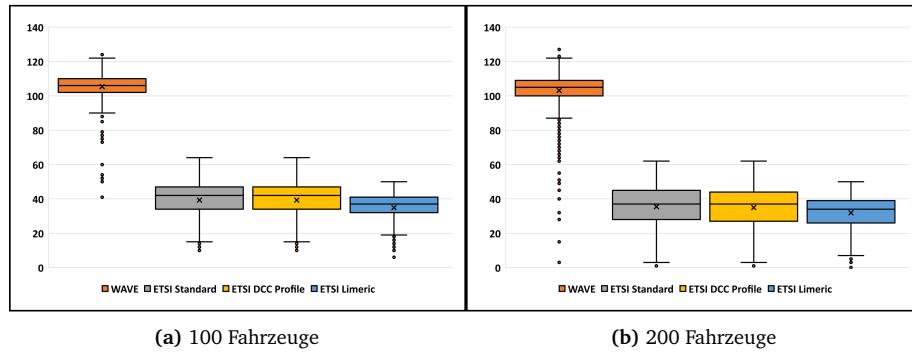
**Abbildung B.18** – Kanallast (CBR) bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stadt* mit einer Simulationszeit von je 10 Sekunden



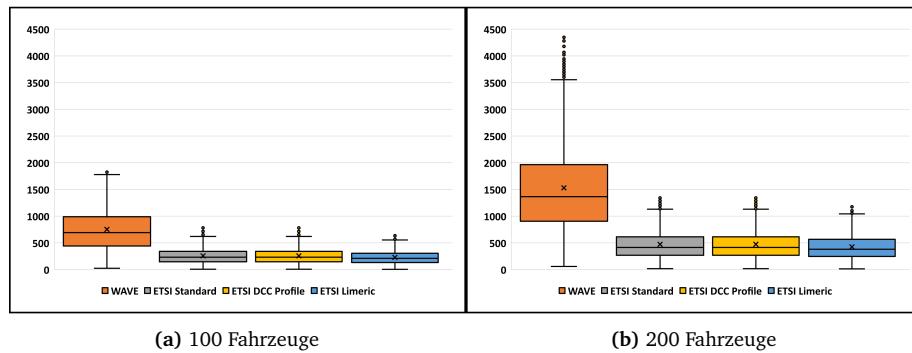
**Abbildung B.19** – Kanallast (CBR) bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stadt* mit einer Simulationszeit von je 10 Sekunden



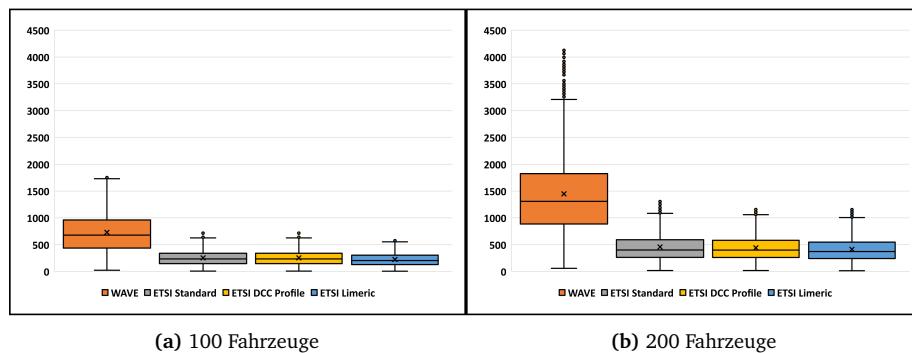
**Abbildung B.20** – Gesendete Beacons bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stadt* mit einer Simulationszeit von je 10 Sekunden



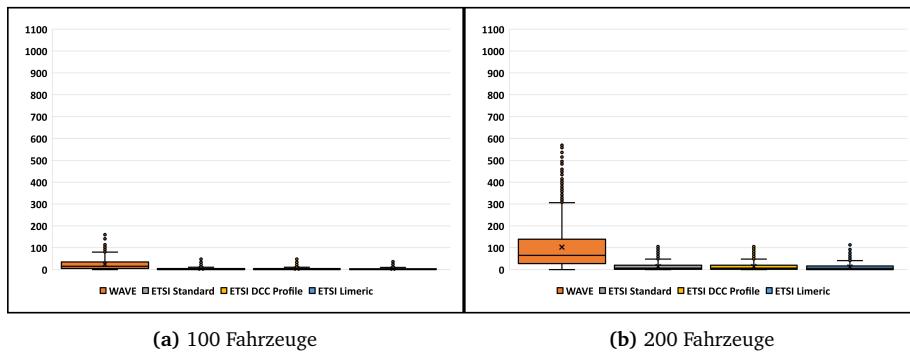
**Abbildung B.21** – Gesendete Beacons bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Autobahnkreuz* mit einer Simulationszeit von je 10 Sekunden



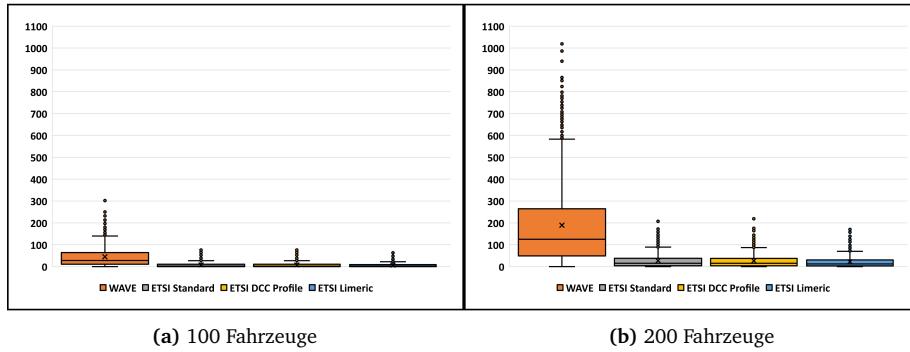
**Abbildung B.22** – Empfangene Beacons bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stadt* mit einer Simulationszeit von je 10 Sekunden



**Abbildung B.23** – Empfangene Beacons bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stadt* mit einer Simulationszeit von je 10 Sekunden

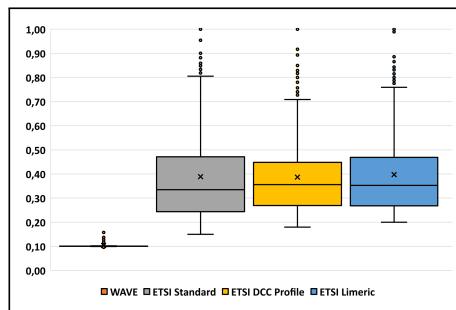


**Abbildung B.24** – Paketverluste bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stadt* mit einer Simulationszeit von je 10 Sekunden



**Abbildung B.25** – Paketverluste bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stadt* mit einer Simulationszeit von je 10 Sekunden

### B.2.2 Ausführung dieses Szenarios mit einer Simulationszeit von 200 Sekunden



**Abbildung B.26** – Generierungsintervall in Sekunden bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Stadt*

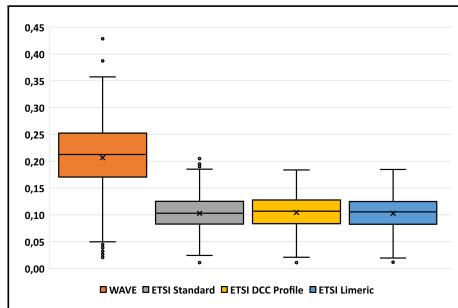


Abbildung B.27 – Kanallast (CBR) bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Stadt*

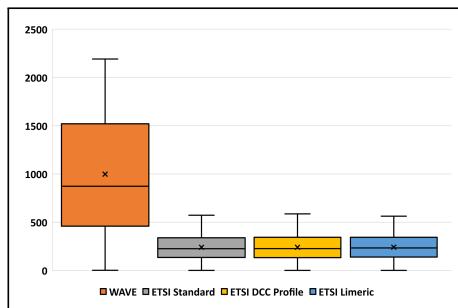


Abbildung B.28 – Gesendete Beacons bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Stadt*

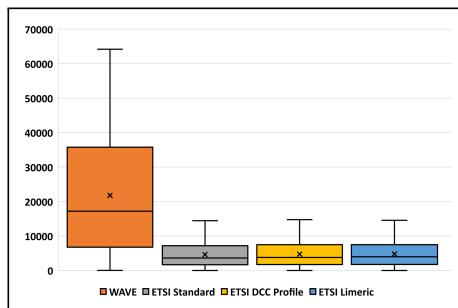
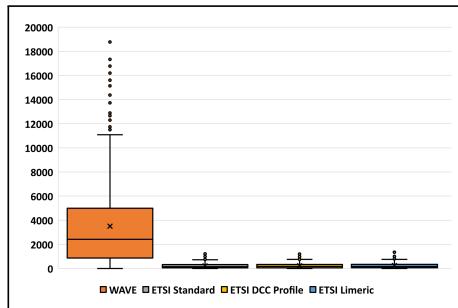


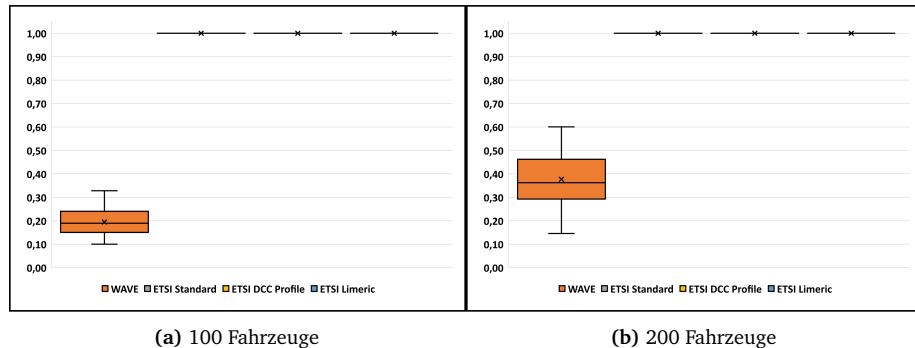
Abbildung B.29 – Empfangene Beacons bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Stadt*



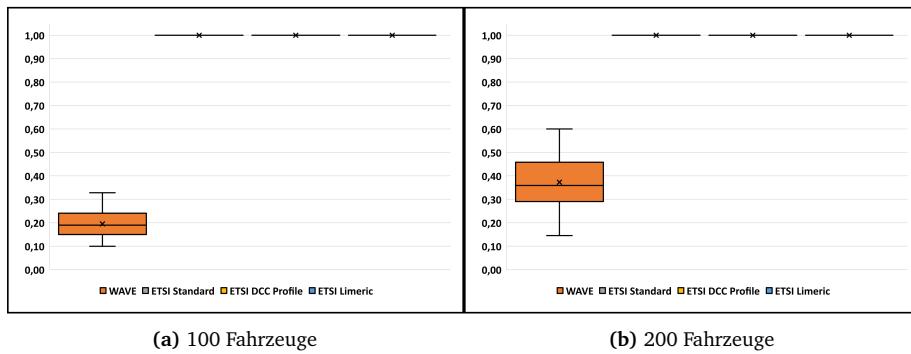
**Abbildung B.30** – Paketverluste bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Stadt*

## B.3 Stau

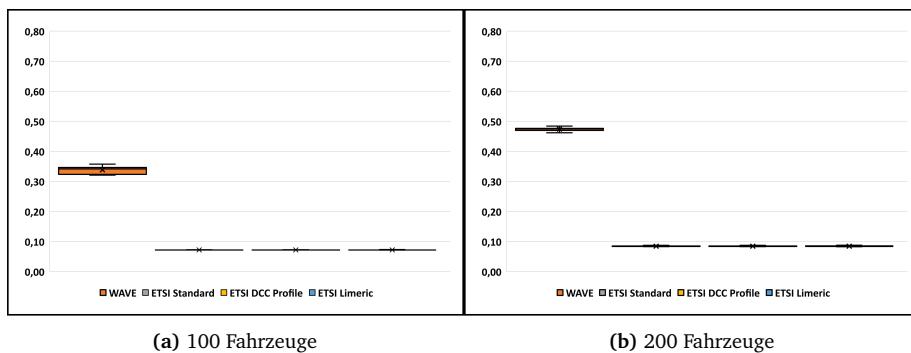
### B.3.1 Sechsfache parallele Ausführung dieses Szenarios mit einer Simulationszeit von je 10 Sekunden



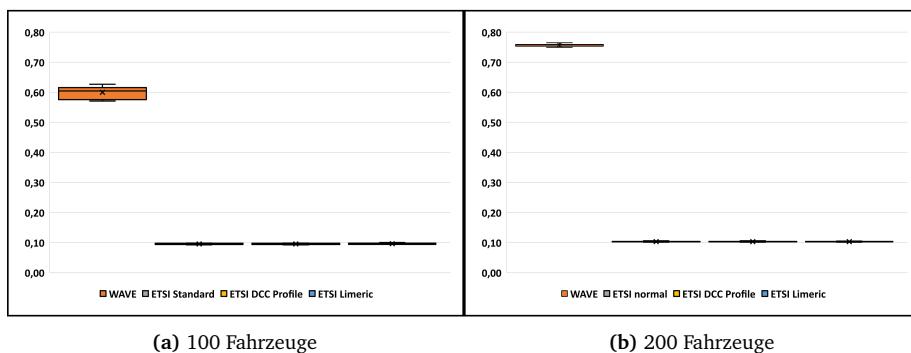
**Abbildung B.31** – Generierungsintervall in Sekunden bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stau* mit einer Simulationszeit von je 10 Sekunden



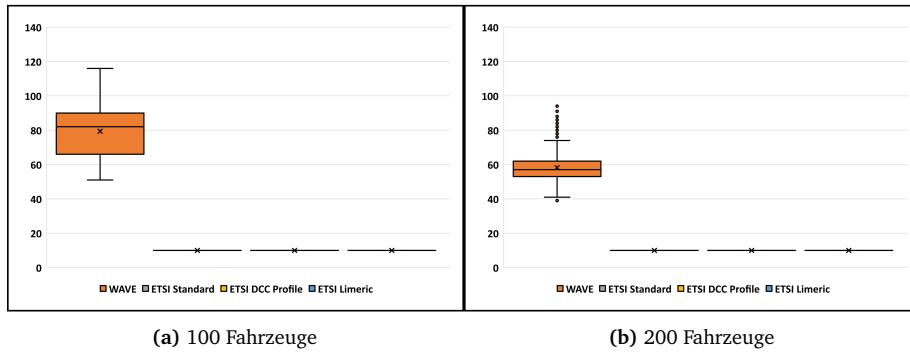
**Abbildung B.32** – Generierungsintervall in Sekunden bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stau* mit einer Simulationszeit von je 10 Sekunden



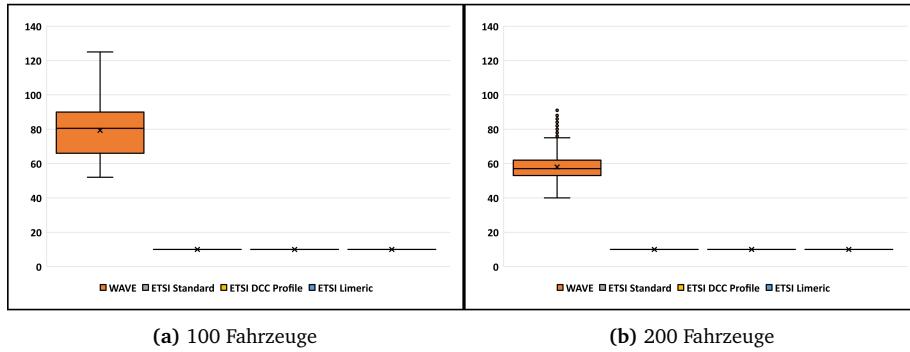
**Abbildung B.33 – Kanallast (CBR) bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stau* mit einer Simulationszeit von je 10 Sekunden**



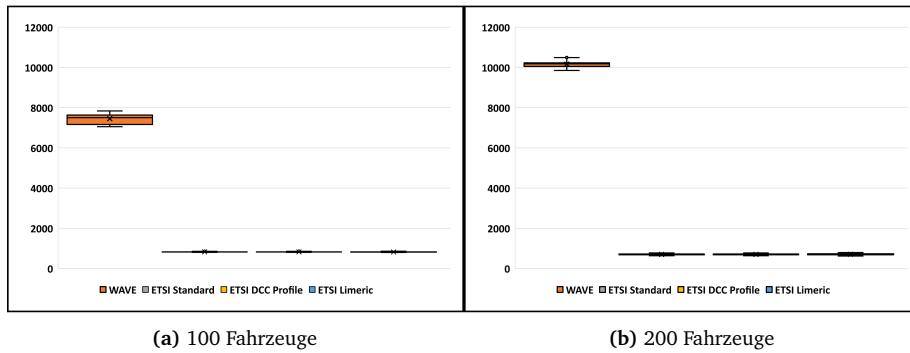
**Abbildung B.34** – Kanallast (CBR) bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stau* mit einer Simulationszeit von je 10 Sekunden



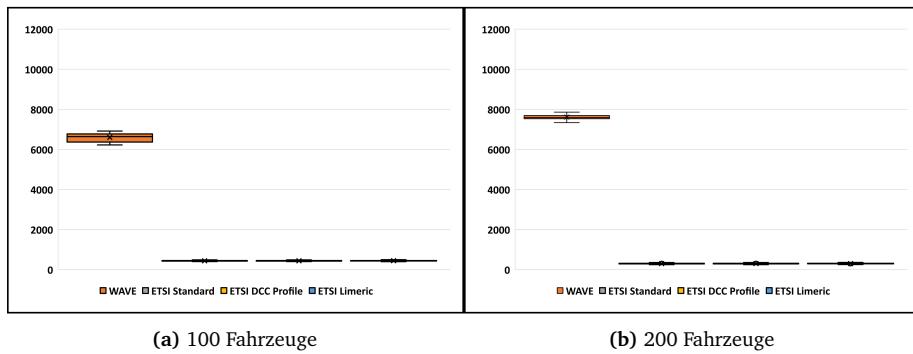
**Abbildung B.35** – Gesendete Beacons bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stau* mit einer Simulationszeit von je 10 Sekunden



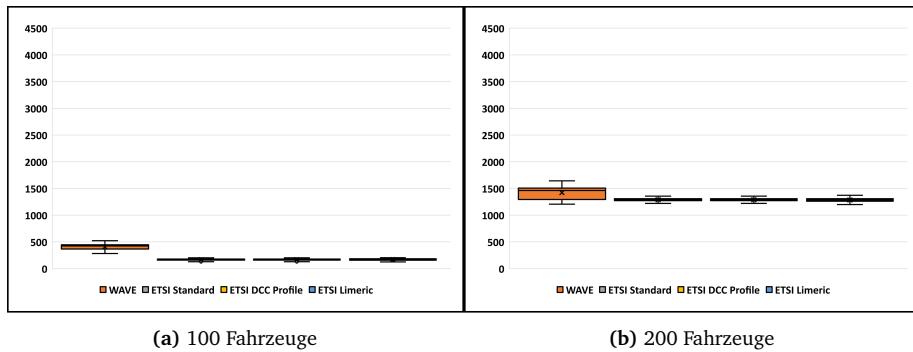
**Abbildung B.36** – Gesendete Beacons bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stau* mit einer Simulationszeit von je 10 Sekunden



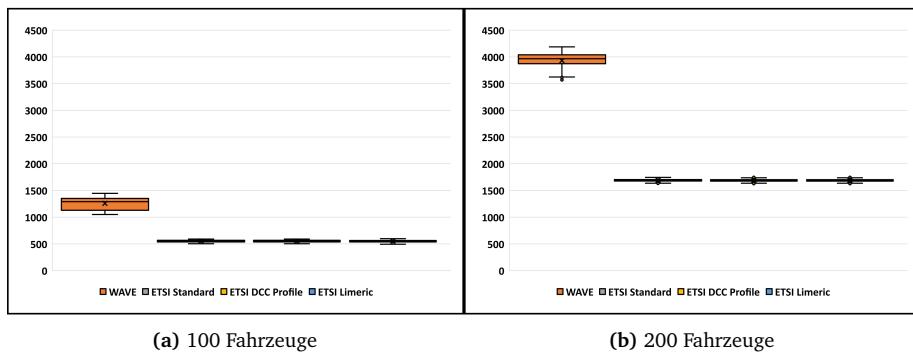
**Abbildung B.37** – Empfangene Beacons bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stau* mit einer Simulationszeit von je 10 Sekunden



**Abbildung B.38** – Empfangene Beacons bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stau* mit einer Simulationszeit von je 10 Sekunden



**Abbildung B.39** – Paketverluste bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stau* mit einer Simulationszeit von je 10 Sekunden



**Abbildung B.40** – Paketverluste bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios *Stau* mit einer Simulationszeit von je 10 Sekunden

### B.3.2 Ausführung dieses Szenarios mit einer Simulationszeit von 200 Sekunden

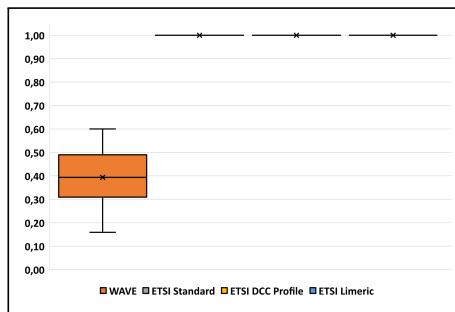


Abbildung B.41 – Generierungsintervall bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Stau*

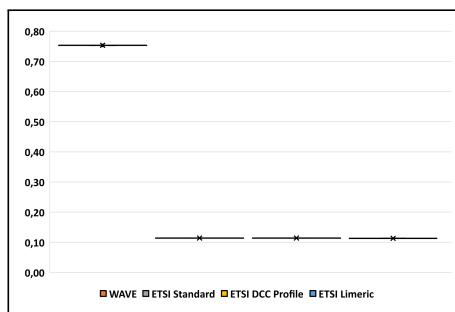


Abbildung B.42 – Kanallast (CBR) bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Stau*

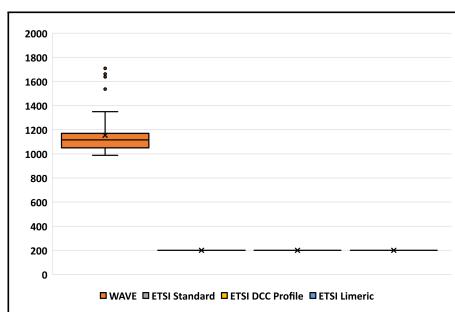
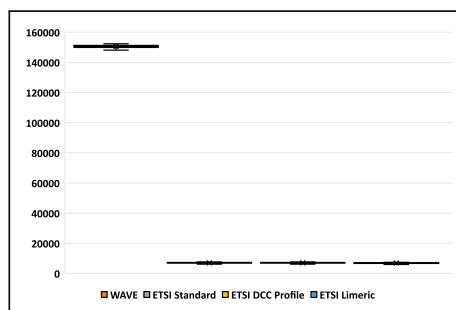
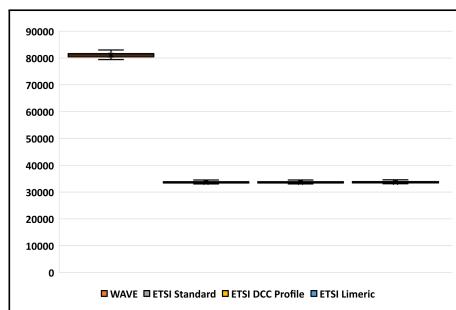


Abbildung B.43 – Gesendete Beacons bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Stau*



**Abbildung B.44** – Empfangene Beacons bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Stau*



**Abbildung B.45** – Paketverluste bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario *Stau*

---

## Anhang C

---

# Inhalt des Datenträgers

---

Auf beigefügtem Datenträger befinden sich alle Source-Code-Dateien sowie LATEX-Dateien zum Erstellen dieser Arbeit. Folgende Ordnerstruktur wurde hierfür verwendet.

**Boxplots** In diesem Verzeichnis befinden sich alle in Abschnitt 4.3 verwendeten Boxplot-Grafiken im PDF-Format. `boxplotsEqual` beinhaltet alle Boxplots mit einheitlichen Y-Achsenwerten für einen einfachen Vergleich aller Szenarien. In `boxplotsIndividual` sind die in Anhang B aufgeführten Boxplots enthalten.

**testVeins** In Abschnitt 3.5 wurde das Testen der Implementierung erläutert. Der hierfür relevante Sourcecode befindet sich in diesem Verzeichnis. Durch Import des gesamten Codes in ein Omnet++-Projekt kann dieser ausgeführt werden.

**thesis** Unter diesem Verzeichnis sind alle LATEX-Dateien zum Erstellen dieser Arbeit vorhanden.

**veins** Hierin befindet sich der Sourcecode, mit dem alle Szenarien ausgeführt wurden. In Abschnitt 3.3 und Abschnitt 3.4 wird auf Sourcecode eingegangen, der sich in diesem Verzeichnis befindet. Durch Import des gesamten Codes in ein Omnet++-Projekt kann dieser ausgeführt werden.

In dem Ordner `veins` wurde folgende Verzeichnisstruktur festgelegt:

- `examples.masterthesis`: Hierin befinden sich die drei Szenarien *Autobahnkreuz*, *Stadt* und *Stau*. In jedem dieser Verzeichnisse befinden sich Unterverzeichnisse für die Ausführung mit dem WAVE- oder ETSI-Mechanismus mittels der jeweiligen `omnetpp.ini`.
- `src.veins.masterthesis`: Hierin befinden sich alle eigens für diese Arbeit erstellten Klassen und NED-Dateien.

- `src/veins/modules/application.*`: Hierin befinden sich die Klassen `BaseWaveApplLayer.cc`, `TraCIDemoRSU11p.cc` und `TraCIDemo11p.cc`.
- `src/veins/modules/mac.*`: In diesem Verzeichnis befindet sich die Klasse `Mac1609_4.cc`.

---

## **Abkürzungsverzeichnis**

---

<b>AC</b>	Access Category
<b>AIFSN</b>	Arbitration Interframe Spacing Number
<b>BSM</b>	Basic Safety Message
<b>BSW</b>	Blind Spot Warning
<b>CAM</b>	Cooperative Awareness Message
<b>CBP</b>	Channel Busy Percentage
<b>CBR</b>	Channel Busy Ratio
<b>CCA</b>	Clear Channel Assessment
<b>CCH</b>	Control Channel
<b>C-ITS</b>	Cooperative Intelligent Transport System
<b>CLW</b>	Control Loss Warning
<b>CW</b>	Contention Window
<b>DCC</b>	Decentralized Congestion Control
<b>DENM</b>	Decentralized Environmental Notification Message
<b>DSC</b>	DCC Sensitivity Control
<b>DSRC</b>	Dedicated Short Range Communications
<b>EDCA</b>	Enhanced Distributed Channel Access
<b>EEBL</b>	Emergency Electronic Brake Lights
<b>ETSI</b>	European Telecommunications Standards Institute
<b>FCC</b>	Federal Communications Commission

<b>FCW</b>	Forward Crash Warning
<b>HV</b>	Host Vehicle
<b>IETF</b>	Internet Engineering Task Force
<b>IPv6</b>	Internet Protocol Version 6
<b>IMA</b>	Intersection Movement Assist
<b>LCW</b>	Lane Change Warning
<b>LDM</b>	Local Dynamic Map
<b>LTA</b>	Left Turn Assist
<b>MaxITT</b>	Maximum BSM Generation Interval
<b>NED</b>	Network Description
<b>NDL</b>	Network Design Limits
<b>PER</b>	Packet Error Ratio
<b>POTI</b>	Position and Time Management
<b>QoS</b>	Quality of Service
<b>RV</b>	Remote Vehicle
<b>SAE Intern.</b>	Society of Automotive Engineers
<b>SCH</b>	Service Channel
<b>SUMO</b>	Simulation of Urban Mobility
<b>TCP</b>	Transport Control Protocol
<b>TDC</b>	Transmit Datarate Control
<b>TPC</b>	Transmit Power Control
<b>TraCI</b>	Traffic Control Interface
<b>TRC</b>	Transmit Rate Control
<b>UDP</b>	User Datagram Protocol
<b>VANET</b>	Vehicular Ad Hoc Network
<b>VDP</b>	Vehicle Data Provider

<b>VEINS</b>	Vehicles in Network Simulation
<b>V2V</b>	Vehicle-to-Vehicle
<b>WAVE</b>	Wireless access in vehicular environments
<b>WSMP</b>	WAVE Short Message Protocol

---

## Algorithmusverzeichnis

---

2.1	Berechnung des <i>Channel Quality Indicators</i> . . . . .	15
2.2	Berechnung des <i>Tracking Errors</i> . . . . .	16
2.3	<i>Transmission Decision</i> . . . . .	17
2.4	<i>Schedule Transmission</i> . . . . .	18
2.5	Berechnung der <i>Radiated Power</i> . . . . .	18
2.6	Generierung von BSMs . . . . .	19
2.7	Generierung von CAMs . . . . .	33

---

# Abbildungsverzeichnis

---

2.1	Kanalallokierung bei WAVE . . . . .	6
2.2	Protokollstack bei WAVE . . . . .	7
2.3	BSM-Format . . . . .	9
2.4	Übersicht über die Generierung einer BSM . . . . .	11
2.5	PER Berechnungsintervalle . . . . .	13
2.6	Kanalallokierung bei ETSI . . . . .	20
2.7	Protokollstack bei ETSI . . . . .	21
2.8	CAM Basic Service . . . . .	22
2.9	CAM Format . . . . .	23
2.10	DCC Architektur . . . . .	25
2.11	DCC Access . . . . .	26
2.12	DCC Zustandsmaschine für den CCH . . . . .	28
2.13	Beispiel der verschiedenen Kanallastwerte des Geonetworking Protokolls	31
3.1	Architektur des VEINS-Frameworks . . . . .	35
3.2	SUMO Screenshot während der Ausführung des Szenarios <i>Luxemburg</i> mit farblicher Unterscheidung von Straßen (schwarz), Gebäuden (rot) und Parkplätzen (grau) . . . . .	36
4.1	SUMO-Gui-Screenshot des Szenarios <i>Autobahnkreuz</i> . . . . .	56
4.2	Kanallastverlauf des Fahrzeuges <i>Node2</i> bei WAVE während des Szenarios <i>Autobahnkreuz</i> . . . . .	61
4.3	Kanallastverlauf des Fahrzeuges <i>Node2</i> bei ETSI (Standardvariante) während des Szenarios <i>Autobahnkreuz</i> . . . . .	61
4.4	SUMO-Gui-Screenshot des Szenarios <i>Stadt</i> . . . . .	63
4.5	Auswirkung der Änderung von <i>vPERRange</i> auf <i>Smooth Vehicle Density in Range</i> bei Szenario <i>Stau</i> . . . . .	71
4.6	Auswirkung der Änderung von <i>vPERRange</i> auf die Kanallast bei Szenario <i>Stau</i> . . . . .	71

4.7 Anzahl des <i>TxDecisionDynamics</i> -Flags mit Wert true bei dem Szenario <i>Stau</i> . . . . .	72
B.1 Generierungsintervall in Sekunden bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Autobahnkreuz</i> mit einer Simula- tionszeit von je 10 Sekunden . . . . .	85
B.2 Generierungsintervall in Sekunden bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Autobahnkreuz</i> mit einer Simula- tionszeit von je 10 Sekunden . . . . .	85
B.3 Kanallast (CBR) bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Autobahnkreuz</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	86
B.4 Kanallast (CBR) bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Autobahnkreuz</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	86
B.5 Gesendete Beacons bei 300 Bytes und sechsfacher paralleler Ausfüh- rung des Szenarios <i>Autobahnkreuz</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	86
B.6 Gesendete Beacons bei 600 Bytes und sechsfacher paralleler Ausfüh- rung des Szenarios <i>Autobahnkreuz</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	87
B.7 Empfangene Beacons bei 300 Bytes und sechsfacher paralleler Aus- führung des Szenarios <i>Autobahnkreuz</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	87
B.8 Empfangene Beacons bei 600 Bytes und sechsfacher paralleler Aus- führung des Szenarios <i>Autobahnkreuz</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	87
B.9 Paketverluste bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Autobahnkreuz</i> mit einer Simulationszeit von je 10 Sekunden	88
B.10 Paketverluste bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Autobahnkreuz</i> mit einer Simulationszeit von je 10 Sekunden	88
B.11 Generierungsintervall in Sekunden bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Autobahnkreuz</i> . . . . .	89
B.12 Kanallast (CBR) bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Autobahnkreuz</i> . . . . .	89
B.13 Gesendete Beacons bei 600 Bytes, 200 Fahrzeugen und Simulations- zeit 200 s bei Szenario <i>Autobahnkreuz</i> . . . . .	89
B.14 Empfangene Beacons bei 600 Bytes, 200 Fahrzeugen und Simula- tionszeit 200 s bei Szenario <i>Autobahnkreuz</i> . . . . .	90

B.15 Paketverluste bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Autobahnkreuz</i> . . . . .	90
B.16 Generierungsintervall in Sekunden bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stadt</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	91
B.17 Generierungsintervall in Sekunden bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stadt</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	91
B.18 Kanallast (CBR) bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stadt</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	92
B.19 Kanallast (CBR) bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stadt</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	92
B.20 Gesendete Beacons bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stadt</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	92
B.21 Gesendete Beacons bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Autobahnkreuz</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	93
B.22 Empfangene Beacons bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stadt</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	93
B.23 Empfangene Beacons bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stadt</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	93
B.24 Paketverluste bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stadt</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	94
B.25 Paketverluste bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stadt</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	94
B.26 Generierungsintervall in Sekunden bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Stadt</i> . . . . .	94
B.27 Kanallast (CBR) bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Stadt</i> . . . . .	95
B.28 Gesendete Beacons bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Stadt</i> . . . . .	95
B.29 Empfangene Beacons bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Stadt</i> . . . . .	95
B.30 Paketverluste bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Stadt</i> . . . . .	96
B.31 Generierungsintervall in Sekunden bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stau</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	96

B.32 Generierungsintervall in Sekunden bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stau</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	97
B.33 Kanallast (CBR) bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stau</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	97
B.34 Kanallast (CBR) bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stau</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	97
B.35 Gesendete Beacons bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stau</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	98
B.36 Gesendete Beacons bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stau</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	98
B.37 Empfangene Beacons bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stau</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	98
B.38 Empfangene Beacons bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stau</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	99
B.39 Paketverluste bei 300 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stau</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	99
B.40 Paketverluste bei 600 Bytes und sechsfacher paralleler Ausführung des Szenarios <i>Stau</i> mit einer Simulationszeit von je 10 Sekunden . . . . .	99
B.41 Generierungsintervall bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Stau</i> . . . . .	100
B.42 Kanallast (CBR) bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Stau</i> . . . . .	100
B.43 Gesendete Beacons bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Stau</i> . . . . .	100
B.44 Empfangene Beacons bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Stau</i> . . . . .	101
B.45 Paketverluste bei 600 Bytes, 200 Fahrzeugen und Simulationszeit 200 s bei Szenario <i>Stau</i> . . . . .	101

---

## **Listingverzeichnis**

---

3.1 Auszug aus omnetpp.ini . . . . .	37
3.2 Berechnung der <i>PER</i> . . . . .	41
3.3 Methode <code>setStateParams</code> . . . . .	46

---

# Tabellenverzeichnis

---

2.1	EDCA-Parameter bei WAVE . . . . .	8
2.2	Beispiele von WAVE Safety Applications . . . . .	10
2.3	EDCA-Parameter bei ETSI . . . . .	22
2.4	Vergleich von CAM und DENM . . . . .	24
2.5	Standardparameter der DCC . . . . .	29
2.6	Für CAMs relevante DCC Standardparameter . . . . .	29
2.7	Für CAMs relevante DCC Parameter nach DCC Profile 2 . . . . .	29
2.8	CBR Werte des Geonetworking Protokolls . . . . .	30
4.1	Allgemeiner Vergleich beider Mechanismen . . . . .	53
4.2	Simulationsparameter . . . . .	55
4.3	Individuelle Eigenschaften von Szenario <i>Autobahnkreuz</i> . . . . .	57
4.4	Vergleich der Kennwerte des Szenarios <i>Autobahnkreuz</i> anhand der beobachteten Median-Werte . . . . .	60
4.5	Individuelle Eigenschaften von Szenario <i>Stadt</i> . . . . .	63
4.6	Vergleich der Kennwerte des Szenarios <i>Stadt</i> mittels der beobachteten Median-Werte . . . . .	65
4.7	Individuelle Eigenschaften von Szenario <i>Stau</i> . . . . .	66
4.8	Vergleich der Kennwerte des Szenarios <i>Stau</i> mittels der beobachteten Median-Werte . . . . .	69
4.9	Gegenüberstellung der ETSI-Varianten anhand des Szenarios <i>Autobahnkreuz</i> bei 6 parallelen Ausführungen für je 10 s . . . . .	75
4.10	Gegenüberstellung der ETSI-Varianten anhand des Szenarios <i>Stadt</i> bei 6 parallelen Ausführungen für je 10 s . . . . .	76
4.11	Gegenüberstellung der ETSI-Varianten anhand des Szenarios <i>Stau</i> bei 6 parallelen Ausführungen für je 10 s . . . . .	77

---

## Literaturverzeichnis

---

- [1] Tagesschau, “Tödlicher unfall mit selbstfahrendem tesla,” 2016-07-01. [Online]. Available: <https://www.tagesschau.de/ausland/tesla-unfall-101.html>
- [2] www.tagesschau.de, “Technik war nicht schuld an tesla-unfall,” 2017-01-20. [Online]. Available: <https://www.tagesschau.de/wirtschaft/tesla-unfall-109.html>
- [3] H. Winner, S. Hakuli, F. Lotz, and C. Singer, *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*, 3rd ed., ser. ATZ/MTZ-Fachbuch. Wiesbaden: Springer Vieweg, 2015.
- [4] “Ieee 802.11p-2010: Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Wireless access in vehicular environments.” [Online]. Available: <http://standards.ieee.org/findstds/standard/802.11p-2010.html>
- [5] C. Sommer, R. German, and F. Dressler, “Bidirectionally coupled network and road traffic simulation for improved ivc analysis,” *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, 2011.
- [6] C. Sommer, “Veins - the open source vehicular network simulation framework.” [Online]. Available: <http://veins.car2x.org/>
- [7] “What is omnet++?” [Online]. Available: <https://omnetpp.org/intro>
- [8] “Sumo - simulation of urban mobility.” [Online]. Available: [http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931\\_read-41000/](http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/)
- [9] D. Eckhoff, N. Sofra, and R. German, “A performance study of cooperative awareness in etsi its g5 and ieee wave,” in *10th IEEE/IFIP Conference on Wireless On demand Network Systems and Services (WONS 2013)*. Banff, Canada: IEEE, 2013, pp. 196–200.

- [10] “Etsi ts 102 687 v1.1.1 (2011-07): Intelligent transport systems (its); decentralized congestion control mechanisms for intelligent transport systems operating in the 5 ghz range; access layer part.” [Online]. Available: [http://www.etsi.org/deliver/etsi\\_ts/102600\\_102699/102687/01.01.60/ts\\_102687v010101p.pdf](http://www.etsi.org/deliver/etsi_ts/102600_102699/102687/01.01.60/ts_102687v010101p.pdf)
- [11] F. Marzouk, R. Zagrouba, A. Laouiti, and P. Muhlethaler, “An empirical study of unfairness and oscillation in etsi dcc,” in *International Conference on Performance Evaluation and Modeling in Wired and Wireless networks PEMWN’ 2015*, Hammamet, Tunisia, 2015. [Online]. Available: <https://hal.archives-ouvertes.fr/hal-01211466>
- [12] G. Bansal, B. Cheng, A. Rostami, K. Sjoberg, J. B. Kenney, and M. Gruteser, “Comparing limeric and dcc approaches for vanet channel congestion control,” in *2014 IEEE 6th International Symposium on Wireless Vehicular Communications (WiVeC 2014)*, 2014, pp. 1–7.
- [13] T. Lorenzen, “Performance analysis of the functional interaction of awareness control and dcc in vanets,” in *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, 2016, pp. 1–7.
- [14] A. Autolitano, C. Campolo, A. Molinaro, R. M. Scopigno, and A. Vesco, “An insight into decentralized congestion control techniques for vanets from etsi ts 102 687 v1.1.1,” in *2013 IFIP Wireless Days (WD)*, 2013, pp. 1–6.
- [15] S. Yang and H. Kim, “Configuring etsi dcc parameters for better performance and stability,” *Electronics Letters*, vol. 52, no. 8, pp. 667–669, 2016.
- [16] J. B. Kenney, “Dedicated short-range communications (dsrc) standards in the united states,” *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1162–1182, 2011.
- [17] “Sae j2945/1\_201603: On-board system requirements for v2v safety communications,” 2016-03-30. [Online]. Available: [http://standards.sae.org/j2945/1\\_201603/](http://standards.sae.org/j2945/1_201603/)
- [18] “Ieee 1609.0-2013: Ieee guide for wireless access in vehicular environments (wave) - architecture.” [Online]. Available: <https://standards.ieee.org/findstds/standard/1609.0-2013.html>
- [19] “Ieee 1609.1-2006: Trial-use standard for wireless access in vehicular environments (wave) - resource manager.” [Online]. Available: <http://standards.ieee.org/findstds/standard/1609.1-2006.html>
- [20] “Ieee 1609.2-2016: Ieee standard for wireless access in vehicular environments–security services for applications and management messages.” [Online]. Available: <https://standards.ieee.org/findstds/standard/1609.2-2016.html>

- [21] “Ieee 1609.3-2016: Ieee standard for wireless access in vehicular environments (wave) – networking services.” [Online]. Available: <http://standards.ieee.org/findstds/standard/1609.3-2016.html>
- [22] “Ieee 1609.4-2016: Ieee standard for wireless access in vehicular environments (wave) – multi-channel operation.” [Online]. Available: <https://standards.ieee.org/findstds/standard/1609.4-2016.html>
- [23] “Ieee 1609.11-1020: Ieee standard for wireless access in vehicular environments (wave)- over-the-air electronic payment data exchange protocol for intelligent transportation systems (its).” [Online]. Available: <http://standards.ieee.org/findstds/standard/1609.11-2010.html>
- [24] U. N. E. P. A. J. d. S. Roberto A. Uzcátegui and G. I. o. T. Guillermo Acosta-Marum, “Wave: a tutorial,” Mai 2009. [Online]. Available: <http://www.slideshare.net/gamarun/wave-a-tutorial>
- [25] “Fcc 47 cfr.” [Online]. Available: <https://www.fcc.gov/general/rules-regulations-title-47>
- [26] X. Zhang and D. Qiao, *An Overview of the DSRC/WAVE Technology: 7th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, QShine 2010, and Dedicated Short Range Communications Workshop, DSRC 2010, Houston, TX, USA, November 17-19, 2010, Revised Selected Papers*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 74.
- [27] J. Hui and M. Devetsikiotis, “A unified model for the performance analysis of ieee 802.11e edca,” *IEEE Transactions on Communications*, vol. 53, no. 9, pp. 1498–1510, 2005.
- [28] “Etsi en 302 663 v1.2.1 (2013-07): Intelligent transport systems (its); access layer specification for intelligent transport systems operating in the 5 ghz frequency band.” [Online]. Available: [http://www.etsi.org/deliver/etsi\\_en/302600\\_302699/302663/01.02.01\\_60/en\\_302663v010201p.pdf](http://www.etsi.org/deliver/etsi_en/302600_302699/302663/01.02.01_60/en_302663v010201p.pdf)
- [29] “Sae j2735\_201603: Dedicated short range communications (dsrc) message set dictionary.” [Online]. Available: [http://standards.sae.org/j2735\\_201603/](http://standards.sae.org/j2735_201603/)
- [30] “Vehicle safety communications – applications (vsc-a), final report, dot hs 811 492a, september 2011.” [Online]. Available: <https://www.nhtsa.gov/DOT/NHTSA/NVS/Crash%20Avoidance/Technical%20Publications/2011/811492A.pdf>

- [31] "Vehicle safety communications – applications (vsc-a) final report: Appendix volume 2 communications and positioning, dot hs 811 492c september 2011." [Online]. Available: <https://one.nhtsa.gov/DOT/NHTSA/NVS/Crash%20Avoidance/Technical%20Publications/2011/811492C.pdf>
- [32] "Etsi en 302 571 v2.1.1 (2017-02): Intelligent transport systems (its); radio-communications equipment operating in the 5 855 mhz to 5 925 mhz frequency band; harmonised standard covering the essential requirements of article 3.2 of directive 2014/53/eu." [Online]. Available: [http://www.etsi.org/deliver/etsi\\_en/302500\\_302599/302571/02.01.01\\_60/en\\_302571v020101p.pdf](http://www.etsi.org/deliver/etsi_en/302500_302599/302571/02.01.01_60/en_302571v020101p.pdf)
- [33] "Etsi es 202 663 v1.1.0 (2010-01): Intelligent transport systems (its); european profile standard for the physical and medium access control layer of intelligent transport systems operating in the 5 ghz frequency band." [Online]. Available: [http://www.etsi.org/deliver/etsi\\_es/202600\\_202699/202663/01.01.00\\_60/es\\_202663v010100p.pdf](http://www.etsi.org/deliver/etsi_es/202600_202699/202663/01.01.00_60/es_202663v010100p.pdf)
- [34] "Etsi eg 202 798 v1.1.1 (2011-01): Intelligent transport systems (its); testing; framework for conformance and interoperability testing." [Online]. Available: [http://www.etsi.org/deliver/etsi\\_eg/202700\\_202799/202798/01.01.01\\_60/eg\\_202798v010101p.pdf](http://www.etsi.org/deliver/etsi_eg/202700_202799/202798/01.01.01_60/eg_202798v010101p.pdf)
- [35] "Etsi tr 101 607 v1.1.1 (2013-05): Intelligent transport systems (its); cooperative its (c-its); release 1." [Online]. Available: [http://www.etsi.org/deliver/etsi\\_tr/101600\\_101699/101607/01.01.01\\_60/tr\\_101607v010101p.pdf](http://www.etsi.org/deliver/etsi_tr/101600_101699/101607/01.01.01_60/tr_101607v010101p.pdf)
- [36] "Etsi ts 102 636-4-2 v1.1.1 (2013-10): Intelligent transport systems (its); vehicular communications; geonetworking; part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; sub-part 2: Media-dependent functionalities for its-g5." [Online]. Available: [http://www.etsi.org/deliver/etsi\\_ts/102600\\_102699/1026360402/01.01.01\\_60/ts\\_1026360402v010101p.pdf](http://www.etsi.org/deliver/etsi_ts/102600_102699/1026360402/01.01.01_60/ts_1026360402v010101p.pdf)
- [37] "Etsi tr 102 863 v1.1.1 (2011-06): Intelligent transport systems (its); vehicular communications; basic set of applications; local dynamic map (ldm); rationale for and guidance on standardization." [Online]. Available: [http://www.etsi.org/deliver/etsi\\_tr/102800\\_102899/102863/01.01.01\\_60/tr\\_102863v010101p.pdf](http://www.etsi.org/deliver/etsi_tr/102800_102899/102863/01.01.01_60/tr_102863v010101p.pdf)
- [38] "Etsi en 302 637-2 v1.3.2: Intelligent transport systems (its); vehicular communications; basic set of applications; part 2: Specification of cooperative awareness basic service." [Online]. Available: <http://www.etsi.org/standards-search#search=ETSIEN302637-2V1.3.2>

- [39] “Etsi ts 102 894-2 v1.2.1 (2014-09): Intelligent transport systems (its); users and applications requirements; part 2: Applications and facilities layer common data dictionary.” [Online]. Available: [http://www.etsi.org/deliver/etsi\\_ts/102800\\_102899/10289402/01.02.01\\_60/ts\\_10289402v010201p.pdf](http://www.etsi.org/deliver/etsi_ts/102800_102899/10289402/01.02.01_60/ts_10289402v010201p.pdf)
- [40] “Etsi ts 102 637-3 v1.1.1 (2010-09): Intelligent transport systems (its); vehicular communications; basic set of applications; part 3: Specifications of decentralized environmental notification basic service.” [Online]. Available: [http://www.etsi.org/deliver/etsi\\_ts/102600\\_102699/10263703/01.01.01\\_60/ts\\_10263703v010101p.pdf](http://www.etsi.org/deliver/etsi_ts/102600_102699/10263703/01.01.01_60/ts_10263703v010101p.pdf)
- [41] “Etsi ts 102 724: Intelligent transport systems (its); harmonized channel specifications for intelligent transport systems operating in the 5 ghz frequency band.” [Online]. Available: [http://www.etsi.org/deliver/etsi\\_ts/102700\\_102799/102724/01.01.01\\_60/ts\\_102724v010101p.pdf](http://www.etsi.org/deliver/etsi_ts/102700_102799/102724/01.01.01_60/ts_102724v010101p.pdf)
- [42] “Etsi tr 101 612 v1.1.1 (2014-09): Intelligent transport systems (its); cross layer dcc management entity for operation in the its g5a and its g5b medium; report on cross layer dcc algorithms and performance evaluation.” [Online]. Available: [http://www.etsi.org/deliver/etsi\\_tr/101600\\_101699/101612/01.01.01\\_60/tr\\_101612v010101p.pdf](http://www.etsi.org/deliver/etsi_tr/101600_101699/101612/01.01.01_60/tr_101612v010101p.pdf)
- [43] “Etsi draft ts 102 687 v1.1.2 (2014-01): Intelligent transport systems (its); decentralized congestion control mechanisms for intelligent transport systems operating in the 5 ghz range; access layer part.”
- [44] J. B. Kenney, G. Bansal, and C. E. Rohrs, “Limeric: A linear message rate control algorithm for vehicular dsrc systems,” in *Proceedings of the Eighth ACM International Workshop on Vehicular Inter-networking*, ser. VANET ’11. New York, NY, USA: ACM, 2011, pp. 21–30. [Online]. Available: <http://doi.acm.org/10.1145/2030698.2030702>
- [45] “Etsi en 302 636-3 v1.2.1 (2014-12): Intelligent transport systems (its); vehicular communications; geonetworking; part 3: Network architecture.” [Online]. Available: [http://www.etsi.org/deliver/etsi\\_en/302600\\_302699/30263603/01.02.01\\_60/en\\_30263603v010201p.pdf](http://www.etsi.org/deliver/etsi_en/302600_302699/30263603/01.02.01_60/en_30263603v010201p.pdf)
- [46] “Etsi tr 101 613 v1.1.1 (2015-09): Intelligent transport systems (its); cross layer dcc management entity for operation in the its g5a and its g5b medium; validation set-up and results.” [Online]. Available: [http://www.etsi.org/deliver/etsi\\_tr/101600\\_101699/101613/01.01.01\\_60/tr\\_101613v010101p.pdf](http://www.etsi.org/deliver/etsi_tr/101600_101699/101613/01.01.01_60/tr_101613v010101p.pdf)
- [47] C. Sommer and F. Dressler, *Vehicular Networking*. Cambridge: Cambridge University Press, 2015.

- [48] R. Riebl, H.-J. Günther, C. Facchi, and L. Wolf, “Artery - extending veins for vanet applications,” in *4th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS 2015)*. Budapest, Hungary: IEEE, 2015.