Aus dem Institut für Telematik
der Universität zu Lübeck

Direktor:
Prof. Dr. rer. nat. Stefan Fischer

Organic-Computing-Konzepte und deren Umsetzung für dezentrale Anwendungen im Straßenverkehr

Inauguraldissertation
zur
Erlangung der Doktorwürde
der Universität zu Lübeck
– Aus der Technisch-Naturwissenschaftlichen Fakultät –

Vorgelegt von
Herrn Dipl.-Inform. Axel Wegener
aus Holzminden

Lübeck, 2009

Erster Berichterstatter:	Prof. Dr. rer. nat. Stefan Fischer
Zweiter Berichterstatter:	Prof. Dr. Sándor P. Fekete
Tag der mündlichen Prüfung:	4. September 2009

Zum Druck genehmigt.
Lübeck, den 9. September 2009

Kurzfassung

Individuelle Mobilität ist für eine moderne Gesellschaft eine wichtige Grundlage, sowohl unter wirtschaftlichen als auch unter gesellschaftlichen Aspekten. Allerdings häufen sich mit steigender Auslastung des Straßennetzes Behinderungen und Staus. Daher werden Verkehrsinformationssysteme eingesetzt, um Verkehrsteilnehmer zu warnen und auf alternative Strecken umzuleiten.

Unerwünschte Verkehrsstrukturen wie z. B. Staus werden jedoch erst erfasst, nachdem der Stau eine gewisse Größe erreicht hat und somit schon eine unmittelbare Behinderung darstellt. Dabei taucht ein Stau jedoch nicht unvermittelt auf, sondern entsteht als emergente Eigenschaft basierend auf den Interaktionen zwischen Fahrzeugen auf mikroskopischer Ebene.

Aufgrund der zentralisierten Architektur heutiger Verkehrsinformationssysteme ist eine derart feingranulare Erfassung und Verarbeitung der Verkehrslage nur schwer möglich. Zudem beschränkt sich die Erfassung der Verkehrslage – bedingt durch die benötigte teure Infrastruktur – fast ausschließlich auf Autobahnen. Ein dezentraler Ansatz erlaubt hingegen eine sehr detaillierte Erfassung der Verkehrslage; dabei kommunizieren die beteiligten Fahrzeuge über ein sog. *Vehicular Ad-hoc Network* (VANET) direkt miteinander. Durch die dadurch mögliche Datenverarbeitung „vor Ort“ kann ein sich anbahnender Stau schon in der Entstehungsphase erkannt und seiner Ausbreitung durch lokale Beeinflussung entgegengewirkt werden.

In dieser Arbeit wird ein solcher dezentraler Ansatz entwickelt, der basierend auf Organic Computing-Prinzipien die Entwicklung dezentraler Anwendungen im Straßenverkehr erlaubt. Grundlegend neu in diesem Ansatz ist das Konzept der Datenwolke (engl. *Hovering Data Cloud*, HDC). Eine HDC dient als dynamische Datenstruktur, die ein lokales Verkehrsphänomen erfasst und verfolgt. Dazu nutzt eine HDC Fahrzeuge als Informationsträger, kann sich aber unabhängig von diesen „bewegen“, so dass sie sich immer im Bereich des von ihr beobachteten Verkehrsphänomens aufhalten kann. Durch hierarchische Aggregation so gewonnener Daten lassen sich räumlich ausgedehnte Verkehrsstrukturen erfassen, die als *Organic Information Complexes* (OICs) bezeichnet werden und deren Zustand wiederum in HDCs abgelegt wird. Zudem kann ausgehend von den lokal verfügbaren HDCs eine Adaption des jeweiligen Fahrverhaltens erfolgen; so kann beispielsweise im Falle eines Staus nachfolgender Verkehr umgeleitet, oder die Fahrgeschwindigkeit innerhalb des Staus harmonisiert werden.

Sowohl zur Kommunikation innerhalb einer HDC als auch zur Kommunikation der

HDCs untereinander und zum Verbreiten von Verkehrsinformationen wurde im Rahmen dieser Arbeit das Protokoll *AutoCast* zur effizienten Verteilung von Daten entwickelt. *AutoCast* ist ein hybrides Protokoll, das sein Verhalten je nach lokaler Netztopologie selbständig anpasst. Die einzelnen Protokoll-Bausteine werden ausführlich beschrieben und evaluiert.

Darauf aufbauend setzt das *AutoNomos*-System die Konzepte der HDC und OIC um und ermöglicht die Implementierung von VANET-Anwendungen auf Basis definierter Schnittstellen.

Zur prototypischen Umsetzung und Evaluation wurde eine Simulationsumgebung entwickelt, die einen Verkehrssimulator mit einem Netzwerksimulator bidirektional koppelt. Über die bidirektionale Kopplung werden nicht nur die Bewegungsdaten der Fahrzeuge vom Verkehrssimulator an den Netzwerksimulator geliefert, sondern es kann auch die im Netzwerksimulator implementierte VANET-Anwendung kann auf diverse Aspekte der Verkehrssimulation zugreifen und das Verhalten der Fahrzeuge während der Simulation beeinflussen. Somit können auch die Auswirkungen einer VANET-Anwendung auf den Straßenverkehr analysiert werden.

Neben der Simulationsumgebung dient ein Demonstrator, bestehend aus 18 Lego-Mindstorms-Fahrzeugen, dazu, die betrachteten Konzepte aus der Simulation in ein „reales“ Verkehrsszenario zu überführen und veranschaulicht die Vorteile der vorgestellten Konzepte.

Auf Basis der Simulationsumgebung und des Demonstrators werden drei VANET-Anwendungen exemplarisch vorgestellt, die nach den in dieser Arbeit entwickelten Prinzipien umgesetzt wurden und die Machbarkeit sowie die positive Auswirkung auf den Verkehrsfluss belegen.

Inhaltsverzeichnis

Danksagung

An dieser Stelle möchte ich mich bei all denen bedanken, die mir in den letzten Jahren, in denen diese Arbeit entstanden ist, zur Seite standen.

Zuallererst danke ich meiner lieben Frau Stefanie, die mir stets Halt gegeben hat und mich mit voller Kraft untertützt hat. Unserem Sohn Jannes danke ich für die großen Augen, mit denen er seit zwei Jahren die Welt entdeckt und mir quasi nebenher den nötigen Ausgleich zur Arbeit beschert.

Meinen Eltern danke ich von ganzem Herzen für die lebenslange Unterstützung in allen Belangen und ihr Interesse an meiner Arbeit.

Ein großer Dank gebührt meinem Doktorvater Herrn Prof. Stefan Fischer für die großzügigen Forschungsbedingungen die er mir im Rahmen des Forschungsprojekts AutoNomos eingeräumt hat und die Betreuung meiner Arbeit in den letzten vier Jahren. Ein herzliches Dankeschön gilt auch allen Beteiligten dieses Forschungsprojekts: Herrn Prof. Sándor P. Fekete, der auch das Koreferat übernommen hat, Prof. Horst Hellbrück, Dr. Elad Schiller, Christiane Schmidt, Christopher Tessars und Björn Hendriks.

Für die Motivation in Form einer (verlorenen) Wette danke ich Prof. Hellbrück; ebenso für die zahlreichen Diskussionen und die gemeinsame Arbeit am AutoCast-Protokoll.

Für die tägliche nette Arbeitsatmosphäre, zahlreiche fachliche Diskussionen und das freundschaftliche Miteinander danke ich allen Kollegen am Institut für Telematik gleichermaßen: Claudia Becker, Dr. Carsten Buschmann, Dr. Alexander Carôt, Maick Danckwardt, Nils Glombitza, Daniela Krüger, Martin Lipphardt, Dr. Dennis Pfisterer, Stephan Pöhlsen, Stefan Ransom, Peter Rothenpieler, Dirk Schmidt, Birgit Schneider und Prof. Christian Werner. Es hat Spaß gemacht, mit Euch zu arbeiten!

Kapitel 1

Einleitung

Verkehrsstaus sind sowohl für individuelle Verkehrsteilnehmer als auch für die Volkswirtschaft ein nicht zu unterschätzender Kostenfaktor. Eine aktuelle Studie [167] veranschlagt den volkswirtschaftlichen Schaden in Deutschland auf jährlich 102 Mrd €. Bei einer ständig steigenden Anzahl von zugelassenen Fahrzeugen kann eine Verringerung dieses Schadens nur durch Schaffung neuer Straßen oder eine Erhöhung der Kapazität von bestehenden Straßen erreicht werden. Eine Erweiterung der Verkehrskapazität durch Neu- und Ausbau von Straßen ist allerdings nur begrenzt möglich und sinnvoll. Intelligente Systeme zur Verkehrslenkung versprechen andererseits die Kapazität bestehender Straßen besser auszunutzen.

Mit Hinblick auf intelligente Verkehrslenkung wurden in dieser Arbeit Grundlagen für ein Verkehrsinformationssystem entwickelt, das im Gegensatz zu bestehenden zentralisierten Lösungen vollkommen dezentral arbeitet.

1.1 Motivation

Vor einer genaueren Betrachtung des dezentralen Ansatzes sei kurz die Funktionsweise eines „traditionellen“ zentralisierten Verkehrsinformationssystems erläutert. Der Prozess der Datenverarbeitung lässt sich in folgende drei Schritte unterteilen:

Erfassung Zunächst muss die aktuelle Verkehrslage erfasst werden. Dazu wird entweder eine fest installierte Infrastruktur eingesetzt (z. B. Induktionsschleifen oder Kameras), die den Verkehr von „außen“ beobachtet, oder die Fahrzeuge selbst geben ihre Positionsdaten für eine weitere Verarbeitung preis. Aufgrund des hohen Installations- und Wartungsaufwands ist die Verbreitung fest installierter Messsysteme zumeist auf das Autobahnnetz beschränkt. Staus auf Umgehungsstraßen und im innerstädtischen Bereich werden daher oft nicht erfasst. Alternativ lassen sich die Fahrzeuge selbst zu mobilen Sensoren aufrüsten, die ihre Positionsdaten an eine zentrale Instanz senden. Diese Technik wird im englischen mit *Floating Car Data* (FCD) bezeichnet (vgl. [114]). Bereits vorhandene Daten, wie die Funkzellen

von Mobiltelefonen, die in den Fahrzeugen mitgeführt werden, lassen sich ebenfalls für ein Verkehrsinformationssystem nutzen (vgl. HD Traffic[1]).

Verarbeitung Die erfassten Verkehrsdaten müssen im Anschluss zu einer Verkehrslage zusammengefügt werden. Dies geschieht in der Regel in regionalen Verkehrsmanagementzentralen. Bei ungünstiger Verkehrslage müssen Beeinflussungsmaßnahmen generiert werden, die sich positiv auf die Verkehrslage auswirken sollen. Oft wird die Verkehrslage manuell bewertet und es werden bei Bedarf Verkehrsmeldungen erzeugt. Projekte wie *Autobahn.NRW* [112, 158] zielen auf eine Automatisierung der Stauerkennung und -prognose mithilfe eines zentralisierten Systems hin.

Verbreitung Zuletzt müssen die nötigen Beeinflussungsmaßnahmen den Verkehrsteilnehmern bekannt gemacht werden. Entsprechende Verkehrsmeldungen werden regelmäßig über Radiosender verbreitet. Diese werden sowohl vom Moderator vorgelesen als auch in digitaler Form über den *Traffic Message Channel* (TMC) gesendet, so dass sie direkt vom Navigationsgerät berücksichtigt werden können. Bedingt durch die geringe Datenrate von 60 bit/s und das vergleichsweise große Sendegebiet beschränken sich TMC-Verkehrsmeldungen auf das Autobahnnetz und beziehen sich auf ganze Autobahn-Streckenabschnitte, in der Regel den Bereich zwischen zwei Autobahnanschlussstellen. Zudem führen Irrtümer und Übermittlungsfehler in der langen Verarbeitungskette zu verspäteten und daher teilweise falschen Meldungen. Um die geringe Datenrate von TMC zu umgehen werden individualisierte Verkehrsmeldungen vermehrt über zelluläre Mobilfunknetze (GSM oder UMTS) übertragen.

Auch in einem dezentralen Verkehrsinformationssystem sind die drei vorgenannten Datenverarbeitungsschritte enthalten; abgewichen wird jedoch von der sequenziellen und zentralisierten Datenverarbeitung. Vielmehr wird ein Ad-hoc-Netz verwendet, über das Fahrzeuge, die sich in Funkreichweite zueinander befinden, direkt miteinander kommunizieren können. Fahrzeuge in Funkreichweite werden als *benachbart* bezeichnet. Statt der deutschen Übersetzung *Fahrzeug-Ad-hoc-Netz* wird im Rest dieser Arbeit der gebräuchlichere englische Begriff *Vehicular Ad-hoc Network* (VANET) verwendet. Darauf aufbauende Anwendungen werden als *VANET-Anwendungen* bezeichnet.

Systembedingt ist in einem VANET eine direkte Interaktion nur zwischen benachbarten Fahrzeugen möglich. Eine zentrale Instanz, die z. B. die gesamte Verarbeitung der erfassten Verkehrsdaten vornimmt, ist nicht vorgesehen. Stattdessen werden Informationen von Fahrzeug zu Fahrzeug weitergeleitet und so verbreitet. Dieses Vorgehen erlaubt eine sehr detaillierte Datenerfassung und -verarbeitung am jeweiligen Ursprung

[1] `http://www.tomtom.com/hdtraffic/`

der Daten. Die aus den erfassten Daten abgeleiteten Verkehrsinformationen werden immer weiter aggregiert, während sie sich vom Ort ihrer Messung entfernen.

Darüber hinaus weisen dezentrale VANET-Anwendungen gegenüber zentralisierten Anwendungen die nachfolgend beschriebenen Vorteile auf.

VANET-Anwendungen sind nicht vom Vorhandensein einer festen Infrastruktur abhängig, die aus Kostengründen zumeist nur auf Autobahnen und auf ausgewählten Straßen im innerstädtischen Bereich installiert ist; der Funktionsbereich erstreckt sich vielmehr auf alle Orte, an denen sich entsprechend ausgestattete Fahrzeuge befinden. Tatsächlich stellt der als *Ausstattungsgrad* bezeichnete Anteil an Fahrzeugen, die am VANET partizipieren, eine kritische Größe dar.

In einem VANET können die benötigten Messwerte, wie Verkehrsdichte oder mittlere Geschwindigkeit, direkt von den Fahrzeugen ermittelt und kombiniert werden, ohne auf externe Sensoren angewiesen zu sein, die z. B. die Verkehrsdichte nur jeweils an ihrer Messstelle erfassen. Ein Stau wird daher von fest installierten Messstellen solange nicht erkannt, wie er sich nur zwischen zwei Messstellen befindet. In einem VANET hingegen, kann bereits das erste stehende Fahrzeug erkannt werden. Beeinflussungsmaßnahmen können somit eher und mit kleinerer Auswirkung auf den gesamten Verkehrsfluss durchgeführt werden.

In zentralisierten Systemen führt zudem die benötigte Zeit zum Durchlaufen der gesamten Datenverarbeitungskette häufig zu veralteten Verkehrsmeldungen. Auch die Detailtiefe der Verkehrsmeldungen wird durch die beschränkte Bandbreite des am häufigsten als Rückkanal verwendeten TMC zu den Verkehrsteilnehmern eingeschränkt. „Bessere“ Verkehrsmeldungen sind zumeist an Bezahldienste geknüpft [145]. In VANET-Anwendungen hängen Detailtiefe und Aktualität der Meldungen eher von der eigenen Entfernung zum jeweiligen Datenursprung ab.

Schließlich ist die Umsetzung von Strategien zur Verbesserung des gesamten Verkehrsflusses in zentralisierten Systemen nur sehr eingeschränkt möglich, da eine verkehrsbeeinflussende Maßnahme, wie eine dynamische Geschwindigkeitsbeschränkung, zu weiträumigen Einschränkungen führt, und zudem die Reaktion der Verkehrsteilnehmer auf eine Verkehrsmeldung nur mit deutlicher Verzögerung erfasst wird. In der Realität führen Staumeldungen häufig zu einem Übergreifen des Staus auf die Umgehungsstraßen.

Vor einigen Jahren entstand unter dem Oberbegriff *Organic Computing* eine neue Herangehensweise an immer komplexer werdende Computersysteme [117, 155, 156, 179]. Dabei werden von der Natur inspirierte Verfahren entwickelt, und auf Computersysteme angewendet.

Auch beim Straßenverkehr handelt es sich um ein sehr komplexes System, das aus einer Vielzahl mobiler Einheiten (d. h. Fahrzeugen) besteht, die keiner zentralen Kontrolle unterliegen. Vielmehr strebt jeder Verkehrsteilnehmer sein persönliches Fahrtziel an und handelt auf dem Weg dorthin vollkommen autonom und auf den eigenen Vorteil bedacht. Zum Erreichen des Fahrtziels müssen sich allerdings alle Verkehrsteilnehmer

auf einem gemeinsamen Straßennetz bewegen, was den persönlichen Interessen, z. B. bei stockendem Verkehr, Grenzen setzt.

Anstatt dieses komplexe System durch fest installierte Sensoren von außen zu beobachten, bietet sich eine Koordination der einzelnen Fahrzeuge durch direkte Interaktion an. Im Tierreich kann man ein solches Schwarmverhalten bei Fischen, Vögeln, Insekten und anderen Tieren beobachten. Es existieren bereits etliche Forschungsarbeiten, die das Verhalten, beispielsweise von Ameisenkolonien [136] oder Vogelschwärmen [124], untersuchen.

Auf den Straßenverkehr lässt sich die Grundregel übertragen, dass auf Basis lokal wirkender Regeln größere Strukturen entstehen, die das Gesamtsystem charakterisieren. Dieser Vorgang wird *Emergenz* genannt (vgl. [34, 116]). In Bezug auf den Straßenverkehr gilt es, die richtigen lokalen Regeln zu finden, die zur Entstehung oder zum Ausbleiben von Verkehrsstrukturen führen. Dann kann auf lokaler Ebene eingegriffen werden, um die emergente Entstehung bestimmter Verkehrsstrukturen zu fördern bzw. zu verhindern.

Organic-Computing-Konzepte im Straßenverkehr anzuwenden ist somit besonders vielversprechend, weil sich der Straßenverkehr als Ganzes schon wie ein „organisches“ System verhält.

Die Deutsche Forschungsgemeinschaft (DFG) fördert im Rahmen des Schwerpunktprogramms 1183 [5] die Erforschung des Organic Computing. Im Rahmen dieser Förderung ist der Autor in das Projekt „AutoNomos: A Distributed and Self-Regulating Approach for Organizing a Large System of Mobile Objects“ eingebunden, in dem die oben angesprochenen Fragestellungen behandelt werden [40]. Die in diesem Projekt erarbeiteten Konzepte und Techniken werden im *AutoNomos*-System zusammengefasst und bilden eine Grundlage für die Entwicklung von VANET-Anwendungen.

1.2 Wissenschaftliche Beiträge und Aufbau dieser Arbeit

In dieser Arbeit wurde eine neuartige Herangehensweise für die Entwicklung dezentraler VANET-Anwendungen unter Verwendung von Organic-Computing-Prinzipien erforscht. Das umfasst im Einzelnen:

- Die Entwicklung einer Datenstruktur (*Hovering Data Cloud*, HDC), die Daten einem Ort bzw. einem beobachteten Phänomen zuordnet und deren Inhalt kontinuierlich einen Konsens anstrebt.
- Das *AutoNomos*-System, das auf hierarchischer Aggregation von HDCs basiert und als emergente Eigenschaft beispielsweise Verkehrsstrukturen erkennt und beobachet.
- Die Entwicklung und umfassende Evaluation eines Protokolls zur effizienten Verteilung von Daten (*AutoCast*).

Abbildung 1.1: Eine *Hovering Data Cloud* (HDC) am Stauende warnt ankommende Fahrzeuge.

- Eine Simulationsumgebung, die einen Netzwerksimulator mit einem Verkehrssimulator kombiniert und auch die Auswirkung einer VANET-Anwendung auf den Verkehr erfassen kann.
- Die beispielhafte Umsetzung von VANET-Anwendungen unter Verwendung der erarbeiteten Konzepte und Techniken.

Das folgende Kapitel 2 führt das Anwendungsgebiet und die damit verbundenen Grundlagen ein, auf denen diese Arbeit aufbaut. Abschnitt 2.1 erläutert Grundlagen des Straßenverkehrs sowie Methoden zu seiner Modellierung und Simulation. Abschnitt 2.2 enthält Grundlagen zu Ad-hoc-Netzen und erläutert Modelle zu deren Simulation, insbesondere werden auch die Besonderheiten eines Ad-hoc-Netzes zwischen Fahrzeugen eingeführt. Abschnitt 2.3 stellt dezentrale Anwendungen im Straßenverkehr als Problemfeld vor, das in dieser Arbeit behandelt wird, und gibt eine Übersicht über verwandte Arbeiten aus diesem Umfeld.

Kapitel 3 enthält eine Einführung zum Thema Organic Computing. In Abschnitt 3.1 wird der Straßenverkehr auf bereits vorhandene Organic-Computing-Aspekte hin betrachtet. Nachfolgend werden in Abschnitt 3.2 die Basiskonzepte vorgestellt, die zusammengenommen das *AutoNomos*-System bilden.

In Abschnitt 3.2.1 wird die „schwebende Datenwolke“ (engl. Hovering Data Cloud, HDC) als Datenstruktur vorgestellt, die berücksichtigt, dass Verkehrsphänomene – wie z. B. ein Verkehrsstau – keine statische Größe darstellen, sondern sich dynamisch verändern. Daher ist eine HDC in der Lage dynamisch einem beobachteten Phänomen zu folgen und ihren Dateninhalt auf Basis der beteiligten Fahrzeuge anzupassen. So kann eine HDC z. B. Aufschluss über die zeitliche Entwicklung eines Stauendes geben.

Wie in Abbildung 1.1 dargestellt, verteilen sich die in einer HDC gespeicherten Daten redundant auf alle Fahrzeuge in ihrem Bereich. Jedes dieser Fahrzeuge kann neue Daten in die HDC integrieren und somit sowohl den Datenstand als auch die Position der HDC beeinflussen. Ausgehend von einer HDC werden Verkehrsmeldungen generiert, die an alle Fahrzeuge innerhalb eines bestimmten Verbreitungsgebiets verteilt werden.

In Abschnitt 3.2.2 wird das Konzept des *Organic Information Complexes* (OIC) vorgestellt. Durch hierarchische Aggregation der von den HDCs erfassten Daten wird ein

Verkehrsstau als *Organic Information Complex* (OIC) in seiner Gesamtheit erfasst und als dynamische Verkehrsstruktur wahrgenommen.

Kapitel 4 beschäftigt sich mit der technischen Umsetzung dieser Konzepte. Dazu liefert Abschnitt 4.1 zunächst eine Übersicht über die Funktionalität bestehender Middleware-Lösungen aus dem Umfeld der Ad-hoc-Netze. In Abschnitt 4.2 werden die Entwurfsziele für das *AutoNomos*-System besprochen. Die Architektur des *AutoNomos*-Systems wird in Abschnitt 4.3 behandelt. Abschnitt 4.4 zeigt die Umsetzung des *AutoNomos*-Systems auf Basis objektorientierter Programmierung und definiert die Schnittstellen für die Entwicklung von VANET-Anwendungen.

Das Verteilen von Daten im VANET stellt einen wichtigen Dienst im *AutoNomos*-System dar. In Kapitel 5 wird es daher ausführlich behandelt. Abschnitt 5.1 betrachtet verwandte Arbeiten. Der Protokollaufbau wird in Abschnitt 5.2 detailliert dargelegt. Abschnitt 5.3 beschreibt die Implementierung des *AutoCast*-Protokolls sowie den Protokollablauf anhand eines Beispiels. Abschnitt 5.4 enthält eine ausführliche Evaluation, in der *AutoCast* in einem allgemeinen sowie einem VANET-Szenario untersucht wird. Zudem werden die Protokoll-Eigenschaften „Adaptivität“, „Realitätsnähe“ sowie „Skalierbarkeit“ jeweils in einem eigenen Abschnitt betrachtet.

In Kapitel 6 werden Anforderungen und die Umsetzung einer VANET-Simulationsumgebung beschrieben. Durch die bidirektionale Kopplung eines Netzwerksimulators mit einem Verkehrssimulator können auch die Rückwirkungen von Anwendungen auf den Straßenverkehr erfasst werden. Abschnitt 6.1 stellt verwandte Ansätze für VANET-Simulationsumgebungen vor. Abschnitt 6.2 beschreibt den Lösungsansatz basierend auf elementaren Fahrmanövern, die während der Simulationslaufzeit zur Beeinflussung des Fahrverhaltens eingesetzt werden. Die Systemarchitektur der VANET-Simulationsumgebung wird in Abschnitt 6.3 vorgestellt. Abschnitt 6.4 beschreibt kurz die Implementierung, bevor in Abschnitt 6.5 eine Evaluation die Effizienz des vorgestellten Ansatzes belegt.

Aufbauend auf dem beschriebenen *AutoNomos*-System werden in Kapitel 7 drei beispielhaft implementierte Anwendungen untersucht, die die Arbeitsweise des vorgestellten Ansatzes verdeutlichen und die Umsetzbarkeit belegen. Die Evaluation erfolgt anhand der vorgestellten VANET-Simulationsumgebung und eines Demonstrators, der ein Stau-Szenario anschaulich wiedergibt. Abschnitt 7.1 enthält eine Übersicht verwandter Arbeiten, bei denen die Auswirkungen von VANET-Anwendungen betrachtet wurden. Nachfolgend wird in Abschnitt 7.2 eine Ampelassistenz-Anwendung vorgestellt, die durch die Bereitstellung zusätzlicher Informationen im Fahrzeug und einer entsprechend angepassten Fahrweise zur Einsparung von Treibstoff beiträgt. Abschnitt 7.3 beschreibt eine Anwendung zur adaptiven Routenplanung. Hier werden HDCs an Kreuzungspunkten platziert, die Informationen zu den Fahrtzeiten der nachfolgenden Streckenabschnitte enthalten. Abschnitt 7.4 beschreibt eine Anwendung zur dezentralen Stauerkennung. Dabei wird ein Stau durch hierarchische Aggregation von Daten erkannt.

Die Arbeit schließt in Kapitel 8 mit einer Zusammenfassung und weiterführenden Ansätzen, die in späteren wissenschaftlichen Untersuchungen bearbeitet werden können.

Zusammenarbeit mit anderen Forschern

Die in dieser Arbeit vorgestellten Ergebnisse sind in enger Kooperation mit Kollegen des Instituts für Telematik der Universität zu Lübeck sowie den Projektpartnern des *AutoNomos*-Projekts [40], d. h. der von Prof. Sándor Fekete geleiteten Abteilung Algorithmik des Instituts für Betriebssysteme und Rechnerverbund der TU Braunschweig, entstanden und wären ohne die zahlreichen fruchtbaren Diskussionen sicher nicht erzielt worden.

Die Konzepte der Hovering Data Clouds und Organic Information Complexes und das darauf aufbauende *AutoNomos*-System [43, 184, 188] wurden in Zusammenarbeit mit Prof. Sándor P. Fekete, Prof. Horst Hellbrück, Prof. Stefan Fischer, Björn Hendriks, Christopher Tessars, Christiane Schmidt und Dr. Elad M. Schiller bearbeitet.

AutoCast [68, 185] wurde gemeinsam mit Prof. Horst Hellbrück, Prof. Stefan Fischer, Prof. Sándor P. Fekete und Christiane Schmidt entwickelt.

Das *Traffic Control Interface* (TraCI) [187] wurde entwickelt in Kooperation mit Michał Piórkowski, Maxim Raya und Prof. Jean-Pierre Hubaux von der EPFL sowie Prof. Horst Hellbrück und Prof. Stefan Fischer. Involviert waren zudem Daniel Krajzewicz und Michael Behrisch vom Institut für Verkehrssystemtechnik des DLR Berlin.

Das Anwendungsbeispiel *Ampelassistenz* [186] ist aus einer Kooperation mit Prof. Horst Hellbrück sowie Christian Wewetzer und Andreas Lübke von der Volkswagen AG hervorgegangen.

Im Rahmen dieser Arbeit wurden vom Autor eine Reihe von Studien-, Bachelor-, Diplom- und Masterarbeiten betreut, deren Ergebnisse hier eingeflossen sind. So hat sich Sönke N. Nommensen in seiner Masterarbeit mit der dezentralen Stauerkennung beschäftigt [129]. Torsten Teubler hat in seiner Studienarbeit das *AutoCast*-Protokoll im Netzwerksimulator Shawn umgesetzt und validiert [171]. Ein Stau-Demonstrator wurde im Rahmen mehrerer aufeinander aufbauender Arbeiten realisiert. So wurde zunächst in den Studienarbeiten von Tilo Mentler und Zbigniew Sikorski ein auf Bluetooth basierendes Kommunikationsmodell [113] und ein Fahrzeug-Folgemodell [161] entwickelt. Während Moritz Baum danach in seiner Bachelorarbeit eine dezentrale Stauerkennung [15] implementiert und evaluiert hat, wurde von Sergej Heckel im Rahmen seiner Bachelorarbeit eine grafische Oberfläche entwickelt, die das Kommunikationsmodell kapselt und eine zentrale Steuerung des Demonstrators erlaubt. Schließlich hat Zbigniew Sikorski in seiner Diplomarbeit Verfahren zur dynamischen Verkehrsführung entworfen [162].

Kapitel 2

Anwendungsgebiet

Das in dieser Arbeit verfolgte Ziel ist die Entwicklung von Konzepten und Techniken für dezentrale Anwendungen, die sich positiv auf den Straßenverkehr auswirken. In diesem Kapitel werden zunächst einige Grundlagen besprochen, die den Straßenverkehr und die Kommunikation in einem Ad-hoc-Netz und spezieller in einem VANET betreffen. Ebenso werden bereits bestehende VANET-Anwendungen kategorisiert und so die interessanten Anwendungsfälle herausgearbeitet. Abschließend werden bekannte Arbeiten aus dem Gebiet der VANET-Anwendungen vorgestellt.

2.1 Straßenverkehr

In diesem Abschnitt werden wichtige Begriffe aus dem Bereich der Verkehrsforschung definiert, die der üblichen Darstellung entsprechen (vgl. [104]). Zudem wird das im weiteren Verlauf der Arbeit genutzte Bewegungsmodell zur Simulation des Straßenverkehrs beschrieben.

Allgemein bezieht sich der *Verkehr* in dieser Arbeit auf *Kraftfahrzeuge*, oder kurz *Fahrzeuge*, die sich auf einem *Straßennetz* bewegen. Das Straßennetz setzt sich zusammen aus *Kreuzungen*, zwischen denen *Streckenabschnitte* oder einfacher *Straßen* verlaufen.

Die Bewegungen der Fahrzeuge auf einer Straße ergeben einen *Verkehrsfluss*, der in unterschiedlicher Weise gemessen werden kann. So gibt die *Verkehrsdichte* k die Anzahl der Fahrzeuge pro Wegstrecke an, während die *Verkehrsstärke* q die Anzahl der Fahrzeuge angibt, die einen bestimmten Punkt auf der Straße pro Zeit passieren. Die Verkehrsdichte und die Verkehrsstärke können mithilfe der mittleren momentanen Geschwindigkeit v_{mom} in Zusammenhang gebracht werden: $q = v_{mom} \cdot k$. Die *Kapazität* einer Straße ist definiert als die größte Verkehrsstärke, die auf der entsprechenden Straße erreicht werden kann.

Fahrzeuge, die eine Straße nacheinander befahren, werden als *vorausfahrend* bzw. *nachfolgend* bezeichnet. Der Begriff *benachbart* wird in dieser Arbeit nur in Bezug auf die Kommunikation zwischen den Fahrzeugen verwendet. Aus dem Zusammenwirken der hintereinander herfahrenden Fahrzeuge entstehen mehr oder weniger gewollte *Verkehrsstrukturen*, wie beispielsweise eine Kolonne oder ein Verkehrsstau. Eine *Verkehrslage*

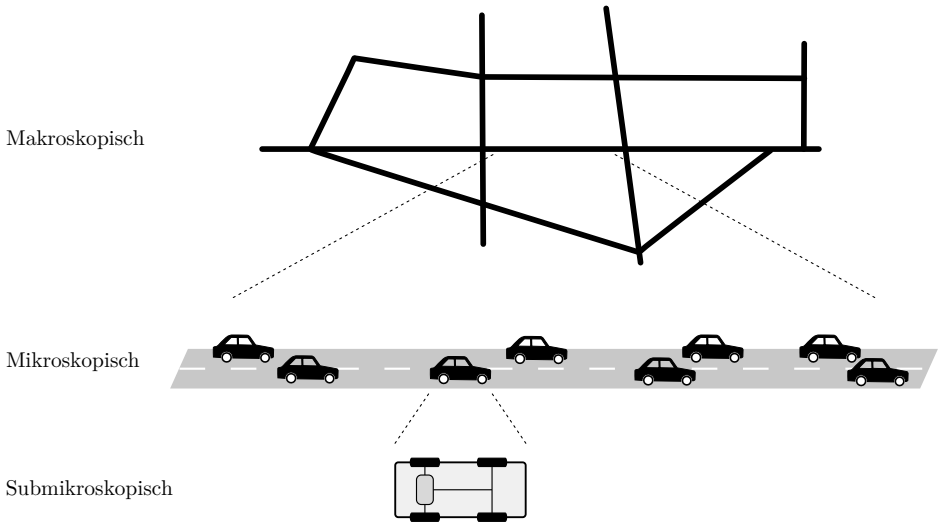


Abbildung 2.1: Verschiedene Granularitäten der Verkehrssimulation.

bezieht sich auf einen Bereich des Streckennetzes und umfasst die momentanen Verkehrsstärken und/oder die momentanen Verkehrsstrukturen, die letztlich einen Rückschluss auf die zu erwartende Verkehrsstärke zulassen.

Wie in Abbildung 2.1 dargestellt, kann der Straßenverkehr in unterschiedlicher Detailtiefe modelliert werden. Chowdhury u. a. geben in [30] einen umfassenden Überblick über die Modellierung des Straßenverkehrs in den unterschiedlichen Detailtiefen.

Makroskopische Modelle ignorieren die Verkehrsstrukturen innerhalb eines Streckenabschnitts. Fahrzeuge werden vielmehr anhand von Verkehrsflussmodellen mit einer gewissen Verzögerung von Kreuzung zu Kreuzung verschoben. Da in dieser Arbeit die Kommunikation der Fahrzeuge untereinander betrachtet wird, ist ein Modell erforderlich, das einem Fahrzeug zu jedem Zeitpunkt einen Ort innerhalb des Streckenabschnitts zuordnen kann. Ein makroskopisches Modell kann diese Position im besten Fall durch Annahme einer gleichförmigen Bewegung entlang des Streckenabschnitts berechnen. Darüber hinaus soll die Untersuchung von Verkehrsstrukturen – wie einem Verkehrsstau inklusive seiner exakten Position und internen Struktur – durch die VANET-Anwendungen ermöglicht werden. Da dies mit makroskopischen Modellen nicht möglich ist, werden diese hier nicht weiter behandelt.

Mikroskopische Modelle, auch Fahrzeug-Folgemodelle genannt, behandeln ein Fahrzeug als kleinste Einheit. Das Verhalten eines Fahrzeugs wird im Wesentlichen durch das vorausfahrende Fahrzeug bestimmt. Brockfeld u. a. vergleichen in [22] eine Vielzahl gängiger mikroskopischer Modelle miteinander.

In dieser Arbeit wird das von Krauß in [95] vorgestellte, und nachfolgend als *Krauß-*

Modell bezeichnete, mikroskopische Bewegungsmodell verwendet, welches im etablierten Verkehrssimulator *Simulator of Urban Mobility* (SUMO) verwendet wird [93, 94]. SUMO ist trotz seines Namens in der Lage auch außerstädtischen Verkehr zu simulieren und kann auch große Straßennetze mit mehr als 100.000 Fahrzeugen verarbeiten.

Submikroskopische Modelle berücksichtigen eine Vielzahl von Parametern innerhalb jedes Fahrzeugs und verfügen meist über ausgefeilte psychologische Modelle des Fahrerverhaltens. Als Bewegungsmodell für die Untersuchung von VANET-Anwendungen sind diese Modelle zu detailliert, da die zu untersuchenden Verkehrsstrukturen bereits von mikroskopischen Modellen erzeugt werden. Eine Betrachtung des Fahrerverhaltens wird allerdings dann interessant, wenn die Akzeptanz von VANET-Anwendungen untersucht wird. Dies liegt jedoch nicht im Fokus dieser Arbeit.

Das Krauß-Modell arbeitet in diskreten Zeitschritten der Dauer Δt. Für jedes Fahrzeug wird ausgehend von seiner Position und Geschwindigkeit v sowie dem Abstand g zum vorausfahrenden Fahrzeug und dessen Geschwindigkeit v_p zum Zeitpunkt t, die Position und Geschwindigkeit zum Zeitpunkt $t + \Delta t$ berechnet. Dazu wird zunächst die maximale sichere Geschwindigkeit v_{safe} ermittelt, die der Anforderung genügt, nicht mit dem vorausfahrenden Fahrzeug zu kollidieren:

$$v_{safe}(t) = v_p(t) + \frac{g(t) - \tau v_p(t)}{\frac{\bar{v}}{b} + \tau} \tag{2.1}$$

In diese Formel geht zudem der Betrag der möglichen Bremsverzögerung b ein, sowie die Durchschnittsgeschwindigkeit des betrachtet und des vorausfahrenden Fahrzeugs $\bar{v} = [v(t) + v_p(t)]/2$. Die Reaktionszeit wird als τ bezeichnet und muss mindestens einen Zeitschritt betragen [95, S. 23].

Als Nächstes wird die gewünschte Geschwindigkeit v_{des} berechnet als das Minimum aus der sicheren Geschwindigkeit, einer vorgegebenen Höchstgeschwindigkeit und der aufgrund der beschränkten Beschleunigung a maximal in einem Zeitschritt erreichbaren Geschwindigkeit:

$$v_{des} = \min\left[v_{safe}(t),\ v_{max},\ v(t) + a\Delta t\right] \tag{2.2}$$

Die Geschwindigkeit des Fahrzeugs für den nächsten Zeitschritt wird unter Berücksichtigung einer zufälligen Verzögerung σa berechnet, welche die menschliche und technischen Unzulänglichkeit widerspiegelt, eine Geschwindigkeit konstant zu halten:

$$v(t + \Delta t) = \max\left[0,\ \mathrm{rand}(v_{des} - \sigma a,\ v_{des})\right] \tag{2.3}$$

Die Funktion $\mathrm{rand}(x,\ y)$ liefert einen gleichverteilten Zufallswert im Bereich $[x, y]$. Schließlich werden die neuen Fahrzeugpositionen aufgrund der neuen Geschwindigkeiten fortgeschrieben.

2.2 Ad-hoc-Netze

Die drahtlose Kommunikation bezieht sich in dieser Arbeit auf *Ad-hoc-Netze* (vgl. [172]). Ein Ad-hoc-Netz ist eine Ansammlung sogenannter *Netzknoten*, oder einfach *Knoten*, die drahtlos miteinander kommunizieren. Jeder dieser Knoten verfügt mindestens über eine Recheneinheit, Speicher und eine Funkschnittstelle. Darüber hinaus erlauben die Knoten oftmals Zugriff auf diverse Sensoren, die Umgebungsparameter zurückliefern, wie beispielsweise die Temperatur oder die GPS-Position des Knotens.

Ein Ad-hoc-Netz zeichnet sich dadurch aus, dass die *Netztopologie* nicht fest vorgegeben ist, wie dies z. B. bei einem kabelgebundenen Netzwerk der Fall ist. Vielmehr wird die Netztopologie über die Kommunikationseigenschaften der einzelnen Knoten definiert. Ein Knoten A gilt als *benachbart* zu bzw. *Nachbar* von Knoten B, wenn ein Datenaustausch von Knoten B zu A möglich ist, so dass A die Gegenwart von B feststellen kann.

Wenn im Text nicht anders dargestellt, wird von einer bidirektionalen Kommunikation ausgegangen, so dass zwei Knoten als *benachbart* bzw. *Nachbarn* bezeichnet werden, wenn sie sich gegenseitig im *Kommunikationsgebiet* des anderen befinden.

Durch Bewegung der Knoten kann die Netztopologie und damit auch die Nachbarschaftsbeziehungen einer ständigen Änderung unterworfen sein.

Kommunizieren zwei benachbarte Knoten wird dies auch als *Single-Hop-Kommunikation* bezeichnet. Ist eine Kommunikation hingegen nur mit Unterstützung eines oder mehrerer dazwischenliegender Knoten möglich, wird dies als *Multi-Hop-Kommunikation* bezeichnet. Dabei bezeichnet der *Hopcount* die Anzahl der mindestens benötigten Weiterleitungen.

Zudem gibt es in einem Ad-hoc-Netz keinen ausgezeichneten Knoten, der bei der Kommunikation die Rolle eines Koordinators übernimmt und über den die gesamte Funkkommunikation abgewickelt wird. Stattdessen sind alle Knoten gleichberechtigt und in der Lage alle gesendeten *Broadcast-Nachrichten* seiner benachbarten Knoten zu verarbeiten. Eine *Broadcast-Nachricht* ist im Gegensatz zu einer *Unicast-Nachricht* nicht an einen speziellen Empfänger-Knoten gerichtet, sondern wird von allen empfangenden Knoten ausgewertet. Diese Interoperabilität wird möglich durch die Einhaltung internationaler Standards, wie z. B. IEEE 802.11 im Ad-hoc-Modus [71].

Die pro Zeit übertragene Datenmenge wird *Datenrate* genannt. Die Datenrate über die Funkschnittstelle ist aufgrund physikalischer Gegebenheiten begrenzt (vgl. [143]) und wird als *maximale Datenrate* bezeichnet. [1]

Charakteristisch für die Netztopologie eines Ad-hoc-Netzes ist die *Knotendichte* oder *Nachbarschaftsgröße*, die die Anzahl Nachbarn eines Knotens wiedergibt. Dieser Wert

[1] Die Begriffe *Bandbreite* und *Datenrate* werden in der Informatik häufig synonym verwendet. Da die *Bandbreite* in der Physik und Nachrichtentechnik die Breite eines Frequenzbandes angibt und somit bereits mit einer anderen Bedeutung belegt ist, wird in dieser Arbeit der Begriff *Datenrate* verwendet.

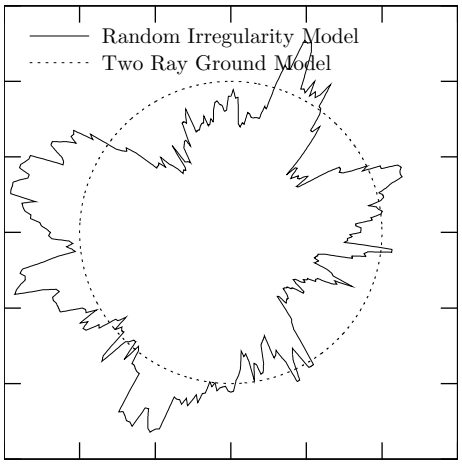


Abbildung 2.2: Kommunikationsgebiet unter Verwendung der Funkausbreitungsmodelle *Two Ray Ground* und *Random Irregularity*.

kann für jeden einzelnen Knoten sehr unterschiedlich ausfallen. Oft wird jedoch zur Klassifizierung eines Szenarios eine *mittlere Knotendichte* angegeben.

Im Folgenden wird davon ausgegangen, dass jeder Knoten über eine eindeutige *Adresse* verfügt, die ihn zumindest unter den benachbarten Knoten eindeutig identifiziert.

2.2.1 Modellierung der drahtlosen Kommunikation in Ad-hoc-Netzen

In dieser Arbeit wird davon ausgegangen, dass die Knoten eines Ad-hoc-Netzes in der Lage sind, Broadcast-Nachrichten zu versenden, die prinzipiell von allen benachbarten Knoten empfangen werden.

Da die Untersuchungen im Wesentlichen auf Computersimulationen der Ad-hoc-Netze basieren, werden entsprechende Modelle eingesetzt, welche die Funktionalität der OSI-Schichten Bitübertragung und Sicherung modellieren (vgl. ISO/OSI-Referenzmodell [75]).

Zu diesem Zweck werden als Kompromiss zwischen der von den Modellen benötigten Rechenzeit und der erreichten Realitätsnähe die folgenden Modelle verwendet.

Ein Funkausbreitungsmodell definiert das Kommunikationsgebiet eines Knotens. Hier wird das *Two-Ray-Ground*-Modell (vgl. [143, Kap. 3.6]) eingesetzt, welches in Abhängigkeit von der Distanz zwischen Sender und Empfänger eine Empfangsfeldstärke für jeden Empfänger errechnet. Aus diesem Modell folgt ein kreisförmiges Kommunikationsgebiet mit einem *Kommunikationsradius* r. Ein weiteres Funkausbreitungsmodell, das *Radio Irregularity Model* (RIM), berücksichtigt die Anisotropie der ausgesendeten Signalstärke, was zu einer richtungsabhängigen Kommunikationsdistanz führt [197]. Abbildung 2.2 zeigt einen Vergleich des Kommunikationsgebiets unter Verwendung der beiden vorge-

nannten Modelle.

Da sich in einem Ad-hoc-Netz alle Knoten das Funkmedium teilen müssen, regelt die Sicherungsschicht den Zugriff auf das gemeinsame Medium. Der Standard IEEE 802.11 setzt dafür ein Verfahren namens *Carrier Sense Multiple Access* (CSMA) ein. In Bezug auf Broadcast-Nachrichten wird bei der Verwendung von CSMA das Funkmedium vor dem Senden einer Broadcast-Nachricht abgehört. Nur wenn die Empfangsfeldstärke unter einem bestimmten Grenzwert liegt, d. h. wenn gerade keine fremde Nachricht gesendet wird, wird die eigene Broadcast-Nachricht gesendet.

Falls zwei Knoten, die sich beide in der Nachbarschaft eines potenziellen Empfängers befinden, gleichzeitig mit dem Senden einer Broadcast-Nachricht beginnen, spricht man von einer *Kollision*, mit der Folge, dass keine der beiden Nachrichten empfangen wird. Ist der Abstand zwischen den sendenden Knoten größer als der Kommunikationsradius, bemerken diese die Kollision nicht einmal. Generell nehmen Kollisionen mit zunehmender *Auslastung* des Funkmediums zu.

2.2.2 Kommunikation in VANETs

Ein *Vehicular Ad-hoc Network* (VANET) zeichnet sich dadurch aus, dass sich die Knoten des Ad-hoc-Netzes auf Fahrzeugen befinden. Dementsprechend spricht man in einem VANET von miteinander kommunizierenden Fahrzeugen.

Eine weitere Besonderheit ist das Bewegungsmodell, das bereits in Abschnitt 2.1 vorgestellt wurde. So können beispielsweise Daten nicht nur per drahtloser Kommunikation von einem Ort zum anderen befördert werden, sondern darüber hinaus kann auch die Bewegung der Fahrzeuge zur Datenbeförderung genutzt werden. So kann eine Mitnahme von Daten durch den Gegenverkehr durchaus hilfreich sein.

Während der *Markteinführung* eines VANETs wird der Anteil der mit Funkschnittstelle ausgestatteten Fahrzeuge langsam ansteigen. Dieser Parameter wird als *Ausstattungsgrad* bezeichnet. Aufgrund des vorgegebenen Bewegungsmodells an das sich alle Fahrzeuge unabhängig von der Ausstattung halten, ist die Knotendichte im Ad-hoc-Netz direkt vom Ausstattungsgrad abhängig.

Um kooperative dezentrale VANET-Anwendungen studieren zu können, werden im Folgenden einige Basistechniken als gegeben vorausgesetzt. So wird im weiteren Verlauf dieser Arbeit davon ausgegangen, dass Broadcast-Nachrichten über ein drahtloses Ad-hoc-Netz gesendet werden können und von allen Fahrzeugen innerhalb des Kommunikationsradius entweder fehlerfrei oder gar nicht empfangen werden, da Übertragungsfehler durch die Sicherungsschicht ausgeschlossen werden. Medienzugriff, Integritätsprüfung, sowie die Codierung der Daten auf der Funkschnittstelle werden ebenfalls nicht weiter betrachtet. Es ist zu erwarten, dass sich in naher Zukunft der als *Dedicated Short Range Communication* (DSRC) bezeichnete und als IEEE 802.11p standardisierte Funkstandard für die Fahrzeug-Fahrzeug-Kommunikation durchsetzt [72, 76, 77, 169].

In jedem Fahrzeug wird der lokale Zugriff auf die Fahrgeschwindigkeit und den Status von Assistenzsystemen wie ABS und ESP vorausgesetzt. Ebenso wird vom Vorhandensein eines GPS-Navigationssystems in jedem Fahrzeug ausgegangen, das Zugriff auf die eigene Position sowie die Straßenkarte erlaubt.

2.2.3 Sicherheitsaspekte in VANETs

Sicherheitsaspekte der Kommunikation werden in dieser Arbeit explizit nicht betrachtet. Das in dieser Arbeit vorgestellte Kommunikationsprotokoll kann jedoch problemlos mit der im EU-Projekt *Secure Vehicular Communication* (SeVeCom) entwickelten umfangreichen Sicherheitsarchitektur für VANET-Anwendungen [82, 132] kombiniert werden. Die von Papadimitratos u. a. in [132] vorgestellte Sicherheitsarchitektur wahrt die Anonymität aller am VANET teilnehmenden Knoten (Fahrzeug, Zugriffspunkt, etc.) im normalen Betrieb, während bei einem Netzwerkangriff der Absender einer Nachricht von staatlicher Seite zweifelsfrei einer Person zugeordnet werden kann.

Dazu werden vertrauenswürdige (staatliche) Zertifizierungsstellen vorausgesetzt, die jeden Knoten – beispielsweise bei der Fahrzeugzulassung – registrieren und ihm ein Schlüsselpaar zur asymmetrischen Verschlüsselung sowie ein Zertifikat zur langfristigen Identifizierung ausstellen. Zur eigentlichen Kommunikation innerhalb des VANETs lassen sich die Knoten in regelmäßigen Abständen von ihrer Zertifizierungsstelle einen Satz kurzzeitig gültiger anonymisierter Zertifikate ausstellen. Die Kommunikation mit den Zertifizierungsstellen erfolgt entweder über Zugriffspunkte oder über eine dedizierte Mobilfunk-Verbindung. Jedes Zertifikat wird nur für einen kurzen Zeitraum verwendet, um versendete Nachrichten zu signieren. Da diese Zertifikate von einer vertrauenswürdigen Zertifizierungsstelle unterschrieben sind, kann jeder Empfänger einer Broadcast-Nachricht den vertrauenswürdigen Ursprung sowie die Integrität der übertragenen Daten überprüfen. Darüber hinaus ist nur die Zertifizierungsstelle in der Lage die wahre Identität des Absenders zurückzuverfolgen. Durch die fortlaufend wechselnden Zertifikate ist auch eine Verfolgung der Fahrtroute eines ausgewählten Fahrzeugs nicht ohne Weiteres möglich.

2.3 Dezentrale Anwendungen im Straßenverkehr

In dieser Arbeit werden dezentrale Anwendungen betrachtet, deren Ziel die Unterstützung von Verkehrsteilnehmern ist. Die durch diese Anwendungen gewonnenen Informationen über die Verkehrslage sollen beispielsweise zu einer Verbesserung des Verkehrsflusses führen oder durch frühzeitige Warnmeldungen die Verkehrssicherheit erhöhen.

Technisch bauen die betrachteten Anwendungen auf einem *Vehicular Ad-hoc Network* (VANET) auf, welches – wie im letzten Abschnitt beschrieben – eine direkte Interaktion benachbarter Fahrzeuge mittels drahtloser Kommunikation erlaubt. Folglich sind die

im Folgenden als *VANET-Anwendungen* bezeichneten Anwendungen per se dezentral aufgebaut.

Es folgt eine Kategorisierung der potenziell interessanten VANET-Anwendungen. Abgeschlossen wird dieser Abschnitt mit einer Übersicht verwandter Arbeiten.

2.3.1 Kategorisierung

Das *CAR-2-CAR Communication Consortium* (C2C-CC), eine Initiative europäischer Automobilhersteller, stellt in einem Manifest [10] potenzielle Szenarien für VANET-Anwendungen vor. Diese werden vom C2C-CC in drei Kategorien eingeteilt:

Safety: In diese Kategorie fallen Anwendungen, die die Sicherheit in bestimmten Szenarien erhöhen. Beim Auftreten eines Verkehrsszenarios, das vom jeweiligen Anwendungsfall abgedeckt wird, erhöht sich die Sicherheit für die aktiv beteiligten Verkehrsteilnehmer.

Als Anwendungsbeispiele aus dieser Kategorie werden u. a. eine *kooperative Kollisionswarnung* sowie eine *Warnung vor Gefahrenstellen* genannt. Die *kooperative Kollisionswarnung* hilft bei der Vermeidung von Auffahrunfällen. Dazu tauschen benachbarte Fahrzeuge während des normalen Fahrens Informationen wie Position, Geschwindigkeit und Fahrtrichtung aus. Durch Auswertung der eigenen Fahrsituation in Korrelation mit umliegenden Fahrzeugen können kritische Situationen frühzeitig erkannt und der betreffende Verkehrsteilnehmer gewarnt werden. Bei der *Warnung vor Gefahrenstellen* werden Informationen über schlechte Fahrbedingungen, die z. B. zum Einsatz des ESP führen, umliegenden Fahrzeugen mitgeteilt. Relevante Informationen können dem Verkehrsteilnehmer angezeigt oder direkt zur Optimierung, beispielsweise eines adaptiven Fahrwerks, genutzt werden.

Traffic Efficiency: Diese Anwendungen zielen auf eine Optimierung des Verkehrsflusses und eine verbesserte Auslastung des Verkehrsnetzes. Dazu werden den Betreibern des Verkehrsnetzes bzw. den Verkehrsteilnehmern zusätzliche Informationen zur Verkehrslage bereitgestellt. Das VANET dient dem kollektiven Erfassen, Verarbeiten und Verteilen von Verkehrsinformationen in einer Detailtiefe, die zentralisierte Lösungen nicht bieten können. Durch eine verbesserte Auslastung des Straßennetzes können zum einen individuelle Fahrzeiten verkürzt werden, zum anderen lässt sich die Notwendigkeit zum Neubau von Straßen reduzieren.

In diese Kategorie fällt beispielsweise eine *Navigationsanwendung*, die Echtzeitdaten zur Verkehrslage über das VANET empfängt und in die Routenplanung einbezieht. Durch direkte Interaktion der beteiligten Fahrzeuge können ungewollte Effekte, wie eine Überlastung der Ausweichstrecke, vermieden werden. Eine weitere Anwendung offeriert im innerstädtischen Bereich eine *Geschwindigkeitsemp-*

fehlung zur Nutzung der grünen Welle. Dabei wird den Verkehrsteilnehmern unter Berücksichtigung der Ampelschaltungen auf der jeweiligen Route sowie der lokalen Verkehrsdichte eine optimale Geschwindigkeit vorgeschlagen. Dies resultiert in weniger Brems- und Beschleunigungsvorgängen und folglich in einem reduzierten Energieverbrauch.

Infotainment and Others: In diese Kategorie fallen alle Anwendungen, die weder die individuelle Sicherheit noch den allgemeinen Verkehrsfluss verbessern.

Solche Anwendungen stellen beispielsweise *Informationsdienste* bereit, die das VANET nutzen, um über Zugriffspunkte, die entlang der Straße angeordnet sind, auf das Internet zuzugreifen. Zudem können auch die Zugriffspunkte selbst Informationen über nahe liegende Sehenswürdigkeiten verteilen. Eine *Ferndiagnose* erlaubt Werkstätten die Fahrzeugdiagnose, ohne eine Kabelverbindung zum Fahrzeug herstellen zu müssen. Auch das Aktualisieren der Fahrzeugsoftware wird so möglich.

Wie schon in den Beispielen angedeutet, sind bei VANET-Anwendungen unterschiedliche Akteure involviert. Verkehrsteilnehmer profitieren von Warnmeldungen zur Erhöhung der Verkehrssicherheit und von verbesserten Informationen zur Verkehrslage. Betreiber der Verkehrsnetze können die zusätzlichen Daten zur Verkehrslage nutzen, um den Verkehr z. B. durch adaptive Ampelschaltungen besser zu steuern. Ferner beteiligt sind Betreiber von Zugriffspunkten, die entlang der Straßen aufgestellt sind und Zugang zum Internet und lokalen Informationen gewähren.

In dieser Arbeit liegt der Fokus auf kooperativen dezentralen Anwendungen, bei denen potenziell eine große Anzahl an Fahrzeugen involviert ist. Dabei agieren vornehmlich die Fahrzeuge direkt untereinander. Eine Infrastruktur bestehend aus Zugriffspunkten entlang der Straße wird nicht grundsätzlich vorausgesetzt, kann aber für bestimmte Anwendungen erforderlich sein. Beispielsweise ist die in Abschnitt 7.2 vorgestellte *Ampel-Assistenz*-Anwendung darauf angewiesen, dass Ampeln ihren zukünftigen Signalwechsel kommunizieren können. Verkehrsnetzbetreiber können im zweiten Schritt von der dezentral erfassten Verkehrslage profitieren, indem sie das VANET an signifikanten Punkten des Straßennetzes abhören.

Neben kooperativen Anwendungen, bei denen der verursachte Datenverkehr allen beteiligten Verkehrsteilnehmern zugutekommt, gibt es auch solche Anwendungen, die individuellen Datenverkehr verursachen. Zu diesen Anwendungen zählt beispielsweise die Nutzung des *World Wide Web* (WWW). Auch wenn von Bechler in [16] ein umfassender Ansatz zur „Internet Integration von VANETs“ vorgestellt wird, erscheint die Nutzung zellulärer Mobilfunknetze wie *Universal Mobile Telecommunications System* (UMTS), in denen Daten per se personalisiert in Unicast-Nachrichten übertragen werden, angebrachter.

Die VANET-Anwendungen, die von Reichardt u. a. in [147] kategorisiert werden, decken sich in ihrer Gesamtheit mit den in dieser Arbeit betrachteten kooperativen, dezentralen Anwendungen:

Information and Warning Functions: Anwendungen dieser Kategorie erfassen die Verkehrslage und senden Informations- bzw. Warnmeldungen an ankommende Fahrzeuge. Betrachtete Szenarien umfassen Bremsmanöver, Unfälle, Gefahrenstellen, Verkehrsstaus sowie die Erfassung von Fahrzeugen im toten Winkel.

Communication-Based Longitudinal Control Systems: Diese Kategorie enthält Anwendungen, die beim hintereinander Herfahren assistieren. Zum Beispiel kann eine VANET-Anwendung die Sichtweite erhöhen und Beschleunigungs- und Bremshinweise geben, wodurch die Kapazität einer Straße erhöht werden kann.

Co-Operative Driver Assistance Systems: Das Koordinieren von Fahrzeugen an kritischen Stellen, wie Autobahnauffahrten, Fahrbahnverengungen oder Kreuzungen ist die Aufgabe von Anwendungen dieser Kategorie.

Diese Kategorisierung lässt sich wie folgt auf die Kategorien des C2C-CC abbilden. So entspricht die Kategorie *Information and Warning Functions* der Kategorie *Safety* des C2C-CC. Die C2C-CC-Kategorie *Traffic Efficiency* wird von Reichardt u. a. in die Kategorien *Communication-Based Longitudinal Control Systems* und *Co-Operative Driver Assistance Systems* unterteilt.

Anzumerken ist an dieser Stelle, dass sicherheits- und zeitkritische Anwendungen, die eine zuverlässige Interaktion unter Echtzeitbedingungen erfordern, z. B. die *elektronische Deichsel*, allein mit einem VANET nur schwer realisierbar sind. Ein Störsender könnte beispielsweise die drahtlose Kommunikation unterbrechen, so dass Gefahrenmeldungen nicht mehr übertragen werden könnten. Daher beschränkt sich diese Arbeit auf VANET-Anwendungen, die einem Verkehrsteilnehmer zusätzliche Informationen und auch Warnungen liefern, ihn aber nicht von seiner Verantwortung entbinden.

2.3.2 Verwandte Arbeiten

In den letzten zehn Jahren gab es eine Vielzahl großer und kleiner Forschungsprojekte, die sich mit verschiedenen Aspekten dezentraler Anwendungen im Straßenverkehr beschäftigt haben.

Die großen deutschen Forschungsprojekte in diesem Bereich sind *FLEETNET – Internet on the Road* (2000-2003) [33, 50] und das zeitlich anschließende *Network on Wheels* (2004-2008) [45], gefördert jeweils vom *Bundesministerium für Bildung und Forschung* (BMBF). Im Rahmen dieser Projekte wurden zunächst Aspekte der unteren Schichten, wie Funkschnittstelle, Medienzugriff und (geografisches) Routing behandelt. Es wurde eine Referenzarchitektur für ein VANET inklusive Zugriffspunkten entlang der

Straße und Zugriff auf das Internet entworfen. Basierend auf diesen Projekten wurde das C2C-CC als Zusammenschluss europäischer Automobilhersteller, Zuliefererfirmen und Forschungseinrichtungen gegründet, um an gemeinsamen Kommunikationsstandards sowie der europaweiten Zuteilung von Funkfrequenzen für VANETs zu arbeiten. In der Zwischenzeit ist der Frequenzbereich zwischen 5,875 GHz und 5,905 GHz europaweit für VANET-Anwendungen reserviert, in dem nach dem Standard IEEE 802.11p [72] kommuniziert werden soll. Des Weiteren wurde z. B. der *Protokoll-Stack für Network Mobility* (NEMO) als Internet Draft [12] veröffentlicht.

Eine Forschergruppe innerhalb des *FLEETNET*-Projekts behandelte ein selbstorganisierendes Verkehrsinformationssystem (Self Organizing Traffic Information System, SOTIS) [194]. Dabei wurden Aspekte der Nachrichtenverteilung [191] und des Medienzugriff bzw. der Mediennutzung [193, 195] betrachtet. Allerdings erfolgte die Betrachtung dieser Aspekte immer mit einer starken Fokussierung auf eine Anwendung zur Messung von Durchschnittsgeschwindigkeiten innerhalb von Fahrbahnsegmenten fester Länge [192]. Diese Anwendung wurde auch prototypisch implementiert. Die dezentrale Weiterverarbeitung der gewonnenen Daten wurde allerdings ebenso wenig behandelt wie Maßnahmen zur Verbesserung der Verkehrslage.

Das Verbundprojekt *PReVENT* [159] behandelt neben diversen Anwendungen der Kategorie *Safety* auch Themen wie Geschäftsmodelle und Standards zum Zugriff auf lokal vorgehaltene Straßenkarten.

Im Rahmen des vom *Bundesministerium für Wirtschaft und Technologie* (BMWI) geförderten *AKTIV*-Projekts [6] werden Anwendungen aus den Kategorien *Safety* und *Traffic Efficiency* betrachtet. Als weiterer Schwerpunkt wird in diesem Projekt die Eignung zellulärer Netze für kooperative Anwendungen untersucht. Während Teile der vorgestellten Anwendungen auf einem VANET aufbauen, werden strategische Entscheidungen weiterhin zentralisiert betrachtet.

Im BMBF-Verbundprojekt *ADVEST* [2] werden Simulationsmodelle und Modelle zur mathematischen Optimierung kombiniert. Das Ziel stellt eine adaptive Verkehrssteuerung dar, die in unterschiedlichen Anwendungsszenarien, wie dezentrale Verkehrssteuerung [41], Ampelsteuerungen sowie Evakuierungsszenarien, untersucht werden.

Neben groß angelegten Forschungsprojekten existiert eine große Anzahl an Arbeiten, die dezentrale VANET-Anwendungen betrachten. Dabei werden die zugrunde liegenden Techniken zumeist sehr pragmatisch und auf die jeweilige Anwendung gerichtet ausgewählt. Die nachfolgend exemplarisch beschriebenen Anwendungen könnten von einer Umsetzung auf Basis des *AutoNomos*-Systems profitieren und auch Synergie-Effekte durch Kooperation der einzelnen Anwendungen erzielen.

Cottingham u. a. beschreiben in [31] und [32] dezentrale Verkehrsanwendungen und gehen dabei vom Vorhandensein einer ausgedehnten Zugriffspunkt-Infrastruktur aus. Die Fahrzeuge kommunizieren ausschließlich mit den Basisstationen. Eine Multihop-Kommunikation zwischen den Fahrzeugen ist nicht vorgesehen. So bleibt eine zentrale

Komponente in den ansonsten dezentral gehaltenen Anwendungen bestehen. Analog zu den im Folgenden vorgestellten *Hovering Data Clouds* (HDCs) werden die Zugriffspunkte genutzt, um Daten dezentral im Netzwerk zu speichern. HDCs sind insofern flexibler, als dass sie ohne ausgewiesene Netzknoten auskommen und ihren Speicherort entsprechend der zugrunde liegenden Daten verändern können.

Das in [122] vorgestellte *TrafficView*-System ermittelt die Verkehrslage durch das Weiterleiten der eigenen Geschwindigkeit in Verbindung mit Geschwindigkeitsangaben, die aus der eigenen Fahrtrichtung empfangen werden. Bei jeder Weiterleitung werden die eigenen Daten mit denen der vorausfahrenden Fahrzeuge kombiniert. So kennt jedes Fahrzeug die vor ihm liegende Verkehrslage und kann seine Fahrweise entsprechend anpassen.

Im Projekt *BeeJamA* werden Routenempfehlungen basierend auf einem Multi-Agenten-Ansatz dezentral berechnet [183]. In Anlehnung an das *BeeHive*-Routing [39] werden Positionen, Geschwindigkeiten und Fahrtziele der Fahrzeuge auf den einzelnen Streckenabschnitten von Software-Agenten eingesammelt. Die Verarbeitung erfolgt durch stationäre Einheiten, die jeweils einem Teil des Streckennetzes zugeordnet sind. Diese stationären Einheiten geben den Verkehrsteilnehmern vor jeder Kreuzung Abbiegeempfehlungen auf Basis der Verkehrslage und des jeweiligen Fahrtziels.

In einem konzeptionellen Artikel [121] stellen Morris u. a. das *CarNet* vor, welches letztlich als Motivation für den von denselben Autoren entwickelten *Grid Location Service* dient, der in Verbindung mit geografischer Nachrichtenweiterleitung eine Unicast-Kommunikation innerhalb von Ad-hoc-Netzen erlaubt [99]. Auf die angedeuteten Verkehrsanwendungen wird nicht weiter eingegangen.

Zusammengefasst werden zurzeit eine große Anzahl unterschiedlicher Aspekte von VANET-Anwendungen erforscht. Bei der Methodik zur Anwendungsentwicklung wird jedoch meist pragmatisch und auf den jeweiligen Anwendungsfall zielgerichtet gearbeitet. So sind bereits eine Vielzahl an (dezentralen) VANET-Anwendungen in der Literatur beschrieben. Eine Kombination verschiedener Anwendungen in einem Szenario oder Synergieeffekte durch Datenaustausch unter den Anwendungen scheitern hingegen noch.

An dieser Stelle setzt die vorliegende Arbeit an. Dafür werden im nächsten Kapitel Organic-Computing-Konzepte zur generellen Umsetzung dezentraler Anwendungen in mobilen Ad-hoc-Netzen vorgestellt. Dabei wird direkter Bezug auf VANET-Anwendungen genommen, obgleich die Konzepte prinzipiell in beliebigen Ad-hoc-Netzen anwendbar sind.

Kapitel 3

Organic Computing

Die Komplexität von Rechnersystemen nimmt seit den 1960er Jahren exponentiell zu, das prognostizierte Moore bereits 1965 in Bezug auf die Anzahl der Schaltkreiskomponenten auf einem Computerchip [119]. Diese als *Mooresches Gesetz* bekannte Prognose gilt bis heute: Alle 18 Monate verdoppelt sich die Anzahl der Transistoren pro Flächeneinheit [59, 154]. Die auf Computerchips bezogene Aussage lässt sich auch auf Computersysteme im Allgemeinen übertragen: Die Komplexität dieser Systeme wird immer weiter zunehmen. Die auch zukünftig weiter steigende Komplexität lässt die Handhabung und Administration technischer Systeme immer schwieriger werden. Daher müssen neue Herangehensweisen entwickelt werden, mit deren Anwendung auch zukünftige technische System beherrschbar bleiben.

Betrachtet man das Verhalten von Lebewesen, die auch unter unterschiedlichsten und unvorhergesehenen Bedingungen „funktionieren“, so liegt es nahe, Erkenntnisse über die Funktionsweise von Lebewesen für die Entwicklung von künstlichen Systemen nutzbar zu machen. Dabei können alle Organisationsebenen von Lebewesen, wie Moleküle, Zellen, Organismen oder Gesellschaften, als Vorbild dienen. Dementsprechend ist ein *organischer Computer* gemäß [117] definiert als: „Ein selbstorganisierendes System, das sich den jeweiligen Umgebungsbedürfnissen dynamisch anpasst. Organische Computersysteme haben sogenannte *Selbst-x-Eigenschaften*: Sie sind selbstkonfigurierend, selbstoptimierend, selbstheilend, selbsterklärend und selbstschützend.“

Demzufolge verhalten sich organische Computer eher wie intelligente Assistenten als wie starre Befehlsempfänger. Dabei sind sie flexibel, optimieren ihr Verhalten selbst und konfigurieren sich – beispielsweise beim Ausfall von Komponenten – selbständig um. Der Nachteil der so geschaffenen neuen Freiheitsgrade liegt in mitunter langen Lernprozessen, während derer ein organisches Computersystem auch einmal fehlerhaft reagieren kann, was in technischen, insbesondere sicherheitskritischen Anwendungen nicht tolerierbar ist.

Diese Sichtweise begründet das Forschungsgebiet des *Organic Computing*, in dem drei große Forschungsziele angestrebt werden [117]:

- Das Verständnis der Prinzipien der Selbst-Organisation natürlicher Systeme,
- die Umsetzung dieser Prinzipien in technisch nutzbare Verfahren und Werkzeuge

- und deren praktische Nutzung in technischen Anwendungen.

Ein herkömmlicher Entwicklungsprozess ist heutzutage geprägt durch die Anfertigung eines abstrakten Grobentwurfs, der immer weiter konkretisiert wird, bis alle Funktionen spezifiziert sind. Dieses Vorgehen wird als *Top-Down*-Ansatz bezeichnet.

Ein Blick in die Natur zeigt viele Beispiele, bei denen die Vorgehensweise eine ganz andere ist. So bilden sich viele natürliche Strukturen aus einer Vielzahl von Einzelsystemen, seien es Moleküle, Zellen oder auch komplexe Lebewesen. Durch die Selbst-Organisation der einzelnen Einheiten entstehen somit in der Natur Strukturen wie Fischschwärme, die sich kollektiv bewegen oder einem Angreifer ausweichen. In [26] werden eine Vielzahl solcher selbstorganisierender Phänomene in biologischen Systemen vorgestellt.

Gemeinsam haben diese Beispiele, dass sich globale Strukturen oder Verhaltensweisen aufgrund von lokalen Interaktionen bilden. Dieses Phänomen wird als *Emergenz* bezeichnet [70, 79]. Selbst-Organisation und Emergenz werden in dieser Arbeit nach der Definition von De Wolf und Holvoet [34] verwendet (frei übersetzt aus dem Englischen): Demnach ist Selbst-Organisation ein dynamischer und adaptiver Prozess, durch den Systeme Strukturen erlangen und aufrecht erhalten, ohne dass eine externe Kontrolle vorhanden ist. Emergenz zeigt sich in einem System durch die Bildung von Zusammenhängen auf der Makro-Ebene, die aufgrund von Interaktionen auf der Mikro-Ebene dynamisch auftauchen. Diese so entstehenden Zusammenhänge sind nicht an den einzelnen Elementen der Mikro-Ebene erkennbar.

Ein Entwurfsprozess für technische Systeme, der diesem Vorbild folgt, wird als *Bottom-Up*-Ansatz bezeichnet. Dabei wird zunächst das Verhalten der einzelnen Systemkomponenten festgelegt. Das Verhalten des Gesamtsystems entsteht als emergente Eigenschaft ausgehend von der Selbst-Organisation der einzelnen Komponenten. Somit ist es möglich, dass das Gesamtsystem in unvorhergesehenen Situationen ein Verhalten zeigt, das vom Entwickler nicht explizit vorgesehen war.

Damit das Verhalten des Gesamtsystems dennoch gewisse Grenzen nicht überschreitet, können nebenläufige Kontrollstrukturen eingesetzt werden. Ein Beispiel hierfür stellen sogenannte *Observer/Controller*-Architekturen dar [149]. Dabei wird das technische System fortlaufend durch einen *Observer* überwacht und kritische Systemzustände an den *Controller* weitergegeben, der korrigierend auf einzelne oder Gruppen von Komponenten einwirkt.

Auch schon vor Organic Computing gab es einige von der Natur inspirierte Forschungsansätze, von denen die bekanntesten nachfolgend vorgestellt werden.

In einem zellulären Automaten sind Zellen in einem Zellularraum angeordnet. Der Zellularraum entspricht beispielsweise einer 1- oder 2-dimensionalen Matrix. Der Zustand einer Zelle wird bestimmt durch Zustandsübergangsregeln, die auf einer endlichen Nachbarschaft der Zelle beruhen. Somit setzen zelluläre Automaten das Prinzip der strikten Lokalität um (vgl. [127]). Beispielsweise bei Conways *Spiel des Lebens* [52] bilden sich aus

den lokal wirkenden Regeln emergente größere Strukturen. Auch bei der Simulation des Straßenverkehrs existieren Bewegungsmodelle auf Basis zellulärer Automaten. So konnte mit dem Nagel-Schreckenberg-Modell erstmals ein Stau aus dem Nichts mit einfachen lokalen Regeln innerhalb eines zellulären Automatens nachgebildet werden [123].

Im Bereich der biologisch inspirierten Verfahren stellen genetische Algorithmen [54] sowie evolutionäre Algorithmen [146] die Grundlage für lernfähige technische Systeme dar. Zudem lassen sich durch diese Ansätze auch Lösungen für Problemstellungen finden, die analytisch nicht vollständig erfasst werden können.

IBM propagierte *Autonomic Computing* [86] als unternehmensweites Leitprojekt. Dabei wurde der Begriff der *Selbst-X-Eigenschaften* geprägt. In Bezug auf Rechenzentren und Rechnernetze wurden die folgenden Eigenschaften definiert, die auch allgemein auf technische Systeme anwendbar sind:

Selbst-Konfiguration Die Konfiguration eines technischen Systems erfolgt auf einer höheren Ebene, beispielsweise durch die Vorgabe von Zielen. Die Einzelsysteme adaptieren ihr Verhalten entsprechend der Vorgaben, ohne dass eine externe Kontrolle notwendig ist.

Selbst-Optimierung Das technische System sucht fortlaufend selbständig nach Möglichkeiten, um die Leistung und Effizienz zu verbessern.

Selbst-Heilung Fehlerzustände werden automatisch detektiert und bewertet. Im Fehlerfall führt das technische System selbständig eine Reparatur oder Umkonfiguration durch, die den Fehlerzustand beendet.

Selbst-Schutz Das technische System schützt sich automatisch gegen böswillige Angreifer oder verkettete Fehler, die zum Ausfall des Gesamtsystems führen könnten.

Organic Computing hat nicht den Anspruch alle diese etablierten Forschungsbereiche abzulösen, sondern strebt eine Kombination unterschiedlicher Methodiken an und ist dementsprechend als sehr heterogenes Forschungsgebiet aufgestellt. Dies zeigt sich auch im zur Zeit laufenden DFG-Schwerpunktprogramm 1183 [5], das den Titel „Organic Computing“ trägt.

So werden in diesem Schwerpunktprogramm Organic-Computing-Projekte aus unterschiedlichen Fachrichtungen der Informatik durchgeführt. Beispielsweise wird untersucht, inwiefern biochemische Prozesse als Programmier-Paradigma nutzbar sind [35]. In einem weiteren Projekt werden massiv parallelisierte Chips entwickelt, wodurch z. B. bei einem Kamera-Chip eine Vorverarbeitung der Ergebnisse, wie die Berechnung von Objektkonturen und Objektschwerpunkten, in Echtzeit möglich wird [47]. Einige der Projekte im Schwerpunktprogramm beschäftigen sich auch mit Ad-hoc-Netzen, beispielsweise in Bezug auf energieoptimiertes Routing in Sensornetzen [152] oder in Bezug auf die Bildung

von Schwarmintelligenz [23]. Eine vollständige Liste der Projekte im Schwerpunktprogramm findet sich in [5].

Auch die vorliegende Arbeit wird im Rahmen des DFG-Schwerpunktprogramms 1183 gefördert [40].

Ein weiteres allgegenwärtiges Beispiel für ein technisches System, das den Organic-Computing-Prinzipien entspricht, ist das Internet. Es stellt eine der komplexesten technischen Strukturen dar, die jemals entworfen wurden. Dabei wurde das Internet nicht im Detail geplant, sondern ist über die Zeit immer weiter gewachsen. Dabei sind Ordnungsstrukturen entstanden, die Ähnlichkeit mit denen des Gehirns aufweisen [14].

In Ad-hoc-Netzen und somit auch in VANETs ist durch die fehlende steuernde Instanz und die flexible Netztopologie per se Selbst-Organisation zur Kommunikation nötig. Durch die Vielzahl miteinander interagierender Knoten auf Mikro-Ebene wird ein Ergebnis auf globaler Ebene üblicherweise durch lokal wirkende Regeln angestrebt und bildet sich als emergente Eigenschaft.

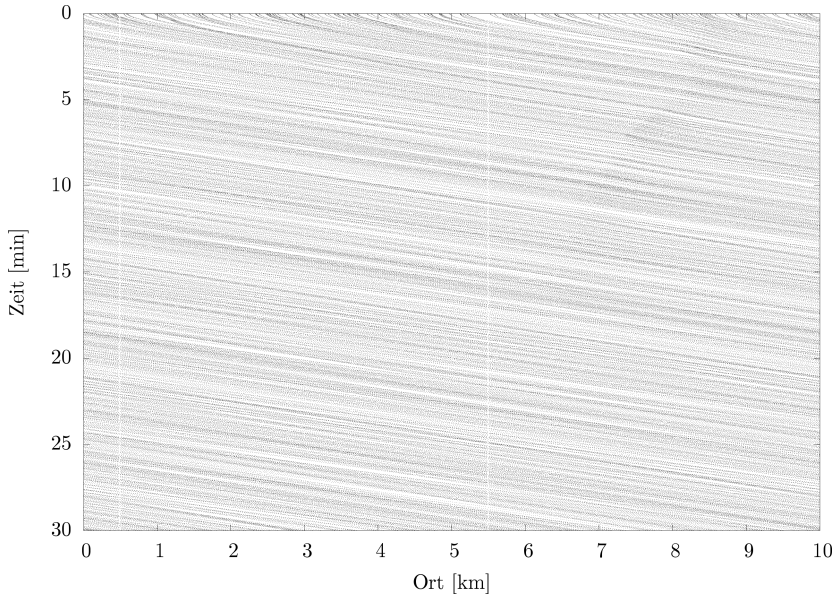
3.1 Organic Computing im Straßenverkehr

Betrachtet man den Straßenverkehr unter Organic-Computing-Gesichtspunkten, lassen sich diverse Parallelen finden. So fehlt im Straßenverkehr eine steuernde zentrale Instanz, stattdessen fährt jeder Verkehrsteilnehmer seinem persönlichen Fahrtziel entgegen. Dabei interagieren die Verkehrsteilnehmer zwangsläufig miteinander, wodurch Verkehrsstrukturen als emergente Phänomene entstehen.

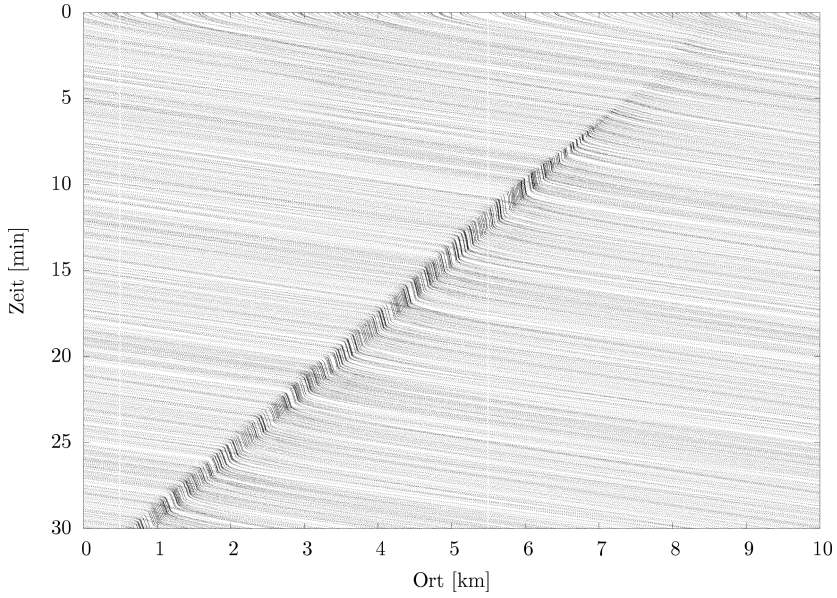
Abbildung 3.1 zeigt anhand eines Beispiels, dass die Entstehung von Verkehrsstrukturen manchmal schon durch eine minimale Änderung des Fahrverhaltens bedingt ist. Zu sehen ist die Auswertung eines simulierten Verkehrsszenarios, in dem sich die Fahrzeuge auf einer 10 km langen Rundstrecke bewegen. In beiden Graphen ist auf der Abszisse der Ort der Fahrzeuge und auf der Ordinate die Zeit aufgetragen. So lassen sich die Fahrzeugpositionen zu Trajektorien verbinden, welche die Bewegung der einzelnen Fahrzeuge wiedergeben.

Während in Abbildung 3.1a der Verkehr flüssig läuft, entwickelt sich in Abbildung 3.1b ein Stau, der an der Diagonalen erkannt werden kann, in der die Trajektorien steiler verlaufen. Die einzige Änderung zwischen den beiden Simulationen ist eine um 5 km/h höhere Geschwindigkeit in Abbildung 3.1b gegenüber Abbildung 3.1a. Es stellt sich hier die Frage, ob tatsächlich die Geschwindigkeit für alle Fahrzeuge begrenzt werden muss (vgl. [176]), oder ob auch eine lokale Geschwindigkeitsbeschränkung den Stau hätte verhindern können. Und wenn das der Fall ist, wie lässt sich die Entstehung eines Staus basierend auf lokalen Informationen vorhersagen und vermeiden?

Ein weiteres Phänomen, bei dem sich eine für alle ungünstige Verkehrssituation einstellt, weil kein Verkehrsteilnehmer benachteiligt werden möchte, ist oft auf Autobahnen



(a) Höchstgeschwindigkeit $v_{max} = 100\,\mathrm{km/h}$



(b) Höchstgeschwindigkeit $v_{max} = 105\,\mathrm{km/h}$

Abbildung 3.1: Verkehrslage dargestellt durch Trajektorien der Fahrzeuge (Raum-Zeit-Kurven). Schon eine minimale Änderung äußerer Vorgaben, wie hier der Höchstgeschwindigkeit, entscheidet über das Auftreten eines Verkehrsstaus (Diagonale mit steileren Trajektorien im unteren Diagramm).

zu beobachten. Es fahren auf der rechten Spur LKWs mit relativ großem Abstand und auf der linken Spur PKWs mit geringem Abstand. In einer solchen Situation sind nur wenige Verkehrsteilnehmer bereit, in die Lücken zwischen den LKWs zu fahren, aus Angst nicht wieder in die linke Spur hineingelassen zu werden. So bleibt eine große Kapazität der Autobahn ungenutzt.

Auch die von den einzelnen Verkehrsteilnehmern getroffenen Entscheidungen bezüglich ihrer Fahrtroute kann zur Bildung von Engstellen führen. Würden einzelne Verkehrsteilnehmer eine unwesentlich längere Fahrtroute in Kauf nehmen, könnten Engstellen mitunter vermieden werden. Das in [21] vorgestellte *Braess Paradoxon* zeigt, dass sogar beim Bau einer neuen Straße die Verkehrsdichte auf einzelnen Streckenabschnitten ansteigen kann.

Der Straßenverkehr erscheint so als besonders vielversprechendes Szenario für den Einsatz von Organic-Computing-Systemen, weil bereits der Straßenverkehr selbst diverse „organische“ Eigenschaften aufweist. Eine wichtige Fragestellung ist, wie lokal wirkende Regeln aussehen, die zu positiven Beeinflussungsmaßnahmen führen und den Verkehrsfluss verbessern. Die im Folgenden vorgestellten Organic-Computing-Konzepte sollen helfen solche Regeln zu finden und systematisch im Rahmen des *AutoNomos*-Systems anzuwenden.

3.2 Organic-Computing-Konzepte für dezentrale Anwendungen

Das *AutoNomos*-System baut auf drei Basiskonzepten auf, die hier vorgestellt werden. Die programmiertechnische Umsetzung wird im nachfolgenden Kapitel 4 behandelt. Die Basiskonzepte wurden bereits in Verbindung mit ihrer Umsetzung in [184] publiziert.

Betrachtet man den Straßenverkehr als komplexes selbstorganisierendes System, ergeben sich zwei Sichtweisen. Aus Sicht der Verkehrsteilnehmer besteht der Straßenverkehr aus den Fahrzeugen in unmittelbarer Umgebung, mit denen sie verschiedenartig interagieren, wobei Verkehrsstrukturen entstehen, die sich über die Zeit verändern. Bei diesen Vorgängen treffen Verkehrsteilnehmer manchmal nur kurz aufeinander, manchmal befinden sie sich aber auch über einen längeren Zeitraum in unmittelbarer Nähe. Betrachtet man darüber hinaus die aus der Interaktion einzelner Verkehrsteilnehmer entstandenen Verkehrsstrukturen, so kann man diese ebenfalls als eigenständige Einheiten auffassen. Ein Stau zum Beispiel kann über Stunden an einem Ort bestehen oder sich sogar rückwärts bewegen, während sich die Menge der Fahrzeuge, die den Stau bilden, ständig verändert und über die Zeit sogar vollständig ausgetauscht werden kann.

Die im Folgenden vorgestellten Basiskonzepte zielen darauf ab, Verkehrsstrukturen losgelöst von einzelnen Fahrzeugen zu erkennen, über die Zeit zu verfolgen und schließlich beeinflussend einzugreifen.

Zur dezentralen Datenhaltung wird die neuartige Datenstruktur einer *Hovering Data*

Clouds (HDCs), zu deutsch „schwebende Datenwolke“, verwendet. HDCs erlauben eine Abstraktion der Daten von ihren physikalischen Trägern, d. h. in diesem Fall den Fahrzeugen; dafür können HDCs von Fahrzeug zu Fahrzeug migrieren, um am Ursprung der in ihnen enthaltenen Daten zu bleiben bzw. ihrem Datenursprung zu folgen. Auch der Inhalt der HDCs ist nicht statisch, sondern kann fortlaufend von beteiligten Fahrzeugen mit neu gewonnenen Daten angereichert werden. Die Umsetzbarkeit des HDC-Konzepts wurde bereits in [188] veröffentlicht.

HDCs entstehen selbstorganisierend aufgrund lokal wirkender Regeln. Im ersten Schritt beschreiben sie Phänomene, die räumlich zusammenhängend sind, z. B. eine Gefahrenstelle, einen Stauanfang oder ein Stauende. Um eine räumlich ausgedehnte Verkehrsstruktur erfassen zu können, werden die Daten mehrerer HDCs von verschiedenen Orten zusammengefasst; dabei entstehen höherwertige Informationen, die wiederum in einer HDC gespeichert werden. Die Gesamtheit dieser hierarchisch organisierten HDCs, die zur Erkennung einer Verkehrsstruktur beitragen, sowie ihre zeitliche Entwicklung werden als *Organic Information Complex* (OIC) bezeichnet.

Durch die neu entwickelte Form der Daten-*Aggregation* im Netz wird eine Art „Fischaugenperspektive“ erreicht. Je näher man einer Verkehrsstruktur kommt, desto detaillierter werden die zugehörigen Informationen. So wird sichergestellt, dass das System auch in großen Netzen gut skaliert.

Nachdem Verkehrsstrukturen und auch die Verkehrslage mit Hilfe von HDCs und OICs erfasst sind, werden adaptive verteilte Strategien (engl. Adaptable Distributed Strategy, ADS) eingesetzt, um die jeweilige Verkehrslage zu verbessern. Optimierungskriterien sind beispielsweise der Energieverbrauch oder die Reisezeit.

3.2.1 Hovering Data Clouds

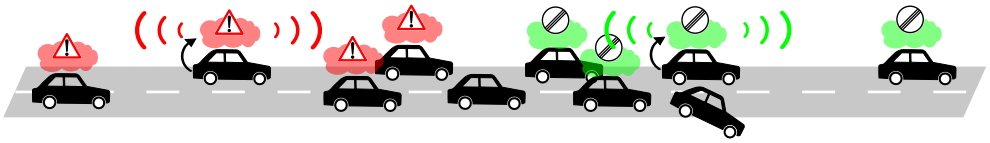
Die von Fahrzeugen erfassten Daten resultieren zumeist aus Phänomenen, die sich in ihrer unmittelbaren Umgebung abspielen. Da sich die Fahrzeuge als Träger der Daten, aber unabhängig von den beobachteten Phänomenen bewegen, ist eine feste Bindung der Daten an einzelne Fahrzeuge nicht sinnvoll. Stattdessen bilden die von allen beteiligten Fahrzeugen erfassten Daten, die dasselbe Phänomen beschreiben, sogenannte *Hovering Data Clouds* (HDCs).

Lebenszyklus

Der Lebenszyklus einer HDC gestaltet sich wie folgt: Ein Fahrzeug erkennt mittels lokaler Sensoren initial ein bestimmtes Phänomen, das von einer HDC beobachtet werden soll. Abbildung 3.2a zeigt, wie eine Unfallstelle erkannt wird; dessen Position speichert das Fahzeug in einer neuen HDC zuerst einmal lokal ab. Der Dateninhalt der HDC wird vom entsprechenden Fahrzeug als Broadcast-Nachricht gesendet; alle benachbarten

(a) Ein erstes Fahrzeug erkennt mittels lokaler Sensoren die Unfallstelle und speichert seine Position in einer neuen HDC.

(b) Die HDC wurde an nachfolgende Fahrzeuge weitergereicht. Durch die Daten, die jedes Fahrzeug der HDC hinzufügt, wird die Position des Stauendes verifiziert. Zudem wird auch die Position des Stauanfangs in einer HDC gespeichert.



(c) Die Stauende-HDC migriert von Fahrzeug zu Fahrzeug, um das Stauende weiter zu beobachten. Jedes Fahrzeug das staubedingt bremst, integriert neue Daten in die HDC.

Abbildung 3.2: Lebenszyklus einer HDC.

Fahrzeuge werten den Inhalt der empfangenen Nachricht aus, stellen fest, dass es sich um eine ihnen unbekannte HDC handelt, und speichern sie zunächst einmal lokal, wie in Abbildung 3.2b illustriert. Das nächste Fahrzeug, das staubedingt bremst, findet somit bereits eine HDC im lokalen Speicher vor, die mit der eigenen Positionsbestimmung des Stauendes *korreliert*. Die lokal erfasste, aktuelle Position des Stauendes wird daraufhin in die bereits in der HDC enthaltenen Daten *integriert*. Im einfachsten Fall wird die Position des Stauendes auf den aktuelleren Messwert gesetzt. Andererseits kann auch über eine Historie der letzten gemessenen Stauende-Positionen mit den zugehörigen Messzeitpunkten die erwartete zukünftige Position des Stauendes sowohl bei der *Korrelation* als auch bei der *Integration* neuer Messwerte berücksichtigt werden. Die so veränderte HDC wird wiederum per Broadcast an die benachbarten Fahrzeuge gesendet, welche die empfangenen Daten auf die gleiche Weise mit vorhandenen HDCs *korrelieren* und gegebenenfalls *integrieren*.

Eine besondere Rolle spielen die Funktionen zum *Korrelieren* und *Integrieren* neuer Daten. Während die *Korrelations*-Funktion die Zusammengehörigkeit neuer Daten mit den bereits lokal gespeicherten HDCs bewertet, sorgt die *Integrations*-Funktion für die Verarbeitung der neuen Daten, infolge derer sich eine lokal gespeicherte HDC ent-

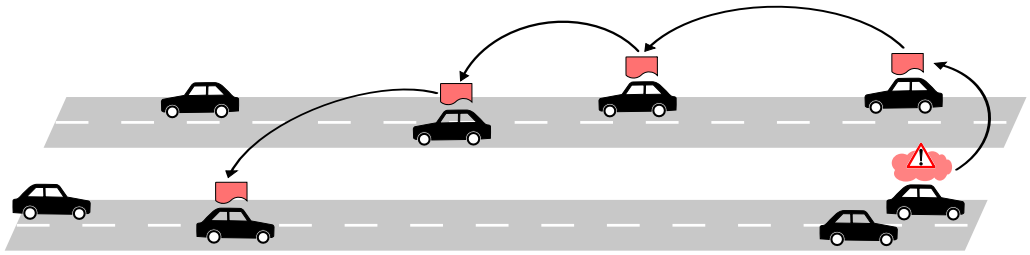


Abbildung 3.3: Überleben einer HDC durch Zwischenspeicherung in einem Datenelement.

sprechend verändert. Die Realisierung dieser beiden Funktionen ist von der Semantik der Daten abhängig und unterliegt somit dem Anwendungsentwickler. Dieser muss sicherstellen, dass die Daten in von ihm implementierten HDCs durch die fortlaufende *Integration* neuer Daten nicht ständig zwischen Extremwerten hin- und herschwingen. Das heißt, die *Integrations*-Funktion muss stets konvergieren.

Durch die ständige Anreicherung einer HDC mit neu gewonnenen Messdaten entwickelt sich diese nicht nur inhaltlich, sondern folgt auch zwangsläufig ihrem Datenursprung. Die Menge der Fahrzeuge, die den Inhalt einer HDC speichern, unterliegt somit einem ständigen Wandel und kann über die Zeit auch vollständig ausgetauscht werden.

Abbildung 3.2c zeigt, wie sich die Stauende-HDC bewegt, um stets am Ort des beobachteten Phänomens, d. h. dem Stauende, zu bleiben.

Verlässt ein Fahrzeug den Bereich einer HDC, ergeben sich zwei Reaktionsmöglichkeiten: Wenn bereits ein Fahrzeug, das sich näher am Ursprung der HDC befindet, Daten zur betreffenden HDC gesendet hat, dann kann die HDC lokal gelöscht werden, da ihr Weiterbestehen in einem anderen Fahrzeug gesichert ist. Falls aber keine Nachricht von einem näherliegenden Fahrzeug empfangen wurde, muss davon ausgegangen werden, dass sich zurzeit kein Fahrzeug im Bereich der HDC befindet.

Abbildung 3.3 zeigt, wie das sich entfernende Fahrzeug in diesem Fall das *AutoCast*-Protokoll nutzt, um den Inhalt der HDC mithilfe des Gegenverkehrs für eine gewisse Zeit am Datenursprung zu halten (*AutoCast* wird in Kapitel 5 ausführlich vorgestellt). Fahrzeuge, die eine so „konservierte“ HDC empfangen, integrieren sie in ihren lokalen Speicher. Somit kann eine HDC auch die kurzzeitige Abwesenheit von Fahrzeugen in ihrem Gebiet „überleben“.

Extraktion von Meldungen

Zusätzlich zur beschriebenen „internen“ Kommunikation, die der Fortentwicklung der HDC dient, können auch (Warn-)Meldungen von HDCs ausgehen, die je nach Inhalt in einem mehr oder weniger großen Umkreis an alle Fahrzeuge verteilt werden. Auch an

dieser Stelle sei auf Kapitel 5 verwiesen, das sich eingehend mit der zu diesem Zweck nötigen Datenverteilung beschäftigt. Bei der internen Kommunikation ist jedes Fahrzeug im Bereich der HDC in der Lage den Inhalt der HDC zu verändern. Dazu werden Daten jeweils per Broadcast an benachbarte Fahrzeuge weitergegeben und dort in die bestehenden HDCs *integriert*. Beim Verteilen von Daten werden diese nur vom Sender, der sich im Bereich der HDC befindet, entsprechend dem Inhalt der HDC aufbereitet, um daraufhin für eine gewisse Zeit in einer bestimmten geografischen Region unverändert verteilt zu werden.

Damit nicht alle Träger einer HDC unkorelliert solche Meldungen produzieren, ist ein einfacher Abstimmungsprozess vorgeschaltet. Grundsätzlich werden neue Meldungen von den Fahrzeugen generiert, die zuletzt neue Daten zur HDC beigetragen haben. Vor der Generierung einer Meldung sendet das entsprechende Fahrzeug den Status der HDC per Broadcast; dabei setzt es ein Flag, das auf die bevorstehende Generierung einer Meldung hinweist. Wird für eine bestimmte Zeit nach diesem Broadcast kein neuer HDC-Status empfangen, so wird eine Meldung generiert und deren Sendezeitpunkt in der HDC abgelegt.

Abgrenzung

In Sensornetzen wird schon seit Längerem eine datenzentrierte Sichtweise propagiert [53, 144, 160]. Ebenso wie bei der Verwendung von HDCs treten dabei die einzelnen Knoten als Träger der Daten in den Hintergrund. Der Ort, an dem Daten gespeichert werden, wird bei der datenzentrierten Sicht von einer verteilten bzw. geografischen Hash-Tabelle (Distributed Hash Table, DHT bzw. Geographical Hash Table, GHT) bestimmt. Gegenüber dem HDC-Konzept ergeben sich dabei allerdings folgende drei Probleme: Erstens muss jeder potenzielle Bezieher vom Vorhandensein bestimmter Daten im Vorfeld wissen, um deren Position mithilfe einer Hash-Tabelle zu bestimmen. Zweitens müssen alle Knoten im Sensornetz ihre Position einschätzen können, um Daten und Anfragen in die richtige Richtung weiterleiten zu können. Und drittens werden Daten anstatt am Ort ihres Entstehens an zufälligen Orten gespeichert; so ist zwar die Aufgabe der Datenspeicherung gleichmäßig im Netz verteilt, mit zunehmender Netzgröße steigt allerdings auch der Kommunikationsaufwand, um die Daten durch das Netz zu befördern.

3.2.2 Organic Information Complexes

Die im letzten Abschnitt vorgestellten HDCs erlauben die Erfassung und Verfolgung von räumlich begrenzten Verkehrsphänomenen auf einfache und effiziente Weise. Um aus den so gewonnenen Daten auf übergeordnete, räumlich ausgedehnte Verkehrsstrukturen schließen zu können, müssen Daten mehrerer räumlich verteilter HDCs *aggregiert* werden. Wie dieses Ziel mithilfe eines *Organic Information Complexs* (OICs) erreicht

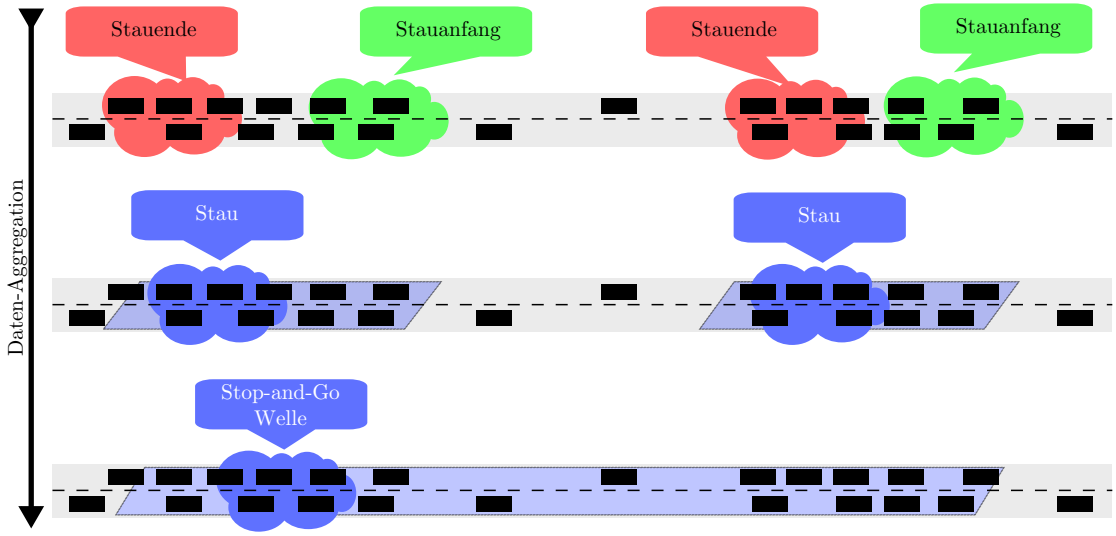


Abbildung 3.4: Organic Information Complex – Daten-Aggregation zur Erkennung eines Verkehrsstaus.

wird, illustriert beispielhaft Abbildung 3.4. Gezeigt wird ein Autobahnabschnitt in drei verschiedenen Stadien der Erkennung eines *Stop-and-Go*-Verkehrsstaus.

Das obere Bild zeigt HDCs, wie sie aufgrund lokaler Messungen der einzelnen Fahrzeuge entstanden sind. *Stauende*-HDCs entstehen bei einer Häufung von korrelierenden Bremsvorgängen und dem Unterschreiten einer bestimmten Grenzgeschwindigkeit. Analog bilden sich *Stauanfang*-HDCs bei korrelierenden Beschleunigungsmanövern. Diese lokal behandelten HDCs verteilen regelmäßig Positionsdaten des entsprechenden Stauanfangs bzw. -endes.

Das Ergebnis der ersten *Aggregation* der Stauanfang- und -ende-HDCs ist im mittleren Teil der Abbildung 3.4 zu sehen. Dabei ist es unerheblich, welche Fahrzeuge zuerst das Vorhandensein zueinander passender HDCs feststellen, die nach ihrer *Aggregation* einen Stau mit seinem Anfang und Ende beschreiben. So entsteht ein OIC, der den gesamten Stau beschreibt und Messdaten enthält, die von einer einzelnen HDC allein nicht erfasst werden können, beispielsweise die Länge des Staus. Die so neu gewonnenen Informationen werden erneut in einer HDC abgelegt, die dort entsteht, wo die von den zueinanderpassenden HDCs ausgesendeten Daten *aggregiert* werden. Die im vorhergehenden beschriebenen *Korrelations-* und *Integrations*-Funktionen von HDCs funktionieren unverändert auch für diese durch Daten-*Aggregation* entstandenen HDCs. Das ist vorteilhaft, da die zugrundeliegenden Daten mindestens von allen Fahrzeugen innerhalb eines Kommunikationsradius zeitgleich detektiert werden. Anstatt mit hohem

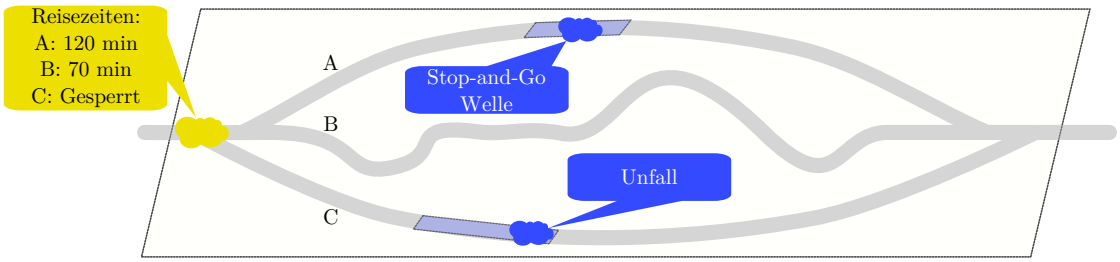


Abbildung 3.5: Organic Information Complex – Ermittlung von Reisezeiten in einem Straßennetz.

Aufwand die Datenkonsistenz zu jedem Zeitpunkt sicherzustellen, sorgt die Konvergenz-Eigenschaft der HDCs für einen Konsens, wobei die Robustheit des gesamten Vorgangs von der redundanten Datenverarbeitung profitiert.

Der Aggregationsprozess setzt sich in hierarchischer Form fort, wie im unteren Teil von Abbildung 3.4 gezeigt. Aus den im vorherigen Schritt erkannten Staus wird aufgrund ihrer Länge und des Abstandes dazwischen auf eine *Stop-and-Go*-Welle geschlossen. Dabei spielt die exakte Stelle letztlich keine Rolle, an welcher die passenden Daten zusammenkommen und ihre *Aggregation* zur Etablierung einer neuen HDC führt. Die *Aggregation* hätte im Beispiel also genauso gut auch im rechten Abschnitt der Strecke stattfinden können. Die Behandlung der Redundanz wird jederzeit durch die *Korrelation* und *Integration* der betreffenden HDCs gewährleistet.

Wenn man ein Straßennetzwerk (vgl. Abbildung 3.5) betrachtet, lässt sich das vorherige Beispiel wie folgt fortführen: Analog zu dessen Beschreibung wurden in Abbildung 3.5 bereits ein Stau und ein Unfall auf den Straßen A bzw. C durch OICs erfasst und beobachtet. Die OICs senden Daten zur jeweiligen Verkehrsbehinderung in Richtung der ankommenden Fahrzeuge.

Sobald diese Daten Kreuzungen im Straßennetz erreichen, an denen eine Umleitung von Verkehrsteilnehmern möglich ist, werden die Daten der einzelnen Verkehrsbehinderungen zu einer voraussichtlichen Reisezeit *aggregiert*. Im gezeigten Beispiel stellt die kurvige Strecke B die zurzeit schnellste Route dar. Die Reisezeiten werden erneut in einer HDC abgelegt, so dass eine fortlaufende Aktualisierung der Routenempfehlung möglich ist.

Dabei sind weder HDCs noch OICs an spezifische Stellen oder ausgewählte Fahrzeuge gebunden. Vielmehr bilden sie sich selbstorganisierend genau dort, wo von der VANET-Anwendung definierte Bedingungen zutreffen. Ebenso ist die *Aggregation* von Daten zu höherwertigen Informationen nicht an bestimmte Stellen im Netz gebunden, sondern findet dort statt, wo passende Daten aufeinandertreffen.

Es obliegt dem Anwendungsentwickler, den Typ der von der jeweiligen Anwendung

benötigten Daten zu bestimmen und Funktionen zur *Korrelation*, *Integration* sowie *Aggregation* auszuwählen. Durch Regeln, die die Häufigkeit sowie die Richtung des Datenversands durch HDCs festlegen, bestimmt der Anwendungsentwickler indirekt den Ort, an dem diese Daten *aggregiert* und die Daten der entsprechenden OICs gehalten werden. So ist es beispielsweise beim Stauszenario vorteilhaft, die *Stauanfang*- und *Stauende*-HDCs am Stauende zu aggregieren, da bei stillstehendem Verkehr die Daten vom Gegenverkehr vom Stauanfang zum Stauende transportiert werden können, während die Multihop-Kommunikation in Fahrtrichtung nicht sichergestellt ist, wenn zu wenige Fahrzeuge mit dem System ausgestattet sind.

3.2.3 Adaptable Distributed Strategies

Zur vollständigen Beschreibung des *AutoNomos*-Systems werden an dieser Stelle Adaptable Distributed Strategies (ADS) eingeführt.

Aufbauend auf der Erfassung von Verkehrsstrukturen mithilfe von HDCs und OICs, ermöglichen adaptive verteilte Strategien (ADS) durch selbstorganisierendes lokales Eingreifen eine Beeinflussung der Verkehrsteilnehmer, so dass ungünstige Verkehrsstrukturen aufgelöst und die Verkehrslage verbessert wird.

Eine genauere Untersuchung und Optimierung dieser verteilten Strategien obliegt den Projektpartnern aus dem *AutoNomos*-Projekt [40], welche bereits ein zum Patent angemeldetes dezentrales Verfahren zur Stauauflösung [44] entwickelt haben. Von besonderem Interesse sind Rückkopplungseffekte, die entstehen, wenn Verkehrsteilnehmer aufgrund von Verkehrsmeldungen ihr ursprüngliches Verhalten ändern und somit mittelbaren Einfluss auf die zugrunde liegende Verkehrsstruktur ausüben. Während im *AutoNomos*-Projekt von einer uneingeschränkten Mitwirkung der einzelnen Verkehrsteilnehmer ausgegangen wird, bearbeitet die Arbeitsgruppe von Prof. Sándor P. Fekete bereits die Auswirkungen, die das egoistische Verhalten der einzelnen Verkehrsteilnehmer mit sich bringt [41].

Als einfaches Beispiel für eine ADS wird in Abschnitt 7.3 eine adaptive Umleitungsstrategie anhand eines Anwendungsfalls eingeführt.

Kapitel 4

AutoNomos-System

Aufbauend auf den im letzten Kapitel vorgestellten Konzepten der *Hovering Data Cloud* (HDC) und des *Organic Information Complex* (OIC) wird in diesem Kapitel deren technische Umsetzung in Form des *AutoNomos*-Systems behandelt, das bereits in [184] publiziert wurde. Zudem wird eine *Programmierschnittstelle* (API) angegeben, mittels derer Anwendungsprogrammierer VANET-Anwendungen unter Verwendung von HDCs und OICs implementieren können.

Die Zielsetzung ist dabei nicht nur allein die Umsetzung der entsprechenden Konzepte, sondern auch die Bereitstellung einer definierten Schnittstelle, welche die Entwicklung modularer und untereinander kompatibler VANET-Anwendungen ermöglicht. Denn wie Abschnitt 2.3.2 zeigt, existieren bereits eine Vielzahl an VANET-Anwendungen, die jedoch fast ausschließlich isoliert betrachtet werden.

Das *AutoNomos*-System weist einige Eigenschaften einer Middleware (vgl. [168, S. 55 ff]) auf und entlastet den Anwendungs-Entwickler von der Erstellung systemnaher Funktionen. Allerdings stellt das *AutoNomos*-Sytem keine Abstraktionsschicht dar, die sich in einem Schichtenmodell anordnen ließe. Vielmehr werden Abläufe und Informationsflüsse vorgegeben. Daher wird das *AutoNomos*-System nicht als Middleware bezeichnet.

Im nachfolgenden Abschnitt werden die Eigenschaften gängiger Middleware-Ansätze für Ad-hoc-Netze betrachtet, deren Funktionalität sich teilweise im *AutoNomos*-System wiederfindet. Dem folgt in Abschnitt 4.2 die Formulierung der Entwurfsziele, die mit dem *AutoNomos*-System angestrebt werden. In Abschnitt 4.3 wird die Funktionsweise des *AutoNomos*-Systems anhand des Informationsflusses aus Sicht eines Fahrzeugs beschrieben. Abgeschlossen wird dieses Kapitel mit der Definition und Erläuterung der API zur Implementierung von Anwendungen im *AutoNomos*-System.

4.1 Verwandte Arbeiten

Im Forschungsbereich der Sensornetze und der *Mobile Ad-hoc Networks* (MANETs) sind bereits eine Vielzahl an Middleware-Ansätzen entwickelt worden.

Unter all den Middleware-Ansätzen, die sich in der Literatur finden lassen, ist jedoch keiner, der Konzepte enthält, die sich mit denen der HDC und OIC vergleichen lassen.

Allerdings gibt es einige Überschneidungspunkte bei der Funktionalität.

Die Übersicht verwandter Arbeiten stellt daher spezifische Merkmale von verwandten Middleware-Ansätzen dar und nennt jeweils einen Vertreter, der das entsprechende Merkmal umsetzt. Dabei wird kein Anspruch auf Vollständigkeit erhoben. Die folgende Auflistung stützt sich maßgeblich auf die von Römer in [150] vorgestellte Übersicht.

Ereignisse: Ausgehend von der Tatsache, dass MANETs oft zur Überwachung ihrer Umgebung eingesetzt werden, stellt die ereignisorientierte Programmierung ein naheliegendes Konzept dar. Klassifizierung von Messdaten und eine Reaktion auf bestimmte Ereignisse bzw. Phänomene definieren das Verhalten der Knoten (vgl. [100]).

Lokale Regeln: In einem Ad-hoc-Netz steigt der Kommunikationsaufwand proportional zur Entfernung der kommunizierenden Knoten an. Somit sollten alle Aktionen auf lokalen oder in näherer Umgebung vorhandenen Daten basieren (vgl. [49, 170]).

Datenzentriert: Wie schon am Ende von Abschnitt 3.2.1 beschrieben, sind in Sensornetzen die erfassten Daten wichtiger als ihre physikalischen Träger. Zum einen werden Daten an beliebigen Orten im Netz gespeichert, um die Datenmenge gleichmäßig im Netz zu verteilen (vgl. [160]). Zum anderen existieren datenbankorientierte Ansätze, die eine Datenanforderung angelehnt an Abfragesprachen für relationale Datenbanken formulieren (vgl. [109]). In beiden Fällen braucht sich der Anwendungsentwickler nicht um Aspekte der Kommunikation zu kümmern.

Virtualisierung: Virtualisierungstechniken sind aus zweierlei Gründen interessant: Einerseits kann über eine Virtualisierungsschicht die Heterogenität einzelner Knoten im Ad-hoc-Netz vor dem Anwendungsentwickler verborgen werden (vgl. [55]). Andererseits lässt sich eine Anwendung als mobiler Agent umsetzen, der sich selbsttätig von einem Knoten zum nächsten kopiert, um seine Aufgabe zu erfüllen. Tatsächlich hat die in [36] vorgestellte *Autonomous Virtual Mobile Node* (AVMN) Ähnlichkeiten zum Konzept der HDC. Nach einer in [43] vorgestellten Stauerkennung mit auf AVMNs basierenden HDCs erwies sich der AVMN-Ansatz jedoch als schwierig erweiterbar und wurde nicht weiter verfolgt.

Abstraktion der Kommunikation: Nahezu alle Middleware-Ansätze enthalten eine unterschiedlich stark ausgeprägte Abstraktion der Kommunikation. Beispielsweise stellt Pfisterer u. a. in [139] einen Ansatz vor, der aus einer abstrakten datentypzentrierten Beschreibung die nötigen Kommunikationsmodule ableitet.

Für weitere Details sei auf die ausführlichen Übersichten in [69, 118, 150] verwiesen. Die genannten Merkmale sind auch in VANETs vorteilhaft. So finden sich einige in den im nächsten Abschnitt beschriebenen Entwurfszielen wieder.

4.2 Entwurfsziele

Bei der Entwicklung des *AutoNomos*-Systems wurde die Umsetzung der im nachfolgenden beschriebenen Systemeigenschaften bzw. Funktionalitäten angestrebt. Diese sind auf eine Umsetzung der im letzten Kapitel vorgestellten Organic-Computing-Konzepte der HDC und OIC hin ausgelegt. Teilweise decken sie sich mit den vorgenannten Eigenschaften von Middleware-Lösungen.

Lokale Datenhaltung / Lokale Regeln: VANET-Anwendungen bauen auf lokal gemessenen Daten auf, beispielsweise der Geschwindigkeit oder Verkehrsdichte. Anstatt diese lokalen Daten an eine zentrale Instanz zur weiteren Bearbeitung weiterzuleiten, bestimmen lokal wirkende Regeln das weitere Vorgehen. Da die Anwendbarkeit bestimmter Regeln auf Messdaten beruht und somit nicht zwangsläufig an feste Zeiten gebunden ist, haben VANET-Anwendungen eine ereignisbasierte Komponente.

Fusionierung von Daten: Die Qualität und Zuverlässigkeit von Sensordaten kann durch Sensordatenfusion von mehreren lokalen Sensoren erhöht werden [102]. Aufgrund der Mobilität der einzelnen Fahrzeuge ist es zusätzlich nötig, Daten auch fahrzeugübergreifend zu fusionieren. So ist z. B. eine vereiste Brücke für die Sensoren eines einzelnen Fahrzeugs nur für einen kurzen Zeitraum sichtbar. Durch die Verwendung einer HDC können Daten über einen längeren Zeitraum am Ort der Messung gehalten werden, wobei neue Messungen jederzeit in die bestehenden Daten integriert werden können.

Aggregation von Daten: Die Aggregation von Daten ist aus zweierlei Gründen nötig. Zum einen lassen sich aus lokal mittels HDCs gewonnenen Daten durch hierarchische Aggregation komplexe Verkehrsstrukturen ableiten. Zum anderen sollte der Detailgrad von Daten im Sinne einer *Fischaugenperspektive* reduziert werden, je weiter sich die Daten von ihrem Ursprung entfernen. Beide Aggregations-Gründe sind inhärent miteinander verbunden. So möchte ein Verkehrsteilnehmer, der sich einen Kilometer hinter einem Stauende befindet, durchaus zeitnah über die exakte Position und Entwicklung des Stauendes informiert werden, während bei einem Stau in mehr als 50 km Entfernung eher die durch den Stau erzeugte Verzögerung sowie mögliche Umleitungsstrecken interessieren.

Selbst-Adaption: Die Kommunikationsschicht des *AutoNomos*-Systems muss sich ohne externes Eingreifen selbständig auf die jeweilige Netztopologie einstellen. Zu Zeiten der Markteinführung werden nur sehr wenige Fahrzeuge am VANET teilnehmen, wogegen bei einem hohen Ausstattungsgrad in einem Stau theoretisch

mehrere hundert Fahrzeuge direkt miteinander kommunizieren können. Zudem variiert der Kommunikationsaufwand je nach vorherrschender Verkehrslage; Priorisierung wichtiger Daten sowie eine adaptive Aggregation zur Datenreduktion helfen Engpässe zu vermeiden.

Robustheit: Die Funktionsfähigkeit von VANET-Anwendungen kann von verschiedenen äußeren Faktoren beeinflusst werden. Bedingt durch die Unzuverlässigkeit der drahtlosen Kommunikation können Nachrichten beim Versand verloren gehen. Hinzu kommt, dass den Fahrzeugen aufgrund der teils hohen relativen Geschwindigkeiten unter Umständen nur wenige Sekunden zum Datenaustausch verbleiben. Zur Erhöhung der Robustheit des Gesamtsystems verzichtet das *AutoNomos*-System auf die Verteilung spezieller Aufgaben an einzelne Fahrzeuge, die fortlaufend verwaltet werden müssten. Dies führt zu einem zustandslosen System, in dem Aufgaben nicht von vorher definierten Fahrzeugen ausgeführt werden; stattdessen wird die Aufgabenverteilung mittels Zufallsvariablen gesteuert. Ebenso werden Daten nicht fest an bestimmte Fahrzeuge gebunden, sondern in Form von HDCs von allen Fahrzeugen innerhalb eines Kommunikationsradius redundant gehalten; dabei hat jedes Fahrzeug die Möglichkeit neue Daten beizusteuern, wodurch sich der HDC-Inhalt über die Zeit entwickeln kann, ohne von der Funktionsfähigkeit ausgewiesener Fahrzeuge abhängig zu sein.

Skalierbarkeit. Sollten sich VANET-Anwendungen eines Tages im Massenmarkt etablieren, so stellt die Skalierbarkeit ein wesentliches Kriterium dar. Laut Kraftfahrt-Bundesamt sind Anfang 2009 allein in Deutschland 49,6 Mio. Kraftfahrzeuge zugelassen [92]. In Sensornetzen wird oft das *Publish-Subscribe*-Prinzip angewendet, bei dem Datensenken ihr Interesse an Daten eines bestimmten Typs im gesamten Netz bekunden und so die Daten zu sich „heranziehen“ (Pull-Mechanismus). Im Gegensatz zu einem Sensornetz, in dem es potenziell nur eine begrenzte Anzahl an Datensenken gibt, ist im VANET jedes Fahrzeug an Daten interessiert; bei individuellen Datenanforderungen jedes einzelnen Fahrzeugs steigt der Aufwand mindestens linear mit der Anzahl der Fahrzeuge im Netz. Um Skalierbarkeit zu gewährleisten, müssen Daten automatisch dorthin geleitet werden, wo sich potenzielle Empfänger aufhalten, die Interesse an den jeweiligen Daten haben (Push-Mechanismus). Skalierbarkeit spielt jedoch nicht nur bei der Entwicklung des *AutoNomos*-Systems eine zentrale Rolle, sondern muss auch bei der Entwicklung jeder einzelnen VANET-Anwendung berücksichtigt werden.

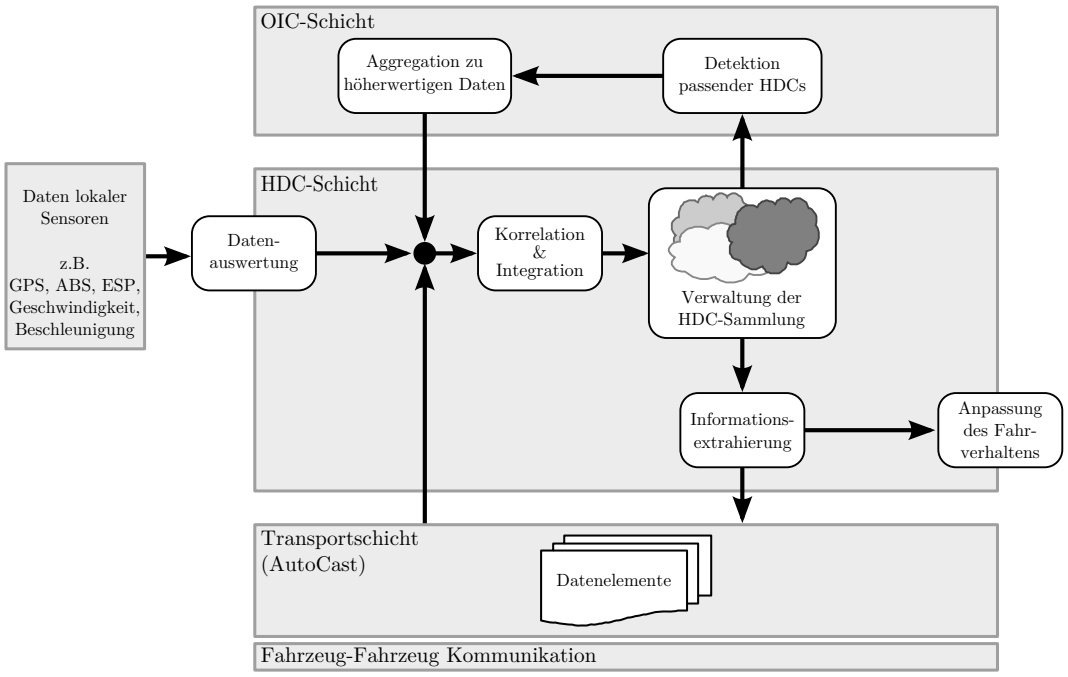


Abbildung 4.1: Informationsfluss innerhalb des *AutoNomos*-Systems.

4.3 Architektur

Die Umsetzung der Konzepte aus Abschnitt 3.2 unter den Vorgaben aus dem vorherigen Abschnitt zeigt Abbildung 4.1. Abgebildet sind die Informationsflüsse, die innerhalb eines jeden Fahrzeugs zwischen den einzelnen Funktionseinheiten auftreten, welche hier als abgerundete Rechtecke dargestellt sind. Die gesamte Softwarearchitektur des *AutoNomos*-Systems ist in drei Schichten unterteilt, die im Folgenden beginnend bei der untersten Schicht erläutert werden.

AutoNomos basiert auf einer Transportschicht, die nicht nur das Senden und Empfangen von Broadcast-Nachrichten erlaubt, sondern auch die eigenständige Verteilung von Daten[1] innerhalb eines bestimmten geografischen Gebiets als Dienst anbietet. Die Transportschicht wiederum setzt einen Broadcast-fähigen Funkstandard voraus, der in Form des Standards IEEE 802.11p [72] als gegeben angenommen wird. Kapitel 5 beschäftigt sich ausführlich mit dem *AutoCast*-Protokoll, das im Rahmen dieser Arbeit für die Datenverteilung entwickelt wurde.

Die *HDC-Schicht* setzt auf den von der Transportschicht bereitgestellten Diensten

[1] Daten, die von der Transportschicht verteilt werden, werden im Folgenden als *Datenelemente* bezeichnet (vgl. Kapitel 5).

zum Verteilen von Datenelementen, sowie zum Senden und Empfangen von Broadcast-Nachrichten auf. Sie ist verantwortlich für die Speicherung und Pflege von Daten in Form von HDCs. Eine HDC ist dabei nicht einem Fahrzeug fest zugewiesen (vgl. Abschnitt 3.2.1), sondern wird, wie in Abbildung 3.2 angedeutet, von allen Fahrzeugen innerhalb eines geografischen Gebiets gehalten. Jedes der beteiligten Fahrzeuge besitzt eine lokale HDC-Instanz, die den vollständigen Datenbestand der HDC enthält. Das *AutoNomos*-System hat allerdings nicht den Anspruch, Daten innerhalb einer HDC stets konsistent zu halten, sondern tauscht regelmäßig Daten zwischen benachbarten Fahrzeugen aus, so dass die zu einer HDC gehörenden Daten in den lokalen HDC-Instanzen aller Fahrzeuge konvergieren.

Jedes Fahrzeug speichert die lokal bekannten HDC-Instanzen in einer *HDC-Sammlung*, die in ihrer Gesamtheit das lokale Wissen über Verkehrsstrukturen in der Umgebung sowie aggregierte Verkehrsdaten aus dem weiteren Umfeld enthalten.

Zur Konvergenz einer HDC wird ein „verteilte Funktion“ verwendet: Regelmäßig extrahiert die Funktionseinheit *Informationsextrahierung* Datenelemente aus den HDC-Instanzen, und sendet diese über die Transportschicht an benachbarte Fahrzeuge. Auf Seite der Empfänger werden die Datenelemente der Funktionseinheit *Korrelation & Integration* zugeführt. Die *Korrelations*-Funktion durchsucht die lokale HDC-Sammlung und liefert – falls vorhanden – die in Bezug auf das beschriebene Phänomen am besten passende HDC-Instanz aus. Dafür kommen Parameter der beobachteten Verkehrsstruktur, wie die Position und deren zeitliche Veränderung, in Betracht. Falls die *Korrelations*-Funktion eine HDC-Instanz zurückliefert, sorgt die *Integrations*-Funktion für die Einarbeitung der im Datenelement enthaltenen Daten in die betreffende HDC-Instanz; anderenfalls wird aus dem empfangenen Datenelement eine neue HDC-Instanz lokal angelegt. Die Kommunikation innerhalb einer verteilt gespeicherten HDC wird nachfolgend als *interne Kommunikation einer HDC* bezeichnet.

Verlässt ein Fahrzeug das Gebiet einer HDC, so wird die entsprechende HDC-Instanz aus der lokalen HDC-Sammlung entfernt. Vorher wird allerdings noch überprüft, ob bereits Daten in die lokale HDC-Instanz integriert wurden, die von einem Fahrzeug stammen, das sich näher am Ursprung der Daten befindet. Ist dies der Fall, kann die lokale HDC-Instanz gelöscht werden. Ansonsten wird die HDC-Instanz vor dem Löschen der Funktionseinheit *Informationsextrahierung* übergeben, welche die HDC-Instanz in ein Datenelement verpackt. Die Transportschicht hält dieses Datenelement für eine gewisse Zeit am letzten bekannten Datenursprung und ermöglicht ein „Überleben“ der HDC, auch wenn sich kurzzeitig keine Fahrzeuge in ihrem Bereich aufhalten.

Die Transportschicht macht sich dabei das besondere Bewegungsmuster der Fahrzeuge im Straßenverkehr zunutze. So können Daten beispielsweise an einem Ort gehalten werden, indem sie immer wieder an ein in entgegengesetzte Richtung fahrendes Fahrzeug weitergegeben werden. Um das Verfahren zu vereinfachen, wird die Transportschicht angewiesen, das Datenelement, welches die HDC repräsentiert, an alle Fahrzeuge in einem

Verbreitungsgebiet zu verteilen, dessen Größe je nach Wichtigkeit der HDC variieren kann.

Zudem wird die Funktionseinheit *Informationsextrahierung* bei Änderungen innerhalb der HDC-Sammlung aktiv. Wie bereits beschrieben, extrahiert sie Informationen aus den HDC-Instanzen, die entweder für Empfehlungen zur *Anpassung des Fahrverhaltens* genutzt werden, oder als Datenelemente an die Transportschicht gegeben werden.

Es gibt drei Gründe für das Versenden von Datenelementen:

- Zur Fortentwicklung einer HDC mittels *Korrelation & Integration* ist die regelmäßige *interne Kommunikation* nötig.
- Die im vorherigen Absatz beschriebene „Überlebensstrategie“ gibt die letzte HDC-Instanz einer HDC an die Transportschicht ab, sobald sich kein Fahrzeug mehr in ihrem Gebiet befindet.
- Des Weiteren werden Datenelemente für die *externe Kommunikation* generiert, d. h. für die Kommunikation verschiedener HDCs untereinander und folglich zur Bildung von OICs. Auch das Senden von Warnmeldungen an ankommende Fahrzeuge fällt unter die *externe Kommunikation*.

Um eine Beschreibung von Phänomenen aus der realen Welt mittels HDCs zu ermöglichen, gibt es in der *HDC-Schicht* die Funktionseinheit *Datenauswertung*, die als Schnittstelle für sensorisch erfasste Daten dient. Im Rahmen der *Datenauswertung* werden die von Sensoren gelieferten Rohdaten vorgefiltert, so dass nur relevante Messdaten in Datenelemente verpackt werden und der *Korrelation & Integration* in gleicher Weise wie Datenelemente aus der Transportschicht zugeführt werden.

Mit den bis hierher beschriebenen Funktionseinheiten ist das *AutoNomos*-System bereits in der Lage, räumlich begrenzte Verkehrsstrukturen mithilfe von HDCs zu erfassen. Es fehlt allerdings noch die hierarchische Aggregation von HDCs, die zur Bildung von Organic Information Complexes (OIC) führt.

Diese Aufgabe übernimmt die *OIC-Schicht*. Sie enthält eine Funktionseinheit zur *Detektion passender HDCs*, welche die lokale HDC-Sammlung observiert und zusammenpassende HDCs ausfindig macht, um sie an die nächste Funktionseinheit weiterzureichen, die eine *Aggregation zu höherwertigen Daten* vornimmt. So kann beispielsweise aus zwei HDCs, die einen Stauanfang und ein Stauende beschreiben, ein Stau mit der aggregierten Information „Staulänge“ abgeleitet werden, falls die HDCs keine weiteren Stauanfänge bzw. -enden zwischen sich haben (vgl. Abschnitt 3.2.2). Die so aggregierten Daten werden wiederum in ein Datenelement verpackt und ebenso wie Datenelemente, die von der Transportschicht oder der Datenauswertung lokaler Sensoren herrühren, zur *Korrelation & Integration* weitergeleitet. Somit erzeugen auch aggregierte Daten HDCs mit allen oben beschriebenen Möglichkeiten.

An dieser Stelle sei angemerkt, dass zwar der Informationsfluss durch das *AutoNomos*-System festgelegt ist, der Anwendungsentwickler jedoch das Verhalten der einzelnen Funktionseinheiten festlegen muss.

Die Entwurfsziele aus Abschnitt 4.2 werden durch das *AutoNomos*-System vollständig umgesetzt. Der Einsatz von HDCs für die Speicherung von Daten erlaubt die Erstellung lokal wirkender Regeln, die auf den HDC-Daten basieren. HDCs erlauben zudem die Fusion von Daten aller beteiligten Fahrzeuge. Eine Aggregation von Daten erfolgt über die Bildung von OICs. Das *AutoCast*-Protokoll, das in Kapitel 5 vorgestellt wird, adaptiert sich selbständig an die gegebene Netztopologie. Eine hohe Robustheit wird erreicht durch die redundante Datenhaltung innerhalb der HDCs und die Vermeidung einer festen Aufgabenverteilung an einzelne Fahrzeuge. Die Skalierbarkeit wird gewährleistet durch die Umsetzung einer Fischaugenperspektive, d. h. die Daten werden immer weiter aggregiert je weiter sie sich von ihrem Ursprung entfernen. Zudem werden Daten nicht auf spezielle Anfragen hin verschickt, sondern werden proaktiv an potenzielle Empfänger verteilt.

4.4 Programmierschnittstelle

Um eine strukturierte Entwicklung von VANET-Anwendungen aufbauend auf dem *AutoNomos*-System zu ermöglichen, wurde eine entsprechende Programmierschnittstelle entworfen. Diese ist hier in Form einer objektorientierten Klassenstruktur gegeben, welche die Funktionseinheiten des im letzten Abschnitt vorgestellten Informationsfluss-Modells umsetzt. Es wird an dieser Stelle nicht der Anspruch erhoben, eine vollständige Spezifikation und technische Dokumentation zu liefern. Stattdessen werden die wichtigsten Schnittstellen beschrieben und die Zusammenhänge in der Objektstruktur erläutert.

Daher zeigt Abbildung 4.2 nur ein vereinfachtes Klassendiagramm des *AutoNomos*-Systems in der *Unified Modeling Language* (UML) 2.0. Dieses baut auf einem Betriebssystem auf und wird von VANET-Anwendungen genutzt. Die Beschreibung beschränkt sich auf die wichtigen, von außen zugänglichen Schnittstellen und Datenstrukturen.

Betriebssystem

Die Basis für die Implementierung des *AutoNomos*-Systems stellt ein übliches Betriebssystem dar, das auf „echter“ Hardware oder innerhalb einer Simulationsumgebung läuft und essenzielle Funktionalität zur Verfügung stellt. Als Hardware kommen beispielsweise eingebettete Systeme infrage. Ein modulares Betriebssystem für Sensornetze, welches zudem einen Entwicklungsprozess basierend auf einer Simulationsumgebung erlaubt, stellt *iSense* [25, 138] dar. Im Gegensatz zu Sensornetzen gibt es in Kraftfahrzeugen keine Einschränkung durch die Energieversorgung, so dass auch leistungsstärkere eingebettete Systeme zum Einsatz kommen können.

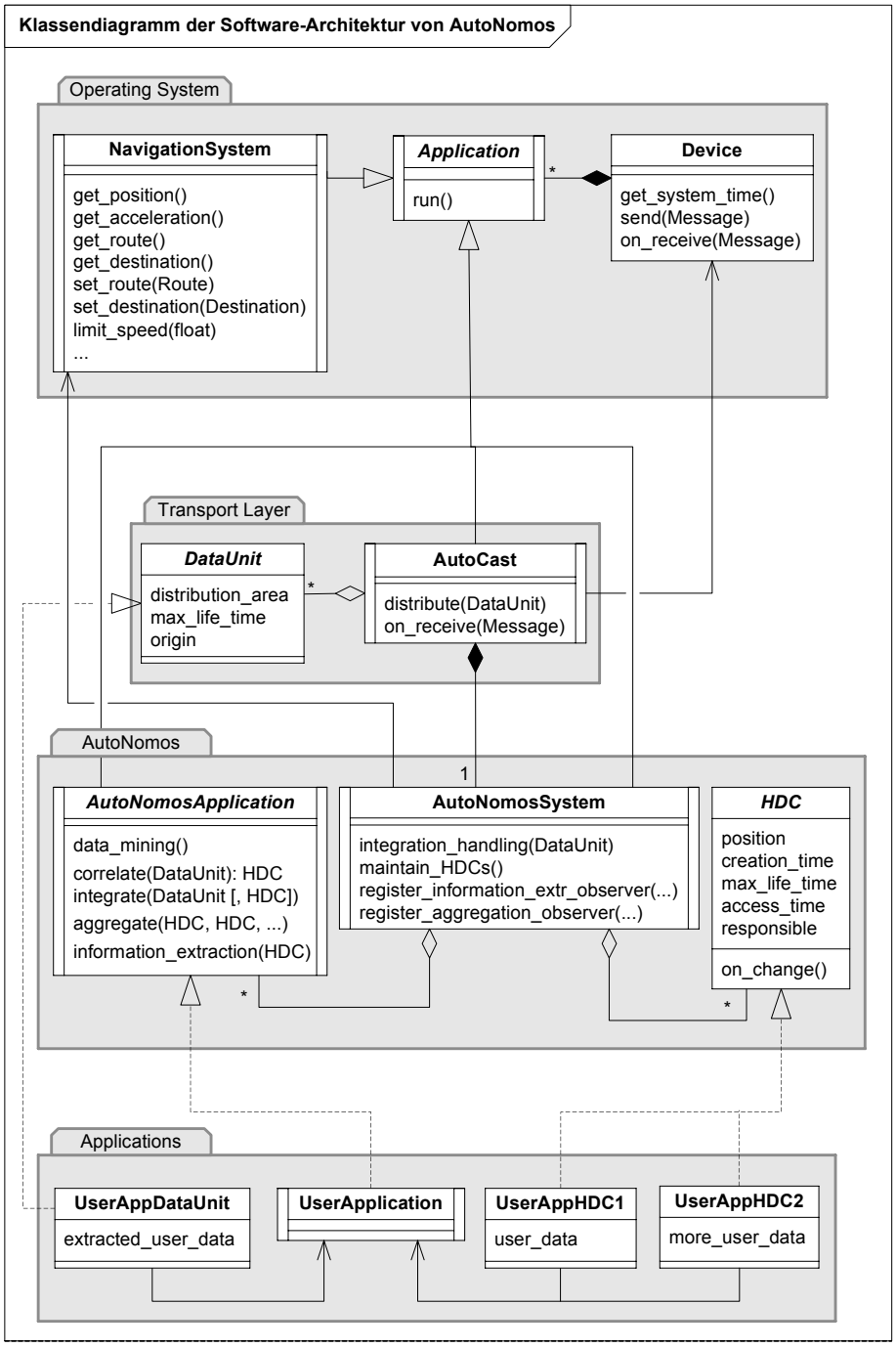


Abbildung 4.2: UML-Klassendiagramm des *AutoNomos*-Systems.

Die mindestens erforderlichen Betriebssystem-Schnittstellen sind in Abbildung 4.2 im oberen Block (*Operating System*) dargestellt. So werden neben dem Zugriff auf eine Echtzeituhr (`get_system_time()`) Methoden zum Senden (`send(`*`Message`*`)`) und Empfangen (`on_receive(`*`Message`*`)`) von Nachrichten über eine Funkschnittstelle vorausgesetzt. Das Empfangen von Nachrichten sollte über eine Callback-Funktion realisierbar sein. Das Betriebssystem sollte multithreading-fähig sein, so dass mehrere *Application*-Objekte parallel ihre `run()`-Methode ausführen können.

Für das angestrebte Einsatzfeld „Straßenverkehr“ wird das Vorhandensein der Anwendung *NavigationSystem* vorausgesetzt, die jederzeit Zugriff auf die eigene Position und auf die gegebenenfalls aktuell geplante Route erlaubt. Daneben sind hier auch Methoden zur Beeinflussung des Fahrverhaltens angegeben, z. B. `set_route(`*`Route`*`)` oder `limit_speed(`*`float`*`)`. Während diese Methoden in einer Simulationsumgebung das Fahrverhalten direkt beeinflussen können, sind in einem realen System entsprechende Meldungen auf der Bedienoberfläche des Navigationssystems denkbar.

Transportschicht

Die Transportschicht (*Transport Layer*) stellt ein weiteres Paket im Klassendiagramm dar. Sie bietet über die Klasse *AutoCast* Methoden zum Verteilen und Empfangen von Datenelementen (engl. data units) an.

Alle Datenelemente sind abgeleitet von der abstrakten Klasse *DataUnit* und enthalten neben den Nutzdaten immer ein Verbreitungsgebiet (*distribution_area*), eine Lebenszeit (*max_life_time*) sowie den Ursprung der Daten (*origin*). Der Datenursprung umfasst eine geografische Position, eine (Pseudo-)ID des Absenders und einen Verweis auf die zugehörige Anwendung. Durch entsprechendes Setzen dieser Variablen wird ein Datenelement entweder nur einmalig per Broadcast gesendet (*interne Kommunikation*), oder über ein größeres Gebiet verteilt (*externe Kommunikation*).

Die Arbeitsweise von *AutoCast* wird im nachfolgenden Kapitel 5 detailliert beschrieben.

AutoNomos

Der Kern des *AutoNomos*-Systems ist im Paket *AutoNomos* zusammengefasst und wird hier in seiner Minimalform bestehend aus nur drei Klassen vorgestellt. Die Klasse *AutoNomosSystem* stellt die zentrale Komponente des *AutoNomos*-Systems dar. Die Klasse *AutoNomosApplication* definiert Schnittstellen, die jede VANET-Anwendung nutzen kann bzw. umsetzen muss. So wird die Funktionalität der einzelnen Funktionseinheiten aus Abbildung 4.1 in einer Reihe von Methoden umgesetzt. Alle Daten des *AutoNomos*-Systems werden in Instanzen von Klassen gehalten, die von der Klasse *HDC* abgeleitet sind.

Die Klasse *AutoNomosSystem* hat Zugriff auf ein *AutoCast*-Objekt, an das einerseits Datenelemente übergeben werden, die von den Anwendungen gesendet werden. Für jedes empfangenene Datenelement wird andererseits die Methode `integration_handling (DataUnit)` in der Klasse *AutoNomosSystem* aufgerufen, welche die eintreffenden Datenelemente in die lokale HDC-Sammlung *integriert*.

Dazu wird zunächst die korrespondierende VANET-Anwendung aus der *DataUnit* ausgelesen. Die entsprechende *AutoNomosApplication* ruft `correlate(DataUnit)` auf, die – falls vorhanden – eine passende HDC aus der lokalen Sammlung heraussucht. Nachfolgend wird das Datenelement von der Methode `integrate(DataUnit [, HDC])` in das bestehende *HDC*-Objekt integriert. Sollte in der lokalen HDC-Sammlung keine passende HDC enthalten sein, so wird `integrate` ohne den Parameter *HDC* aufgerufen und so eine neue lokale HDC erzeugt.

Des Weiteren enthält die Klasse *AutoNomosSystem* die lokale *HDC-Sammlung*, welche durch periodischen Aufruf der Methode `maintain_HDCs()` aufgeräumt wird. Entsprechend der Beschreibung im vorherigen Abschnitt werden veraltete und zu weit entfernte HDCs gelöscht und gegebenenfalls vorher in Form eines Datenelements verteilt. Der Parameter *responsible* eines *HDC*-Objekts zeigt an, ob das Verteilen als Datenelement vor dem Löschen des *HDC*-Objekts nötig ist. Dieser Parameter wird jeweils beim *Integrieren* eines Datenelements entsprechend dem Ursprung des Datenelements gesetzt.

Jede lokale HDC-Instanz wird durch ein *HDC*-Objekt repräsentiert. Enthalten sind die Metadaten aktuelle Position (*position*), die Zeit der lokalen Erstellung der HDC (*creation_time*), die maximale Lebenszeit, die eine HDC ohne die *Integration* neuer Daten überlebt (*max_life_time*) sowie der Zeitpunkt des letzten Zugriffs (*access_time*). Der Zugriff auf diese Metadaten und die Nutzdaten abgeleiteter *HDC*-Klassen wird über entsprechende Methodenaufrufe gekapselt. Von diesen Methoden wird nach jedem schreibenden Zugriff auf eine *HDC* `on_change()` aufgerufen und so der Informationsfluss in Richtung der Funktionseinheiten *Informationsextrahierung* und *Detektion passender HDCs* initiiert.

Um die in Abbildung 4.1 von der HDC-Sammlung ausgehenden Informationsflüsse effizient zu realisieren, registrieren sich die Anwendungen als „Beobachter“ beim *AutoNomosSystem*.

Zur *Informationsextrahierung* registrieren sich die Anwendungen mittels `register_information_extr_observer(...)`; dabei wird als Parameter eine von *HDC* abgeleitete Klasse angegeben. Immer wenn in der HDC-Sammlung ein entsprechendes Objekt verändert wird, wird automatisch die Methode `information_extraction(HDC)` der entsprechenden *AutoNomosApplication* aufgerufen. Basierend auf der ihr übergebenen *HDC*, kann in dieser Methode einerseits Einfluss auf das Fahrverhalten ausgeübt werden. Dazu wird über die Klasse *AutoNomosSystem* auf die Klasse *NavigationSystem* zugegriffen. Andererseits werden Datenelemente erzeugt, die wie im vorherigen Abschnitt beschrieben zur *internen* und *externen Kommunikation der HDC* genutzt werden. Die

Datenelemente werden über die Klasse *AutoNomosSystem* an die Transportschicht weitergegeben.

Mittels `register_aggregation_observer(...)` registrieren sich die Anwendungen, um mehrere *HDC*-Objekte zu höherwertigen HDCs zu aggregieren. Als Parameter für diese Methode wird eine Liste von Klassen angegeben, die von *HDC* abgeleitet sind. Immer wenn ein *HDC*-Objekt aus dieser Liste geändert wird, wird die Methode `aggregate(` *`HDC`*`,` *`HDC`*`, ...)` der jeweiligen Anwendung aufgerufen und ihr werden alle beteiligten *HDC*-Objekte übergeben. Falls die HDCs tatsächlich entsprechend der Anwendungslogik aggregiert werden können, so werden die aggregierten Daten in einem Datenelement verpackt und, wie oben beschrieben, mittels *Korrelation & Integration* in die HDC-Sammlung eingepflegt.

Die Funktionseinheit *Datenauswertung* aus Abbildung 4.1 wird von den Anwendungen in der Methode `data_mining()` implementiert. Diese wird periodisch aufgerufen und erzeugt gegebenenfalls ein Datenelement basierend auf den Daten lokaler Sensoren, welches ebenfalls mittels *Korrelation & Integration* in die HDC-Sammlung eingepflegt wird.

VANET-Anwendungen

Die Bestandteile einer VANET-Anwendung sind in Abbildung 4.2 im Paket *Applications* zusammengefasst. Für jede VANET-Anwendung muss eine Klasse *UserApplication* erstellt werden, die von der zuvor beschriebenen Klasse *AutoNomosApplication* abgeleitet ist. Die in ihr enthaltenen Methoden stellen die Funktionseinheiten aus Abbildung 4.1 dar und müssen anwendungsspezifisch implementiert werden.

Wie oben beschrieben, ruft das *AutoNomos*-System diese Methoden zum richtigen Zeitpunkt auf, um den Informationsfluss entsprechend Abbildung 4.1 zu realisieren.

Die lokal gehaltenen Anwendungsdaten werden in Instanzen von Klassen gespeichert, die von der Klasse *HDC* abgeleitet sind. Daten, die zum Versenden oder zur *Integration* in HDCs vorgesehen sind, werden in Instanzen von Klassen verwaltet, die von *DataUnit* abgeleitet sind.

Die Speicherung darüber hinausgehender Daten liegt im Zuständigkeitsbereich des Anwendungsentwicklers. Dazu gehören beispielsweise Verkehrsmeldungen, die als Datenelement empfangen werden, aber nicht weiter als HDCs verarbeitet werden sollen.

Kapitel 5

Datenverteilung mit AutoCast

Das in den vorangegangenen Kapiteln vorgestellte *AutoNomos*-System nutzt zur Kommunikation den Dienst der *Datenverteilung*. In diesem Zusammenhang wurde im Rahmen dieser Arbeit das Protokoll *AutoCast* [68, 185] entwickelt, welches das Verteilen von Daten als Dienst zur Verfügung stellt.

Der Einsatz des *AutoCast*-Protokolls ist nicht auf die Kommunikation in VANETs beschränkt. Vielmehr eignet es sich zum Verteilen beliebiger Daten in mobilen Ad-hoc-Netzen und kann so in einer Vielzahl verteilter Anwendungen eingesetzt werden. Zur Verallgemeinerung werden daher in diesem Kapitel die miteinander kommunizierenden Fahrzeuge als *Knoten* und die zu verteilenden einzelnen Daten als *Datenelement* bezeichnet. Es wird davon ausgegangen, dass ein Datenelement zum Versenden in eine *Nachricht* passt; größere Datenmengen können dafür von der Anwendung in entsprechende Datenelemente unterteilt werden.

Die Verwendung klassischer Kommunikationsmodelle ist im Allgemeinen in Ad-hoc-Netzen nicht sinnvoll. So bauen beispielsweise Client-Server-Protokolle auf länger andauernde, zuverlässige Verbindungen zwischen zwei Kommunikationspartnern. Aufgrund der erhöhten Fehlerwahrscheinlichkeit in Multi-Hop-Funknetzen gegenüber kabelgebundenen Netzen ist dieser Ansatz mit einem hohen Aufwand in Form von Nachrichtenwiederholungen verbunden. Wenn zudem die Knoten mobil sind und das Netz bedingt durch eine geringe Knotendichte in viele Partitionen zerfällt, wird das Aufrechterhalten von Punkt-zu-Punkt-Verbindungen praktisch unmöglich. Des Weiteren berücksichtigen im Internet eingesetzte Protokolle nur selten die physikalische Position der kommunizierenden Systeme und die daraus resultierenden Kommunikationswege. In Peer-2-Peer-Overlay Netzen ist es beispielsweise durchaus üblich, Systeme, die auf verschiedenen Kontinenten stehen, auf logischer Ebene als „direkt benachbart“ zu betrachten (vgl. [106]). In Ad-hoc-Netzen ist hingegen der Kommunikationsaufwand direkt von der Entfernung der Kommunikationspartner abhängig, da Nachrichten von Knoten zu Knoten weitergereicht werden müssen. Im Übrigen stellt die Verwendung von Unicast-Kommunikation in einem auf Broadcast-Kommunikation basierenden Funknetz eine künstliche Einschränkung dar.

Auch viele Routing-Verfahren, die für Ad-hoc-Netze entwickelt wurden, versuchen

bewährte Unicast-Kommunikationsmodelle in Ad-hoc-Netzen verfügbar zu machen, dazu zählen beispielsweise z. B. AODV [133] oder DSDV [134].

Für Anwendungen, die auf mobilen Ad-hoc-Netzen basieren, muss somit eine neue Form der Kommunikation gefunden werden, die berücksichtigt, dass Kommunikationsbeziehungen aufgrund der verwendeten Funkschnittstelle generell unzuverlässig sind und sich zudem die Netztopologie ständig durch Bewegung der Knoten verändert. Da die direkte Kommunikation zwischen zwei beliebig ausgewählten Knoten nicht sichergestellt ist, müssen Daten von dazwischen liegenden Knoten weitergeleitet und gegebenenfalls zwischengespeichert werden, um ihren Weg zum endgültigen Empfänger bzw. zur Gruppe von Empfängern zu finden. Es ist leicht einsehbar, dass ein Kommunikationsmodell, bei dem beliebige – teils weit voneinander entfernte – Knoten paarweise Daten untereinander austauschen, bei zunehmender Größe des Ad-hoc-Netzes nicht skaliert. Bei den betrachteten Anwendungen spielt jedoch der konkrete Kommunikationspartner keine Rolle, so dass keine paarweise Kommunikation benötigt wird. Vielmehr müssen Daten an alle Knoten in einer bestimmten Region verteilt werden. Dieser Vorgang wird in dieser Arbeit mit Datenverteilung (engl. data dissemination) betitelt.

AutoCast erlaubt allen Knoten Datenelemente zu produzieren, die neben den Nutzdaten, ein Verbreitungsgebiet sowie eine Lebenszeit beinhalten. Während ihrer Lebensdauer verteilt *AutoCast* die Datenelemente an möglichst alle Knoten im entsprechenden Verbreitungsgebiet. Dabei beteiligen sich auch solche Knoten an der weiteren Verbreitung der Datenelemente, die nicht an den enthaltenen Daten interessiert sind, wie beispielsweise Fahrzeuge auf der Gegenfahrbahn. Zur Reduktion der Komplexität werden keinerlei Beziehungen zwischen Knoten aufgebaut und auch keine Annahmen über Multi-Hop-Routen getroffen.

Neben dem Verteilen von Datenelementen an alle Knoten innerhalb eines Verbreitungsgebiets beherrscht *AutoCast* auch einen *Broadcast*-Dienst, der ein Datenelement einmalig aussendet, so dass es von allen direkten Nachbarn des Knotens empfangen wird. Dieser *Broadcast*-Dienst wird für die interne Kommunikation der HDCs verwendet.

Im Sinne von Organic Computing nimmt sich das *AutoCast*-Protokoll eine soziale Kommunikationsform als Vorbild: „Klatsch und Tratsch“. Denn dabei werden höchst effizient Informationen verbreitet; einerseits durch direkte (Gruppen-)Kommunikation und andererseits durch „Mitnehmen“ der Informationen an entfernte Orte und dortiges Weitererzählen. Menschliche Kommunikationspartner geben recht schnell bekannt, ob ihnen gewisse Informationen bereits bekannt sind, so dass zum nächsten Thema übergegangen werden kann. Besonders interessante Nachrichten werden bevorzugt weitererzählt, was einer einfachen Priorisierung entspricht. Im Gegensatz zum Tratschen verzichtet *AutoCast* natürlich auf die rein menschliche Art der Übertreibung und Verzerrung von Informationen.

Neben dieser Grundidee soll das *AutoCast*-Protokoll die folgenden Anforderungen erfüllen, um eine möglichst zuverlässige und effiziente Datenverteilung zu gewährleisten:

1. Die Funktion ist in verschiedenen Verkehrssituationen und den daraus entstehenden Netztopologien sichergestellt. So ergibt sich beispielsweise bei einem Verkehrsstau und hohem Ausstattungsgrad ein sehr dichtes Netz, in dem ein Knoten hundert und mehr Nachbarn haben kann, die sich über einen längeren Zeitraum nicht verändern. Bei fließendem Autobahnverkehr hingegen partitionieren große Abstände zwischen ausgestatteten Fahrzeugen das Netz, und entgegenkommenden Fahrzeugen verbleiben nur einige Sekunden zum Datenaustausch.

2. Möglichst alle Knoten im Verbreitungsgebiet, die theoretisch über Multi-Hop-Kommunikation und Mitnahme der Daten erreichbar sind, sollen die zu verteilenden Daten auch empfangen. Diese Eigenschaft wird in der Evaluation als Bewertungsmaß für die Qualität des Protokolls dienen.

3. Die Daten sollen so schnell wie möglich zugestellt werden, dabei aber eine möglichst geringe Datenrate erzeugen. Hier gilt es, einen geeigneten Kompromiss zu erzielen.

4. Zusammengefasst muss sichergestellt sein, dass das emergente Verhalten, welches auf globaler Ebene durch die vielen lokalen Interaktionen der Knoten untereinander entsteht, stets kontrollierbar bleibt. So darf das Protokoll beispielsweise nicht die gesamte zur Verfügung stehende Datenrate ausnutzen.

Dieses Kapitel gliedert sich wie folgt: Zunächst werden in Abschnitt 5.1 verwandte Arbeiten vorgestellt und der Bedarf für einen neuen Ansatz erläutert. In Abschnitt 5.2 werden die einzelnen Bausteine des Protokolls besprochen. Die Implementierung wird in Abschnitt 5.3 behandelt. Dem folgt in Abschnitt 5.4 eine ausführliche Evaluation der maßgeblichen Leistungsparameter des Protokolls im Vergleich zu anderen Ansätzen. Daran schließt sich eine Evaluation des Simulationswerkzeugs in Hinblick auf Laufzeit und Speicherverbrauch an. Das Kapitel schließt mit einer Zusammenfassung der Ergebnisse.

5.1 Verwandte Arbeiten

Eine Vielzahl wissenschaftlicher Arbeiten beschäftigen sich im weitesten Sinne mit der Beförderung von Daten in (mobilen) Ad-hoc-Netzen. Diese Arbeiten lassen sich unterteilen in Ansätze zum *Routing* und zum *Verteilen von Daten*. Beim Routing werden Daten zu einem oder mehreren *eindeutig vorgegebenen Empfängern* gesendet, die sich an beliebiger Stelle im Ad-hoc-Netz befinden können. Beim Verteilen von Daten werden diese an alle Knoten gesendet, die sich in einer *eindeutig vorgegebenen Region* befinden; oder allgemeiner an alle Knoten, die eine bestimmte gemeinsame Eigenschaft besitzen.

AutoCast zählt zur Gruppe der Protokolle für das Verteilen von Daten. Dabei befindet sich der ursprüngliche Sender eines Datenelements immer innerhalb des Verbreitungsgebiets. Protokolle zum Routing könnte *AutoCast* dahin gehend erweitern, dass Daten

zunächst über weite Distanzen geroutet werden, bevor sie ihr Verbreitungsgebiet erreichen, um dort mit *AutoCast* verteilt zu werden. Kandidaten für ein solches hybrides Protokoll wären sowohl Unicast-, Multicast- als auch geografische Routing-Protokolle, die nachfolgend kurz vorgestellt werden.

Unicast-Protokolle haben ihren Ursprung zumeist in kabelgebundenen Netzen und können mehr oder weniger schlecht mit Knotenbewegungen umgehen. Prominente Vertreter sind *Ad-hoc On-Demand Distance Vector Routing* (AODV) [133] als reaktives Verfahren und *Destination-Sequenced Distance Vector Routing* (DSDV) [134] als proaktives Verfahren. Eine Übersicht findet sich in [8]. Multicast-Protokolle verwalten nicht nur Routen zu einzelnen Knoten, sondern ermöglichen anhand beliebiger Parameter die Bildung von Gruppen, die als Empfänger von Nachrichten dienen (vgl. [80] für eine Übersicht). Beim geografischen Routing wird die Position eines Knotens zur Adressierung verwendet. Beim *Greedy Perimeter Stateless Routing* (GPSR) [84] wird davon ausgegangen, dass jeder Knoten seine eigene Position kennt und so entscheiden kann, ob er beim Weiterleiten einer Nachricht behilflich sein kann. Eine Übersicht über geografisches Routing findet sich in [111]. Die Gruppe der geografischen Routing-Protokolle erscheint für eine Kombination mit *AutoCast* in VANETs am sinnvollsten, da jedes Fahrzeug seine eigene GPS-Position bestimmen kann und somit geografische Positionen direkt als Zieladresse verwendet werden können.

Als bestehendes Beispiel für ein hybrides Protokoll sei *Location-Aided Routing* (LAR) [90] genannt, das Wissen über die räumliche Position von Knoten ausnutzt, um die Suche nach einer Route zwischen zwei Knoten auf eine „Abfrage-Zone“ zu beschränken. Sobald die Route gefunden ist, wird beispielsweise AODV zum Austausch von Daten verwendet werden.

Bei Protokollen zur Datenverteilung werden die Zielknoten aufgrund einer bestimmten Eigenschaft ausgewählt. Diese Eigenschaft kann sich entweder auf eine beliebige Eigenschaft der Knoten beziehen, die keine Rückschlüsse auf räumliche Gegebenheiten zulässt oder die Eigenschaft stellt geografische Positionen dar und definiert somit eine Zielregion für die Daten.

Im erstgenannten Fall sind Multicast-Verfahren notwendig, um die Mitgliedschaft von Knoten in Empfänger- bzw. Multicast-Gruppen zu organisieren; hier verschwimmt die Abgrenzung zwischen Routing und Datenverteilung, da die Multicast-Datenverteilung im Extremfall über eine Vielzahl von Unicast-Routing-Prozessen realisiert werden muss.

Ein typisches Verfahren zur Datenverteilung in drahtlosen Sensornetzen, das zu Multicast-Kommunikation führt, ist Directed Diffusion [74]. Dabei fluten potenzielle Empfänger, auch Datensenken genannt, die sie interessierenden Datentypen im gesamten Ad-hoc-Netz. Entlang der so entstehenden Pfade werden im weiteren Verlauf die entsprechenden Datentypen zu den Datensenken weitergeleitet. Dieses Konzept arbeitet zufriedenstellend in statischen Ad-hoc-Netzen, wenn die Anzahl der Datensenken begrenzt und die Datentypen vorhersehbar sind. Es ist aber unzureichend zur Verteilung

dynamisch generierter Daten an eine große Zahl empfangender Knoten. Im Allgemeinen stellt die Verwaltung der Multicast-Gruppen den Engpass bei allen Multicast-Ansätzen dar.

In zweiten Fall lassen sich die Empfänger der Datenverteilung anhand ihrer geografischen Position bestimmen. Dabei kann der Hopcount genauso als Kriterium dienen wie GPS-Positionen, vorausgesetzt alle Knoten verfügen über einen entsprechenden GPS-Empfänger. Somit entfällt die Notwendigkeit, dedizierte Empfängergruppen zu verwalten. Da *AutoCast* sich in diese Kategorie einordnen lässt, folgen verwandte Ansätze auf diesem Gebiet.

Der einfachste und in der Literatur vielfach verwendete Ansatz ist einfaches Fluten. Es wird beispielsweise zum Auffinden von Routen in Unicast-Routing-Protokollen verwendet (vgl [78]). Beim einfachen Fluten sendet ein Knoten, der eine ihm unbekannte Nachricht empfängt, diese umgehend. Wird eine bereits bekannte Nachricht empfangen, so wird sie verworfen. Der Nachteil dieses sehr einfachen Schemas ist das Fehlen einer Adaption an die Knotendichte. So führt eine geringe Knotendichte zu einer geringen Zuverlässigkeit, da eine Nachricht immer nur genau einmal von einem Knoten weitergeleitet wird, während in sehr dichten Netzen „Broadcast Storms“ [177] die Funktionalität einschränken.

Zur Erhöhung der Effizienz in Netzen mit hoher Knotendichte wurde das einfache Fluten um eine probabilistische Komponente erweitert, so dass eine Nachricht nur noch mit einer bestimmten Wahrscheinlichkeit weitergeleitet wird – beim einfachen Fluten beträgt diese Wahrscheinlichkeit immer 100 %. Protokolle, die diese Idee umsetzen finden sich in der Literatur unter den Namen selektives Fluten, probabilistisches Fluten sowie Gossiping – im Folgenden wird der Begriff *probabilistisches Fluten* verwendet. Die Weiterleitungswahrscheinlichkeit

- basiert entweder auf einer globalen Konstante [58, 153, 177],
- richtet sich nach einer Eigenschaft der Netztopologie, z. B. der Knotendichte oder deren Geschwindigkeit [13, 60],
- oder basiert auf einer Eigenschaft, die pro weiterzuleitender Nachricht bestimmt wird, wie z. B. der Position relativ zum sendenden Knoten oder der Anzahl von Sendewiederholungen der entsprechenden Nachricht [81, 101, 128, 174].

Vergleiche verschiedener probabilistischer Ansätze aus den drei vorgenannten Kategorien gegenüber einfachem Fluten finden sich in [27, 64, 190]. Die Idee des probabilistischen Flutens basierend auf der lokalen Knotendichte findet sich als Protokollbaustein im *AutoCast*-Protokoll wieder.

Ein weiterer Ansatz zur Verringerung des Nachrichtenaufkommens beim Verteilen von Daten im Vergleich zum einfachen Fluten ist die proaktive Rollenverteilung bzw. das

Bilden funktionaler Cluster. Das von Khan u. a. beschriebene Verfahren *Parameterless Broadcasting from Static to Mobile* (PBSM) [87] basiert auf der proaktiven und kontinuierlichen Bestimmung eines Connected Dominating Sets, d. h. es wird eine Teilmenge der Knoten so ausgewählt, dass die in der Teilmenge befindlichen Knoten untereinander verbunden sind und sich alle übrigen Knoten in ihrem Kommunikationsgebiet befinden. Nur die so ausgezeichneten Knoten beteiligen sich am Weiterleiten von Daten. Die Autoren zeigen, dass dieses Verfahren auch in mobilen Szenarien effizient arbeitet, gehen dabei allerdings vom Unit-Disk-Graph-Modell aus. In der Realität sind die Kommunikationsbeziehungen leider nicht immer bidirektional und so stabil wie von diesem Modell angenommen.

Alle probabilistischen Ansätze sowie PBSM reduzieren die benötigte Datenrate beim Verteilen einer Nachricht und gehen dabei von einem voll konnektierten Netz mit hoher Knotendichte aus. In einem Netz mit geringer Knotendichte führen diese Verfahren allein allerdings nicht zum Ziel, da eine Nachricht höchstens innerhalb einer Netzpartition verteilt wird. In Szenarien mit mobilen Knoten – wie im Straßenverkehr – verändern sich die Netzpartitionen mit der Zeit aufgrund der Knotenbewegung. Eine Wiederholung der Übertragung von Nachrichten in veränderter Umgebung erscheint hier vielversprechend. Diese – im Folgenden vorgestellten – Ansätze werden auch mit Store-and-Forward bezeichnet.

Wu u. a. beschreiben in [196] die Nutzung mobiler Knoten als „Fähren“ für die Verbesserung des Unicast Routings. Dabei kontrolliert das vorgestellte Protokoll die Bewegung aller Knoten, was in dieser Arbeit nicht vorgesehen ist. Für den Fall, dass einige mobile Knoten vom Routingprotokoll an strategische Positionen bewegt werden können, finden sich in [67, 131] entsprechende Lösungen. Wenn die Bewegung der einzelnen Knoten nicht vom Routingprotokoll gesteuert werden kann, stellt die Auswahl des jeweils richtigen Knotens – der eine Nachricht ein Stück seines Weges mitnimmt – eine große Herausforderung dar, wobei Grossglauser und Tse in [57] eine Erhöhung des Datendurchsatzes bei einer zufälligen Bewegung der Knoten voraussagen.

Der in [108] vorgestellte *XCast*-Ansatz sieht ein periodisches Versenden von Daten vor, um einen verteilten Datenspeicher namens *Distributed Virtual Shared Information Space* (DVSIS) [91] zu verwalten. Nicht näher spezifizierte lokale Filter sollen die zu versendenden Daten auswählen und eine Überlastung der Funkschnittstelle vermeiden. Durch das periodische Senden ist *XCast* in der Lage, Daten zwischen temporär nicht verbundenen Netzpartitionen zu übertragen.

Wenn die Bewegung der Knoten nicht kontrolliert werden kann, sind offensichtlich alle Knoten potenzielle Fähren. Somit müssen alle Knoten weiterzuleitende Daten für eine gewisse Zeit speichern, um sie gegebenenfalls zu einem späteren Zeitpunkt erneut zu versenden. Das verteilte Speichern von Daten in einem Ad-hoc-Netz wird beispielsweise vom zuvor genannten DVSIS umgesetzt. Jeder Knoten kennt einen Ausschnitt des DVSIS und neue Daten werden sporadisch zwischen den Knoten ausgetauscht. Dies führt

zu einem unvollständigen, teilweise veralteten bzw. inkonsistenten Datenstand auf den einzelnen Knoten, wobei der Umgang und die Interpretation der Daten der Anwendung überlassen bleibt.

Einen in Bezug auf die Kommunikation sehr optimistischen Ansatz stellt die *geografische Hash-Tabelle* (*Geographical Hash Table*, GHT) dar [144]. Die Daten werden in diesem Ansatz gleichmäßig im Netz verteilt, wobei der Hash-Wert der Daten den Speicherort geografisch festlegt. Zum Abrufen der Daten wird ein geografisches Routingverfahren verwendet. Daten werden somit nicht an dem Ort gespeichert, den sie beschreiben, sondern an einem von ihrem Hash-Wert bestimmten Ort. Das *AutoCast*-Protokoll realisiert einen verteilten Datenspeicher quasi als Seiteneffekt, denn die Datenelemente werden innerhalb ihres Verbreitungsgebiets von allen Knoten zwischengespeichert bis ihre Lebenszeit endet – die geografische Position wird dabei von den Datenelementen selbst festgelegt. Dieser Ansatz ist weit weniger komplex als die Daten gleichmäßig im Netz zu verteilen, und erlaubt beispielsweise die semantische Aggregation während sich die Daten immer weiter von ihrem Ursprung entfernen.

Neben der Bestimmung von geografischen Positionen können Hash-Werte auch zum Reduzieren der benötigten Datenrate verwendet werden. Dazu werden anstatt der vollständigen Daten zunächst nur deren Hash-Werte ausgetauscht. In [65] wird ein Unicast-basierter Ansatz vorgestellt, bei dem die Kommunikation in drei Stufen erfolgt: Als Erstes werden nur Hash-Werte gesendet. Der Empfänger fragt daraufhin nach den ihm unbekannten Hash-Werten, die vollständigen Daten werden im dritten Schritt an ihn gesendet. Das *AutoCast*-Protokoll verwendet ebenfalls Hash-Werte, um Datenelemente eindeutig zu identifizieren, benötigt aber nur einen zweistufigen Prozess und dank Broadcast-Kommunikation profitieren alle Knoten innerhalb des Kommunikationsgebiets.

Jeder der bisher vorgestellten Ansätze arbeitet optimal nur in bestimmten Netztopologien. Eine dynamische Adaption des Protokoll-Verhaltens entsprechend der Knotendichte findet nur begrenzt statt. Im Folgenden werden zwei Ansätze näher betrachtet, die ihr Verhalten vom probabilistischen Fluten bis hin zu Store-and-Forward anpassen können.

In [180] stellen Viswanath und Obrazcka ein Protokoll names *Adaptive Flooding* vor, bei dem jeder Knoten zwischen drei Betriebsmodi umschalten kann. Dies sind Fluten, probabilistisches Fluten und ein Modus der zwischengespeicherte Pakete an jeden neu ins Kommunikationsgebiet kommenden Knoten sendet. Das Umschalten erfolgt aufgrund der Knotengeschwindigkeit und der Auslastung des Funknetzes. Leider führt das unkorrelierte Weiterleiten zu einem hohen Protokoll-Overhead in Netzen mit geringer Knotendichte.

Beim *Hypergossiping*, das Khelil u. a. in [88] vorstellen, wird probabilistisches Fluten mit Sendewiederholungen kombiniert. Das probabilistische Fluten wird anhand der lokalen Knotendichte adaptiert. Allerdings muss die Weiterleitungswahrscheinlichkeit auf

einem relativ hohen Maß gehalten werden, damit eine Flutwelle garantiert alle Knoten innerhalb einer Netzpartition erreicht. Dies ist notwendig, weil zwischengespeicherte Pakete nur dann erneut gesendet werden, wenn Knoten aufeinander treffen, die längere Zeit nicht miteinander kommuniziert haben und deren Liste der zuletzt empfangenen Pakete über einem bestimmten Maß voneinander abweicht. Die Notwendigkeit, ein einzelnes Paket erneut zu senden, wird somit nicht detektiert. *AutoCast* ist an dieser Stelle sensibler und erkennt das Fehlen einzelner Datenelemente anhand ihrer Hash-Werte.

5.2 Protokollaufbau

Ausgehend von den am Anfang des Kapitels beschriebenen Anforderungen ist *AutoCast* als adaptives Protokoll aufgebaut, das zwei Arten der Beförderung von Daten in Ad-hoc-Netzen nutzt. Zum einen erlaubt die Funkschnittstelle einen sehr schnellen Datenaustausch zwischen Knoten, die sich innerhalb der Kommunikationsradius befinden („Multi-Hop-Forwarding“). Wenn die Knoten zudem mobil sind, kommt als zweite Variante „Store-And-Forward“ hinzu. Daten werden dabei von mobilen Knoten gespeichert und an einem anderen Ort erneut gesendet.

Bei Netzen mit einer hohen Knotendichte führt bereits Multi-Hop-Forwarding zum Ziel, denn alle Knoten bilden eine einzige Netzpartition. Sobald aber die Knotendichte abnimmt und das Netz in mehrere Partitionen zerfällt, ist zusätzlich die Store-And-Forward-Protokoll-Komponente nötig, um die Grenzen der Netzpartitionen zu überwinden.

Wie im vorherigen Abschnitt 5.1 beschrieben, decken die meisten verwandten Protokolle zur Datenverteilung diesen Arbeitsbereich nicht komplett ab, sondern verwenden nur eine der Varianten – entweder Multi-Hop-Forwarding oder Store-And-Forward. *AutoCast* hingegen passt seine Arbeitsweise ausgehend von lokalen Regeln an die jeweilige Knotendichte an. Dabei kommen die folgenden vier Protokollbausteine zum Einsatz:

Multi-Hop-Forwarding: Um Nachrichten innerhalb von Netzpartitionen per drahtloser Kommunikation von Knoten zu Knoten zu verteilen, greift *AutoCast* auf probabilistisches Fluten zurück (vgl. Abschnitt 5.1). Auf Basis von lokalen Parametern entscheidet jeder Knoten, ob er sich am Weiterleitungsprozess eines Datenelements beteiligt.

Selbst-Adaption: Basierend auf dem Grundgedanken, dass es keine Rolle spielt, durch welche Knoten ein Datenelement weitergeleitet wird, dient die Nachbarschaftsgröße als Metrik zur Selbst-Adaption. Je weniger Nachbarn ein Knoten besitzt, desto aktiver beteiligt er sich an der Kommunikation. Durch regelmäßige *Beacon*-Nachrichten ermittelt jeder Knoten die Anzahl seiner Nachbarn.

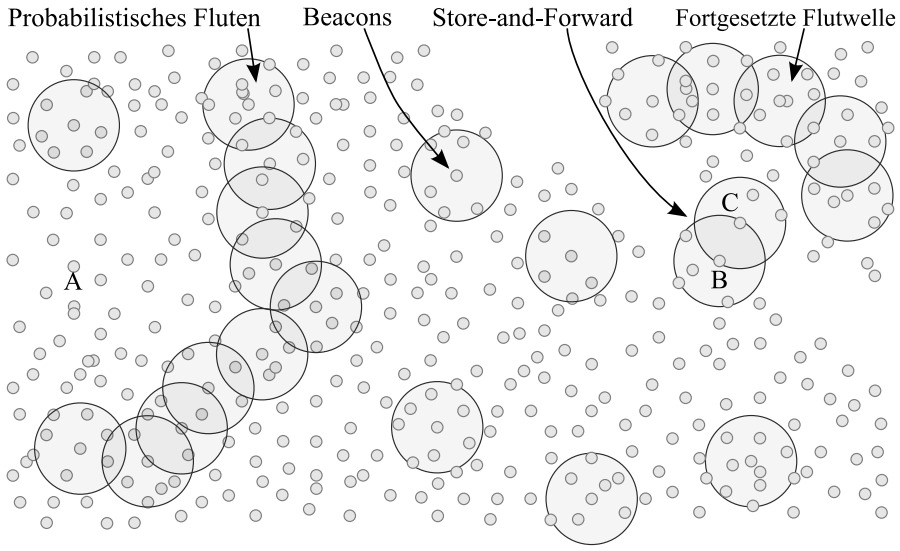


Abbildung 5.1: Aktionsmuster des *AutoCast*-Protokolls.

Store-And-Forward: Durch Bewegung von Knoten können auf den Knoten zwischengespeicherte Daten in entfernte Netzpartitionen getragen werden. Ein Store-And-Forward-Protokollbaustein sorgt für ein wiederholtes Versenden eines Datenelements, falls ein oder mehrere benachbarte Knoten das entsprechende Datenelement noch nicht kennen. Durch den gegenseitigen Abgleich von komprimierten Wissensständen werden nur exakt die Datenelemente übertragen, die den benachbarten Knoten fehlen. Dieser Protokollbaustein greift auch, falls der Prozess des probabilistischen Flutens zum Erliegen kommt.

Verbreitungsgebiete: Ein Datenelement ist in der Regel nur innerhalb eines bestimmten Gebiets für eine bestimmte Zeitdauer von Interesse. Dieser Baustein sorgt dafür, dass alle Daten nur innerhalb der gegebenen Grenzen verbreitet werden.

Abbildung 5.1 veranschaulicht die Funktionsweise von *AutoCast*. Im gezeigten Ad-hoc-Netz werden einige Datenelemente simultan verteilt, so dass sich alle Protokollbausteine in der Momentaufnahme wiederfinden.

So wird ein von Knoten *A* ausgesendetes Datenelement mithilfe des probabilistischen Flutens innerhalb der Netzpartition verbreitet. Aufgrund der Selbst-Adaption beteiligt sich nur ein Bruchteil der Knoten im Ad-hoc-Netz an diesem Prozess, der in Abbildung 5.1 in Form einer Flutwelle zu sehen ist.

Regelmäßig sendet jeder Knoten ein Beacon, welches einerseits zur Bestimmung der Knotendichte und somit zur Selbst-Adaption verwendet wird, und andererseits zum Auffinden neuer benachbarter Knoten dient.

So sorgen diese regelmäßigen Beacons dafür, dass sich die Knoten B und C gegenseitig über ihre Anwesenheit informieren, sobald sie in Funkreichweite zueinander kommen. Da die Knoten B und C als Erstes die beiden abgebildeten Netzpartitionen verbinden, tauschen sie untereinander die Datenelemente aus, die der jeweils andere noch nicht kennt (Store-and-Forward), um ihren Wissensstand untereinander abzugleichen.

Ausgehend vom Knoten C werden die von Knoten B zwischengespeicherten Datenelemente in der nächsten Netzpartition durch probabilistisches Fluten verteilt und die Flutwelle somit fortgesetzt.

In den folgenden Abschnitten wird die Funktionsweise und das Zusammenspiel der vorgenannten vier Bausteine ausführlich hergeleitet und beschrieben.

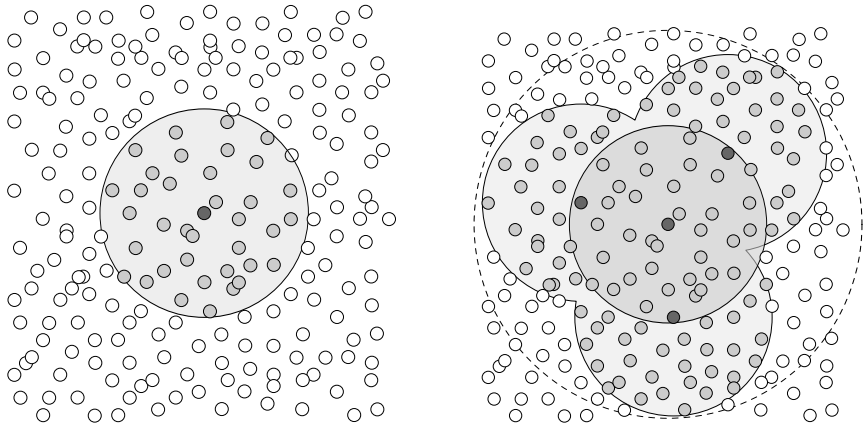
5.2.1 Probabilistisches Fluten (Multi-Hop-Forwarding)

Ein sehr einfaches Protokoll zum Verteilen von Daten ist *einfaches Fluten*. Es wird vielfach in der Literatur genannt, beispielsweise als Technik zum Auffinden von Routingpfaden beim *Dynamic Source Routing* (vgl. [78]).

Es funktioniert wie folgt: Jeder Knoten, der eine Nachricht empfängt, vergleicht diese mit seinen bisher empfangenen Nachrichten. Falls er die Nachricht zuvor noch nicht empfangen hat, versendet er diese umgehend als Broadcast-Nachricht und fügt sie in die Liste bisher empfangener Nachrichten ein. Bezogen auf die hier verwendete Nomenklatur heißt das, dass ein empfangenes Datenelement so schnell wie möglich und genau einmal von jedem Knoten weitergeleitet wird. Fluten verbreitet Daten in zusammenhängenden Netzen extrem schnell. Allerdings werden nur Knoten innerhalb einer Netzpartition erreicht.

In Netzen mit hoher Knotendichte erzeugt Fluten zudem einen „Broadcast Storm“, wie in [177] beschrieben. Da beim einfachen Fluten ein neues Datenelement von allen Empfängern weitergeleitet wird, steigt die Netzauslastung zunächst proportional mit der mittleren Nachbarschaftsgröße. Sobald die von allen benachbarten Knoten weitergeleiteten Datenelemente beim Zugriff auf das Funkmedium miteinander kollidieren, steigt die Netzauslastung durch wiederholte Sendeversuche der Medienzugriffsschicht überproportional an (bei Verwendung des Medienzugriffsverfahrens CSMA), so dass der Prozess des Flutens zum Erliegen kommt.

Um die Netzauslastung beim Fluten in Netzen mit hoher Knotendichte zu vermindern und den beschriebenen Broadcast Storm zu vermeiden, lässt sich *probabilitisches Fluten* einsetzen. Dabei leitet ein Knoten ein neu empfangenes Datenelement nur mit einer bestimmten Wahrscheinlichkeit weiter (beim einfachen Fluten beträgt diese Wahrscheinlichkeit 100 %). Abbildung 5.2a zeigt beispielhaft den Anfang eines Flut-Prozesses, d. h. zunächst wird ein Datenelement vom ursprünglichen Sender an seine direkten Nachbarn gesendet. In diesem Fall wird das Datenelement von 36 Knoten empfangen. Bei geschickter Wahl der weiterleitenden Knoten werden im zweiten Schritt weitere 80 Kno-



(a) Erste Broadcast-Nachricht vom ursprünglichen Sender

(b) Selektive Weiterleitung

Abbildung 5.2: Probabilistisches Fluten.

ten innerhalb der hell schattierten Fläche von nur drei weiterleitenden Knoten erreicht, wie Abbildung 5.2b zeigt. Falls hingegen alle 36 Knoten das Datenelement weiterleiten, werden alle 138 Knoten erreicht, die sich innerhalb des gestrichelten Kreises befinden, dessen Radius dem doppelten Kommunikationsradius entspricht. Somit lassen sich in diesem Beispiel durch geschickte Selektion der weiterleitenden Knoten 58 % der Knoten mit nur 8 % der gesendeten Nachrichten erreichen, die beim einfachen Fluten nötig wären.

Unter der Anforderung ein Datenelement an alle Knoten im Netz zu verteilen, stellt die optimale Selektion der weiterleitenden Knoten schon in statischen Netzen und bei Vorhandensein von globalem Wissen ein komplexes Problem dar. In den hier betrachteten mobilen Ad-hoc-Netzen ist per se kein globales Wissen verfügbar; Nachbarschaftsinformationen müssen stattdessen über die Funkschnittstelle ausgetauscht werden. Da diese Nachbarschaftsinformationen durch die Bewegung der Knoten veralten, muss ihr Austausch periodisch wiederholt werden. Auch das Funkmedium ist in der Realität nicht so berechenbar, wie im hier zur analytischen Betrachtung verwendeten Unit-Disk-Graph-Modell zugrunde gelegt wird. Nachrichten können sporadisch durch äußere Störungen verloren gehen und auch der Kommunikationsradius variiert in der Realität abhängig von äußeren Faktoren.

In einer entsprechend konstruierten Netztopologie (vgl. Abbildung 5.3) schlägt selbst einfaches Fluten durch Verzicht auf jeglichen Overhead ausgereifte Protokolle, wenn die Nachricht von jedem Knoten im Netz weitergeleitet werden muss.

Probabilistisches Fluten wird in der Literatur oftmals mit einer konstanten Weiterleitungswahrscheinlichkeit eingesetzt (vgl. [58, 153, 177]). In diesen Arbeiten wird je

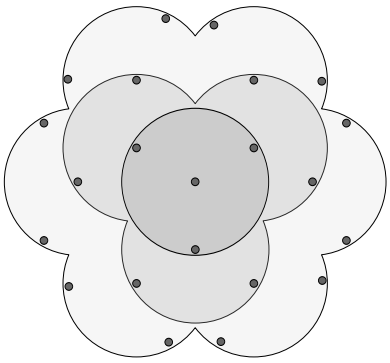


Abbildung 5.3: Netztopologie, bei der einfaches Fluten optimal ist.

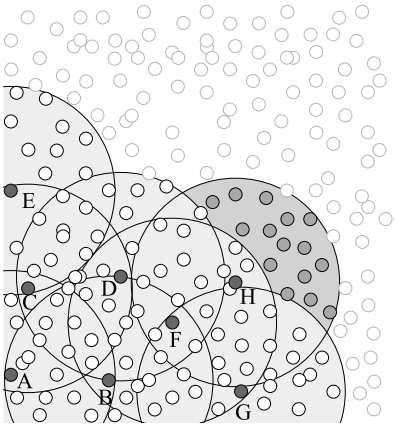


Abbildung 5.4: Informationsverteilung durch probabilistisches Fluten.

nach Aufbau des Simulationsszenarios und der daraus resultierenden Netztopologie eine optimale Weiterleitungswahrscheinlichkeit zwischen 60 % und 80 % angegeben.

Das in *AutoCast* implementierte probabilistische Fluten arbeitet hingegen in beliebigen Netztopologien effizient. Dabei basiert es vollständig auf lokalem Wissen der Knoten. Die grundlegende Idee ist, dass unabhängig von der lokalen Knotendichte die Anzahl weiterleitender Knoten auf konstantem Niveau bleiben soll. Diese im Mittel konstante Zahl weiterleitender Knoten wird $\#forwarder$ genannt. An dieser Stelle sei jedem Knoten die Anzahl seiner Nachbarn ($\#neighbors$) bekannt; eine detaillierte Betrachtung dieser Problematik folgt in Abschnitt 5.2.2.

Eine empfangene Nachricht soll grundsätzlich nur von Knoten weitergeleitet werden, die diese Nachricht zum ersten Mal empfangen; das in Bereichen mit geringer Knotendichte nötige wiederholte Senden von Datenelementen wird in Abschnitt 5.2.3 betrachtet.

Zur Veranschaulichung zeigt Abbildung 5.4 eine „Flutwelle“, wie sie beim Verteilen einer Nachricht durch probabilistisches Fluten entsteht. Nachdem die Knoten A bis G durch Senden der Nachricht bereits die Knoten im unteren linken Bereich informiert haben, sendet Knoten H die Nachricht. Grau schattiert sind die Knoten, welche die Nachricht von H zum ersten Mal erhalten. Alle übrigen Knoten in Hs Kommunikationsgebiet haben die Nachricht bereits durch einen vorhergehenden Broadcast erhalten. Somit stellen sich in diesem Beispiel 15 der insgesamt 43 Nachbarn ($\approx 35\,\%$) von H die Frage, ob sie die Nachricht weiterleiten. Der Anteil benachbarter Knoten, der eine gesendete Nachricht zum ersten Mal hört, sei im Folgenden $ratio_{newNeighbors}$.

Betrachtet man von allen Knoten die Überlappung ihrer Kommunikationsgebiete, die im Unit-Disk-Graph-Modell als kreisförmig angenommen werden, dann lässt sich für $ratio_{newNeighbors}$ ein Mittelwert berechnen. Aus Gründen der Übersichtlichkeit findet sich die Herleitung in Anhang A.

So ergibt sich unter der Annahme, dass die Nachbarn eines Knotens auf einer Fläche gleichverteilt sind (lokale Gleichverteilung), $ratio_{newNeighbors} \approx 40\,\%$. Da sich die Knoten in einem typischen Autobahn-Szenario eher auf einer Linie als auf einer Fläche angeordnet sind, ergibt sich ein etwas geringerer Mittelwert $ratio_{newNeighbors} \approx 30\,\%$. Für die weitere Betrachtung wird der im gleichverteilten Szenario ermittelte Wert von $40\,\%$ angenommen.

Beim Empfang eines neuen Datenelements wertet ein Knoten die Ungleichung

$$\text{RND}(0,1) < \frac{\#\,forwarder}{\#\,neighbors \cdot ratio_{newNeighbors}} = \frac{\#\,forwarder}{\#\,neighbors \cdot 0,4} \tag{5.1}$$

aus. Wenn die Ungleichung wahr ist, sendet der entsprechende Knoten die Nachricht weiter.

Der folgende Abschnitt beschäftigt sich mit der Ermittlung der Nachbarschaftsgröße ($\#\,neighbors$). Der optimale Wert für $\#\,forwarder$ wird in Anhang B empirisch ermittelt.

Der Vollständigkeit halber sei noch eine komplexere Variante des probabilistischen Flutens namens *Exchange of Set of Neighborhood* erwähnt, welche vom Autor in [68] veröffentlicht wurde. Dabei wird an jede gesendete Nachricht eine Liste der Nachbarn des sendenden Knotens angehängt. Je weniger diese Nachbarschaftsliste in einer empfangenen Nachricht mit der eines potenziellen weiterleitenden Knotens übereinstimmt, desto größer kann die Distanz zwischen den Knoten angenommen werden und umso eher wird dieser Knoten die Nachricht weiterleiten. Somit steigt die mittlere Distanz zwischen weiterleitenden Knoten und damit auch die Effizienz des Protokolls. Allerdings steigert diese Maßnahme sowohl den Overhead des Protokolls durch das Anhängen der Nachbarschaftslisten an alle Nachrichten als auch die Komplexität des Protokolls. Da sich Organic-Computing-Protokolle durch ihre Einfachheit auszeichnen und das Weglassen dieser Variante die Grundfunktion des Protokolls nicht beeinträchtigt, wird in der vorliegenden Arbeit auf diese Protokollvariante verzichtet.

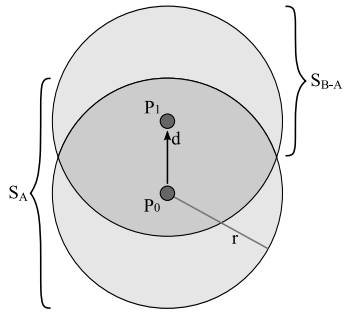


Abbildung 5.5: Veränderung des Kommunikationsgebiets eines Knotens durch dessen Bewegung.

5.2.2 Nachbarschaften (Selbst-Adaption)

Wie im letzten Abschnitt behandelt, basiert das probabilistische Fluten darauf, dass jeder Knoten die Anzahl seiner Nachbarn ($\#\,neighbors$) kennt bzw. abschätzen kann. Die Anwesenheit eines benachbarten Knotens kann in einem Ad-hoc-Netz nur dann festgestellt werden, wenn dieser eine Nachricht sendet und sich so zu erkennen gibt. Der übliche Ansatz, der vielfach in der Literatur zu finden ist, verwendet periodische *Beacon*-Nachrichten, die jeder Knoten versendet, um seine benachbarten Knoten über die eigene Anwesenheit zu informieren (vgl. [9, 135, 166]).

In vorangegangenen Arbeiten wird meist ein konstantes Intervall für die Zeit zwischen zwei Beacon-Nachrichten (t_{beacon}) angenommen; Der Standard IEEE 802.11 sieht beispielsweise im Infrastruktur-Modus vor, dass Basisstationen alle 0,1 s eine Beacon-Nachricht senden. Wischhof u. a. beschreiben in [193] eine Adaptierung des Broadcast-Intervalls welches zumeist zwischen 4 s und 10 s liegt. Dabei beziehen sich die Autoren aber ausschließlich auf das von ihnen vorgestellte Verkehrsinformationssystem SOTIS (vgl. [191]).

Da das *AutoCast*-Protokoll sein Verhalten aufgrund der Anzahl benachbarter Knoten adaptiert, wurde in dieser Arbeit eine systematische Untersuchung des optimalen Zeitintervalls zwischen zwei Beacon-Nachrichten durchgeführt. Dabei soll die Nachbarschaftsgröße im Mittel mit einer gewissen Genauigkeit ermittelt werden. Da diese Betrachtung möglichst allgemeingültig sein soll, wird lediglich ein einfaches Bewegungsmuster angenommen, bei dem sich die Knoten paarweise unabhängig voneinander bewegen. Zudem wird für die Berechnung des optimalen Zeitintervalls erneut das Unit-Disk-Graph-Modell zugrunde gelegt, d. h. das Kommunikationsgebiet eines Knotens ist kreisförmig mit einem Kommunikationsradius r.

Abbildung 5.5 betrachtet zunächst die Bewegung eines einzelnen Knotens. Bewegt sich der Knoten von P_0 nach P_1, verändert sich sein Kommunikationsgebiet und somit

die Nachbarschaft des Knotens. Als Maß für die Veränderung der Nachbarschaft wird die Fläche des veränderten Kommunikationsgebiets S_{B-A} im Verhältnis zur Fläche des gesamten Kommunikationsgebiets S_A verwendet.

Um nun die maximale Abweichung von der tatsächlichen Nachbarschaftsgröße zu begrenzen, muss jeder Knoten ein Beacon senden, wenn sich sein Kommunikationsgebiet entsprechend verschoben hat. Hier gilt es, einen vernünftigen Kompromiss zwischen der maximal tolerierten Abweichung und der dafür nötigen Anzahl an Beacon-Nachrichten zu finden. Im Folgenden sei die maximale Abweichung von der tatsächlichen Nachbarschaftsgröße 10 %, so dass genügend Zeit für den Nachrichtenaustausch verbleibt. Wie in Anhang A ausgeführt und in Abbildung A.1b illustriert, ändern sich 10 % der Nachbarschaft eines Knotens, wenn dieser sich um eine Distanz von $d = 0,14r$ bewegt.

Diese Betrachtung für einen Knoten wird nun übertragen auf Szenarien, in denen sich eine beliebige Anzahl Knoten paarweise unabhängig voneinander bewegen. Dann beträgt die relative Geschwindigkeit zwischen zwei beliebigen Knoten N_i und N_j $v_{rel} = |\overrightarrow{v_i} - \overrightarrow{v_j}|$. Je nach Bewegungsrichtung der Knoten N_i und N_j variiert die relative Geschwindigkeit zwischen $v_{rel} = 0$, wenn sich die Knoten mit gleicher Geschwindigkeit in dieselbe Richtung bewegen, und $v_{rel} = |\overrightarrow{v_i}| + |\overrightarrow{v_j}|$, wenn sie sich in exakt entgegengesetzte Richtungen bewegen. Wenn die Bewegungsrichtungen gleichverteilt sind, beträgt der mittlere Winkel zwischen zwei Bewegungsvektoren 90°.

Für die mittlere relative Geschwindigkeit zweier Knoten ergibt sich somit $v_{avg} = \sqrt{2}v_i$ bzw. $v_{avg} = \sqrt{2}v_j$. Im VANET-Szenario sind die Bewegungsrichtungen zwar nicht gleichverteilt, allerdings kann man annehmen, dass sich die Knoten gleichmäßig auf die Richtungsfahrbahnen aufteilen. Die relative Geschwindigkeit zwischen zwei Knoten ist also entweder sehr niedrig, wenn sie in die gleiche Richtung fahren, oder entspricht $|\overrightarrow{v_i}| + |\overrightarrow{v_j}|$ bei entgegengesetzter Fahrtrichtung. Im Durchschnitt führt dies zum selben Ergebnis.

Unter Berücksichtigung der relativen Geschwindigkeit und mit dem Ziel eine Genauigkeit von 90 % zu erzielen, ergibt sich die Distanz, die ein Knoten maximal zwischen dem Senden zweier Beacons zurücklegen darf, mit $d = 0,14r/\sqrt{2} \approx 0,1r$. Letztlich kann ein Knoten N_i, der sich mit der Geschwindigkeit v_i bewegt, die Zeitdauer zwischen zwei Beacons bestimmen als

$$t_{beacon} = \frac{d}{v_i} = 0,1\frac{r}{v_i}. \tag{5.2}$$

Um das Verschwinden benachbarter Knoten einfacher erfassen zu können, sendet jeder Knoten mit den Beacon-Nachrichten sein aktuelles t_{beacon}. Wenn die nächste Beacon-Nachricht eines benachbarten Knotens überfällig ist, so wird er aus der Nachbarschaftsliste entfernt.

Auch wenn die Berechnung von t_{beacon} von der eher theoretischen Abschätzung der Funkreichweite abhängig ist, ergibt sich eine lineare Abhängigkeit zwischen der Knotengeschwindigkeit und dem Zeitintervall zwischen zwei Beacon-Nachrichten.

5.2.3 Wiederholung von Übertragungen (Store-And-Forward)

Innerhalb einer zusammenhängenden Netzpartition können bereits mit den bislang vorgestellten *AutoCast*-Bausteinen „Multi-Hop-Forwarding“ und „Selbst-Adaption“ Datenelemente effizient verteilt werden. In folgenden Fällen reichen diese beiden Bausteine alleine jedoch nicht aus:

- Spätestens an der Grenze einer Netzpartition stoppt das Multi-Hop-Forwarding. Wenn später durch Bewegung der Knoten zwei Netzpartitionen in Funkreichweite zueinander kommen, muss das weitere Verteilen von Datenelementen erneut angestoßen werden.
- Wird ein Knoten inmitten eines bestehenden Netzes eingeschaltet, so muss dieser Knoten auf den Wissensstand der ihn umgebenden Knoten gebracht werden.
- Auch schon innerhalb einer Netzpartition kann in ungünstigen Fällen das Verteilen eines Datenelements durch die verwendete Zufallskomponente ins Stocken geraten. Nämlich dann, wenn sich beim probabilistischen Fluten einmal kein Knoten für die Weiterleitung eines Datenelements zuständig fühlt.

In jedem der Fälle wird ein Mechanismus benötigt, der die Verteilung von Datenelementen erneut anstößt, sobald ein oder mehrere Datenelemente benachbarten Knoten unbekannt sind. Zu diesem Zweck enthält *AutoCast* eine Komponente zum wiederholten Senden von Datenelementen basierend auf zusätzlich mit den Beacons versendeten Daten, die den Wissensstand des sendenden Knotens repräsentieren.

Diese zusätzlich versendeten Daten müssen alle Datenelemente, die der sendende Knoten kennt, eindeutig repräsentieren. Gleichzeitig sollen sie nur einen Bruchteil des Speicherplatzes der von ihnen beschriebenen Datenelemente belegen. *AutoCast* verwendet hier eine Hash-Funktion $h(d)$ namens *Divisions-Rest-Methode*, mit $h(d) \in [0; 2^{32}[$ (vgl. [89, S. 508 ff]). Für jedes aus l Byte bestehende Datenelement $d = b_0 b_1 ... b_{l-1}$ berechnet sich der Hash-Wert nach dem Horner-Schema wie in Gleichung 5.3 angegeben.

$$h(d) = (\ldots((((b_1 \cdot 2^8 + b_2) \bmod m) \cdot 2^8 + b_3) \bmod m) \cdot 2^8 + \cdots + s_{l-1}) \bmod m \quad (5.3)$$

Die Wahrscheinlichkeit, dass zwei Datenelemente mit unterschiedlichem Inhalt auf denselben Hash-Wert abgebildet werden, liegt laut [107] nicht höher als bei komplizierteren Verfahren. Angenommenen, n Hash-Werte sind über den gesamten Wertebereich $[0..m-1]$ der Hash-Funktion gleichverteilt. Dann ergibt sich die erwartete Anzahl an Kollisionen bzw. die Kollisionswahrscheinlichkeit $P_{col} = \binom{n}{2}\frac{1}{m} = \frac{n(n-1)}{2m}$. Für m wird hier die größte Primzahl kleiner 2^{32} gewählt: $m = 4294967291$. Bei $n = 100$ gleichzeitig lokal bekannten Datenelementen ergibt sich eine geringe Kollisionswahrscheinlichkeit von $P_{col} \approx 10^{-6}$. Somit können für die Untersuchung des *AutoCast*-Protokolls kollidierende

Hash-Werte zunächst vernachlässigt werden. Im Ausblick (Abschnitt 5.5) wird zudem ein Verfahren vorgestellt, das systematische Kollisionen der Hash-Werte verhindert.

Knoten, die eine Nachricht mit einer Liste von Hash-Werten empfangen, vergleichen die empfangenen Hash-Werte mit denen der lokal bekannten Datenelemente. Zusätzlich zum Weiterleiten neuer Datenelemente, wie in Abschnitt 5.2.1 beschrieben, können nun beim Empfang einer Nachricht folgende Situationen eintreten:

- Wenn in der empfangenen Nachricht mehr Hash-Werte enthalten sind als lokal bekannt sind, können die fehlenden Datenelemente angefragt werden. Zur Anfrage eines fehlenden Datenelements ist es ausreichend eine Liste mit den lokal bekannten Hash-Werten zu versenden. Denn dann tritt bei den empfangenden Knoten der folgende Fall ein.

- Fehlen einer empfangenen Nachricht lokal bekannte Hash-Werte, werden die (angefragten) Datenelemente, die offensichtlich einem benachbarten Knoten fehlen, als Antwort erneut gesendet.

Die periodisch versendeten Beacon-Nachrichten, die zur Erkennung der Nachbarschaftsgröße eingeführt wurden, dienen als implizite Anfragen, indem immer eine Liste mit den Hash-Werten der Datenelemente des sendenden Knotens angehängt wird.

Die Beacon-Nachrichten sind so ausgelegt, dass sie Veränderungen in der Nachbarschaft mit einer maximalen Abweichung von 10 % erkennen; dafür sendet ein Knoten immer dann eine Beacon-Nachricht, wenn er sich um 10 % seines Kommunikationsradius bewegt hat (vgl. Gleichung 5.2). Begegnen sich zwei Knoten, so wird im Mittel nach 5 % ihres Kommunikationsradius von einem der beiden Knoten die erste Beacon-Nachricht gesendet. So bleibt genügend Zeit für den nachfolgenden Austausch fehlender Datenelemente. Auch unterbrochene Flutprozesse und neu hinzugekommene Knoten werden durch die periodischen Beacon-Nachrichten zuverlässig erkannt.

Analog zum einfachen Fluten kann auch beim Senden von Anfragen und Antworten eine Überlastung der Funkschnittstelle in Netzen mit hoher Knotendichte auftreten. Wenn beispielsweise ein Knoten inmitten einer Nachbarschaft von 500 weiteren Knoten – die bereits einige Datenelemente ausgetauscht haben – eingeschaltet wird, so würden auf die erste Beacon-Nachricht des neuen Knotens alle 500 Knoten gleichzeitig antworten. Daher senden analog zum probabilistischen Fluten nicht alle Knoten die nötigen Antworten bzw. Anfragen. Stattdessen wird ein Knoten nur dann aktiv, wenn Gleichung 5.1, die bereits für die Entscheidungsfindung beim probabilistischen Fluten benutzt wird, wahr ist.

In dem Beispiel mit einem neuen Knoten inmitten von 500 weiteren Knoten, kennen alle 500 Knoten die Anzahl ihrer Nachbarn ($\#\,neighbors = 500$) und würden somit jeweils mit einer Wahrscheinlichkeit von $\frac{\#\,forwarder}{500 \cdot 0{,}4} = \frac{\#\,forwarder}{200}$ antworten. Insgesamt also $2{,}5 \cdot \#\,forwarder$ Knoten. In diesem extremen Fall sind die Knoten also etwas

zu mitteilsam, da die gesamte Nachbarschaft des anfragenden Knotens die benötigten Datenelemente kennt; im Falle eines unterbrochenen Flutprozesses oder wenn sich zwei Netzpartitionen aufeinander zu bewegen, ist die Abschätzung angemessen, dass nur 40 % der benachbarten Knoten die nötigen Datenelemente kennen, was zur gewünschten Anzahl von antwortenden bzw. anfragenden Knoten führt (*# forwarder*).

Um das Zusammenspiel von probabilistischem Fluten, Anfragen und Antworten zu koordinieren, verwendet *AutoCast* Prioritäten, die durch unterschiedliche Verzögerungen vor dem Senden entsprechender Nachrichten umgesetzt werden. Die Verzögerungen sind Vielfache von δ, was der maximalen Übertragungszeit einer (maximal großen) Nachricht entspricht. Jedes Mal, wenn eine Nachricht empfangen wird, beginnt für alle geplanten Aktionen die jeweilige Verzögerungszeit von Neuem. Somit wird einerseits eine Überlastung des Funkmediums verhindert und andererseits werden auch verschieden priorisierte Vorgänge benachbarter Knoten in die gewünschte Reihenfolge gebracht, d. h. es werden erst alle Aktionen mit einer Verzögerung von δ ausgeführt, bevor eine um 2δ verzögerte Aktion zur Ausführung kommt.

1. Wenn ein Knoten das Fehlen eines ihm bekannten Hash-Wertes in einer empfangenen Nachricht feststellt, so antwortet er nach 1δ mit dem fehlenden Datenelement. So kann dieses in nachfolgende Flutprozesse mit eingebunden werden.

2. Bevor ein Knoten ein Datenelement weiter flutet, wartet er 2δ.

3. Falls in einer empfangenen Nachricht mehr Hash-Werte enthalten sind, als lokal bekannt sind, so wird nach 3δ eine Anfrage in Form einer Beacon-Nachricht gesendet. Das explizite Senden einer Anfrage stellt allerdings die Ausnahme dar, weil schon jede reguläre Beacon-Nachricht die Liste der lokal bekannten Hash-Werte enthält und so einen mehr wissenden benachbarten Knoten implizit zum Senden des fehlenden Datenelements veranlasst.

Um die Anzahl gesendeter Nachrichten zu reduzieren, werden verschieden priorisierte Aktionen auf einem Knoten kombiniert, indem beim Senden einer Nachricht mit hoher Priorität, beispielsweise einer Antwort, niedriger priorisierte Aktionen in die Nachricht mit aufgenommen werden. Zudem werden anstehende Anfragen von Datenelementen abgebrochen, wenn die fehlenden Datenelemente schon innerhalb der Verzögerungszeit empfangen werden.

5.2.4 Verbreitungsgebiete

Einzelne Datenelemente sind im Allgemeinen nur für Knoten innerhalb eines bestimmten Verbreitungsgebiets von Interesse und nicht für alle Knoten innerhalb des gesamten Adhoc-Netzes. Um die Verbreitung von Datenelementen auf ein bestimmtes geografisches

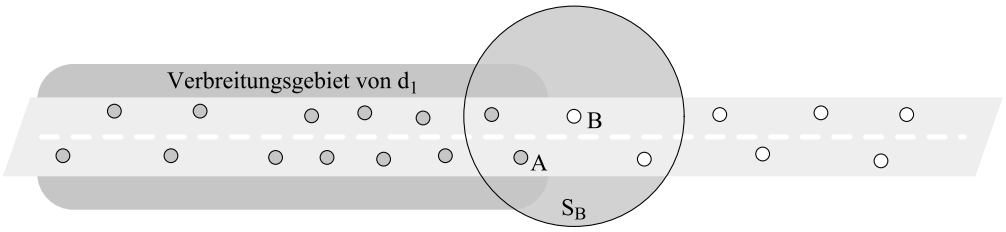


Abbildung 5.6: Verbreitungsgebiet eines Datenelements.

Gebiet begrenzen zu können, wird davon ausgegangen, dass alle Knoten ihre eigene Position bestimmen können. Aufgrund der Möglichkeiten, die bereits heutige Navigationssysteme mitbringen, kann man in VANETs nicht nur vom Vorhandensein geografischer Positionen auf Basis von GPS ausgehen, sondern auch eine Abbildung der geografischen Positionen auf das Straßennetz als gegeben annehmen.

Die Metadaten jedes Datenelements enthalten daher ein Verbreitungsgebiet, das z. B. kreisförmig sein kann, mit Mittelpunkt und Radius, oder durch einen Streckenabschnitt mit Straßen-Kennung sowie Start- und Endpunkt beschrieben sein kann.

Knoten, die innerhalb des Verbreitungsgebiets eines empfangenen Datenelements liegen, verarbeiten dieses, wie in den letzten Abschnitten beschrieben. Im Randgebiet eines Verbreitungsgebiets ergibt sich allerdings ein Problem, das Abbildung 5.6 zeigt. Angenommen ein Datenelement d_1 wurde bereits an alle Knoten innerhalb seines Verbreitungsgebiets verteilt. Knoten B, der sich außerhalb des Verbreitungsgebiets von d_1 befindet, sendet eine periodische Beacon-Nachricht mit allen ihm bekannten Hash-Werten. Knoten A, der sich noch im Verbreitungsgebiet von d_1 befindet, empfängt diese Nachricht und antwortet mit dem Datenelement d_1, welches B fehlt. Dieser verwirft jedoch das neu empfangene Datenelement, da er sich nicht in dessen Verbreitungsgebiet befindet. Dieser vergebliche Datenaustausch wiederholt sich ständig, wenn innerhalb eines Verbreitungsgebiets liegende Knoten mit Knoten außerhalb des Verbreitungsgebiets miteinander kommunizieren.

Zur Lösung dieses Problems pflegen die Knoten neben den Hash-Werten von Datenelementen, in deren Verbreitungsgebiet sie sich befinden, eine Liste „ungültiger“ Datenelemente und ihrer Hash-Werte. Im vorhergehenden Beispiel empfängt B das Datenelement d_1 von A. Da B nicht im Verbreitungsgebiet von d_1 liegt, speichert er d_1 in der Liste ungültiger Datenelemente. Datenelemente, die sich in dieser Liste befinden, werden bei der Generierung von Anfragen ausgelassen. Zudem werden die Hash-Werte der ungültigen Datenelemente jeder gesendeten Nachricht in einer separaten Liste angehängt, um potenzielle Antworten zu unterdrücken.

Sobald sich ein Knoten ins Verbreitungsgebiet eines bis dato ungültigen Datenelements bewegt, wird dieses verarbeitet, ohne dass eine weitere Kommunikation nötig ist. Das

heißt, das Datenelement wird an die übergeordnete (Applikations-)Schicht weitergegeben und aus der Liste ungültiger Datenelemente in die Liste bekannter Datenelemente verschoben.

Bewegt sich ein Knoten aus dem Verbreitungsgebiet eines Datenelements heraus, so wird dieses zunächst in die Liste ungültiger Datenelemente verschoben. Sobald sich ein Knoten um mehr als den doppelten Kommunikationsradius vom Verbreitungsgebiet des Datenelements entfernt hat, wird es lokal vollständig gelöscht.

Ebenso wie die räumliche Begrenzung des Verbreitungsgebiets zum beschriebenen unnötigen Datenaustausch führen kann, so können voneinander abweichende Uhren benachbarter Knoten zu einem unterschiedlichen Verständnis der zeitlichen Gültigkeit von Datenelementen führen – was wiederum unnötigen Austausch von Datenelementen zur Folge hat. Daher wird ein Datenelement auch beim Ablauf seiner Lebenszeit zunächst für eine gewisse Zeit in die Liste ungültiger Datenelemente aufgenommen, bevor es vollständig gelöscht wird. Diese Aufbewahrungszeit muss mindestens der maximalen Zeitabweichung zwischen allen Uhren der beteiligten Knoten entsprechen. Das zur Positionsbestimmung genutzte GPS erlaubt eine Zeitsynchronisation auf Basis des empfangenen GPS-Signals. Laut [17] lässt sich schon mit einem Standard-GPS-Empfänger eine Genauigkeit von $\pm 100\,\text{ns}$ erzielen. Eine Aufbewahrungszeit der Datenelemente von 1 s nach ihrer Lebenszeit ist daher bei einer Zeitsynchronisation über GPS hinreichend.

5.3 Implementierung

Dieser Abschnitt enthält eine kurze Beschreibung der im Rahmen dieser Arbeit angefertigten *AutoCast*-Implementierungen. Zunächst wird das verwendete Nachrichtenformat eingeführt. Im folgenden Abschnitt 5.3.2 wird der Protokollablauf übersichtsartig beschrieben. Abschließend findet sich in Abschnitt 5.3.3 eine Erläuterung des Protokollablaufs an einem Beispiel. Eine detaillierte Beschreibung der *AutoCast*-Implementierung befindet sich in Anhang C.

5.3.1 Nachrichtenformat

AutoCast verwendet für die Übertragung von Nachrichten ein einziges Nachrichtenformat. Dieses enthält einen Nachrichtenkopf gefolgt von drei Listen: Hash-Werte ungültiger Datenelemente, Hash-Werte aller bekannten Datenelemente sowie vollständige Datenelemente. Die Größe der jeweiligen Liste ist im Nachrichtenkopf vermerkt.

Zur Verwaltung der Nachbarschaften enthält der Nachrichtenkopf zudem die Kennung des Absenders, die Anzahl seiner Nachbarn und die Zeitdauer t_{beacon}, innerhalb derer der jeweilige Knoten seine nächste Beacon-Nachricht senden wird.

In jedem Fall enthält eine *AutoCast*-Nachricht die Hash-Werte aller bekannten und ungültigen Datenelemente. Wenn es sich ausschließlich um eine Beacon-Nachricht handelt,

sind keine vollständigen Datenelemente angehängt. Ansonsten folgen den Hash-Werten ein oder mehrere Datenelemente, die geflutet werden oder als Antwort auf eine (implizite) Anfrage dienen. Sollten einmal mehr Datenelemente zum Senden vorgemerkt sein als in eine Nachricht passen, werden zufällig so viele ausgewählt, wie die Nachricht aufnehmen kann.

5.3.2 Protokollablauf

Das *AutoCast*-Protokoll wurde in den diskreten ereignisgesteuerten Netzwerksimulatoren *Network Simulator 2* (ns-2) [38] und Shawn [42] implementiert. ns-2 simuliert die unteren Netzwerkschichten für die Bitübertragung und den Medienzugriff realitätsnäher und gilt als gut validierter Simulator. Shawn hingegen konzentriert sich mehr auf die verteilte Anwendung und modelliert nicht alle physikalischen Effekte, sondern nur deren Auswirkung. Daher lassen sich in Shawn im Vergleich zu ns-2 sehr viel größere Netze in kürzerer Simulationslaufzeit und mit weniger Arbeitsspeicherbedarf simulieren.

Für die in diesem Kapitel durchgeführten Evaluationen wird zunächst ns-2 verwendet, um die generelle Funktionsweise von *AutoCast* zu untersuchen (vgl. Abschnitt 5.4). Da ns-2 allerdings in der Simulationsgröße auf wenige tausend Knoten beschränkt ist, folgt in Abschnitt 5.4.6 eine Beschreibung der Implementierung in Shawn, die die Simulation von weit größeren Szenarien erlaubt. In der anschließenden Evaluation wird das mit Shawn ermittelte Protokollverhalten mit den Ergebnissen von ns-2 qualitativ verglichen und das Laufzeitverhalten beider Simulatoren untersucht.

Die genaue Implementierung inklusive Pseudocode findet sich aus Gründen der Übersichtlichkeit in Anhang C. Kurz gefasst lässt sich die Implementierung auf zwei Methoden zum Senden und Empfangen von Nachrichten reduzieren, die jeweils die vorgestellten Protokollbausteine auf Sender- und Empfängerseite umsetzen.

Die Methode zum Senden wird durch verschiedene Zeitgeber aufgerufen, wie am Ende von Abschnitt 5.2.3 beschrieben. Nachdem die Zugehörigkeit aller Datenelemente zu den bekannten bzw. ungültigen Datenelementen überprüft ist, wird eine *AutoCast*-Nachricht erstellt und als Broadcast-Nachricht gesendet. Schließlich wird ein Zeitgeber für das nächste Senden einer Beacon-Nachricht gestartet.

Beim Empfangen einer *AutoCast*-Nachricht werden zunächst alle Zeitgeber entsprechend ihrer Priorität verschoben. Falls sich der Knoten aufgrund seiner Nachbarschaftsgröße entscheidet, aktiv zu werden, werden die in der Nachricht enthaltenen neuen Datenelemente zum Fluten vorgemerkt. Falls in der Nachricht Hash-Werte von Datenelementen fehlen, die lokal bekannt sind, werden diese zum Antworten vorgemerkt. Falls darüber hinaus die empfangene Nachricht Hash-Werte enthält, die lokal nicht bekannt sind, so werden diese angefragt. Alle Aktionen werden gemäß ihrer Priorität per Zeitgeber geplant und führen beim Ablauf eines Zeitgebers zum Aufruf der Sendemethode.

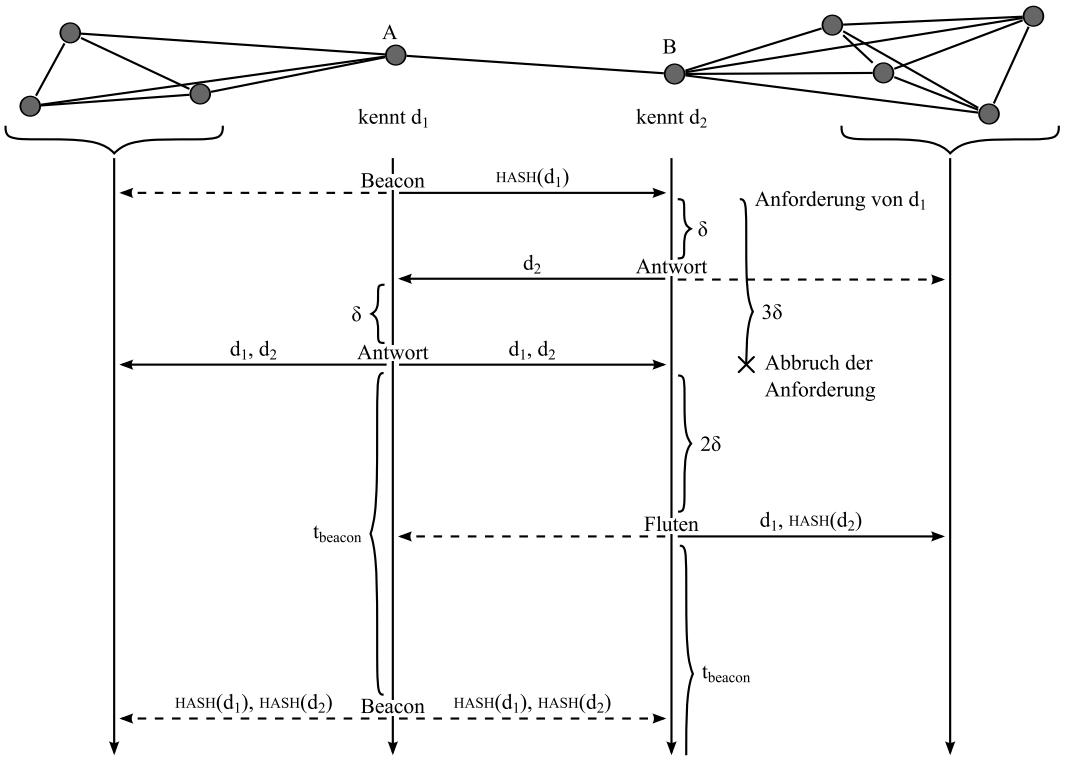


Abbildung 5.7: *AutoCast*-Protokoll – Weg-Zeit-Diagramm beim Aufeinandertreffen zweier Netzpartitionen.

5.3.3 Beispiel

Nach der Vorstellung des Nachrichtenformats sowie des Protokollablaufs illustriert das folgende Beispiel das Zusammenwirken der einzelnen Bausteine des *AutoCast*-Protokolls. Abbildung 5.7 zeigt eine Situation, in der sich zwei Netzpartitionen einander nähern. Die Knoten A und B, die sich in je einer der Netzpartitionen befinden, verbinden als Erste die beiden Partitionen.

Im Vorfeld wurde bereits das Datenelement d_1 vollständig an alle Knoten in As Partition verteilt, während Knoten B und alle weiteren Knoten in Bs Partition bereits das Datenelement d_2 kennen. Das Beispiel beschränkt sich auf die Nachrichten, die von den Knoten A und B versendet werden. Beide Knoten antworten und fluten mit einer Wahrscheinlichkeit von 100 %, da ihre Nachbarschaft ausreichend klein ist (vgl. Abschnitt 5.2.1).

Die erste Nachricht im Beispiel ist eine von Knoten A versendete Beacon-Nachricht. Sie enthält den Hash-Wert des Datenelements d_1 und stellt den ersten Kontakt zu Knoten

B und dessen Netzpartition her. B empfängt diese Nachricht, trägt A in seine Nachbarschaftsliste ein und stellt fest, dass Knoten A das Datenelement d_2 nicht kennt, während ihm selbst der Hash-Wert von d_1 unbekannt ist. Somit plant B nach einer Verzögerung δ eine Antwort zu versenden, die das Datenelement d_2 enthält. Ebenso plant B eine Anfrage nach 3δ, um das Datenelement d_2 in Erfahrung zu bringen.

Nach Ablauf der Verzögerung sendet B als Antwort das Datenelement d_2. Der Empfang dieser Nachricht löst bei Knoten A zwei Aktionen aus. Zum einen speichert er d_2 lokal und merkt es zum Fluten mit einer Verzögerung von 2δ vor. Zum anderen erkennt A das Fehlen des Hash-Wertes von d_1 in Bs Nachricht und plant eine Antwortnachricht mit einer Verzögerung von δ. Beide Aktionen werden nach Ablauf des Antwort-Zeitgebers zusammengefasst; die gesendete Nachricht enthält sowohl d_1 als auch d_2.

Datenelement d_2 wird daraufhin in As Netzpartition weiter verbreitet. Knoten B empfängt zum ersten Mal d_1, woraufhin er die Anfrage für dieses Datenelement abbricht, es lokal speichert und zum Fluten mit der Verzögerung 2δ vorsieht. Zur Vereinfachung wird die weitere Verbreitung der Datenelemente in den entsprechenden Netzpartitionen in diesem Beispiel nicht betrachtet.

Nach dem Austausch der Datenelemente d_1 und d_2 fahren die Knoten A und B fort mit dem Senden von Beacon-Nachrichten mit der nur von ihrer eigenen Geschwindigkeit abhängigen Verzögerung t_{beacon} (vgl. Abschnitt 5.2.2).

5.4 Evaluation

Zur Evaluation von *AutoCast* wurde eine umfangreiche Untersuchung durchgeführt, die nachfolgend beschrieben wird. Nach der Beschreibung des Evaluationsverfahrens wird *AutoCast* in Abschnitt 5.4.2 und 5.4.3 in zwei verschiedenen Szenarien getestet. Dem folgt eine Untersuchung von *AutoCast* hinsichtlich seiner Adaptivität (vgl. Abschnitt 5.4.4) und Robustheit (vgl. Abschnitt 5.4.5). Abschließend wird die Fähigkeit zweier Netzwerksimulatoren untersucht, *AutoCast* auch in großen VANET-Szenarien evaluieren zu können. Dafür werden sowohl Laufzeit und Speicherverbrauch als auch die Aussagekraft der Ergebnisse ausgewertet.

Zum Vergleich mit *AutoCast* und als Referenz dienen folgende drei Protokolle:

Einfaches Fluten: Jedes empfangene Datenelement wird von jedem Knoten ohne Verzögerung genau einmal weitergeleitet. Dieses Protokoll wird in der Literatur vielfach verwendet und ist sehr einfach aufgebaut.

Probabilistisches Fluten: Dieses Protokoll setzt das in Abschnitt 5.2.1 beschriebene probabilistische Fluten um, welches in voll konnektierten Netzen eine verminderte Datenrate gegenüber dem einfachen Fluten verspricht. Die lokale Knotendichte

bestimmt jeder Knoten gemäß Abschnitt 5.2.2. Die Anzahl weiterleitender Knoten ist gemäß den Ergebnissen aus Anhang B gesetzt auf $\#\,forwarder = 2$.

Theoretisches Optimum: Diese Implementierung verwendet globales Wissen, um einen Knoten ein Datenelement in genau dem Moment senden zu lassen, in dem ein neuer Knoten in seine Kommunikationsgebiet kommt, der das entsprechende Datenelement noch nicht kennt. Zur Erkennung benachbarter Knoten sind keine Beacon-Nachrichten nötig; vielmehr ist jedem Knoten die gesamte Netztopologie bekannt. Somit ist der Datenverkehr auf den reinen Austausch der Datenelemente beschränkt. Die Datenrate beträgt 1 MBit/s; es treten allerdings weder Übertragungsfehler noch Kollisionen auf dem Funkmedium auf. Verarbeitungszeiten auf einzelnen Knoten werden vernachlässigt. Ansonsten arbeitet dieses Protokoll rundenbasiert und überprüft alle 100 ms jeden Knoten auf neue benachbarte Knoten.

In der folgenden Leistungsbewertung wird das einfache Fluten die untere Schranke darstellen, während das theoretische Optimum eine obere Schranke für alle Protokolle ist. Als Leistungsparameter dienen die folgenden Kriterien:

Erfolgsquote: Diese Metrik gibt an, wie vielen der potenziellen Empfängerknoten ein Datenelement zugestellt werden konnte; sie ist somit ein direktes Maß für die Qualität der Datenverteilung. Der Wert wird ermittelt, indem für jedes Datenelement die Knoten ermittelt werden, die sich während der Lebenszeit im Verbreitungsgebiet befinden bzw. (kurz) befunden haben. Der Anteil dieser Knoten, die das entsprechende Datenelement empfangen haben, im Verhältnis zur Gesamtzahl ergibt den Anteil erfolgreich verteilter Datenelemente. Der Wert liegt zwischen 0 % und 100 %, wobei ein Wert von 100 % angestrebt wird, d. h. alle Datenelemente wurden an alle Knoten im Verbreitungsgebiet verteilt.

Zustellungsgeschwindigkeit: Diese Metrik zeigt die Geschwindigkeit, mit der Datenelemente vom ursprünglichen Sender ausgehend verteilt werden. Dieser Wert wird ermittelt, indem für jedes zum ersten Mal empfangene Datenelement die Distanz zwischen ursprünglichem Sender und dem empfangenden Knoten durch die Differenz zwischen Erzeugungs- und Empfangszeitpunkt berechnet wird. Am Ende der Simulation wird der Mittelwert über alle einzelnen Messungen gebildet. Diese in km/h angegebene Metrik erlaubt eine Abschätzung, wie schnell Daten durch das Ad-hoc-Netz transportiert werden können.

Übertragene Datenmenge: Dieser Leistungsparameter zeigt die Effizienz des Protokolls. Je geringer die übertragene Datenmenge, desto effizienter arbeitet das Protokoll. Die übertragene Datenmenge wird auf zwei Arten ausgewertet. Einerseits wird die **übertragene Datenmenge pro Knoten pro Sekunde** ermittelt. Durch den Bezug der übertragenen Datenmenge auf einen Knoten ist diese Metrik in

unterschiedlich großen Szenarien aussagekräftig. Zur Ermittlung der tatsächlichen Auslastung des Funkkanals wird die mittlere Nachbarschaftsgröße des jeweiligen Szenarios mit einbezogen. Des Weiteren wird die insgesamt **übertragene Datenmenge pro empfangenem Datenelement** bestimmt, um Mehraufwand zu erlauben, sofern er zur Erhöhung des Anteils erfolgreich verteilter Datenelemente führt.

5.4.1 Simulationsparameter

Für die im Folgenden beschriebene Leistungsbewertung des *AutoCast*-Protokolls wurde *AutoCast* für den diskreten ereignisgesteuerten Simulator *Network Simulator 2* (ns-2) [38] implementiert. Es wird ns-2 in der Version 2.30 mit der in ns-2 vorhandenen Implementierung des Übertragungsstandards IEEE 802.11 [71] verwendet. Daher ist die Datenrate für Broadcasts begrenzt auf 1 MBit/s; im weiteren Verlauf der Evaluation wird sich herausstellen, dass diese Vorgabe keine Einschränkung darstellt. Da die maximale Länge der Nutzdaten in IEEE 802.11 auf 2304 Byte begrenzt ist, benötigt die Übertragung der Nutzdaten rechnerisch maximal 23 ms. Die Verzögerungen der unteren Schichten inklusive Wartezeiten, Preambel und MAC-Header liegen unterhalb 1 ms. Die in Abschnitt 5.2.3 eingeführte Verzögerung δ, die im Allgemeinen größer sein soll als die maximal zur Übermittlung einer Nachricht benötigte Zeit, wird für die Evaluation aufgerundet auf $\delta = 30\,\text{ms}$. Auf alle Verzögerungszeiten wird eine zufällige gleichverteilte Abweichung von $(\pm 2,5\,\text{ms})$ addiert, um systematische Kollisionen auf dem Funkmedium in der Simulation einzuschränken.

Als Funkausbreitungsmodell dient das *Two-Ray-Ground*-Modell (vgl. [143, Kap. 3.6] und [46]), welches ein kreisförmiges Kommunikationsgebiet simuliert. Der Kommunikationsradius wird auf $r = 250\,\text{m}$ gesetzt, was beispielsweise laut [163, 173] als realistische Näherung angesehen werden kann. Eine Betrachtung des Protokolls unter Verwendung eines realistischeren Funkausbreitungsmodells folgt in Abschnitt 5.4.5.

Alle im Folgenden vorgestellten Simulationsergebnisse basieren auf Simulationsläufen von 1000 s Dauer (simulierte Zeit). Während dieser Zeit erzeugt eine fiktive dezentrale Anwendung sekündlich ein neues Datenelement, das im gesamten Simulationsszenario verteilt wird und jeweils eine Lebenszeit von 50 s hat. Nach einer Initialisierungsphase von 50 s werden somit bis zum Simulationsende gleichzeitig immer 50 gültige Datenelemente verteilt. Jedes Datenelement hat eine Größe von 200 Byte. Für die Berechnung der benötigten Datenrate wird nur die von *AutoCast* verursachte Datenmenge berücksichtigt. Der von IP sowie der 802.11 Medienzugriffs- und Bitübertragungsschicht verursachte Overhead wird vernachlässigt.

Tabelle 5.1: Mittlere Knotendichte im Standard-Szenario (quadratische Simulationsfläche).

Seitenlänge [m]	100	200	500	1000	1500	2000	2500	3000	3500	4000	4500	5000	5500	6000
Knotendichte	399,0	398,8	192,9	62,5	30,0	17,6	11,5	8,1	6,0	4,6	3,7	3,0	2,5	2,1
	One-Hop		Multi-Hop								Partitioniert			

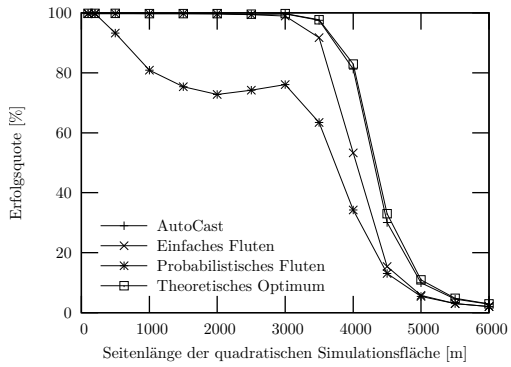
5.4.2 Standard-Szenario

In diesem Szenario bewegen sich die Knoten auf einer quadratischen Fläche nach dem *Gauß-Markov*-Bewegungsmodell, wie von Hellbrück in [66] vorgestellt. Die Knoten bewegen sich unkorreliert, beeinflussen sich also nicht gegenseitig. Das Bewegungsmuster der Knoten setzt sich wie folgt aus geradlinigen atomaren Bewegungen mit konstanter Geschwindigkeit v zusammen: Zu Beginn jeder atomaren Bewegung wird die Bewegungsrichtung α aus dem Intervall $[0; 360[$ berechnet zu $\alpha_n = (\alpha_{n-1} + x_n) \bmod 360$ mit x_n als normalverteilte Zufallszahl mit Mittelwert $\mu = 0$ und der Standardabweichung $\sigma = 250\,°$. Jede atomare Bewegung wird für die Zeitdauer $t_{move} = 10\,\mathrm{s}$ ausgeführt.

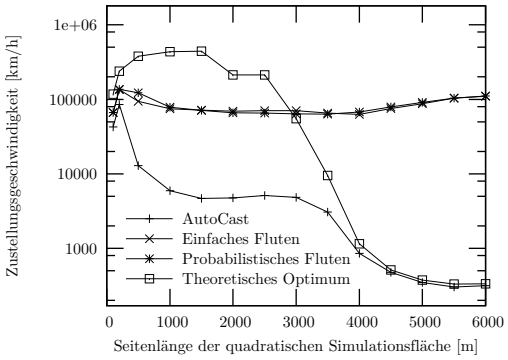
Erreicht ein Knoten die Begrenzung der Simulationsfläche, so wird seine Bewegungsrichtung an der Flächenbegrenzung reflektiert und der Knoten – wie eine an der Bande abprallende Billardkugel – zurück auf die Simulationsfläche gelenkt. Dieses Bewegungsmodell führt wie in [66] gezeigt zu einer stationären Gleichverteilung der Knoten auf der Simulationsfläche.

Um verschiedene Knotendichten innerhalb eines einzigen Szenarios zu simulieren, wird die Größe der quadratischen Simulationsfläche zwischen $100\,\mathrm{m} \times 100\,\mathrm{m}$ und $6000\,\mathrm{m} \times 6000\,\mathrm{m}$ variiert, während die Anzahl der Knoten konstant 400 beträgt. Aufgrund der unterschiedlich großen Simulationsfläche variiert die Knotendichte in den einzelnen Simulationsläufen entsprechend Tabelle 5.1. Zudem bilden sich aus der Knotendichte drei charakteristische Bereiche in denen unterschiedliche Strategien zur Datenverteilung nötig sind. Bei einer Simulationsfläche von $100\,\mathrm{m} \times 100\,\mathrm{m}$ und $200\,\mathrm{m} \times 200\,\mathrm{m}$ können alle Knoten direkt miteinander kommunizieren (One-Hop-Dichte); somit hat jeder Knoten 399 Nachbarn. Die richtige Strategie bei kleiner Simulationsfläche ist demnach probabilistisches Fluten. Je größer die Simulationsfläche wird, desto kleiner wird die mittlere Nachbarschaftsgröße. Bis zu einer Simulationsfläche von $4000\,\mathrm{m} \times 4000\,\mathrm{m}$ kann man das Netz noch als zusammenhängend betrachten (Multi-Hop-Dichte). Bei darüber hinausgehender Simulationsfläche zerfällt das Netz in mehrere Partition bis bei einer Fläche von $6000\,\mathrm{m} \times 6000\,\mathrm{m}$ jeder Knoten im Mittel nur zwei Nachbarn hat (partitionierte Dichte) – die Datenverteilung ist dann auf die Store-and-Forward Strategie angewiesen.

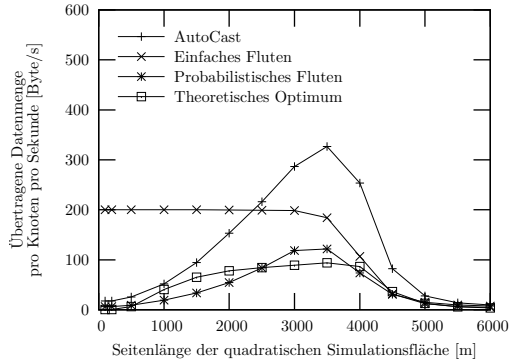
Die in Tabelle 5.1 angegebenen Knotendichten, wurden empirisch bestimmt und berücksichtigen den Effekt, dass Knoten am Rand der Simulationsfläche prinzipiell weniger Nachbarn haben als solche deren Kommunikationsgebiet komplett innerhalb der Simu-



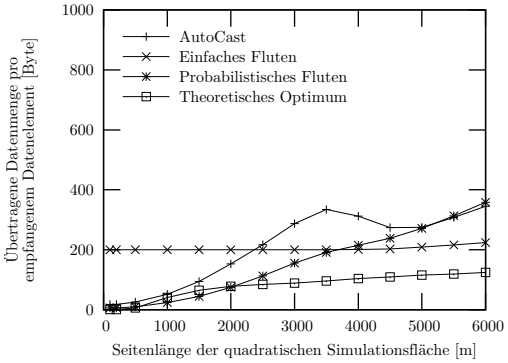
(a) Erfolgsquote



(b) Zustellungsgeschwindigkeit



(c) Übertragene Datenmenge pro Knoten pro Sekunde



(d) Übertragene Datenmenge pro empfangenem Datenelement

Abbildung 5.8: *AutoCast*-Leistungsbewertung im Standard-Szenario (Knotengeschwindigkeit $v = 1\,\mathrm{m/s}$).

lationsfläche liegt.

Die Simulationen wurden mit drei verschiedenen Knotengeschwindigkeiten durchgeführt ($v = 1\,\mathrm{m/s}$, $v = 10\,\mathrm{m/s}$ und $v = 25\,\mathrm{m/s}$), um das Protokollverhalten bei unterschiedlicher Dynamik zu erfassen. Für jeden der in Abschnitt 5.4 eingeführten Leistungsparameter wird je ein Diagramm gezeigt, das den Leistungsparameter in Abhängigkeit von der Größe der Simulationsfläche abbildet. Jede Kurve innerhalb dieser Diagramme stellt das Verhalten eines der betrachteten Protokolle dar.

Die Resultate werden in den nachfolgenden Abschnitten diskutiert.

Standard-Szenario mit geringer Dynamik (Knotengeschwindigkeit $v = 1\,\mathrm{m/s}$)

Abbildung 5.8 zeigt die Ergebnisse aller Simulationsläufe für das oben beschriebene

Standard-Szenario mit der Knotengeschwindigkeit $v = 1\,\mathrm{m/s}$.

Die Erfolgsquote (vgl. Abbildung 5.8a) liegt in kleinen Szenarien, in denen zur Datenverteilung keine Multi-Hop-Kommunikation notwendig ist, bei 100 %. Mit ansteigender Größe der Simulationsfläche fällt das probabilistische Fluten auf $\approx 70\,\%$ ab. Dieses Verhalten entspricht der Vorhersage aus Anhang B. Die übrigen Protokolle erreichen bis zu einer Simulationsfläche von $3000\,\mathrm{m} \times 3000\,\mathrm{m}$ eine Erfolgsquote von 100 %. Bei weiter zunehmender Größe der Simulationsfläche fällt die Erfolgsquote rasch ab. Die Erfolgsquote von *AutoCast* ist dabei über alle Simulationsflächen hinweg nahezu identisch mit dem theoretischen Optimum. In diesem Szenario mit geringer Knotengeschwindigkeit wird die Store-and-Forward Strategie nur selten gebraucht – so liegt auch das einfache Fluten nicht weit unterhalb von *AutoCast*.

Die in Abbildung 5.8b gezeigte Zustellungsgeschwindigkeit offenbart den grundsätzlichen Unterschied der beiden Fluten-Protokolle gegenüber *AutoCast* und theoretischem Optimum. Beim Fluten werden Datenelemente sofort nach dem Empfang weitergeleitet. Eine weitere Verzögerung durch späteres erneutes Versenden von Datenelementen tritt nicht auf. Vielmehr ist die Zustellungsgeschwindigkeit beim einfachen und probabilistischen Fluten ausschließlich von der Verzögerung beim Senden und Empfangen abhängig; die von den Protokollen verursachte Verarbeitungszeit auf den Knoten wird vernachlässigt. So ergibt sich beim Fluten eine Zustellungsgeschwindigkeit zwischen 63 000 km/h und 135 000 km/h; wobei die Zustellungsgeschwindigkeit zunächst mit der Größe der Simulationsfläche abnimmt, da immer mehr Hops zur Verbreitung der Datenelemente nötig sind. Im partitionierten Netz steigt die Zustellungsgeschwindigkeit erneut an, da die Datenelemente nicht mehr über die gesamte Simulationsfläche verteilt werden.

Das theoretische Optimum ermittelt die maximale Zustellungsgeschwindigkeit unter der Vorgabe, so viele Knoten wie möglich zu erreichen. Da im Single-Hop-Szenario alle Knoten direkt miteinander kommunizieren können, basiert die Zustellungsgeschwindigkeit auf einer einzigen Datenübertragung pro zu verteilendem Datenelement. Bei zunehmender Größe der Simulationsfläche müssen immer öfter Netzpartitionen durch Knotenbewegungen überbrückt werden, wodurch die Zustellungsgeschwindigkeit absinkt.

AutoCast erreicht im One-Hop-Szenario nahezu dieselbe Zustellungsgeschwindigkeit wie die Fluten-Protokolle. Sobald die Datenverteilung über mehrere Hops erfolgen muss, pendelt sich die Zustellungsgeschwindigkeit bei ca. 4500 km/h ein. Diese Begrenzung resultiert aus der um 2δ verzögerten Weiterleitung von Datenelementen. Im partitionierten Netz sinkt die Zustellungsgeschwindigkeit analog zum theoretischen Optimum.

Die von den Protokollen übertragenen Datenmengen sind in zwei Bezugsgrößen angegeben. Zum einen zeigt Abbildung 5.8c die von jedem Knoten im Durchschnitt übertragene Datenmenge pro Sekunde und zum anderen ist die übertragene Datenmenge pro empfangenem Datenelement in Abbildung 5.8d abgebildet. In beiden Fällen ist das theoretische Optimum unter Berücksichtigung der Erfolgsquote am effizientesten, da per Definition keinerlei Datenverkehr zur Nachbarschaftserkennung und zum Ermitteln des

Wissensstandes benachbarter Knoten nötig ist.

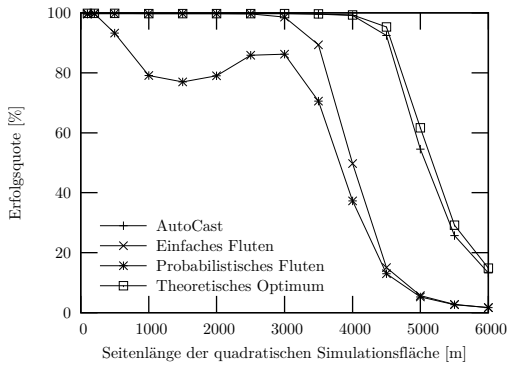
Abbildung 5.8c zeigt für das einfache Fluten zunächst eine konstante Datenmenge von 200 Byte/s, da jede Sekunde ein neues – 200 Byte großes – Datenelement verteilt, und somit von jedem Knoten genau einmal gesendet wird. Mit sinkender Erfolgsquote nimmt auch die Datenmenge ab, die jeder Knoten pro Sekunde zu übertragen hat. Das probabilistische Fluten zeigt sich bei One- und Multi-Hop-Dichten deutlich effizienter als das einfache Fluten, da durchschnittlich immer nur zwei Knoten weiterleiten und im Übrigen nicht alle Knoten im Szenario erreicht werden. Auch *AutoCast* ist in kleinen Szenarien sehr effizient; mit ansteigender Größe der Simulationsfläche und zunehmender Partitionierung des Netzes müssen Datenelemente aber immer öfter wiederholt übertragen werden, um eine hohe Erfolgsquote zu erzielen. Das Maximum ist bei einer Simulationsfläche von 3500 m × 3500 m erreicht, bei der pro Knoten eine Datenmenge von 327 Byte/s übertragen werden. Im partitionierten Netz können die Datenelemente nicht mehr an alle Knoten verteilt werden. Die übertragene Datenmenge pro Knoten fällt daher wieder deutlich ab.

Um zu verdeutlichen wie viel Aufwand die Protokolle im Mittel betreiben, um ein Datenelement an einen Knoten auszuliefern, bezieht Abbildung 5.8d die insgesamt übertragene Datenmenge auf die Anzahl empfangener Datenelemente. Bis zu einer Simulationsfläche von 3000 m × 3000 m sind die Ergebnisse analog zu denen aus Abbildung 5.8c, da in diesen Szenarien eine Erfolgsquote von 100 % erreicht wird. Nur das probabilistische Fluten erreicht hier eine geringere Erfolgsquote, was im Vergleich zu Abbildung 5.8c zu einem leicht erhöhten Kurvenverlauf führt. Mit sinkender Erfolgsquote für größere Szenarien zeigt diese Auswertung, dass *AutoCast* sowie das probabilistische Fluten einen zunehmend höheren Aufwand pro empfangenem Datenelement betreiben. Die Ursache hierfür ist die zunehmende Gewichtung der Nachbarschaftserkennung und das wiederholte Senden von Datenelementen. Im Gegensatz zum probabilistischen Fluten führt dieser Mehraufwand bei *AutoCast* zu einer nahezu idealen Erfolgsquote in Bezug auf das theoretische Optimum.

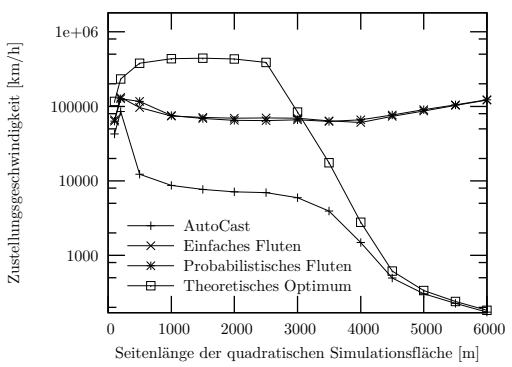
Standard-Szenario mit mittlerer Dynamik (Knotengeschwindigkeit $v = 10\,\mathrm{m/s}$)

In dieser Simulationsreihe soll gegenüber den Simulationen aus dem vorhergehenden Abschnitt die Dynamik im Netz ansteigen. Zu diesem Zweck wird die Knotengeschwindigkeit von $v = 1\,\mathrm{m/s}$ auf $v = 10\,\mathrm{m/s}$ erhöht. Durch die höhere Mobilität der einzelnen Knoten ist zu erwarten, dass *AutoCast* einen verstärkten Nutzen aus der Store-and-Forward-Strategie erzielen kann.

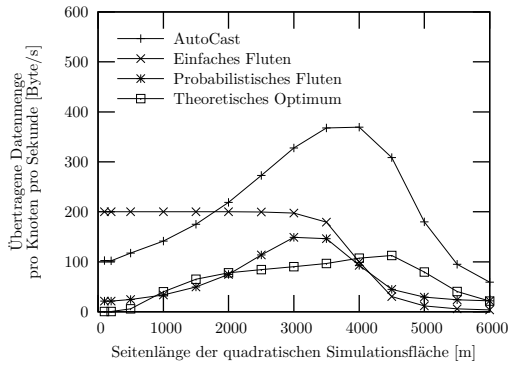
Die Simulationsergebnisse für eine Knotengeschwindigkeit von 10 m/s sind in Abbildung 5.9 zusammengefasst. Die einzelnen Kurven zeigen die gleichen Leistungsparameter, die bereits im vorigen Abschnitt besprochen wurden. Daher konzentriert sich der folgende Abschnitt auf die Änderungen, die sich durch die erhöhte Mobilität ergeben. Für



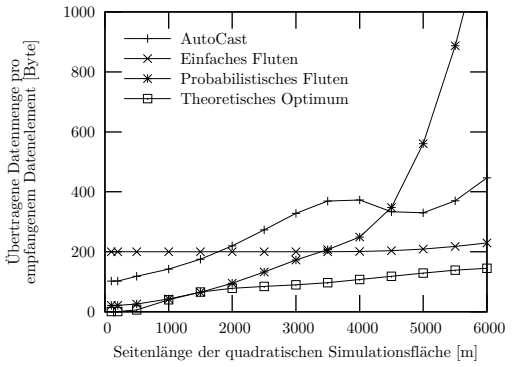
(a) Erfolgsquote



(b) Zustellungsgeschwindigkeit



(c) Übertragene Datenmenge pro Knoten pro Sekunde



(d) Übertragene Datenmenge pro empfangenem Datenelement

Abbildung 5.9: *AutoCast*-Leistungsbewertung im Standard-Szenario (Knotengeschwindigkeit $v = 10\,\mathrm{m/s}$).

eine bessere Vergleichbarkeit sind auch die Skalierungen der Achsen identisch gegenüber denen in Abbildung 5.8.

Die in Abbildung 5.9a dargestellte Erfolgsquote zeigt den prognostizierten Zugewinn für das *AutoCast*-Protokoll. Während bei einer Knotengeschwindigkeit von $v = 1\,\mathrm{m/s}$ und einer Simulationsfläche von $4500\,\mathrm{m} \times 4500\,\mathrm{m}$ nur 30 % der Knoten erreicht werden, beträgt mit der jetzt betrachteten Knotengeschwindigkeit von $v = 10\,\mathrm{m/s}$ die Erfolgsquote auf derselben Simulationsfläche 92 %. Dabei bleibt *AutoCast* wieder sehr dicht unter der durch das theoretische Optimum ermittelten maximalen Erfolgsquote. Gegenüber den Fluten-Protokollen ist *AutoCast* bei zunehmender Größe der Simulationsfläche durch die Store-and-Forward-Strategie deutlich im Vorteil. Das probabilistische Fluten zeigt bei einer Simulationsfläche von $3000\,\mathrm{m} \times 3000\,\mathrm{m}$ ein deutliches lokales Maximum,

das in geringerem Umfang auch schon in Abbildung 5.8a zu erkennen ist. Es ist ein Hinweis darauf, dass die Nachbarschaftserkennung bei zunehmender Geschwindigkeit die tatsächliche Knotendichte leicht unterschätzt. Somit beteiligen sich im Mittel mehr als zwei Knoten am Weiterleiten, was zu einer erhöhten Erfolgsquote führt. Laut Abbildung B.2 liegt der Erwartungswert für probabilistisches Fluten bei etwa 70 %.

Die Zustellungsgeschwindigkeit (vgl. Abbildung 5.9b) erhöht sich für gut konnektierte Szenarien (bis 3000 m × 3000 m) auf durchschnittlich 7000 km/h. Gegenüber dem Szenario mit geringer Mobilität sinkt die Geschwindigkeit allerdings bei weiter zunehmender Größe der Simulationsfläche etwas mehr ab. Die Ursache dafür ist der größere Anteil an Datenelementen, die zugestellt werden nachdem sie von einem Knoten mit einer Geschwindigkeit von 10 m/s über die Simulationsfläche bewegt wurden.

Die in Abbildung 5.9c und 5.9d gezeigten übertragenen Datenmengen weisen die gleiche Charakteristik auf wie im vorhergehenden Abschnitt. Erwähnenswert ist allerdings die im Allgemeinen größere übertragene Datenmenge des *AutoCast*-Protokolls. Hier zeigen sich einerseits die Kosten für die höhere Erfolgsquote für große Simulationsflächen (d. h. für geringere Knotendichten) und andererseits der erhöhte Aufwand der Nachbarschaftserkennung gegenüber dem Szenario mit geringer Dynamik. Wie in Abschnitt 5.2.2 beschrieben, besteht ein linearer Zusammenhang zwischen der Knotengeschwindigkeit und der Häufigkeit mit der Beacon-Nachrichten gesendet werden. Gegenüber Abbildung 5.8c wäre somit eine um den Faktor 10 höhere übertragene Datenmenge zu erwarten. Dass denoch die übertragene Datenmenge pro Knoten im One-Hop-Netz nur von 18 Byte/s auf 102 Byte/s steigt und die maximal übertragene Datenmenge pro Knoten nur 370 Byte/s beträgt – also ähnlich wie bei $v = 1\,\mathrm{m/s}$ – liegt daran, dass Beacon-Nachrichten nur dann gesendet werden müssen, wenn keine weiteren Aktionen (*Antworten*, *Fluten* bzw. *Anfragen*) nötig sind.

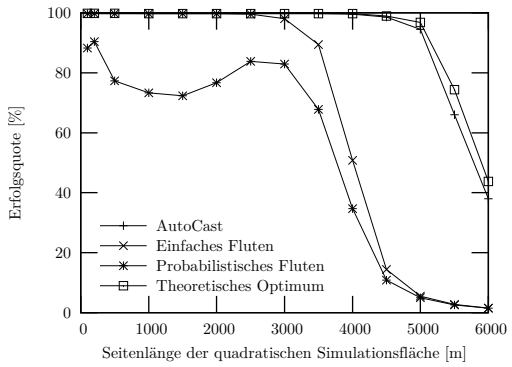
Auch in Bezug auf empfangene Datenelemente wird von *AutoCast* etwas mehr Datenverkehr erzeugt (vgl. Abbildung 5.9d). Der erhöhte Aufwand der Nachbarschaftserkennung bei gleichzeitig sehr niedriger Erfolgsquote sorgt beim probabilistischen Fluten für einen überproportionalen Kurvenverlauf für partionierte Netze.

Standard-Szenario mit hoher Dynamik (Knotengeschwindigkeit $v = 25\,\mathrm{m/s}$)

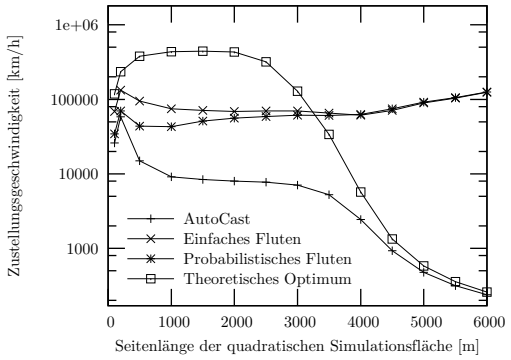
In der dritten Simulationsreihe wurde erneut die Dynamik im Netz erhöht. Dazu wurde die Knotengeschwindigkeit auf $v = 25\,\mathrm{m/s}$ gesetzt, was ungefähr $v = 90\,\mathrm{km/h}$ entspricht. In der Realität ist zwar mit dem gewählten Mobilitätsmodell – bei dem sich die Knoten unkorreliert in beliebiger Richtung bewegen – eine derart hohe Geschwindigkeit nicht zu erwarten, dafür dient dieses Setup als Überleitung zum im nachfolgenden besprochenen Autobahn-Szenario.

Die Evaluationsergebnisse dieser Simulationsreihe zeigt Abbildung 5.10.

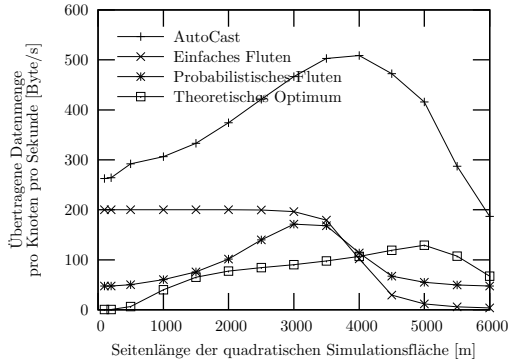
Wie in Abbildung 5.10a zu sehen ist, sorgt die abermals erhöhte Knotengeschwin-



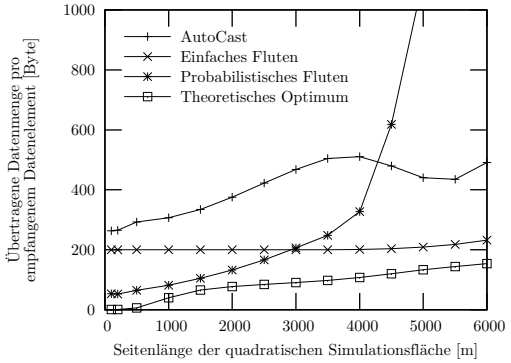
(a) Erfolgsquote



(b) Zustellungsgeschwindigkeit



(c) Übertragene Datenmenge pro Knoten pro Sekunde



(d) Übertragene Datenmenge pro empfangenem Datenelement

Abbildung 5.10: *AutoCast*-Leistungsbewertung im Standard-Szenario (Knotengeschwindigkeit $v = 25\,\mathrm{m/s}$).

digkeit für einen weiteren Anstieg der Erfolgsquote für große Simulationsflächen. Dabei erreicht *AutoCast* erneut nahezu den vom theoretischen Optimum ermittelten Maximalwert. Die Erfolgsquote der Fluten-Protokolle verändert sich hingegen nur minimal, da diese Protokolle nicht von der Mobilität der Knoten profitieren.

Die in Abbildung 5.10b dargestellte Zustellungsgeschwindigkeit unterscheidet sich nur wenig von der im vorigen Abschnitt. Es ist lediglich ein weiterer leichter Anstieg der Zustellungsgeschwindigkeit des *AutoCast*-Protokolls festzustellen. Beim Vergleich der beiden Fluten-Protokolle zeigt sich, dass das einfache Fluten in kleinen Szenarien die Datenelemente etwas schneller verteilt als das probabilistische Fluten. Dieser Effekt resultiert aus der zufälligen Auswahl der weiterleitenden Knoten beim probabilistischen Fluten, was im Vergleich zum einfachen Fluten zu einer geringeren mittleren Distanz

zwischen zwei weiterleitenden Knoten führt.

Die von *AutoCast* übertragene Datenmenge nimmt durch die höhere Knotengeschwindigkeit und der sich daraus ergebenden höheren Frequenz von Beacon-Nachrichten erneut zu (vgl. Abbildung 5.10c). Die pro Knoten maximal übertragene Datenmenge von 509 Byte/s wird beim Übergang vom Multi-Hop zum partitionierten Netz erreicht. Die gleiche Tendenz ist in Abbildung 5.10d festzustellen. Die von *AutoCast* übertragene Datenmenge pro Datenelement steigt aufgrund der höheren Anzahl an Beacon-Nachrichten unabhängig von der Größe der Simulationsfläche an, wobei die Charakteristik des Kurvenverlaufs erhalten bleibt.

Standard-Szenario – Zusammenfassung

Die Auswertung der Leistungsparameter des *AutoCast*-Protokolls im Standard-Szenario offenbart die folgenden Protokoll-Eigenschaften, die aufzeigen, dass die am Anfang von Kapitel 5 postulierten Anforderungen an das *AutoCast*-Protokoll erfüllt werden:

- *AutoCast* ist in der Lage ein Datenelement an nahezu alle Knoten auszuliefern, die sich in dessen Verbreitungsgebiet befinden. Dabei werden mobile Knoten zur Überbrückung von Netzpartitionen genutzt.
- In gut konnektierten Netzen wird eine Zustellungsgeschwindigkeit von mehr als 4500 km/h erreicht. Informationen lassen sich somit innerhalb von 80 s über eine Distanz von 100 km befördern. Zerfällt das Netz in mehrere Partitionen, ist die Zustellungsgeschwindigkeit maßgeblich von der Knotengeschwindigkeit abhängig.
- Die von *AutoCast* verursachte Datenrate ist abhängig von der individuellen Knotengeschwindigkeit sowie der lokalen Knotendichte. Je höher die Geschwindigkeit eines Knotens ist, desto häufiger sendet dieser Knoten eine Beacon-Nachricht. In Bezug auf die Knotendichte ist die höchste Datenrate zu erwarten, wenn ein Knoten 4 bis 6 Nachbarn besitzt (vgl. Tabelle 5.1 für ein Szenario zwischen 3500 m × 3500 m und 4000 m × 4000 m). Erhöht sich die Knotendichte, nimmt die Aktivität jedes einzelnen Knotens ab. Fällt die Knotendichte unter 3, so ist auch bei einer hohen Knotengeschwindigkeit die Datenverteilung nicht gewährleistet.

5.4.3 Autobahn-Szenario

Nachdem sich *AutoCast* im vorangegangenen Abschnitt in einem Standard-Szenario aus dem MANET-Umfeld bewährt hat, werden im Folgenden dieselben Leistungsparameter für ein typisches VANET-Szenario bestimmt. In diesem Fall stellen die Fahrzeuge die Knoten des Ad-hoc-Netzes dar. Das Szenario umfasst einen Autobahnabschnitt von 10 km Länge, der in beide Fahrtrichtungen zweispurig ausgelegt ist. Die Geschwindigkeit ist beschränkt auf 100 km/h. Die Verkehrsdichte beträgt ungefähr 14 Fahrzeuge pro

Kilometer und Fahrspur, was laut Handbuch für die Bemessung von Straßenverkehrsanlagen [48] zwar einer hohen Auslastung entspricht, allerdings noch keinen Verkehrsstau provoziert.

Die Bewegung der Fahrzeuge basiert auf dem von Krauß in [95] vorgestellten Fahrzeugfolgemodell, das im mikroskopischen Verkehrssimulator SUMO (Simulator of Urban Mobility, [93]) implementiert ist. Die Bewegungen aller Fahrzeuge werden in atomare lineare Bewegungen von jeweils einer Sekunde Dauer zerlegt und in einer Bewegungsdatei zwischengespeichert.

Durch die hohe Geschwindigkeit von durchschnittlich 25 m/s ($\approx$ 90 km/h) weist dieses Autobahn-Szenario eine vergleichbare Dynamik auf wie das zuletzt betrachtete Standard-Szenario mit hoher Dynamik. Im Gegensatz zum Standard-Szenario mit quadratischer Simulationsfläche bewegen sich die Knoten jetzt auf festen Fahrspuren; durch die Bildung von Fahrzeugkolonnen zerfällt das Netz deutlich eher in Partitionen als beim Random-Direction-Bewegungsmodell.

Um auch in diesem Szenario Aussagen über verschiedene Knotendichten – und somit auch über das Protokollverhalten während der Markteinführung – treffen zu können, kommuniziert nur ein bestimmter Anteil der Fahrzeuge miteinander. Dieser Anteil an miteinander kommunizierenden Fahrzeugen im Verhältnis zur Gesamtanzahl der Fahrzeuge wird im Folgenden als Ausstattungsgrad bezeichnet. Er variiert zwischen 5 % und 100 %. Bei einem Ausstattungsgrad von 5 % liegt die mittlere Nachbarschaftsgröße bei 0,75, also noch geringer als im 6000 m $\times$ 6000 m großen Standard-Szenario. Ein Ausstattungsgrad von 100 % resultiert in einer mittleren Nachbarschaftsgröße von 15.

Um die beiden Szenarien in Hinblick auf die Knotendichte vergleichen zu können, ist eine Zuordnung nötig zwischen den Simulationsflächen im Standard-Szenario und den im Autobahn-Szenario verwendeten Ausstattungsgraden. Abbildung 5.11 zeigt entsprechend die mittleren Knotendichten, die sich im Autobahn-Szenario aus unterschiedlichen Ausstattungsgraden ergeben. Zudem sind die in Hinblick auf die Knotendichte korrelierenden Simulationsflächen in der Grafik aufgetragen. Dabei zeigt sich, dass ein Ausstattungsgrad von nahezu 100 % der 2000 m $\times$ 2000 m großen Simulationsfläche entspricht, während eine 6000 m $\times$ 6000 m große Simulationsfläche mit einem Ausstattungsgrad von gut 10 % vergleichbar ist. Die im Folgenden vorgestellten Diagramme des Autobahn-Szenarios entsprechen somit der rechten Hälfte der Standard-Szenario-Diagramme.

Die Ergebnisse dieser Simulationsreihe sind in Abbildung 5.12 dargestellt. Wieder wurden die am Anfang von Abschnitt 5.4 vorgestellten vier Leistungsparameter ausgewertet und in je einem Diagramm aufgetragen.

Bei der Erfolgsquote (vgl. Abbildung 5.12a) markiert das theoretische Optimum wieder per Definition die Obergrenze des technisch Machbaren. Laut theoretischem Optimum können ab einem Ausstattungsgrad von 30 % mehr als 95 % der ausgestatteten Fahrzeuge erreicht werden. Ein direkter Vergleich zum Standard-Szenario mit hoher Dynamik suggeriert hingegen, dass bereits ab einem Ausstattungsgrad von 17 % (entspricht einer

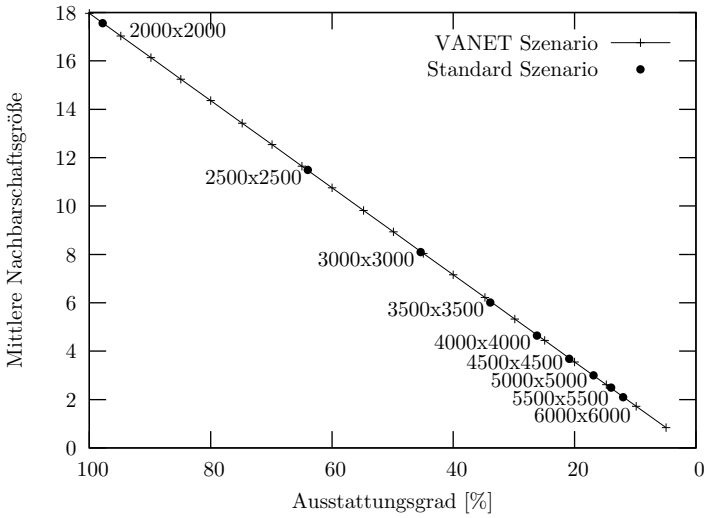
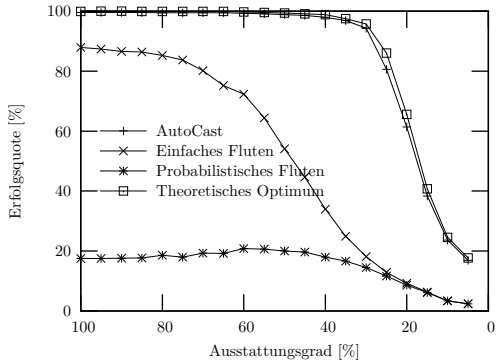


Abbildung 5.11: Mittlere Knotendichten im Autobahn-Szenario und Vergleich zum Standard-Szenario.

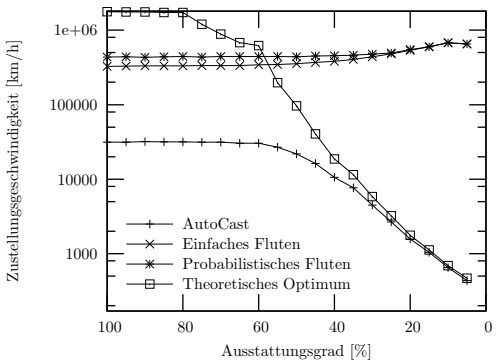
Simulationsfläche von 5000 m × 5000 m) eine Erfolgsquote von 95 % zu erwarten sei. Hier zeigt sich jedoch der Einfluss des Bewegungsmodells, was die Notwendigkeit untermauert *AutoCast* im Autobahn-Szenario gesondert zu untersuchen.

Die von *AutoCast* erreichte Erfolgsquote zeigt trotz der – durch das Szenario bedingten – etwas schlechteren Grundbedingungen, dass die Datenelemente an nahezu alle erreichbaren Fahrzeuge verteilt werden. Bei der Betrachtung der Fluten-Protokolle fällt auf, dass das Netz auch bei einem Ausstattungsgrad von 100 % (entspricht einer Simulationsfläche von 2000 m × 2000 m) partitioniert ist; weswegen das einfache Fluten nur eine Erfolgsquote von maximal 88 % und das probabilistische Fluten sogar nur 21 % erreicht. Im Autobahn-Szenario ist also die Store-and-Forward Strategie zwingend erforderlich, genauso wie die Einbeziehung des Gegenverkehrs in die Datenverteilung, selbst wenn die dort fahrenden Fahrzeuge keinen Nutzen aus dem Inhalt der Datenelemente ziehen können.

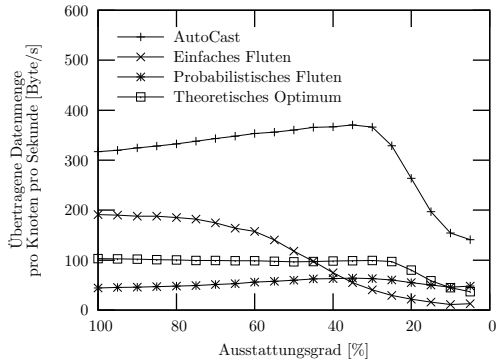
Durch die nahezu eindimensionale Form der Simulationsfläche werden Zustellungsgeschwindigkeiten erreicht, die um den Faktor vier bis fünf höher liegen als im Standard-Szenario (vgl. Abbildung 5.12b). Bei den Fluten-Protokollen liegt die Zustellungsgeschwindigkeit über alle Ausstattungsgrade über 300 000 km/h, da Datenelemente ausschließlich innerhalb einer Flutwelle zugestellt werden. Dass dieses Verfahren bei Weitem nicht ausreichend ist, zeigt Abbildung 5.12a. *AutoCast* erreicht ab einem Ausstattungsgrad von 60 % eine Zustellungsgeschwindigkeit von über 30 000 km/h, die von der Wahl der Weiterleitungsverzögerung δ bestimmt wird. Mit sinkendem Ausstattungsgrad steigt



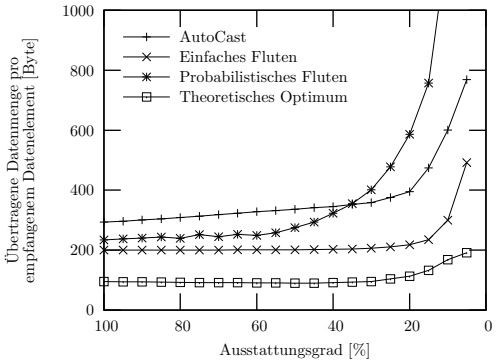
(a) Erfolgsquote



(b) Zustellungsgeschwindigkeit



(c) Übertragene Datenmenge pro Knoten pro Sekunde



(d) Übertragene Datenmenge pro empfangenem Datenelement

Abbildung 5.12: *AutoCast*-Leistungsbewertung im VANET-Szenario (Knotengeschwindigkeit ca. 25 m/s).

die Partitionierung des Netzes und folglich der Anteil der Store-and-Forward Strategie. Bei sehr niedrigem Ausstattungsgrad wird die Zustellungsgeschwindigkeit im Wesentlichen bestimmt durch die Geschwindigkeit der Fahrzeuge; so wird bei einem Ausstattungsgrad von 5 % eine Zustellungsgeschwindigkeit von 435 km/h erreicht. Der Verlauf der Kurve des theoretischen Optimums legt nahe, dass dies eine Beschränkung des zugrundeliegenden Netzes ist.

Bezüglich der übertragenen Datenmenge zeigen Abbildung 5.12c und 5.12d auf Basis des Standard-Szenarios erwartete Resultate. Allerdings fällt die *AutoCast*-Kurve für die übertragene Datenmenge pro Knoten etwas niedriger aus als im Standard-Szenario mit hoher Dynamik (max. 370 Byte/s bei einem Ausstattungsgrad von 35 %). Hier wirkt sich offenbar die Form des Szenarios und speziell die Bildung von Clustern durch das Fahren

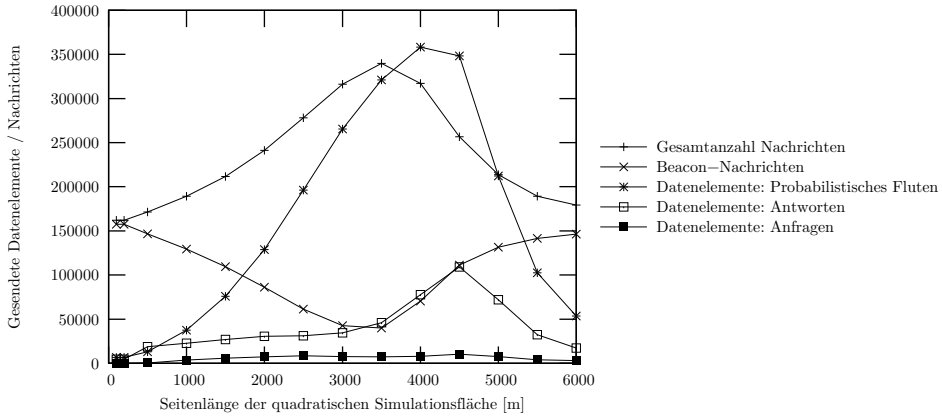


Abbildung 5.13: *AutoCast*– Ursachen des Nachrichtenaustauschs (Standard-Szenario, Two Ray Ground Model, Knotengeschwindigkeit 10 m/s).

in Kolonnen positiv aus.

Zusammenfassend zeigt sich auch im Autobahn-Szenario, dass *AutoCast* die Zielvorgabe, möglichst alle Fahrzeuge zu erreichen, erfüllt. Auch die Zustellungsgeschwindigkeit liegt auf einem guten Niveau. Einerseits wird sie von der Verzögerung δ beeinflusst, die auch für einen sinnvollen Trade-Off zwischen der Zustellungsgeschwindigkeit und der übertragenen Datenmenge sorgt. Andererseits sorgt bei niedrigem Ausstattungsgrad die Fahrgeschwindigkeit sowie der Datenaustausch mit dem Gegenverkehr dafür, dass Datenelemente mindestens mit der Geschwindigkeit der beteiligten Fahrzeuge verteilt werden. Diesen Aspekt sollte man bei der Entwicklung dezentraler Anwendungen im Straßenverkehr beachten.

5.4.4 Adaptivität

Die folgende Auswertung befasst sich mit der Adaptivität des *AutoCast*-Protokolls. Zu diesem Zweck werden die Gründe untersucht, die zum Senden von Nachrichten bzw. Datenelementen führen. Diese Auswertung wurde im Standard-Szenario mit mittlerer Dynamik (Knotengeschwindigkeit $v = 10\,\mathrm{m/s}$) durchgeführt, da dieses Szenario eine größere Varianz der Knotendichte aufweist als das Autobahn-Szenario.

Für *AutoCast* gibt es folgende Gründe zum Senden von Nachrichten bzw. Datenelementen: Beacon-Nachrichten zur Nachbarschaftserkennung (vgl. Abschnitt 5.2.2), sowie die in Abschnitt 5.2.1 und 5.2.3 vorgestellten ereignisgesteuerten Aktionen *probabilistisches Fluten*, *Antworten* und *Anfragen*. Jede dieser Aktionen wird durch eine Kurve in Abbildung 5.13 dargestellt. Die x-Achse dieser Abbildung zeigt die Seitenlänge der quadratischen Simulationsfläche. Da sich immer 400 Knoten mit einer Geschwindigkeit

von $v = 10\,\mathrm{m/s}$ in zufällige Richtungen auf der Simulationsfläche bewegen, hat die Größe der Simulationsfläche direkten Einfluss auf die Knotendichte (vgl. Abschnitt 5.4.2 und Tabelle 5.1). Auf der y-Achse ist die Anzahl der während eines Simulationslaufs von 1000 s Dauer von *AutoCast* gesendeten Nachrichten bzw. Datenelemente aufgetragen.

Die gesendeten Nachrichten bzw. Datenelemente werden wie folgt aufsummiert: Beim Ablaufen einer der Zeitgeber für probabilistisches Fluten, Antworten oder Anfragen, wird nicht nur der Zähler der entsprechenden Aktion hochgezählt, sondern auch weitere geplante Aktionen berücksichtigt, die implizit in der gesendeten Nachricht enthalten sind. Angenommen, auf einem Knoten veranlasst der Antwort-Zeitgeber mit einer Verzögerung von δ das Senden einer Nachricht und zudem sind noch das Fluten einiger Datenelemente nach 2δ sowie das Anfragen weiterer Datenelemente nach 3δ geplant; in diesem Fall werden die Zähler aller drei Aktionen entsprechend der betroffenen und in der gesendeten Nachricht enthaltenen Datenelemente hochgezählt. Der Zähler für Beacon-Nachrichten wird ausschließlich beim Ablauf des Beacon-Zeitgebers hochgezählt. An dieser Stelle sei daran erinnert, dass jede gesendete Nachricht den Beacon-Zeitgeber zurücksetzt und, dass jede gesendete Nachricht mehr als ein Datenelement enthalten kann, womit die Zählung von Datenelementen die Anzahl gesendeter Nachrichten übersteigen kann.

Abbildung 5.13 zeigt die Anzahl der gesendeten Nachrichten unter Verwendung des in Abschnitt 5.4.1 vorgestellten Two-Ray-Ground-Funkausbreitungsmodells. Falls alle Knoten direkt miteinander kommunizieren können (Simulationsfläche von $100\,\mathrm{m} \times 100\,\mathrm{m}$), wird der Großteil der Nachrichten als reine Beacon-Nachrichten gesendet, welche keine Datenelemente enthalten. Aufgrund der konstanten Knotengeschwindigkeit von $v = 10\,\mathrm{m/s}$ und einem Kommunikationsradius von $r = 250\,\mathrm{m}$ ergibt sich gemäß Gleichung 5.2 $t_{beacon} = 2,5\,\mathrm{s}$. Bei 400 Knoten und einer simulierten Zeit von 1000 s müssten somit 160 000 Beacon-Nachrichten gesendet werden. Tatsächlich sind es 157 458 Beacon-Nachrichten. Die übrigen Beacon-Nachrichten werden durch das probabilistische Fluten von insgesamt 1000 Datenelementen unterdrückt. Insgesamt verursacht das probabilistische Fluten das Senden von 6547 Datenelementen. Davon entfallen 1000 Datenelemente auf das initiale Senden der Datenelemente; das erste Weiterleiten verursacht somit das Senden von 5547 Datenelementen, d. h. beim ersten Weiterleiten eines Datenelements beteiligen sich im Mittel rund 5,5 Knoten. Dies scheint der Wahl von $\#\,forwarder = 2$ zu widersprechen, bei genauerer Betrachtung offenbart sich allerdings ein willkommener Seiteneffekt bei der Abschätzung der weiterleitenden Knoten: Beim ersten Aussenden eines Datenelements trifft die Annahme nicht zu, dass 60 % der benachbarten Knoten das Datenelement bereits kennen. Daher leiten im Mittel nicht zwei, sondern fünf Knoten das Datenelement weiter, was im besten Fall zu einer sternförmigen Ausbreitung des neuen Datenelements führt. Dass im Mittel 5,5 statt 5 Knoten ein Datenelement weitergeleitet haben, legt nahe, dass einige Beacon-Nachrichten kollidiert sind und folglich die Knotendichte leicht unterschätzt wurde.

Bei zunehmender Größe der Simulationsfläche muss ein Datenelement immer öfter wei-

tergeleitet werden, bis es über die gesamte Simulationsfläche verteilt ist; entsprechend der Zunahme der Simulationsfläche steigt die Anzahl an Flut-Nachrichten an, bis bei einer Simulationsfläche von 4000 m × 4000 m das Netz immer öfter partitioniert ist, so dass probabilistisches Fluten allein nicht mehr zielführend ist. Durch das Senden von Antwort-Nachrichten ist *AutoCast* noch bis zu einer Simulationsfläche von 4500 m × 4500 m in der Lage die Erfolgsquote über 90 % zu halten (vgl. Abbildung 5.9a). Der Rückgang der Gesamtanzahl an Nachrichten bei einer Simulationsfläche über 3500 m × 3500 m lässt sich wie folgt erklären: Sobald Knoten, die nur temporär miteinander kommunizieren können, in Reichweite zueinander kommen, wird der Datenstand mittels der Mechanismen *Antworten* und notfalls *Anfragen* abgeglichen. Bei diesem Vorgang können mehrere Datenelemente in einer Nachricht zusammengefasst werden. Entsprechend steigt die Anzahl an Antworten ab einer Simulationsfläche von 3500 m × 3500 m deutlich an. Ab einer Simulationsfläche von 5000 m × 5000 m ist das Netz soweit partitioniert, dass die Datenelemente nicht mehr an alle Knoten ausgeliefert werden können. Daher geht die Anzahl versendeter Datenelemente deutlich zurück bis bei einer Simulationsfläche von 6000 m × 6000 m die Beacon-Nachrichten den Großteil der versendeten Nachrichten ausmachen.

5.4.5 Realitätsnähe der Simulation

Bislang wurde für die Evaluation des *AutoCast*-Protokolls das Two-Ray-Ground-Funkausbreitungsmodell verwendet, aus dem ein kreisförmiges Kommunikationsgebiet folgt. Da in der Realität die Funkausbreitung allerdings richtungsabhängig variiert [51] sowie zeitlichen Schwankungen unterliegt, wird in diesem Abschnitt das realitätsnähere *Radio Irregularity Model* (RIM) verwendet (vgl. Abbildung 2.2.1), um *AutoCast* mit mehr Realitätsnähe in Bezug auf eine unzuverlässige Nachrichtenübermittlung zu evaluieren. Im Gegensatz zum Two-Ray-Ground-Modell, bei dem unter Angabe des Kommunikationsradius r ein kreisförmiges Kommunikationsgebiet angenommen wird, variiert im RIM der Kommunikationsradius richtungsabhängig, wobei letztendlich die Fläche des Kommunikationsgebietes dem eines Kreises mit Radius r entspricht. Somit ergibt sich die gleiche mittlere Knotendichte wie im bislang genutzten Two-Ray-Ground-Modell. Als Parameter für die Unförmigkeit des Kommunikationsbereiches (engl. Degree of Irregularity, auch DOI) wurde $DOI = 0,01$ gewählt, was als realistische Größe für Ad-hoc-Netze angenommen werden kann [197]. Abbildung 2.2 zeigt einen beispielhaften Kommunikationsbereich des RIMs unter den verwendeten Parametern.

Das RIM ist ebenfalls in ns-2 implementiert und baut auf der Arbeit von Buschmann u. a. auf [24]. Um auch zeitliche Schwankungen zu berücksichtigen, wurde das Modell dahin gehend erweitert, dass nicht jeder Knoten einen statischen Kommunikationsbereich hat. Vielmehr wird jeweils beim Senden eines Pakets aus einer Vielzahl im Vorfeld der Simulation berechneter Radio-Irregularity-Kommunikationsbereiche zufällig einer aus-

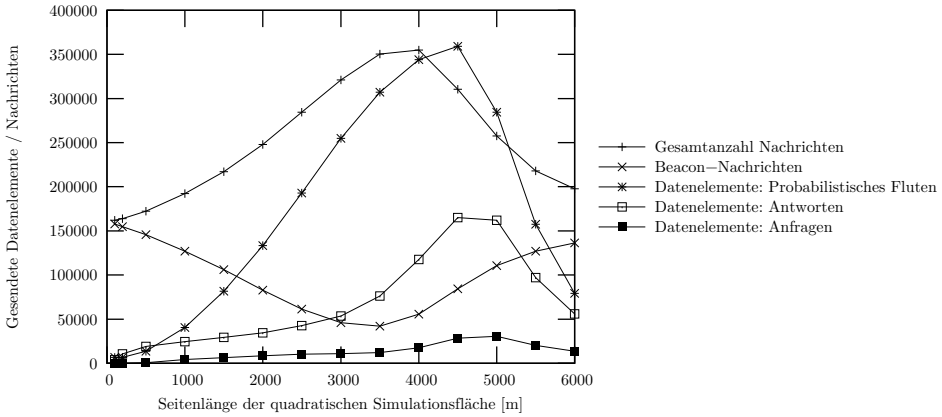


Abbildung 5.14: *AutoCast*– Ursachen des Nachrichtenaustauschs (Standard-Szenario, Random Irregularity Model, Knotengeschwindigkeit 10 m/s).

gewählt.

Die Auswertung der vier in Abschnitt 5.4 spezifizierten Leistungsparameter offenbart lediglich eine geringe Abweichung gegenüber den in Abschnitt 5.4.2 vorgestellten Ergebnissen. Aufgrund der hohen Ähnlichkeit der Ergebnisse wird an dieser Stelle auf eine Abbildung verzichtet.

Obgleich der Kommunikationsbereich im RIM dem des Two-Ray-Ground-Modells entspricht, ist *AutoCast* durch den höheren maximalen Kommunikationsradius von 375 m in der Lage, eine etwas höhere Erfolgsquote bei großen Simulationsflächen zu erzielen. Bis zu einer Simulationsfläche von 3000 m × 3000 m wird ebenfalls eine Erfolgsquote von 100 % erreicht.

Die Zustellungsgeschwindigkeit liegt mit dem RIM tendenziell etwas höher als mit dem Two-Ray-Ground-Modell, was ebenfalls auf den höheren maximalen Kommunikationsradius zurückzuführen ist.

Der Kommunikationsaufwand steigt mit dem RIM erwartungsgemäß etwas an. So liegt das Maximum der übertragenen Datenmenge pro Knoten bei 414 Byte/s bei einer Simulationsfläche von 4000 m × 4000 m (gegenüber 370 Byte/s bei Verwendung des Two-Ray-Ground-Modells).

Abbildung 5.14 zeigt die Ursachen des Nachrichtenaustauschs unter Verwendung des RIMs und verdeutlicht sehr gut die Reaktion des *AutoCast*-Protokolls auf die Änderung des Funkausbreitungsmodells. Im Vergleich zu Abbildung 5.13, die dieselbe Auswertung unter Verwendung des Two-Ray-Ground-Modells zeigt, ist zu erkennen, dass die Gesamtanzahl gesendeter Nachrichten ab einer Simulationsfläche von 4000 m × 4000 m um ca. 15 % zunimmt, während bei kleineren Simulationsflächen und somit höherer Knotendichte kein Unterschied hinsichtlich der Gesamtanzahl gesendeter Nachrichten besteht.

Während auch die Anzahl der Datenelemente, die per probabilistischem Fluten verteilt werden, nahezu identisch ist, zeigt die starke Zunahme an Datenelementen, die als Antwort gesendet werden, dass die Kommunikation unter Verwendung des RIMs prinzipiell unzuverlässiger wird. Ebenso steigt die Anzahl an Anfragen, was die Notwendigkeit dieser Aktion bei unzuverlässiger Datenübertragung bekräftigt.

Letztendlich zeigt sich, dass *AutoCast* sehr robust ist; erreicht wird dies durch den fortlaufenden Abgleich der Datenelemente bzw. ihrer Hash-Werte. Die Zielvorgaben, in unterschiedlichen Netztopologien möglichst alle Datenelemente an alle erreichbaren Knoten auszuliefern und dabei einen vernünftigen Trade-Off zwischen der übertragenen Datenmenge und der Zustellungsgeschwindigkeit zu finden, werden weiterhin erfüllt.

5.4.6 Skalierbarkeit der Simulation

Der Netzwerksimulator ns-2 hat sich nicht zuletzt aufgrund der detaillierten Modellierung des Nachrichtenaustauschs für die Simulation von mobilen Ad-hoc-Netzen etabliert. Dazu werden die Funktionen der unteren Schichten, wie Medienzugriff, Funkausbreitung, Sende- und Empfangsstärke auch unter Berücksichtigung des Hintergrundrauschens im Detail modelliert und bei jedem Nachrichtenaustausch berücksichtigt.

Die Untersuchung verteilter Verkehrsanwendungen, für die *AutoCast* entwickelt wurde, erfordert die Simulation deutlich größerer Szenarien als solche, die im vorherigen zur Evaluation von *AutoCast* verwendet wurden.

So ist beispielsweise zur systematischen Untersuchung eines Verkehrsstaus die Betrachtung eines größeren Streckenabschnitts mit einer hohen Fahrzeugdichte notwendig.

Im Gegensatz zu ns-2 fokussiert der Netzwerksimulator Shawn [42, 96, 137, 140] nicht auf die exakte Modellierung physikalischer Vorgänge, sondern modelliert die von den physikalischen Vorgängen ausgehenden Effekte auf möglichst effiziente Weise. Dazu verwendet Shawn abstrakte und austauschbare Modelle, womit auch die Simulation großer Netze ermöglicht wird. Die Vorzüge von Shawn, auch im Hinblick auf die Softwarearchitektur, hat Pfisterer ausführlich in [137] vorgestellt.

In Shawn wird der Nachrichtenaustausch durch das Zusammenspiel dreier Modelle simuliert:

Communication Model: Das Kommunikationsmodell legt fest, welche Knoten prinzipiell miteinander kommunizieren können. Die einzige Schnittstelle dieses Modells bildet die Methode CAN_COMMUNICATE_UNI(A, B), die zurückliefert, ob Knoten A grundsätzlich eine Nachricht an Knoten B senden kann.

Edge Model: Das Kantenmodell stellt eine Graphenrepräsentation des Netzes zur Verfügung. Die miteinander kommunizierenden Knoten entsprechen den Knoten des Graphen, die gerichteten Kanten des Graphen markieren die möglichen Kommunikationsrichtungen. Der Graph wird mithilfe des Kommunikationsmodells aufge-

baut – für jede prinzipielle Kommunikationsbeziehung wird eine gerichtete Kante in den Graphen eingefügt. Über die Schnittstelle dieses Modells lassen sich die Nachbarn eines Knotens erfragen.

Transmission Model: Zum Versenden einzelner Nachrichten wird letztlich das Übertragungsmodell verwendet. Hier entscheidet sich, welche potenziellen Nachbarn eine bestimmte Nachricht erhalten. Dazu ist das Übertragungsmodell in der Lage, Nachrichten zu verzögern, zu verwerfen oder auch zu stören. Auch die Beeinträchtigung durch parallel stattfindende Nachrichtenübertragungen, die sich gegenseitig beeinflussen, können hier berücksichtigt werden.

In dieser Arbeit werden die folgenden Implementierungen der vorgenannten Modelle verwendet:

Als Kommunikationsmodell dient das *Unit-Disk-Graph*-Modell. Eine Kommunikation zwischen zwei Knoten ist somit immer dann möglich, wenn der Abstand zwischen den Knoten geringer als der Kommunikationsradius ist. Der Vorteil dieses Modells liegt in seiner Einfachheit und der damit verbundenen Nachvollziehbarkeit. In Abschnitt 5.4.5 wurde bereits der Einfluss des realitätsnäheren RIMs untersucht, mit der Schlussfolgerung, dass sich das Verhalten von *AutoCast* aus Perspektive der Anwendung praktisch nicht verändert.

Aufgrund der Mobilität der Knoten und der damit verbundenen stetigen Veränderung der Kommunikationsbeziehungen lohnt es sich nicht, für das Kantenmodell den Kommunikationsgraphen proaktiv zu berechnen. Das hier verwendete *Grid*-Kantenmodell liefert dennoch auf effiziente Weise die Nachbarn eines Knoten. Dazu werden alle Knoten in einem quadratischen Flächenraster angeordnet, dessen Kantenlänge dem maximalen Kommunikationsradius entspricht. Beim Auffinden benachbarter Knoten müssen somit nur Knoten in der eigenen und den umgebenden acht Zellen des Flächenrasters betrachtet werden. Das Mobilitätsmodell sorgt während der Bewegung eines Knotens auch für das Verschieben des Knotens von einer Zelle zur nächsten, so dass die Einordnung der Knoten in das Raster stets aktuell ist.

Als Übertragungsmodell dient das CSMA-Modell. Auch die Implementierung dieses Modells ist sehr effizient und benötigt nur unwesentlich mehr Rechenleistung als eine direkte Zustellung aller Nachrichten ohne Berücksichtigung von Kollisionen. Dazu verwendet das CSMA-Modell auf jedem Knoten einen einfachen Zustandsautomaten mit den Zuständen *Leerlauf*, *Empfang* und *Kollision*. Ein sendender Knoten überprüft zunächst seinen Status – ist dieser nicht *Leerlauf*, so wird das Senden gemäß dem IEEE 802.11 Backoff-Algorithmus verzögert, bis der sendende Knoten den *Leerlauf*-Zustand erreicht oder die maximale Anzahl an Versuchen erreicht ist. Im letzteren Fall wird die Nachricht verworfen. Das eigentliche Senden wird durch zwei Ereignisse realisiert, die am Anfang und Ende der Übertragung jeweils bei allen Nachbarn des Senders aufgerufen

werden. Das erste Ereignis überführt den Zustand aller Nachbarn von *Leerlauf* in *Empfang*, bzw. von *Empfang* in *Kollision*. Nur wenn beim zweiten Ereignis, das nach dem Übermitteln der Nachricht aufgerufen wird, der Zustand des empfangenden Knotens noch immer *Empfang* ist, kann er die Nachricht verarbeiten und in den Status *Leerlauf* wechseln. Ansonsten wartet er, bis er von allen kollidierten Nachrichten das zweite Ereignis empfangen hat und wechselt seinen Status erst dann von *Kollision* in *Leerlauf*. Somit wird eine Nachricht nur empfangen, wenn die Übertragung nicht durch eine weitere Nachricht gestört wird. Zudem setzt das CSMA-Modell das zeitliche Verhalten des IEEE 802.11 Standards [71] um.

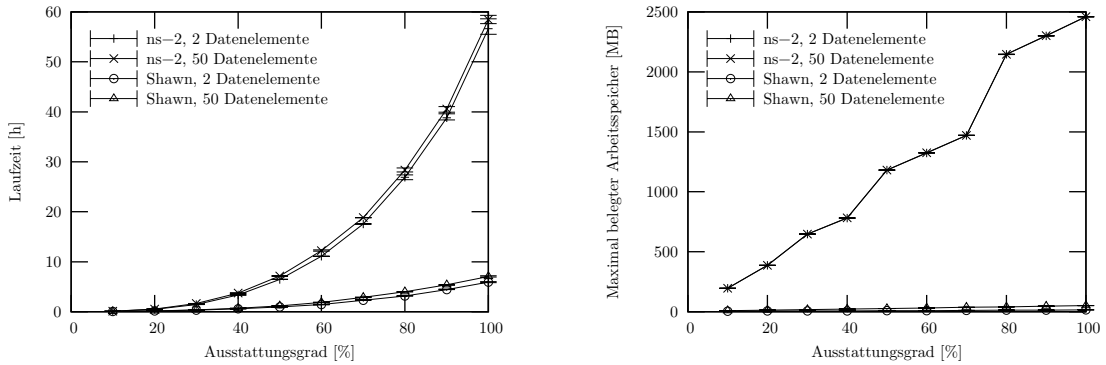
Wie schon in Abschnitt 5.3.2 erwähnt, wurde *AutoCast* ebenfalls in Shawn mit derselben Funktionalität implementiert.

Zur Evaluation der Skalierbarkeit wird das Autobahn-Szenario aus Abschnitt 5.4.3 vergrößert, um die Grenzen von ns-2 aufzuzueigen. Die Strecke besteht aus einer ringförmigen Straße mit jeweils zwei Spuren pro Fahrtrichtung. Die Länge der Rundstrecke mit einem Radius von 12,5 km beträgt ca. 78,5 km. Auf dieser Strecke fahren insgesamt 3000 Fahrzeuge, was einer mittleren Verkehrsdichte von ca. 9,5 Fahrzeugen/km/Fahrspur entspricht. Es wird der Zeitraum von einer Stunde simuliert. Auch hier wird *AutoCast* zum Verteilen von Datenelementen genutzt.

Da an dieser Stelle die Komplexität der Simulation im Vordergrund steht, wird lediglich ein einfaches Kommunikationsmuster umgesetzt, bei dem pro Zeitintervall von einem einzelnen Fahrzeug ein 200 Byte großes Datenelement generiert wird, dessen Verbreitungsgebiet die gesamte Strecke umfasst und dessen Lebenszeit 10 min beträgt. Es werden Simulationen mit zwei verschiedenen Zeitintervallen durchgeführt. Ein Zeitintervall von 12 s entspricht 50 gleichzeitigen Datenelementen und bei einem Zeitintervall von 300 s werden nur zwei Datenelemente gleichzeitig im Netz verteilt.

Für verschiedene Ausstattungsgrade zwischen 10 % und 100 % (entsprechend 300 bis 3000 Fahrzeugen) wurde die benötigte Rechenzeit der Simulation, im weiteren Laufzeit genannt, sowie die maximale Speicherauslastung auf einem aktuellen Standard-PC mit Intel Pentium D Prozessor (3,4 GHz Taktfrequenz) und 2 GB Arbeitsspeicher ermittelt. Das Vorhandensein zweier Prozessorkerne beschleunigt die Simulation allerdings nicht, da beide Simulatoren jeweils nur einen Prozessorkern nutzen. Als Betriebssystem kam ein Debian Linux mit Kernel 2.6.24 zum Einsatz. Beide Netzwerksimulatoren wurden mit GCC 3.3.5 unter Verwendung der maximalen Compileroptimierung (mittels Compiler-Option `-O3`) kompiliert. Jedwede Protokollierung der Mobilität, der gesendeten und empfangenen Nachrichten, des Protokollverhaltens, usw. sind bei beiden Simulatoren deaktiviert.

Das Ergebnis der Laufzeit- und Speichermessung ist Abbildung 5.15 zu entnehmen. Die Diagramme zeigen auf der Abszisse den Ausstattungsgrad; 100 % entsprechen 3000 ausgestatteten Fahrzeugen. Auf der Ordinate sind die durch die jeweilige Simulation benötigten Systemressourcen Laufzeit bzw. maximal belegter Speicher aufgetragen. Jede



(a) Laufzeit (b) Maximale Speicherauslastung

Abbildung 5.15: Vergleich des Laufzeitverhaltens der Netzwerksimulatoren ns-2 und Shawn.

Simulationsreihe resultiert in einer Kurve im Diagramm.

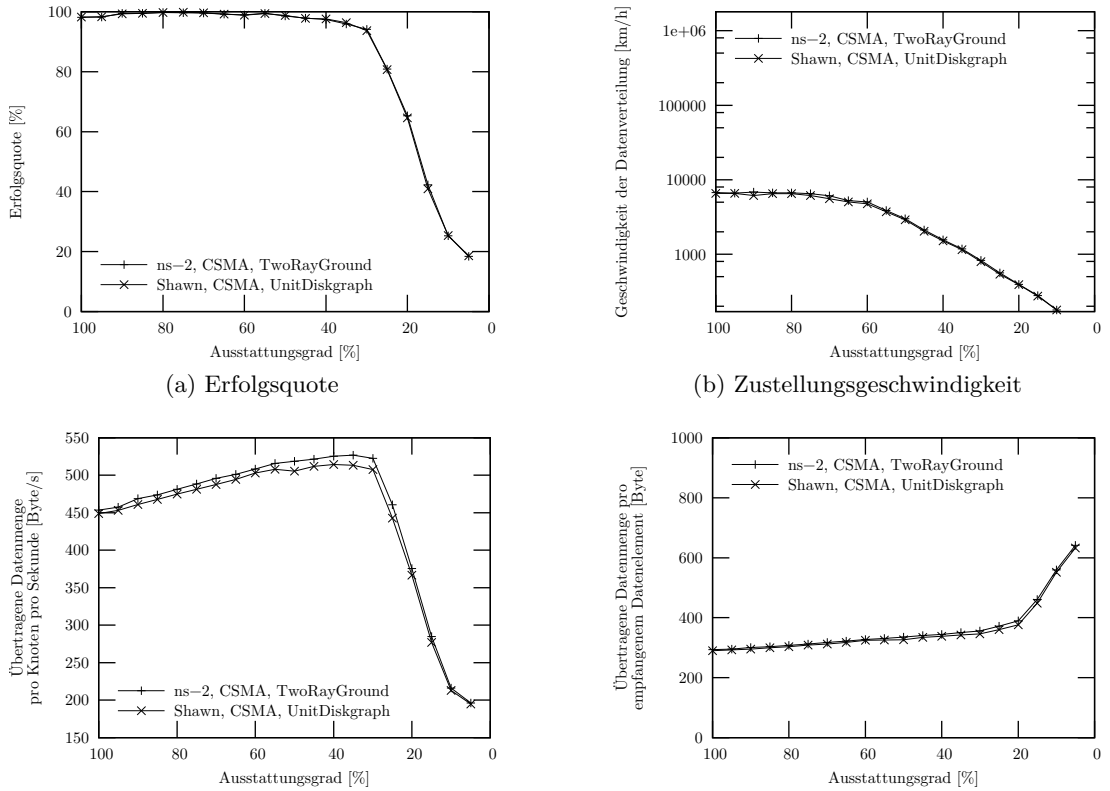
Sowohl bei der Laufzeit als auch beim Speicherbrauch zeigt sich Shawn deutlich ressourcenschonender. Die um eine Größenordnung höhere Laufzeit von ns-2 ist mit den sehr viel detaillierteren Modellen der unteren Netzwerkschichten zu begründen. So läuft eine Simulation mit 100 % Ausstattungsgrad und 50 gleichzeitig verteilten Datenelementen in ns-2 über 57 Stunden. In Shawn ist dieselbe Simulation bereits nach 7 Stunden durchgelaufen.

Warum ns-2 so viel Speicher verbraucht, obwohl jegliche Protokollierung in der Simulationsbeschreibung deaktiviert ist, ist nicht nachvollziehbar. Bei einem Ausstattungsgrad von 100 % und 50 gleichzeitig verteilten Datenelementen benötigt ns-2 2460 MB Arbeitsspeicher, während Shawn lediglich 50 MB beansprucht.

Es zeigt sich, dass die von *AutoCast* übertragene Datenmenge (2 oder 50 gleichzeitige Datenelemente) nur einen geringen Einfluss auf Laufzeit und Speicherverbrauch haben. Vielmehr ist sowohl die Laufzeit als auch der Speicherverbrauch in erster Linie von der Anzahl der simulierten Knoten und der damit verbundenen größeren Anzahl an (Beacon-)Nachrichten abhängig.

Während der Simulation des Standard-Szenarios (vgl. Abschnitt 5.4.2), bei dem die Gesamtzahl der Knoten konstant ist, zeigte sich, dass auch die Knotendichte, d. h. die Anzahl der Empfänger einer Nachricht, einen großen Einfluss auf die Laufzeit der ns-2-Simulation hat.

Aufgrund der hier gezeigten Ergebnisse erscheint es sinnvoll, die Knotenanzahl in ns-2 auf ca. 1200 (entspricht hier einem Ausstattungsgrad von 40 %) zu beschränken. Eine Limitierung in dieser Größenordnung findet sich häufiger in der Literatur, z. B. in [137]. Für größere Szenarien ist eine Vereinfachung der verwendeten Modelle nötig, wie sie von



(a) Erfolgsquote

(b) Zustellungsgeschwindigkeit

(c) Übertragene Datenmenge pro Knoten pro Sekunde

(d) Übertragene Datenmenge pro empfangenem Datenelement

Abbildung 5.16: Qualitativer Vergleich der Netzwerksimulatoren ns-2 und Shawn anhand der *AutoCast*-Leistungsparameter.

Shawn umgesetzt wird.

Nach Betrachtung des Laufzeitverhaltens der Netzwerksimulatoren ns-2 und Shawn stellt sich die Frage, ob Shawn trotz deutlich geringerer Verwendung von Systemressourcen qualitativ mit ns-2 mithalten kann. Abbildung 5.16 beantwortet diese Frage für das *AutoCast*-Protokoll im oben beschriebenen Szenario. Zu sehen ist erneut die Auswertung der vier in Abschnitt 5.4 eingeführten Leistungsparameter. Zur besseren Vergleichbarkeit mit den Ergebnissen in den zuvor betrachteten Szenarien ist die Skalierung der Achsen unverändert.

Die in Abbildung 5.16a dargestellte Erfolgsquote zeigt, dass in diesem Szenario, in dem die Datenelemente ein Verbreitungsgebiet von 78,5 km Länge haben, eine Erfolgsquote von 100 % nicht mehr ganz erreicht wird. In der Shawn-Simulation werden die Daten-

elemente im Mittel an 0,08 % weniger Fahrzeuge verteilt als in der ns-2-Simulation. Bei dieser ohnehin sehr geringen Abweichung ist teils ns-2 und teils Shawn geringfügig besser. Der größte Unterschied entsteht bei einem Ausstattungsgrad von 15 % – hier erreicht ns-2 1,4 % mehr Fahrzeuge als Shawn.

Bei der Zustellungsgeschwindigkeit zeigt sich die Shawn-Simulation zwischen ungefähr 0 % und bis zu 10 % langsamer als die ns-2-Simulation. Im Mittel liegt die Abweichung bei 3,5 %.

Wie in Abbildung 5.16c und 5.16d zu sehen ist, erzeugt die ns-2-Simulation systematisch eine im Mittel 1,8 % höhere übertragene Datenmenge, sowohl in Bezug auf Knoten pro Sekunde, als auch pro übertragenem Datenelement. Hier zeigt sich die Auswirkung der vereinfachten Implementierung von Nachrichtenkollisionen. In Shawn werden nur Kollisionen berücksichtigt, bei denen beide Sender innerhalb des Kommunikationsradius des Empfängers liegen, während in ns-2 durch Summierung von Signalstärken auch weiter entfernter Übertragungen das Hintergrundrauschen berechnet wird, was zu einer leicht erhöhten Anzahl kollidierter Nachrichten führt und folglich durch wiederholte Übertragungen kompensiert werden muss.

Die Auswertung zeigt somit, dass Shawn qualitativ nahezu gleichwertige Resultate bei sehr viel geringerem Ressourcenverbrauch liefert.

Vergleicht man zudem die Softwarearchitektur beider Simulatoren, fällt auf, dass ns-2 zwar deutlich mehr Modelle, Protokolle und Algorithmen bereithält, diese aber in ihrer Gesamtheit ns-2 zu einem schwer beherrschbaren und teilweise ebenso schwer nachvollziehbaren Werkzeug machen. Shawn hingegen reduziert seine Modelle auf die oben vorgestellten Kommunikations-, Kanten- und Übertragungsmodelle und erlaubt eine Fokussierung auf die eigentliche Anwendung. Dabei weist Shawn ein durchgehendes Objektmodell auf. Mitunter resultiert dieser Vorteil auch aus dem geringen Alter des Simulators Shawn gegenüber ns-2. Insgesamt dominieren die Vorteile der Nutzung von Shawn für die weiteren Netzwerksimulationen, die hauptsächlich auf Aspekte der Anwendungsschicht fokussieren.

Ungeachtet der bisherigen Ausführung ist die Akzeptanz des Simulators ns-2 und auch die Akzeptanz der damit erzielten Messergebnisse aufgrund seiner großen Verbreitung deutlich höher als die von Shawn. Letztlich sollte also je nach Fokus der Simulation und der Größe des simulierten Szenarios der passende Simulator ausgewählt werden.

5.5 Zusammenfassung und Ausblick

In diesem Abschnitt wurde *AutoCast* als adaptives und robustes Protokoll zum effizienten Verteilen von Daten vorgestellt. *AutoCast* ist entworfen für Anwendungen bei denen Daten und Kontext wichtiger sind als einzelne Absender und Empfänger.

Das *AutoCast*-Protokoll zeichnet sich aus durch seine hohe Anpassungsfähigkeit an

verschiedene Knotendichten und -geschwindigkeiten. Dazu wird ein hybrider Ansatz aus probabilistischem Fluten und Store-and-Forward verwendet. Beim Entwurf galt es einen Kompromiss zwischen Erfolgsquote, Zustellungsgeschwindigkeit und übertragener Datenmenge zu finden.

Durch die analytische Betrachtung der einzelnen Protokollbausteine wurde dieser Kompromiss bereits weit gehend im Vorfeld der Evaluation angepeilt und durch die vorgestellten Simulationen bestätigt. Eine wichtige Eigenschaft von *AutoCast* zeigte sich allerdings erst durch ausgiebige Simulationen. So waren anfangs die Aktionen *Antworten*, *Fluten* und *Anfragen* nicht durch verschiedene zeitliche Verzögerungen voneinander entkoppelt. Als Folge traten beim Zusammentreffen großer Netzpartitionen mit unterschiedlichem Wissensstand in seltenen Fällen sehr hohe Datenraten auf, die zu einer Überlastung des Netzes führten. Es zeigte sich, dass zunächst eine hohe Anzahl an Anfragen generiert wurde, die relativ lange in der Medienzugriffsschicht auf das Versenden warteten. Die somit veralteten Anfragen führten zu einer Vielzahl unnötiger Antworten. Durch die Umsetzung des Entwurfsprinzips *Aufmerksamkeit* in Form der Strategie „Zuhören vor dem Reden“ konnte dieses Problem vollständig gelöst werden. Die entsprechenden zeitlichen Verzögerungen der einzelnen Aktionen sind in Abschnitt 5.2.3 beschrieben.

In der Evaluation wurde *AutoCast* verglichen mit einfachem und probabilistischem Fluten sowie einem theoretischem Protokoll, das die Obergrenze der Leistungsfähigkeit in jedem Szenario bestimmt. Die Effizienz des Protokolls hängt von den im Folgenden zusammengefassten Parametern ab.

AutoCast erreicht in allen Fällen die primäre Vorgabe, Datenelemente an möglichst alle Knoten zu verteilen, was ein Vergleich der Erfolgsquote von *AutoCast* mit dem theoretischen Optimum belegt. Unter dieser Vorgabe erreicht *AutoCast* in gut konnektierten Netzen Zustellungsgeschwindigkeiten oberhalb von 5000 km/h. Die übertragene Datenmenge ist direkt abhängig von der Anzahl und Größe der Datenelemente sowie der Knotengeschwindigkeit. Der von *AutoCast* verursachte Overhead in Form von Hash-Werten aller Datenelemente wird umso kleiner, je größer die Datenelemente werden. Die Wiederholrate der Beacon-Nachrichten, die jeder Knoten sendet, hängt linear von der Knotengeschwindigkeit ab. Dieser Aufwand ist allerdings nötig, um zum einen die Anzahl benachbarter Knoten abschätzen zu können und zum anderen, um den Wissensstand benachbarter Knoten zu erfahren.

Es werden im untersuchten Fall, bei dem 50 Datenelemente mit je 200 Byte zu verteilen waren, selbst bei hohen Geschwindigkeiten nicht mehr als 510 Byte/s pro Knoten übertragen. Im Autobahn-Szenario wurde im Maximalfall bei einem Ausstattungsgrad von 100 % eine Datenrate von lediglich 100 kbit/s innerhalb eines doppelten Kommunikationsgebiets benötigt. Für weitere Protokolle und Anwendungen sollte somit noch ausreichend Netzkapazität zur Verfügung stehen.

Obwohl *AutoCast* bereits in der jetzigen Umsetzung überzeugende Ergebnisse liefert,

lässt es sich in der Zukunft noch um einige Optionen erweitern.

In allen gesendeten Nachrichten nehmen die Hash-Werte der Datenelemente relativ viel Platz ein, der linear von der Anzahl der Datenelemente abhängig ist, die der sendende Knoten kennt. Durch Verwendung von Bloom-Filtern [18, 115] könnte der Platz, der zur Identifikation aller bekannten Datenelemente notwendig ist, auf eine konstante Größe beschränkt werden – unabhängig von der Gesamtanzahl bekannter Datenelemente. Mithilfe eines Bloom-Filters kann ein Empfänger-Knoten zweifelsfrei das Fehlen eines Datenelements beim Sender feststellen, aufgrund der mit der konstanten Größe des Bloom-Filters einhergehenden Komprimierung werden allerdings nicht alle dem Sender unbekannten Datenelemente erkannt. Durch die Verwendung eines zufälligen (mitgesendeten) Initialwertes könnte allerdings bei jedem Senden ein unterschiedlicher Bloom-Filter das Fehlen unterschiedlicher Datenelemente offenbaren.

Des Weiteren benötigt *AutoCast* zwar keinerlei Infrastruktur, könnte in größeren Szenarien aber durchaus von der Integration einiger Infrastruktur-Knoten profitieren. Die Infrastruktur-Knoten sind mit dem Internet verbunden, und können somit Datenelemente um ein Vielfaches schneller über große Entferungen transportieren als dies mittels Multi-Hop-Kommunikation möglich ist. Zur Kommunikation der Infrastruktur-Knoten untereinander muss ein Protokoll entworfen werden, welches ein schnelles Auffinden von Infrastruktur-Knoten innerhalb eines Verbreitungsgebiets erlaubt. Ein Peer-to-Peer-Netz [7], das die Infrastruktur-Knoten ohne zentrale Instanz miteinander verbindet, erscheint hier als vielversprechender Ansatz.

Kapitel 6

VANET-Simulationsumgebung mit Rückkopplungsschleife

Bei der Evaluation des *AutoCast*-Protokolls im vorhergehenden Kapitel lag der Fokus im Bereich des Protokollverhaltens in Bezug auf das Funknetz. Die Bewegung der Knoten im Ad-hoc-Netz wurde im Vorfeld der Netzwerksimulation anhand eines Bewegungsmodells festgelegt.

Wenn sich der Fokus der Evaluation allerdings in Richtung des Verhaltens und der Auswirkung von VANET-Anwendungen verschiebt, dann ist nicht nur der Einsatz eines Netzwerksimulators erforderlich, der in der Lage ist, die Ad-hoc-Kommunikation zu modellieren, sondern darüber hinaus ist auch eine adäquate Modellierung der Fahrzeugbewegungen notwendig. Nur so können VANET-Anwendungen bestimmte Verkehrsstrukturen erkennen. Soll sich der Einsatz der VANET-Anwendungen auch positiv auf die Verkehrslage auswirken, so muss darüber hinaus die Möglichkeit bestehen, zur Simulationslaufzeit auf das Fahrverhalten der einzelnen Fahrzeuge Einfluss zu nehmen.

Besonders der letzte Punkt ist zur Bewertung des Nutzens von VANET-Anwendungen absolut erforderlich. Dennoch war bislang keine frei verfügbare Simulationsumgebung bekannt, die die Auswirkungen des durch die VANET-Anwendungen vorhandenen, zusätzlichen Wissens auf Seiten der Verkehrsteilnehmer berücksichtigen konnte.

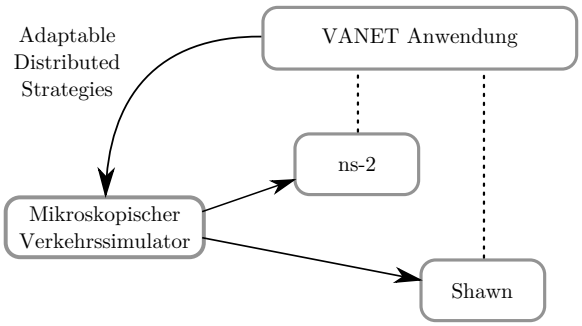


Abbildung 6.1: Informationsfluss in der VANET-Simulationsumgebung.

Der angestrebte Informationsfluss innerhalb der Simulationsumgebung ist in Abbildung 6.1 illustriert. Die VANET-Anwendung wird in einem ereignisbasierten Netzwerksimulator implementiert. In Abschnitt 5.4.1 und 5.4.6 wurden bereits die Netzwerksimulatoren ns-2 und Shawn eingeführt.

Während ns-2 die physikalischen Aspekte der drahtlosen Kommunikation detailliert modelliert, erlaubt Shawn unter Verwendung optimierter Kommunikationsmodelle die Simulation größerer Szenarien. Zumindest bei Verwendung des *AutoCast*-Protokolls produzieren beide Simulatoren nahezu identische Ergebnisse (vgl. Abschnitt 5.4.6).

Der passende Netzwerksimulator muss je nach Zielsetzung der jeweiligen Simulation gewählt werden.

Zusätzlich zum Netzwerksimulator wird ein mikroskopischer Verkehrssimulator eingesetzt, der ein Straßennetz modelliert, auf dem sich Fahrzeuge in vorgegebener Weise bewegen. „Mikroskopisch“ bedeutet in diesem Zusammenhang, dass die Fahrzeuge die elementare Einheit im Simulator darstellen (vgl. Abschnitt 2.1). Die Bewegungsdaten werden während der Simulationslaufzeit an den Netzwerksimulator weitergegeben.

Ebenfalls zur Simulationslaufzeit meldet die VANET-Anwendung gewünschte Anpassungen des Fahrverhaltens an den Verkehrssimulator, so dass sich diese direkt auf die Bewegung der Fahrzeuge auswirken. Die VANET-Anwendung kann somit die Auswirkungen ihrer Fahrempfehlungen nachvollziehen, wodurch sich die Rückkopplungsschleife im Informationsfluss schließt.

Das im Rahmen dieser Arbeit entwickelte *Traffic Control Interface* (TraCI) dient als bidirektionale Schnittstelle zwischen einem Verkehrssimulator und einen Netzwerksimulator. Es wurde bereits in [186, 187] veröffentlicht. Die Schnittstelle erlaubt die Nutzung etablierter Simulatoren sowohl für die Netzwerk- als auch für die Verkehrssimulation, anstatt grundlegende Modelle neu zu entwickeln bzw. zu implementieren.

6.1 Verwandte Arbeiten

In den vergangenen Jahren wurden diverse Simulationswerkzeuge für VANETs vorgestellt. Sie lassen sich in Bezug auf die Interaktion von Verkehrs- und Netzwerksimulation in drei Kategorien unterteilen.

Offline-Kopplung: Bei diesem Ansatz werden im ersten Schritt Bewegungsmuster erzeugt und in einer Datei abgelegt. Der Netzwerksimulator liest die Bewegungsmuster aus der Datei aus und bewegt die Knoten entsprechend. Diese Methode wird bei nahezu allen Implementierungen von Bewegungsmodellen unterstützt. Es finden sich einfache Ansätze, die auf einer realistischen Straßenkarte entweder ein Random-Waypoint-Bewegungsmuster umsetzen [151] und solche, die auch Fahrzeug-Folgemodelle mit einbeziehen [29, 83]. Ebenso bieten eigenständige Verkehrssimulatoren, wie SUMO [93] oder VISSIM [105], die Möglichkeit, Bewegungs-

muster in Dateien abzulegen. Für SUMO wurde beispielsweise im Rahmen dieser Arbeit der *TraceExporter* [175] entwickelt, der die Ausgabe des Verkehrssimulators in eine für ns-2 lesbare Form überführt. Auch die Nutzung realer Messdaten ist mit dieser Methode möglich [126].

Unidirektionale Online-Kopplung: Bei dieser Methode laufen der Verkehrs- und Netzwerksimulator gleichzeitig, so dass die vom Verkehrssimulator erzeugten Fahrzeugbewegungen nicht in Dateien zwischengespeichert werden müssen. Verkehrs- und Netzwerksimulator sind entweder eigenständige Produkte und der Verkehrssimulator übergibt zur Laufzeit Bewegungsdaten an den Netzwerksimulator [37, 103], oder ein mehr oder weniger realistisches Bewegungsmodell ist direkt im Netzwerksimulator implementiert [62, 110, 181].

Bidirektionale Online-Kopplung: Die vielversprechendste Lösung besteht in einer bidirektionalen Verbindung zwischen Verkehrs- und Netzwerksimulator. So können nicht nur Bewegungsdaten in den Netzwerksimulator eingespeist werden, sondern auch zur Simulationslaufzeit das Fahrverhalten angepasst werden. Die Beeinflussung des Fahrverhaltens wird in [103] unter Verwendung des Verkehrssimulators VISSIM und des Netzwerksimulators ns-2 erwähnt, aber nicht näher spezifiziert. Wenn das Bewegungsmodell direkt im Netzwerksimulator implementiert ist (vgl. [181]), ist der Zugriff auf das Fahrverhalten zur Simulationslaufzeit aus technischer Sicht sehr einfach. Allerdings geben die Autoren in [181] lediglich an, diese Funktion sei „einfach durch den Anwendungsentwickler zu implementieren“ (frei übersetzt).

Das in dieser Arbeit entwickelte TraCI zielt auf eine bidirektionale Online-Kopplung. Es stellt die erste frei verfügbare Lösung für die Kopplung etablierter Verkehrs- und Netzwerksimulatoren dar und wurde bereits von Dritten in nachfolgenden Arbeiten verwendet und weiterentwickelt [141, 164, 165]. Der aktuelle Status der TraCI-Entwicklung findet sich im SUMO-Wiki [4]. In Abschnitt 6.6 werden einige bereits umgesetzte Erweiterungen beschrieben.

Bei der (Neu)-Implementierung eines Bewegungsmodells innerhalb eines Netzwerksimulators, wie in [62, 110, 181] umgesetzt, ergeben sich zwei gravierende Nachteile gegenüber der Verwendung eines etablierten Verkehrssimulators: So ist nicht nur die (Neu)-Implementierung eines Bewegungsmodells mit ähnlicher Funktionalität wie sie bestehende Verkehrssimulatoren aufweisen, mit einem erheblichen Aufwand verbunden. Darüber hinaus stellt auch die Validierung des implementierten Bewegungsmodells eine nicht zu unterschätzende Herausforderung dar.

6.2 Lösungsansatz basierend auf elementaren Fahrmanövern

Jedes komplexe Fahrverhalten entsteht als Resultat vieler kleine Entscheidungen eines Verkehrsteilnehmers. Dementsprechend lässt es sich aufspalten in eine Sequenz *elementarer Fahrmanöver*, die jeweils ein Resultat einer einzelnen Entscheidung darstellen. *Elementare Fahrmanöver* können sehr kurzfristig wirken, wie beispielsweise ein Spurwechsel oder die Änderung der Geschwindigkeit, oder eine eher langfristige Auswirkung haben, wie z. B. bei der Anpassung der Fahrtroute. Unabhängig von VANET-Anwendungen werden im Verkehrssimulator ohnehin fortlaufend *elementare Fahrmanöver* ausgeführt; die zusätzlich von den VANET-Anwendungen kommenden *elementaren Fahrmanöver* müssen daher mit einer entsprechend höheren Priorität in das Bewegungsmodell integriert werden.

TraCI bietet eine Schnittstelle, um die Ausführung *elementarer Fahrmanöver* von bestimmten Fahrzeugen zu einem exakten Zeitpunkt mittels TraCI-Nachrichten vorzugeben. Der Verkehrssimulator ist für die sinnhafte Ausführung der TraCI-Nachrichten verantwortlich. Eine unrealistisch hohe Beschleunigung sollte genausowenig möglich sein wie eine Kollision aufgrund einer *Spurwechsel*-Nachricht.

Die Verwendung *elementarer Fahrmanöver* sei am Beispiel einer VANET-Anwendung zur Warnung vor Einsatzfahrzeugen erläutert. Das Einsatzfahrzeug verpackt seinen Standort, seine Geschwindigkeit und seine vorgesehene Fahrtroute in einer HDC, von der periodisch Warnmeldungen ausgehen. Die Empfänger einer solchen Warnmeldung, die sich auf der Fahrtroute des Einsatzfahrzeugs befinden, könnten beispielsweise koordiniert einen *Spurwechsel* auf die rechte Spur vollziehen und ihre *Geschwindigkeit reduzieren*. Sobald das Einsatzfahrzeug passiert ist, kann die *Geschwindigkeit erhöht* werden und die *Einschränkung der Fahrspur aufgehoben* werden.

Um alle notwendigen *elementaren Fahrmanöver* zu identifizieren, wurden die vom C2C-CC in [10] vorgeschlagenen Anwendungsfälle studiert; eine Auswahl ist in Tabelle 6.1 aufgeführt.

Letztlich zeigt sich, dass jedes komplexe Fahrverhalten durch eine Sequenz der folgenden *elementaren Fahrmanöver* dargestellt werden kann. Die folgende Auflistung sortiert die Fahrmanöver nach ihrer zeitlichen Wirkungsweise:

Geschwindigkeitsänderung: Ein Fahrzeug tendiert bei Verwendung von Fahrzeug-Folgemodellen üblicherweise dazu, die eigene Geschwindigkeit so hoch wie möglich zu wählen. Dabei werden der Abstand zum vorausfahrenden Fahrzeug, die maximale Beschleunigung und weitere Einschränkungen (z. B. Ampeln) berücksichtigt. Zum Ändern der Geschwindigkeit reicht somit das Definieren einer individuellen Höchstgeschwindigkeit aus. Das Setzen der Höchstgeschwindigkeit auf $v_{max} = 0$ führt beispielsweise zu einer spontanen Vollbremsung.

Spurwechsel: Mit diesem Fahrmanöver kann man ein Fahrzeug zur Nutzung einer be-

Tabelle 6.1: Eine Auswahl der VANET-Anwendungsfälle des C2C-CC [10], klassifiziert gemäß [147] (vgl. Abschnitt 2.3.1, S. 18) mit den resultierenden *elementaren Fahrmanövern*.

Anwendungsfall	**Elementare Fahrmanöver**				
	Geschwindigkeitsänderung	Stop	Spurwechsel	Fahrtroute anpassen	Fahrtziel anpassen
Information und Warnung					
Stauwarnung	+		+	+	+
Adaptive Navigation	+	+			
Longitudinale Fahrassistenz					
Unfallvorhersage	+	+	+		
Geschwindigkeitsbeschränkung	+				
Adaptiver Tempomat	+	+	+		
Ampel-Assistenz	+	+			
Kooperative Fahrassistenz an kritischen Stellen					
Kreuzung-Kollisionswarnung	+	+			
Spurwechsel-Assistent	+	+	+		
Warnung vor Einsatzfahrzeugen	+	+	+		
Glatteis-Warnung	+				
Fahrbahnzustands-Warnung	+		+	+	
Warnung vor niedriger Brücke	+			+	
Baustellen-Warnung	+		+	+	
Nebel-Warnung	+				
Unfall-Warnung	+	+	+	+	
Warnung vor „engen“ Kurven	+				
Warnung vor Gefahrenstellen	+	+	+	+	+

stimmten Fahrspur zwingen. Es lässt sich eine Zeit angeben, nach der die Einschränkung der Fahrspur wieder aufgehoben wird. Der Spurwechsel wird unter Berücksichtigung umgebender Fahrzeuge erst dann vollzogen, wenn er kollisionsfrei möglich ist.

Stop: Dieses Fahrmanöver sorgt für ein Anhalten an einem definierten Ort entlang der Fahrtroute. Dabei lässt sich die Fahrspur wählen, auf der gehalten wird, und die Wartezeit einstellen, nach der das Fahrzeug seine Fahrt fortsetzt. So lassen sich Unfälle genauso simulieren wie Bushaltestellen.

Fahrtroute anpassen: Für die Planung der Fahrtroute besitzt jeder Streckenabschnitt im Straßennetz eine vordefinierte *Fahrzeit*, die sich aus der Länge der Strecke und der möglichen Fahrgeschwindigkeit errechnet. Durch Setzen individueller *Fahrzeiten* für einzelne Streckenabschnitte und anschließende Neuberechnung der Fahrt-

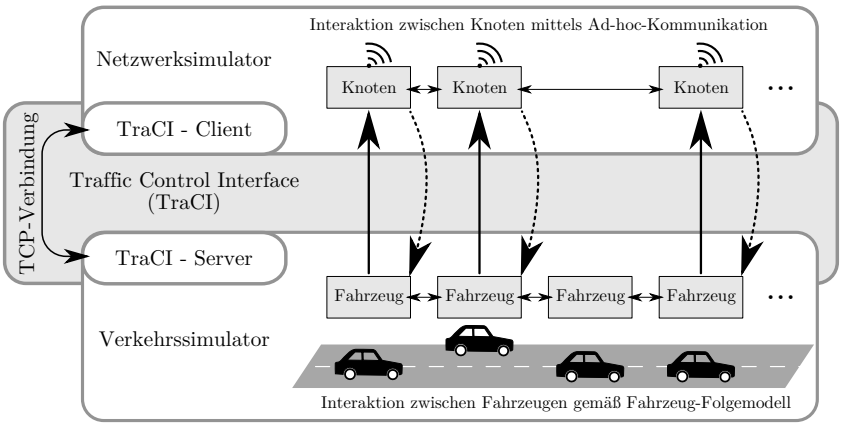


Abbildung 6.2: VANET-Simulationsumgebung mit Rückkopplungsschleife zur Beeinflussung des Fahrverhaltens.

route, wird diese gegebenenfalls angepasst. So ist sowohl die Berücksichtigung von kleineren Verzögerungen als auch von Vollsperrungen möglich.

Fahrtziel anpassen: Mit diesem Fahrmanöver lässt sich das Fahrtziel eines Fahrzeugs auf einen beliebigen Streckenabschnitt innerhalb des Straßennetzes festlegen. Die Fahrtroute wird automatisch unter Berücksichtigung der individuellen *Fahrzeiten* neu berechnet.

Im folgenden Kapitel wird die Systemarchitektur des Traffic Control Interfaces beschrieben. Die *elementaren Fahrmanöver* werden dabei in eine Reihe von TraCI-Nachrichten überführt, die der Netzwerksimulator an den Verkehrssimulator senden kann. Der genaue Aufbau dieser TraCI-Nachrichten befindet sich aus Gründen der Übersichtlichkeit in Abschnitt D.3.

6.3 Systemarchitektur des Traffic Control Interfaces

Wie in Abbildung 6.2 gezeigt, stellt das *Traffic Control Interface* (TraCI) das Bindeglied zwischen dem gleichzeitig laufenden Verkehrs- und Netzwerksimulator dar. Der Verkehrssimulator enthält eine Menge an Fahrzeugen und ist verantwortlich für die realistische Bewegung dieser Fahrzeuge anhand eines Fahrzeug-Folgemodells. Jedem Fahrzeug wird im Netzwerksimulator ein Knoten zugeordnet. Durch Setzen des Parameters *Ausstattungsgrad* werden für eine entsprechende Teilmenge der Fahrzeuge korrespondierende Knoten im Netzwerksimulator angelegt. So ist die Funktionsfähigkeit von VANET-Anwendungen zu verschiedenen Zeiten der Markteinführung evaluierbar.

Die Knoten im Netzwerksimulator bewegen sich entlang derselben Strecken, wie die zugehörigen Fahrzeuge im Verkehrssimulator. Die im Netzwerksimulator imlementierte VANET-Anwendung ist aufgrund der realistischen Bewegung der Knoten in der Lage Verkehrsstrukturen zu erkennen. Die von der VANET-Anwendung an den Verkehrssimulator gesendeten *elementaren Fahrmanöver* schließen die Rückkopplungsschleife.

Technisch gesehen stellt eine TCP-Verbindung die Verbindung zwischen den Simulatoren her. Für beide Simulatoren ist jeweils ein TraCI-Modul notwendig, das entsprechend dem Softwarekonzept des jeweiligen Simulators zu gestalten ist. Der Verkehrssimulator übernimmt die Rolle des TraCI-Servers und wartet vor Simulationsbeginn auf einem definierbaren Port auf eine eingehende TCP-Verbindung. Der im Netzwerksimulator implementierte TraCI-Client verbindet sich – ebenfalls noch vor Beginn der Simulation – mit dem TraCI-Server, so dass die TCP-Verbindung zu Beginn der Simulation aufgebaut ist. Nach dem erstmaligen TCP-Verbindungsaufbau nimmt der TraCI-Server keine weiteren eingehenden Verbindungen an. Somit steht zur Simulationslaufzeit eine symmetrische TCP-Verbindung zwischen genau einem Verkehrs- und Netzwerksimulator zur Verfügung.

Nach dem erfolgreichen Verbindungsaufbau übernimmt der Netzwerksimulator die aktive Rolle und sendet im Rahmen von *Anfragen* TraCI-Nachrichten an den Verkehrssimulator. Dieser wertet die TraCI-Nachrichten aus und sendet für jede eingegangene TraCI-Nachricht wiederum eine oder mehrere TraCI-Nachrichten als *Antwort* an den Netzwerksimulator.

Sobald die Simulation in einem der Simulatoren endet, schließt dieser die TCP-Verbindung. Daraufhin wird die Simulation im anderen Simulator ebenfalls beendet.

Die TraCI-Nachrichten gliedern sich in die folgenden Aufgabenbereiche:

Simulationsablauf: Während der Simulation wird alternierend je ein Simulationsschritt vom Verkehrs- und Netzwerksimulator ausgeführt. Die TraCI-Nachrichten aus dieser Gruppe steuern diesen Ablauf und versorgen den Netzwerksimulator periodisch mit neuen Fahrzeugbewegungen.

Elementare Fahrmanöver: Die zweite Gruppe bilden die in Abschnitt 6.2 eingeführten elementaren Fahrmanöver, die in Form von TraCI-Nachrichten an einzelne Fahrzeuge gesendet werden.

Zugriff auf das Verkehrsszenario: Zur realitätsnahen Simulation der Fahrzeugbewegungen sind im Verkehrssimulator üblicherweise diverse Aspekte des Szenarios, wie Straßennetz, Ampeln, Sehenswürdigkeiten, usw. modelliert. TraCI-Nachrichten aus dieser Kategorie erlauben den Zugriff auf verschiedene Aspekte des Szenarios und davon abhängigen Informationen, wie z. B. Entfernungsberechnungen.

Das exakte Nachrichtenformat, das mehrere TraCI-Nachrichten in einem Nachrichten-Container kapselt, befindet sich aus Gründen der Übersichtlichkeit in Anhang D. Im

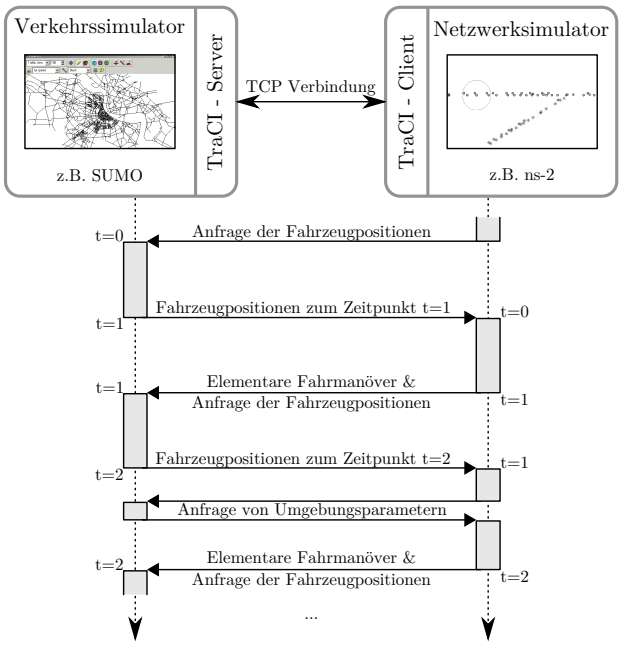


Abbildung 6.3: Interaktion des Verkehrs- und Netzwerksimulators zu Beginn der Simulation im Sequenzdiagramm.

Folgenden liegt der Fokus auf der Arbeitsweise und dem Zusammenspiel der TraCI-Nachrichten aus den drei vorgenannten Gruppen.

6.3.1 Steuerung des Simulationsablaufs

Die Steuerung des Simulationsablaufs hat die Aufgabe, die simulierte Zeit im Verkehrs- und Netzwerksimulator synchron zu halten und die im Verkehrssimulator erzeugten Fahrzeugbewegungen auf die Knoten im Netzwerksimulator zu übertragen.

Um die Synchronität der simulierten Zeit in beiden Simulatoren zu gewährleisten, wird die Simulation in diskrete Simulationsschritte unterteilt. Die Simulatoren arbeiten je einen Simulationsschritt alternierend ab. Das in Abbildung 6.3 gezeigte Sequenzdiagramm veranschaulicht diese Arbeitsweise.

Vor jedem Simulationsschritt sendet der Netzwerksimulator eine *Simulation Step*-Nachricht. Der Verkehrssimulator berechnet daraufhin die Fahrzeugbewegungen innerhalb des nächsten Simulationsschritts und antwortet mit *Move Node*-Nachrichten für jedes Fahrzeug, das sich an der VANET-Anwendung beteiligen soll. Daraufhin simuliert der Netzwerksimulator denselben Simulationsschritt, der bereits im Verkehrssimulator vollzogen wurde und bewegt die Knoten entsprechend der Angabe des Verkehrssimula-

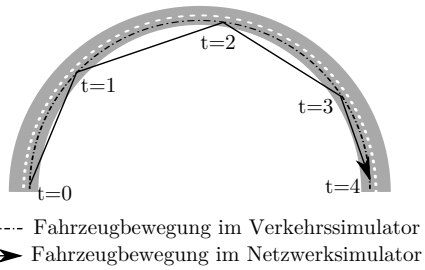


Abbildung 6.4: Die Umsetzung der Fahrzeugbewegungen beim Übergang vom Verkehrssimulator zum Netzwerksimulator.

tors.

Die alternierende Arbeitsweise von TraCI stellt somit sicher, dass nicht ein Simulator den anderen „überholt“ und die Uhren in beiden Simulatoren maximal einen Simulationsschritt differieren. Trotz der Verwendung zweier Simulatoren ist so immer nur ein Thread aktiv. Weil zudem die Kommunikation über eine lokale TCP-Verbindung um Größenordnungen schneller ist als die Kommunikation zwischen zwei separaten Rechnern, wird die Ausführung beider Simulatoren auf demselben Rechner empfohlen, obwohl die von TraCI verwendete TCP-Verbindung die Ausführung der beiden Simulatoren auf verschiedenen Rechnern prinzipiell erlaubt.

Neben der eben beschriebenen Zeitsynchronisation muss auch die Synchronität in Bezug auf die Fahrzeugpositionen berücksichtigt werden. Die von TraCI ausgeführte Umsetzung der Fahrzeugbewegungen zeigt Abbildung 6.3 im Sequenzdiagramm. Hier wird von einem Simulationsschritt der Dauer 1 s ausgegangen, was den üblichen Standard bei mikroskopischen Verkehrssimulatoren darstellt.

Jedes vom Verkehrssimulator zum Zeitpunkt t in die Simulation eingeführte Fahrzeug wird dem Netzwerksimulator zum Zeitpunkt $t + 1$ in einer *Move Node*-Nachricht mitgeteilt. Da diese Nachricht allerdings nur den Zielort des Fahrzeugs zum Zeitpunkt $t + 1$ enthält, befindet es sich im Netzwerksimulator während seines initialen Simulationsschritts in der Zeit von t bis $t + 1$ statisch am für $t + 1$ gemeldeten Zielort. Diese Entwurfsentscheidung wurde zugunsten eines einfacheren TraCI-Protokolls getroffen und betrifft jeweils nur den initialen Simulationsschritt eines Fahrzeugs.

Während der Verkehrssimulator den exakten Straßenverlauf kennt und anhand dessen die Fahrzeugpositionen bei gegebener Geschwindigkeit berechnet, werden zur Reduktion der Komplexität im Netzwerksimulator alle Bewegungen während eines Simulationsschritts als lineare Bewegungen mit konstanter Geschwindigkeit aufgefasst. Dabei ist jedoch immer sichergestellt, dass die Position eines Fahrzeugs und seines korrespondierenden Knotens am Ende eines jeden Simulationsschritts exakt übereinstimmen.

Bei einem kurvigen Straßenverlauf führt dies, wie Abbildung 6.4 zeigt, zu leicht abwei-

chenden Fahrzeugpositionen und aufgrund der geradlinigen Fahrstrecke im Netzwerksimulator zu einer geringfügig geringeren Fahrgeschwindigkeit.

Ausgehend von Simulationsschritten von 1 s Dauer, einer sehr engen Kurve mit einem Radius von $r_{Kurve} = 8\,\text{m}$ und einer sehr sportlichen Fahrgeschwindigkeit von $v = 30\,\text{km/h}$ führt der so implizierte Fehler zu einer Abweichung der Geschwindigkeit von weniger als 5 %. Die absolute Abweichung des Streckenverlaufs beträgt in diesem Fall maximal 1,06 m. Bei konstanter Zentripetalkraft wächst der Kurvenradius quadratisch mit der Fahrgeschwindigkeit, daher nimmt die Abweichung mit zunehmender Fahrgeschwindigkeit ab.

6.3.2 Elementare Fahrmanöver

Die *elementaren Fahrmanöver* wurden bereits im Abschnitt 6.2 ausführlich besprochen.

Für jedes der dort vorgestellten Fahrmanöver existiert eine zugehörige TraCI-Nachricht, mit der man für jeweils ein Fahrzeug eine gewünschte Aktion veranlassen kann. Das Nachrichtenformat dieser TraCI-Nachrichten findet sich in Abschnitt D.3.

Durch die alternierende Arbeitsweise der Simulatoren können die *elementaren Fahrmanöver*, die im Netzwerksimulator während eines Simulationsschritts in Form von TraCI-Nachrichten an den Verkehrssimulator gesendet werden, erst im nächsten Simulationsschritt des Verkehrssimulators berücksichtigt werden. Diese Einschränkung muss bei der Wahl der Dauer eines Simulationsschritts berücksichtigt werden. Bei der Simulation kooperativer VANET-Anwendungen zur Optimierung des Verkehrsflusses darf ein Simulationsschritt mitunter 5 s und länger dauern, da die Beeinflussung des Fahrverhaltens primär die Routenwahl betrifft und somit langfristig wirkt. Dem hingegen stellen zeitkritischen Anwendungen, wie beispielsweise ein Spurwechselassistent, besondere Anforderungen an die Reaktionszeit der *elementaren Fahrmanöver*. TraCI erlaubt daher die Dauer der Simulationsschritte, entsprechend der simulierten Anwendung zu wählen, um dem für die Anwendung nötigen Zeitverhalten gerecht zu werden.

6.3.3 Zugriff auf das Verkehrsszenario

Im Verkehrssimulator ist die Modellierung des Straßennetzes notwendig, um die Fahrzeugbewegungen innerhalb des Straßennetzes simulieren zu können. In gängigen Verkehrssimulatoren können darüber hinaus weitere Aspekte des Szenarios definiert werden, beispielsweise Ampeln, Sehenswürdigkeiten sowie allgemeine Polygone.

Um das Verkehrsszenario nicht manuell innerhalb des Netzwerksimulators erneut modellieren zu müssen, erlaubt TraCI den Zugriff und teilweise auch die Modifikation verschiedener Aspekte des Verkehrsszenarios.

Die zu diesem Zweck verwendete *Scenario*-Nachricht unterteilt das Verekhrsszenario in die Domänen Straßennetz, Fahrzeuge, Ampeln, Sehenswürdigkeiten und Polygone.

Innerhalb der einzelnen Domänen stehen verschiedene Variablen zur Verfügung, wie z. B. die Anzahl an Fahrzeugen, deren Positionen und Fahrtrouten oder die zukünftigen Phasenwechsel von Ampeln.

Darüber hinaus ist die Berechnung der Entfernung zwischen zwei Punkten im Straßennetz möglich; entweder unter Berücksichtigung des Straßennetzes als Fahr-Distanz oder ohne dessen Berücksichtigung als Luftlinien-Distanz.

Auch bei der Konvertierung verschiedener Repräsentationen kann der Verkehrssimulator helfen und kartesische Koordinaten auf Positionen im Straßennetz abbilden und umgekehrt.

TraCI-Nachrichten aus dieser Gruppe können zu jedem Zeitpunkt der Simulation an den Verkehrssimulator gesendet werden, wie in Abbildung 6.3 zwischen $t = 1$ und $t = 2$ ausgeführt. Der Aufruf dieser TraCI-Nachrichten ist ebenfalls blockierend, d. h. eine *Anfrage* beim Verkehrssimulator unterbricht die Netzwerksimulation, bis die entsprechende *Antwort* empfangen wird.

Für eine detaillierte Auflistung der verfügbaren Variablen in den einzelnen Domänen und das Format der dafür verwendeten TraCI-Nachrichten sei auf Anhang D verwiesen.

6.4 Implementierung

Eines der Entwurfsziele von TraCI war die Entwicklung einer frei verfügbaren Simulationsumgebung. Zudem sollen sowohl der Netzwerk- als auch der Verkehrssimulator austauschbar sein. Entsprechend niedrig sind auch die Mindestanforderungen an die jeweiligen Simulatoren. Alle Objekte, auf die TraCI Bezug nimmt, werden durch numerische *ObjectIds* identifiziert; der TraCI-Server bzw. -Client sorgt – falls nötig – für eine entsprechende Konvertierung. Der Verkehrssimulator muss mindestens die Funktionalität zur Steuerung des Simulationsablaufs implementieren. Die Implementierung aller weiteren TraCI-Nachrichten ist optional und muss letztlich nur der angestrebten Simulation genügen.

Ein TraCI-Server wurde für den Verkehrssimulator SUMO [93] (*Simulator of Urban Mobility* (SUMO)) entwickelt. Seit August 2008 ist der TraCI-Server als fester Bestandteil in SUMOs Release-Version enthalten. SUMO ist ein frei verfügbarer, mikroskopischer Verkehrssimulator, der in der Lage ist, auch große Straßennetze in akzptabler Zeit zu simulieren. Er nutzt das Fahrzeug-Folgemodell von Krauß [95] (vgl. Abschnitt 2.1).

Im Bereich der Netzwerksimulatoren wurden TraCI-Clients für ns-2 [38] sowie Shawn [42] entwickelt.

Die TraCI-Module für alle drei Simulatoren sind – wie die Simulatoren selbst – in C++ implementiert. SUMO wird bei Verwendung von TraCI mit einem speziellen Kommandozeilenschalter gestartet, der den Simulator veranlasst auf externe Steuerung über eine TCP-Verbindung zu warten (vgl. Abschnitt 6.3.1). Für ns-2 und Shawn sind je-

weils Klassen entwickelt worden, die die Bewegung der Knoten über TraCI erfragen und C++-Schnittstellen für die TraCI-Nachrichten bereitstellen. In ns-2 sind diese Schnittstellen zudem auch von der Simulationsbeschreibung in TCL [189] zugänglich. Die genaue Beschreibung der Implementierungen inklusive ihrer Verwendung befindet sich in Abschnitt D.4.

Auch wenn sich in dieser Arbeit die Beschreibung der Implementierung auf die drei vorgenannten Simulatoren beschränkt, sei erwähnt, dass TraCI weder an spezielle Verkehrs- noch Netzwerksimulatoren gebunden ist. Vielmehr ist TraCI ausreichend generisch, um nahezu beliebige Verkehrs- und Netzwerksimulatoren über eine klar definierte Schnittstelle miteinander zu verbinden.

6.5 Evaluation

In der folgenden Evaluation soll gezeigt werden, dass sich TraCI als bidirektionale Online-Kopplung zwischen dem Verkehrssimulator SUMO und dem Netzwerksimulator ns-2 in Bezug auf die Simulationslaufzeit mit der Verwendung von Bewegungsdateien messen kann.

Zu diesem Zweck wurden die folgenden Ansätze implementiert, die die Mobilität der Knoten auf unterschiedliche Art generieren.

Nur Laufzeit des Netzwerksimulators: Zur Abschätzung der Zeit, die ns-2 mit sich selbst beschäftigt ist, werden in diesem Ansatz die Knoten lediglich statisch platziert und nicht weiter bewegt. Somit wird nur die Laufzeit des Netzwerksimulators ermittelt.

Erstellung von Bewegungsdateien: Bei diesem Ansatz wird zunächst die Mobilität der Fahrzeuge von SUMO berechnet und in einer Logdatei gespeichert. Diese wird nachfolgend mit dem *TraceExporter* [175] in Bewegungsdateien konvertiert, die ns-2 einlesen kann. Anschließend führt ns-2 seine Simulation basierend auf der erzeugten Bewegungsdatei aus. Die Simulationslaufzeit ist somit die Summe aus der Laufzeit von SUMO, TraceExporter und ns-2, die sequenziell ausgeführt werden.

Bidirektionale Kopplung mittels TraCI: Dieser Ansatz verbindet SUMO und ns-2 mittels TraCI. Die Fahrzeugbewegungen werden, wie in diesem Kapitel beschrieben, während der simultanen Ausführung von SUMO und ns-2 erzeugt und pro Simulationsschritt von 1 s Dauer an ns-2 übergeben.

Um bei dieser Simulationsreihe ausschließlich die Auswirkungen der Verwendung von TraCI zu ermitteln, tut der Netzwerksimulator weiter nichts, als die Knoten entsprechend der Vorgabe des Verkehrssimulators zu bewegen. Es ist keine VANET-Anwendung aktiv

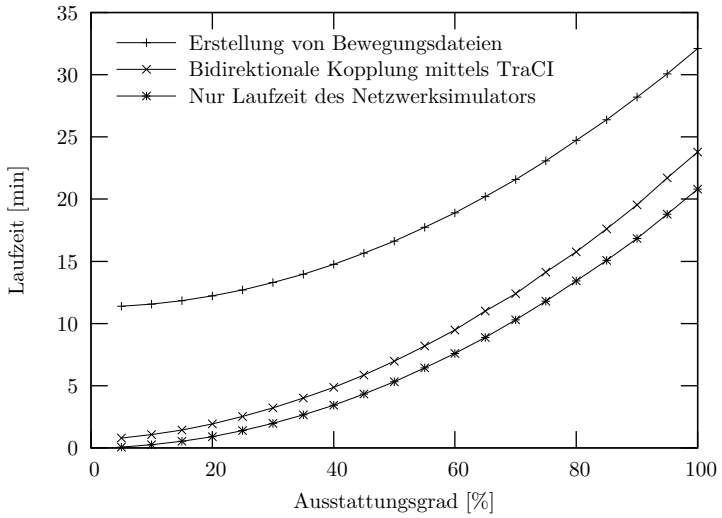


Abbildung 6.5: Vergleich des Laufzeitverhaltens bei Verwendung verschiedener Ansätze zur Bewegung der Knoten.

und es findet auch kein weiterer Datenaustausch zwischen den Knoten im Netzwerksimulator statt.

Aufseiten des Verkehrssimulators fahren die Fahrzeuge auf einer zweispurigen Rundstrecke mit 78,5 km Länge (12,5 km Radius). Insgesamt befinden sich 1500 Fahrzeuge auf der Rundstrecke, was im Mittel einer Verkehrsdichte von 9,5 Fahrzeugen/km/Fahrspur entspricht. Es wird der Zeitraum von zwei Stunden simuliert.

Durch Variation des Parameters *Ausstattungsgrad* wird der Anteil an Fahrzeugen variiert, der an den Netzwerksimulator weitergereicht wird.

Alle Simulationslaufzeiten wurden auf einem Standard-PC mit Intel Pentium D Prozessor (3,4 GHz Taktfrequenz) und 2 GB Arbeitsspeicher ermittelt. Ebenso wie bei der in Abschnitt 5.4.6 durchgeführten Laufzeitmessung kam ein Linux mit Kernel 2.6.24 zum Einsatz. Die Simulatoren wurden von GCC 3.3.5 unter Anwendung maximaler Optimierung (`-O3`) kompiliert.

Abbildung 6.5 zeigt das Ergebnis dieser Evaluation. Dabei ist auf der Abszisse der Ausstattungsgrad aufgetragen; ein Ausstattungsgrad von 100 % entspricht 1500 Fahrzeugen. Die Simulationslaufzeit dient als Leistungsparameter und ist dementsprechend auf der Ordinate aufgetragen. Jede Kurve im Diagramm repräsentiert einen der im Vorfeld beschriebenen Ansätze.

Der auf der Erstellung von Bewegungsdateien basierende Ansatz benötigt bei jedem Ausstattungsgrad die höchste Laufzeit, die sich wie folgt zusammensetzt. SUMO benötigt 78 s, um die Verkehrssimulation durchzuführen und eine Logdatei im XML-Format von

knapp 597 MB Größe zu erzeugen. Der TraceExporter liest diese Logdatei ein und erstellt innerhalb von ca. 10 min eine Bewegungsdatei im TCL-Format von 915 MB Größe. Diese Datei kann prinzipiell für jeden Ausstattungsgrad verwendet werden, da jede Zeile nach dem Muster

```
\small
if { $opt(penetration) > 0.132 }
   { $ns_ at 1.0 "$node_(198) setdest 24386.963 16349.25 0.38" }
```

aufgebaut ist. Der Ausstattungsgrad kann somit von ns-2 vor dem Einbinden der Bewegungsdatei über die TCL-Variable `$opt(penetration)` gesetzt werden. Die Laufzeit von ns-2 ist abhängig vom Ausstattungsgrad bzw. der Anzahl der Fahrzeuge in der Simulation und liegt zwischen 4 s bei 5 % Ausstattungsgrad (75 Knoten) und 21 min bei 100 % Ausstattungsgrad (1500 Knoten).

Bei Verwendung von TraCI entfällt das Anlegen und Konvertieren der Bewegungsdatei. Allerdings hängt in diesem Fall auch die Laufzeit von SUMO vom Ausstattungsgrad bzw. von der Anzahl der mittels TraCI übertragenen Fahrzeuge ab. Sie beträgt zwischen 38 s bei 5 % Ausstattungsgrad und 47 s bei 100 % Ausstattungsgrad und somit in jedem Fall weniger, als bei der Erstellung einer Logdatei. Die Laufzeit von ns-2 verschlechtert sich geringfügig und liegt zwischen 9 s (5 % Ausstattungsgrad) und 23 min (100 % Ausstattungsgrad).

Die dritte Kurve in Abbildung 6.5 zeigt die Laufzeit, die ns-2 ohne jede Bewegung der Knoten benötigt. Es zeigt sich, dass der durch die Bewegung der Knoten von SUMO und TraCI verursachte Overhead insgesamt lediglich zwischen 5 s und 131 s liegt.

Somit ist TraCI deutlich performanter als herkömmliche Verfahren, welche die Erstellung von Bewegungsdateien erfordern. Durch den Wegfall der Bewegungsdateien wird nicht nur eine verbesserte Simulationslaufzeit erzielt, sondern auch der Speicherplatz für die Bewegungsdateien eingespart.

6.6 Zusammenfassung

In diesem Kapitel wurde eine VANET-Simulationsumgebung vorgestellt, welche einer VANET-Anwendung die Beeinflussung des Fahrverhaltens einzelner Fahrzeuge während der Simulation erlaubt. Somit können die durch eine VANET-Anwendung implizierten Veränderungen im Verkehrsgeschehen mit einer frei verfügbaren Simulationsumgebung erfasst werden.

Das vorgestellte *Traffic Control Interface* (TraCI) verbindet zur Simulationslaufzeit einen Verkehrssimulator mit einem Netzwerksimulator über eine TCP-Verbindung. Im Rahmen dieser Arbeit wurden *TraCI-Module* für den Verkehrssimulator SUMO und die Netzwerksimulatoren ns-2 und Shawn entwickelt.

Die Beeinflussung des Fahrverhaltens wird durch eine Sequenz *elementarer Fahrmanöver* umgesetzt, die in ihrer Kombination unterschiedlichsten VANET-Anwendungen genügen.

Im vorhergehenden Abschnitt wurde die Auswirkung von TraCI auf die Laufzeit der Simulation untersucht und gezeigt, dass TraCI deutlich performanter ist als die Verwendung von Bewegungsdateien. So ergibt sich auch schon ein Vorteil bei der Nutzung von TraCI, selbst wenn die Beeinflussung des Fahrverhaltens nicht genutzt wird.

Zwischenzeitlich wird TraCI auch in weiteren Forschungsprojekten eingesezt. So bietet das *Traffic and Network Simulation Environment* (TraNS) [141] eine grafische Oberfläche zur Steuerung und Visualisierung von VANET-Simulationen. TraCI stellt dabei die Verbindung zwischen SUMO und ns-2 her. Sommer u. a. beschreiben in [164, 165] eine VANET-Simulationsumgebung, die SUMO mit dem Netzwerksimulator OMNeT++ [178] verbindet. Für die Umsetzung dieses Ansatzes wurde TraCI um einen Abonnement-Dienst erweitert, der den Netzwerksimulator beispielsweise bei einer Veränderung von Ampelphasen informiert. In der Folge werden als Antwort auf eine *Simulation Step*-Nachricht zusätzlich zu den Fahrzeugpositionen die abonnierten Daten zurückgeliefert.

Durch die generische Auslegung von TraCI wurde es auch schon außerhalb des VANET-Umfelds einsetzt. So beschreibt Morenz in [120] eine TraCI-Erweiterung, die in Echtzeit die Verkehrsflüsse in SUMO an real gemessene Verkehrsaufkommen anpasst.

Eine TraCI-Beschreibung, welche stets den aktuellen Entwicklungsstand widerspiegelt, befindet sich im SUMO-Wiki [4].

Diese VANET-Simulationsumgebung bildet die Grundlage für die genauere Untersuchung von VANET-Anwendungen, die das *AutoNomos*-System verwenden. Drei solcher Anwendungen werden exemplarisch im nachfolgenden Kapitel vorgestellt.

Kapitel 7

Anwendungsfälle

Die in den vorhergehenden Kapiteln vorgestellten Konzepte und Techniken werden nun auf drei konkrete Anwendungsfälle bezogen.

Wie bereits in Abschnitt 2.3.2 besprochen, wurden sehr viele Anwendungsfälle im Straßenverkehr identifiziert, bei denen der Einsatz von VANET-Anwendungen vielversprechend ist. Allein das *CAR-2-CAR Communication Consortium* (C2C-CC) hat über einhundert solcher Anwendungsfälle konkretisiert. Aus diesen wurden drei herausgesucht, und für sie exemplarische VANET-Anwendungen entwickelt, die unterschiedliche Aspekte der in dieser Arbeit vorgestellten Konzepte umsetzen.

Die erste Anwendung *Ampelassistenz* hilft beim Energiesparen an Kreuzungen, die mit einer Ampel ausgestattet sind (vgl. Abschnitt 7.2). Dazu werden den Fahrzeugen zukünftige Phasenwechsel mitgeteilt. Basierend auf diesen Daten bremsen die Fahrzeuge sanft ab, anstatt vor der gerade rot gewordenen Ampel scharf bremsen zu müssen. Zudem können wartende Fahrzeuge den Motor ausschalten, wenn die Rotphase noch eine gewisse Zeit anhält.

Bei der zweiten Anwendung wird eine *adaptive Routenplanung* umgesetzt (vgl. Abschnitt 7.3). Basierend auf der Zeit, die vorausfahrende Fahrzeuge für einen Streckenabschnitt benötigen, entscheiden sich nachfolgende Fahrzeuge adaptiv für die Haupt- oder eine Umleitungsstrecke.

Die dritte Anwendung beschäftigt sich mit der dezentralen *Erkennung von Verkehrsstrukturen*, im Speziellen mit der Erkennung eines Verkehrsstaus (vgl. Abschnitt 7.4). Aus einfachen Fahrmanövern einzelner Fahrzeuge wird dabei durch hierarchische Aggregation der Daten auf einen Verkehrsstau geschlossen und seine Position und Länge berechnet.

Die ersten beiden Anwendungen *Ampelassistenz* und *adaptive Routenplanung* wurden mithilfe der in Kapitel 6 vorgestellten Simulationsumgebung evaluiert.

Die dritte Anwendung wurde zwar auch simulativ evaluiert, wird jedoch im Folgenden anhand einer Demonstrationsplattform vorgestellt und betrachtet. Die Demonstrationsplattform umfasst 18 Lego-Mindstorms-Fahrzeuge, die sich auf einer vorgegebenen Fahrbahn bewegen.

Bei der Evaluation der vorgestellten Anwendungen soll der Fokus nicht auf der Kom-

munikation liegen, denn das verwendete AutoCast-Protokoll wurde schon in Kapitel 5 ausgiebig evaluiert. Stattdessen steht die Auswirkung der jeweiligen Anwendung im Vordergrund. Als Evaluationsparameter werden daher der Energieverbrauch und die Fahrtzeit verwendet.

7.1 Verwandte Arbeiten

Wie in den vorhergehenden Kapiteln vorgestellt, gibt es reichlich Arbeiten in den Bereichen VANET-Anwendungen (vgl. Abschnitt 2.3.2), Middleware-Ansätze in Ad-hoc-Netzen (vgl. Abschnitt 4.1) sowie zu Kommunikationsaspekten in Ad-hoc-Netzen und VANETs (vgl. Abschnitt 5.1) und auch einige Ansätze im Bereich der VANET-Simulationsumgebungen (vgl. Abschnitt 6.1).

Demgegenüber sind nur wenige Arbeiten bekannt, die sich mit den Auswirkungen von VANET-Anwendungen auf den Verkehrsfluss auseinandersetzen.

Richter beschäftigt sich in seiner Dissertation mit der „Geschwindigkeitsvorgabe an Lichtsignalanlagen“. Die Grundidee ist, schon frühzeitig mit einer verminderten Geschwindigkeit an eine Ampel heranzufahren, um sie ohne Halt während der Grünphase zu überqueren, anstatt mit hoher Geschwindigkeit bis kurz vor die Ampel zu fahren und dann bis zum Stillstand des Fahrzeugs abbremsen zu müssen. Die von Richter vorgestellte Anwendung hat Ähnlichkeit zu der in dieser Arbeit vorgestellten *Ampelassistenz*-Anwendung, setzt jedoch einen anderen Fokus. So behandelt Richter sehr ausführlich die Auswirkungen verschiedener Verkehrssituationen, abstrahiert allerdings die Kommunikationsschicht und geht einfach vom Vorhandensein des Wissens über die Schaltzustände der Ampeln sowie der Positionen der benachbarten Fahrzeuge aus. Demgegenüber stellt die hier vorgestellte *Ampelassistenz* aus verkehrswissenschaftlicher Sicht lediglich eine erste Studie dar, berücksichtigt dagegen allerdings auch die Kommunikation der Fahrzeuge im VANET.

In [182] stellen Wang u. a. eine VANET-Anwendung zur Unterstützung beim Einordnen am Ende einer Fahrspur bzw. Autobahnauffahrt nach dem Reißverschluss-System vor. Durch eine proaktive Geschwindigkeitsanpassung kann das Einordnen auf eine Fahrspur flüssiger ablaufen und so der Verkehrsfluss an der Verengung bis um den Faktor zwei gesteigert werden. Drahtlose Kommunikation spielt in der Arbeit von [182] hingegen keine Rolle; stattdessen wird davon ausgegangen, jedes Fahrzeug kenne jederzeit die Position und Geschwindigkeit der umgebenden Fahrzeuge.

7.2 Ampelassistenz

Eine Analyse der während eines Vormittags gemessenen Fahrzeugbewegungen in Braunschweig in Verbindung mit den Positionen der Ampeln in Braunschweigs Innenstadt

zeigte ein großes Potenzial zur Einsparung von Treibstoff durch Abschaltung der Treibstoffzufuhr während des Anhaltevorgangs und durch Ausschalten des Motors bei stehendem Fahrzeug während der Rotphase. Diese analytische Betrachtung hilft zwar bei der Abschätzung des Verbesserungspotenzials, sie kann jedoch nicht die Umsetzbarkeit einer so entstandenen Anwendungs-Idee bestätigen. Daher wurde diese *Ampelassistenz*-Anwendung simulativ umgesetzt und unter Variation verschiedener Parameter evaluiert. Die *Ampelassistenz*-Anwendung wurde bereits in [186] publiziert.

Zur Optimierung von Ampel-Kreuzungen wird bisher meist die Schaltung einer Ampel adaptiv an den Verkehrsfluss angepasst. Dabei wird beispielsweise je nach Tageszeit zwischen verschiedenen Ampelsteuerungen umgeschaltet, und so der pendelnde Berufsverkehr berücksichtigt. Auch die Abstimmung der Ampelschaltungen untereinander (grüne Welle) verbessert den Verkehrsfluss.

Abbremsungen und Wartezeiten vor Ampeln können so zwar vermindert werden, letztlich lässt sich beides aber nicht verhindern. Die hier vorgestellte VANET-Anwendung informiert die Verkehrsteilnehmer über zukünftige Schaltvorgänge und erlaubt so eine Anpassung der Fahrweise.

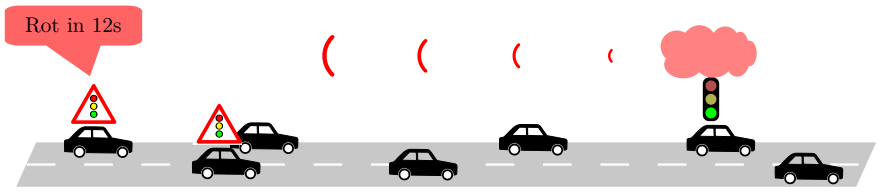
Zu diesem Zweck werden die Ampeln mit einem Kommunikationsmodul ausgestattet, so dass sie als vollwertiger VANET-Knoten mit den Fahrzeugen kommunizieren können. Die zukünftigen Phasenwechsel einer Ampel werden in einer HDC gehalten. Hier tritt der Sonderfall ein, dass die Position einer HDC exakt der Position der zugehörigen Ampel und somit dem VANET-Knoten entspricht. Eine Migration der HDC von Knoten zu Knoten ist daher bei dieser Anwendung nicht erforderlich.

Über die Funktionseinheit *Informationsextrahierung* (vgl. Abschnitt 4.3) wird von der Ampel jeweils $t_{horizon}$ vor einem Phasenwechsel[1] ein Datenelement erzeugt und in einem Radius r_{data} um die Ampel herum verteilt. Das Datenelement enthält die Position der Ampel, die Kennungen des ein- und ausgehenden Streckenabschnitts, den Umschaltzeitpunkt sowie die nächste Phase (rot oder grün). Die Lebenszeit des Datenelements wird so gewählt, dass es bis zum Umschaltzeitpunkt der Ampel gültig ist. Zum Verteilen des Datenelements wird *AutoCast* eingesetzt (vgl. Kapitel 5), so dass die Fahrzeuge die von einer Ampel ausgehenden Datenelemente so schnell wie möglich erhalten, sobald ihr Abstand zur Ampel weniger als r_{data} beträgt. Auf Basis der so verteilten Datenelemente kann jedes Fahrzeug sein Verhalten eigenständig anpassen.

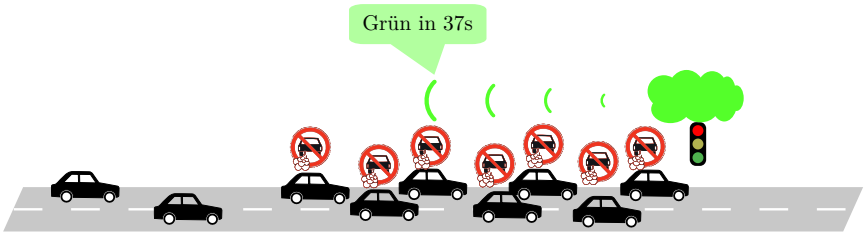
Abbildung 7.1 zeigt zwei Situationen, bei denen von den beteiligten Fahrzeugen jeweils eine der im Folgenden erläuterten Strategien angewendet wird.

1. Wenn eine auf der Fahrtroute liegende Ampel unter Beibehaltung der aktuellen Geschwindigkeit nicht mehr bei Grün zu erreichen ist, wird die Fahrgeschwindigkeit langsam gesenkt, bis das Fahrzeug an der Haltelinie der Ampel zum Ste-

[1] Die Zeitspanne, für die eine Ampel zukünftige Phasenwechsel bestimmen kann, wird im Folgenden *Zeithorizont* genannt und als $t_{horizon}$ angegeben.



(a) Abbremsen, falls die Ampel nicht mehr bei Grün erreicht werden kann



(b) Motor ausschalten, wenn die Ampel noch mindestens 20 s Rot anzeigt

Abbildung 7.1: Ampelassistenz-Anwendung: Fahrstrategien zur Einsparung von Treibstoff.

hen kommt. Dazu kann in den Schub- oder Segelbetrieb gewechselt werden. Beim Schubbetrieb wird die Treibstoffzufuhr zum Motor unterbrochen und so die Motorbremse aktiviert; dabei sinkt der Treibstoffverbrauch auf Null. Beim Segelbetrieb wird der Motor ausgekuppelt. Die Geschwindigkeit sinkt daher aufgrund der geringeren Bremswirkung langsamer als im Schubbetrieb; der Treibstoffverbrauch im Segelbetrieb entspricht dem Leerlaufverbrauch C_{idle}.

Segelbetrieb ist bei höheren Geschwindigkeiten (>50 km/h) sinnvoll, bei niedrigeren Geschwindigkeiten erhöht sich der Einfluss des Leerlaufverbrauchs (pro Zeiteinheit) gegenüber dem Treibstoffverbrauch pro Wegstrecke, so dass Auskuppeln bei niedrigen Geschwindigkeiten ineffizienter ist als der Schubbetrieb. Da die *Ampelassistenz*-Anwendung hauptsächlich im innerstädtischen Verkehr angewendet wird, wird im Folgenden außschließlich der Schubbetrieb eingesetzt.

Abbildung 7.1a visualisiert dieses Vorgehen. So schalten die Fahrzeuge, die die Ampel nicht mehr bei Grün erreichen können, frühzeitig in den Schubbetrieb, während die vorausfahrenden Fahrzeuge die Ampel unbeeinflusst bei Grün überqueren.

2. Falls nach dem Anhalten vor einer Ampel die Rotphase noch mindestens für die Zeit $t_{minEngineOff}$ andauert, kann der Motor des betreffenden Fahrzeugs abgeschaltet werden. Aufgrund der Kraftstoffanreicherung, die beim Motorstart nötig ist, lohnt sich ein Ausschalten des Motors für weniger als $t_{minEngineOff}$ nicht.

   Dieses Verhalten wird in Abbildung 7.1b gezeigt.

7.2.1 Algorithmus zur Adaption des Fahrverhaltens

Wenn die Ampeln die ankommenden Fahrzeuge über die zukünftigen Phasenwechsel der nächsten $t_{horizon}$ informieren, kann jedes Fahrzeug lokal entscheiden, ob eine der oben beschriebenen Anpassungen des Fahrverhaltens sinnvoll ist und so bei Bedarf in den Schubbetrieb wechseln oder den Motor ausschalten.

```
1: t_minEngineOff                ▷ Zeit, ab der sich ein Ausschalten des Motors lohnt
2: a_fuelCutOff                  ▷ Bremsverzögerung beim Unterbrechen der Treibstoffzufuhr zum Motor
3: v, a                          ▷ Aktuelle Geschwindigkeit und Beschleunigung des Fahrzeugs
4: d_trafficLight                ▷ Distanz zur Ampel trafficLight
5: status_trafficLight           ▷ Status (rot oder grün) der Ampel trafficLight
6: phase_trafficLight            ▷ Verbleibende Zeit der aktuellen Rot- bzw. Grün-Phase von Ampel trafficLight
7: dec   ▷ Betrag der Bremsverzögerung, die nötig ist, um an der nächsten roten Ampel zum Stehen zu kommen
```

Algorithmus 7.1: Lokale Variablen bzw. Konstanten.

Algorithmus 7.1 listet die hierfür notwendigen lokalen Variablen und Konstanten auf. Zur Vereinfachung der Simulation wird eine für alle Geschwindigkeiten konstante Bremsverzögerung $a_{fuelCutOff}$ während des Schubbetriebs angenommen.

Algorithmus 7.2 wird in jedem Fahrzeug in einer Endlosschleife ausgeführt und zeigt das Vorgehen zur Adaption des Fahrverhaltens.

Zunächst wird in Zeile 2 bis 8 über das Ausschalten des Motors entschieden. Wenn das Fahrzeug steht und die Rotphase der nächsten Ampel noch länger als $t_{minEngineOff}$ dauert, wird der Motor ausgeschaltet und der Ablauf pausiert, bis die Rotphase beendet ist. Es wird davon ausgegangen, dass sich der Motor automatisch wieder einschaltet, sobald das Gaspedal gedrückt wird.

Die Zeilen 9 bis 19 des Algorithmus berechnen die Bremsverzögerung, die nötig ist, um mit einer gleichförmigen Verzögerung genau an der Haltelinie der nächsten roten Ampel zum Stehen zu kommen. Dazu wird für alle Ampeln, die sich auf den nächsten r_{data} der Fahrtroute befinden, zunächst die Ankunftszeit $t_{arrival}$ an der jeweiligen Ampel berechnet. Wenn die Ampel zum berechneten Zeitpunkt Rot anzeigt (Zeile 16), wird nachfolgend die Bremsverzögerung berechnet, mit der das Fahrzeug exakt an der Haltelinie der dann roten Ampel zum Stillstand kommt. In Zeile 17 wird die größte so ermittelte Bremsverzögerung in der Variablen *dec* gespeichert.

Bei der Berechnung der Ankunftszeit wird davon ausgegangen, dass eine aktuelle positive Beschleunigung bis zum Erreichen der Ampel anhält (Zeile 14) während eine aktuelle negative Beschleunigung nicht berücksichtigt wird (Zeile 12). Die Ankunftszeit $t_{arrival}$ wird somit prinzipiell eher unterschätzt. Ohne die Einbeziehung der aktuellen Beschleunigung würden Fahrzeuge, die sich in einer Warteschlange vor der Ampel befinden und erst gegen Ende der Grünphase anfahren können, zu früh in den Schubbetrieb gehen, obwohl sie die Ampel bei gleichbleibender Beschleunigung noch während der Grünphase

```
1: loop
2:     if (engine is running) ∧ (v = 0) then          ▷ Wenn das Fahrzeug steht, der Motor läuft ...
3:         trafficLight ← (next traffic light ahead)
4:         if (status_trafficLight = red) ∧ (phase_trafficLight > t_minEngineOff) then   ▷ ... und die nächste Ampel noch
                                                                                          länger Rot ist ...
5:             StopEngine                             ▷ ... dann kann der Motor ausgeschaltet werden.
6:             Sleep(phase_trafficLight)
7:         end if
8:     end if

9:     dec ← 0   ▷ Berechnung des Betrags der Bremsverzögerung, um an der nächsten roten Ampel anzuhalten
10:    for all trafficLight ∈ (traffic lights within r_data of the vehicle's route) do
11:        if (a ≤ 0) then
12:            t_arrival ← d_trafficLight / v
13:        else
14:            t_arrival ← sqrt((v/a)^2 + 2 d_trafficLight / v) − v/a
15:        end if
16:        if ((status_trafficLight at t_arrival) = red) then
17:            dec ← max(dec, v^2 / (2 d_trafficLight))        ▷ Die höchste Bremsverzögerung merken
18:        end if
19:    end for

20:    if (dec ≥ |a_fuelCutOff|) then          ▷ Wenn die nötige Bremsverzögerung höher ist, als die
                                                 Bremsverzögerung im Schubbetrieb, ...
21:        FuelCutOff                           ▷ ... dann kann die Treibstoffzufuhr unterbrochen werden.
22:    else
23:        No intervention                      ▷ Ansonsten wird das Fahrverhalten nicht beeinflusst
24:    end if
25:    Sleep(1)                                 ▷ Vor dem nächsten Schleifendurchlauf kurz warten
26: end loop
```

Algorithmus 7.2: Ampelassistenz: Festlegung der Fahrstrategie.

überqueren könnten.

Beträgt die zum Anhalten vor der nächsten roten Ampel nötige Bremsverzögerung *dec* mindestens der Bremsverzögerung $a_{fuelCutOff}$, die im Schubbetrieb auftritt, dann wird durch Abschaltung der Treibstoffzufuhr zum Motor in den Schubbetrieb gewechselt. Ansonsten wird das Fahrverhalten nicht beeinflusst.

Vor dem nächsten Schleifendurchlauf wird in Zeile 25 für die Zeit eines Simulationsschritts gewartet.

7.2.2 Evaluation

Zur Evaluation wurde die *Ampelassistenz*-Anwendung im Netzwerksimulator ns-2 (Version 2.30) [38] nach der in Abschnitt 4.4 vorgestellten Struktur implementiert. Für die Kommunikation im VANET wird das *AutoCast*-Protokoll verwendet.

Für die Verkehrssimulation wird SUMO eingesetzt [93]. Die beiden Simulatoren sind mittels TraCI gekoppelt (vgl. Kapitel 6). Dabei wird in dieser Anwendung nicht nur jedem Fahrzeug ein Knoten im Netzwerksimulator zugeordnet, sondern darüber hinaus wird auch jede Ampel durch einen Knoten im Netzwerksimulator repräsentiert.

Dazu erfragt der Netzwerksimulator über TraCIs *Scenario*-Nachricht die Anzahl und Position der im Verkehrsszenario vorhandenen Ampeln. Während der Simulation wird von den Ampel-Knoten die *Get Traffic Light Status*-Nachricht verwendet, um zukünftige Phasenwechsel innerhalb des Zeithorizonts $t_{horizon}$ zu erfragen.

Die Knoten im Netzwerksimulator bewegen sich entsprechend ihrer assoziierten Fahrzeuge, wie in Abschnitt 6.3.1 vorgestellt. Zur Simulation des Schubbetriebs wird die *Set Maximum Speed*-Nachricht verwendet, um in jedem Simulationsschritt die Höchstgeschwindigkeit so herabzusetzen, dass das Fahrzeug mit der Bremsverzögerung $a_{fuelCutOff}$ abbremst. Sobald das Fahrzeug zum Stillstand kommt oder der Schubbetrieb aus einem anderen Grund beendet wird, wird die individuelle Geschwindigkeitsbeschränkung des Fahrzeugs aufgehoben.

Um den Abstand d_{tl} zu den auf der Fahrtroute befindlichen Ampeln zu berechnen, wird die *Scenario*-Nachricht verwendet, mit der man die Fahrdistanz von einem Objekt des Verkehrsszenarios zu einer beliebigen Straßennetz-Position erfragen kann. In diesem Fall entspricht das Objekt einem Fahrzeug und die Straßennetz-Position der Haltelinie an einer Ampel.

Somit wird TraCI von der *Ampelassistenz*-Anwendung nicht nur zur Bereitstellung der Bewegungsdaten genutzt, sondern auch zur Anpassung des Fahrverhaltens einzelner Fahrzeuge; und darüber hinaus für den Zugriff auf Details des Verkehrsszenarios.

Das Ausschalten des Motors während der Wartezeit vor einer Ampel ist lediglich für die Berechnung des Treibstoffverbrauchs von Interesse und wird vom nachfolgend vorgestellten Verbrauchsmodell berücksichtigt. Eine Interaktion der Simulatoren ist dafür nicht notwendig.

Evaluationsparameter

Wie bereits am Anfang des Kapitels erwähnt, liegt der Fokus der Auswertung auf der Auswirkung der VANET-Anwendung. Dazu wurde ein einfaches Treibstoffverbrauchsmodell entwickelt, das bei Weitem nicht so detailliert und komplex ist, wie beispielsweise das in [85] vorgestellte HBEFA-Modell. Stattdessen werden grundlegende physikalische Gleichungen verwendet, um den Treibstoffverbrauch basierend auf der in jedem Simulationsschritt der Dauer δt gemessenen Geschwindigkeit und Beschleunigung eines Fahrzeugs näherungsweise zu berechnen. Spezielle Effekte wie Steigungen, Kaltstart, Außentemperatur, usw. werden dabei zwar nicht berücksichtigt, jedoch erlaubt das einfache Modell einen Vergleich des Teibstoffverbrauchs mit und ohne VANET-Anwendung in Relation zueinander.

Zur Berechnung des Treibstoffverbrauchs wird zunächst Gleichung 7.1 verwendet, um die zur Bewegung des Fahrzeugs benötigte Kraft $F(t)$ in Abhängigkeit von dessen aktueller Geschwindigkeit $v(t)$ und Beschleunigung $a(t)$ zu berechnen.

$$F(t) = \frac{\rho}{2} c_w A v(t)^2 \ + mg\mu_r + ma(t) \tag{7.1}$$

Die Kraft $F(t)$ setzt sich zusammen aus dem Luftwiderstand, der Rollreibung und der Massenträgheit des Fahrzeugs. Die in der Formel verwendeten Konstanten werden auf ein Mittelklassefahrzeug abgestimmt. Für den Luftwiderstand wird eine Frontfläche von $A = 2\,\mathrm{m}^2$, eine Luftdichte von $\rho = 1,29\,\mathrm{kg/m}^3$ und ein Luftwiderstandsbeiwert von $c_w = 0,4$ angenommen. Die Fahrzeugmasse, die in die Rollreibung und die Massenträgheit eingeht, wird auf $m = 1400\,\mathrm{kg}$ gesetzt. Der Rollwiderstandskoeffizient wird entsprechend einem Reifen auf Beton auf $\mu_r = 0,015$ gesetzt [157]. Zudem wird die Erdbeschleunigung $g = 9,81\,\mathrm{m/s}^2$ verwendet. Dieser Parametersatz wird auf alle Fahrzeuge in der Simulation angewendet.

Die Energie, die zur Bewegung eines Fahrzeugs benötigt wird, ergibt sich als Produkt aus der Kraft $F(t)$ gemäß Gleichung 7.1 und der während eines Simulationsschritts der Dauer δt zurückgelegten Wegstrecke $v(t) \cdot \delta t$. Die so benötigte Energie lässt sich durch das Produkt aus dem Energiegehalt von Benzin und dem Wirkungsgrad des Motors in eine Benzinmenge umrechnen. Für den Energiegehalt von Benzin werden hier $E_{fuel} = 8,9\,\mathrm{kWh/l}$ angenommen, der Wirkungsgrad des Motors wird als Konstante abgeschätzt zu $\eta = 0,3$. Der Leerlaufverbrauch des Motors stellt in jedem Simulationsschritt die untere Grenze dar; als Leerlaufverbrauch wird $C_{idle} = 1\,\mathrm{l/h}$ angenommen.

Der Schubbetrieb sowie das Ausschalten des Motors werden wie folgt berücksichtigt. Ist die Beschleunigung $a(t)$ eines Fahrzeugs kleiner als $a_{fuelCutOff}$, d. h. bremst das Fahrzeug mindestens mit der durch die Motorbremse bewirkten Bremsleistung, wird im entsprechenden Simulationsschritt kein Treibstoff verbraucht. Für den Schubbetrieb wird $a_{fuelCutOff} = -0,38\,\mathrm{m/s}^2$ angesetzt, was dem Mittelwert der Schubverzögerung im Ge-

schwindigkeitsbereich zwischen 0 und 50 km/h entspricht.

Somit ergibt sich im Schubbetrieb bei einer Anfangsgeschwindigkeit von 50 km/h ein Bremsweg von ca. 254 m. Der Radius zum Verteilen der Datenelemente sollte größer gewählt sein als der Bremsweg, um ein Anhalten an der Haltelinie überhaupt zu ermöglichen. Da sich der Anhalteweg im Schubbetrieb bei einer leicht erhöhten Geschwindigkeit von 60 km/h bereits auf ca. 366 m erhöht, wir der Verteil-Radius mit $r_{data} = 500\,\text{m}$ sehr konservativ gesetzt.

Zur Berücksichtigung von Zeiten, in denen der Motor ausgeschaltet ist, wird neben $v(t)$ und $a(t)$ noch der Status des Motors ($engine(t)$) in jedem Simulationsschritt betrachtet. Bei ausgeschaltetem Motor ist $engine(t) = 0$, bei eingeschaltetem Motor gilt $engine(t) = 1$. Ist also in einem Simulationsschritt der Motor ausgeschaltet, wird kein Treibstoff verbraucht. Allerdings wird beim Einschalten des Motors eine Konstante auf den Treibstoffverbrauch addiert, die dem Mehrverbrauch entspricht, der durch den Start des Motors bedingt ist ($C_{idle} \cdot t_{minEngineOff}$). Falls der Motor kürzer als $t_{minEngineOff}$ ausgeschaltet ist, ergibt sich daher keine Einsparung, sondern ein Mehrverbrauch. Im Folgenden wird $t_{minEngineOff} = 20\,\text{s}$ angesetzt.

Damit ergibt sich der Verbrauch C eines Fahrzeugs zu

$$C = \sum_{\delta t} \begin{cases} 0 & \text{falls } (a(t) < a_{fuelCutOff}) \\ 0 & \text{falls } (engine(t) = 0) \\ C_{idle} \cdot t_{minEngineOff} & \text{falls } (engine(t) = 1) \wedge (engine(t-1) = 0) \\ \text{MAX}(C_{idle} \cdot \delta t, \dfrac{F(t) \cdot v(t) \cdot \delta t}{\eta \cdot E_{fuel}}) & \text{sonst.} \end{cases} \tag{7.2}$$

Der mit Gleichung 7.2 errechnete Treibstoffverbrauch C eines Fahrzeugs hat die Dimension Volumen pro Zeit. Um eine Angabe in der gebräuchlicheren Einheit Liter pro 100 Kilometer zu erhalten, wird der Treibstoffverbrauch jedes Fahrzeugs durch die zurückgelegte Wegstrecke dividiert.

Im letzten Schritt wird der durchschnittliche Treibstoffverbrauch der beteiligten Fahrzeuge ermittelt.

Szenario

Die Simulation des Straßenverkehrs in einem realistischen innerstädtischen Verkehrsszenario ist eine aufwendige Angelegenheit und die Auswertung und Interpretation der Simulationsergebnisse hängt von sehr vielen Parametern ab, die mitunter wenig mit der VANET-Anwendung zu tun haben.

Daher wird zur Evaluation der *Ampelassistenz*-Anwendung ein einfaches Szenario gewählt, in dem sich relevante Parameter einfach anpassen lassen. Wie Abbildung 7.2 zeigt,

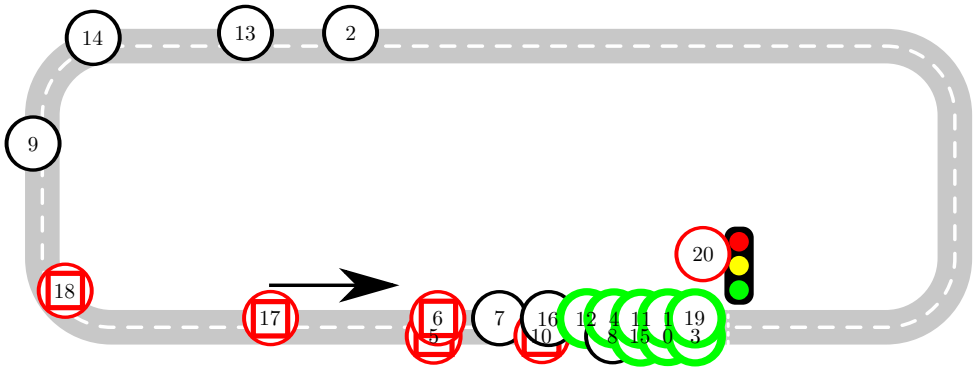


Abbildung 7.2: Für die Evaluation genutztes Verkehrsszenario überlagert mit Fahrzeugen, wie vom Netzwerksimulator gemeldet.

handelt es sich dabei um eine zweispurige Rundstrecke mit einer Ampel. Basierend auf einer groben Abschätzung des mittleren Abstands zwischen Ampeln in deutschen Innenstädten, wurde die Länge der Rundstrecke auf 400 m festgesetzt, d. h. alle 400 m stehen die Fahrzeuge erneut vor einer Ampel.

Zur Variation der Simulation dienen die folgenden drei Parameter:

Anzahl der Fahrzeuge Die Anzahl der Fahrzeuge im Szenario variiert zwischen 2 und 30 Fahrzeugen, entsprechend einer Verkehrsdichte zwischen 2,5 und 37,5 Fahrzeugen pro Kilometer und Fahrspur.

Phasendauer Die Phasendauer beschreibt sowohl die Länge der Rot- als auch der Grünphase. Sie beträgt zwischen 30 s und 70 s.

Ausstattungsgrad Der Anteil der Fahrzeuge, die sich an der VANET-Anwendung beteiligen, beträgt zwischen 0 % und 100 %. Bei einem Ausstattungsgrad von 0 % wird die VANET-Anwendung nicht aktiv. Dieser Fall wird daher als Referenz verwendet.

Jeder Simulationslauf umfasst eine simulierte Dauer von 20 min. Somit wird die Ampel diverse Male von den Fahrzeugen überquert, und es treten je nach Wahl der vorgenannten Parameter verschiedene Verkehrsmuster auf. So entsteht bei günstiger Wahl der Phasendauer eine grüne Welle und die Fahrzeuge müssen kaum anhalten; oder aber sie stehen jede Runde erneut vor einer gerade Rot gewordenen Ampel. Zur Minimierung statistischer Effekte wird jede Simulation 20-mal wiederholt, wobei der Zufallszahlengenerator jeweils unterschiedlich initialisiert wird.

Die maximale Geschwindigkeit der Fahrzeuge beträgt 50 km/h, die für die Kommunikation relevanten Parameter entsprechen denen aus der *AutoCast*-Evaluation (vgl. Abschnitt 5.4.1).

Ergebnisse

Die durchgeführte Simulation wird exemplarisch von Abbildung 7.2 visualisiert. Gezeigt ist ein nachbearbeiteter Screenshot des zu ns-2 gehörenden *Network Animator*. Auf der Fahrbahn bewegen sich 20 Fahrzeuge (Knoten 0–19); die Ampel trägt die Nummer 20.

Der Status der Fahrzeuge hinsichtlich der aktuell angewandten Strategie ist durch eine farbliche und symbolische Markierung dargestellt. Ein dicker grüner Kreis besagt, dass die so markierten Fahrzeuge den Motor ausgeschaltet haben. Fahrzeuge mit einem roten Quadrat befinden sich im Schubbetrieb. Die übrigen Fahrzeuge mit dünnem schwarzen Kreis werden im Moment nicht beeinflusst.

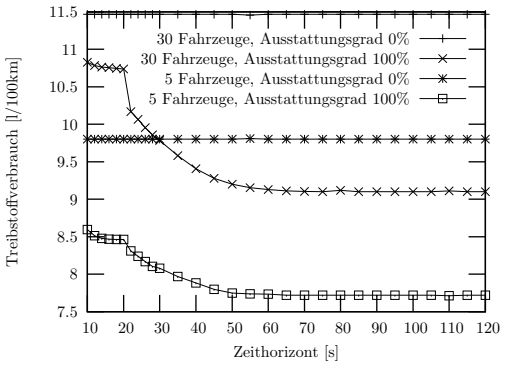
Bei Betrachtung der Fahrzeuge 7, 8 und 16 zeigt sich, dass die vom Verkehrssimulator ausgeführte Mikrosimulation eine höhere Priorität als die Anweisungen der VANET-Anwendung hat. Denn laut Algorithmus 7.2 wird der Schubbetrieb so aktiviert, dass das Fahrzeug genau an der Halteline der Ampel zum Stehen kommt. Hat sich allerdings schon eine Fahrzeugschlange vor der Ampel gebildet, so wird der Schubbetrieb vom Verkehrssimulator durch ein stärkeres Bremsen vorzeitig abgebrochen, um das Fahrzeug am Ende der Schlange anzuhalten. Die Fahrzeuge 7, 8 und 16 befinden sich in Abbildung 7.2 genau in dem Moment des Abbremsens mit einer höheren Bremsverzögerung als $a_{fuelCutOff}$ und werden daher als „nicht beeinflusst“ dargestellt.

Abbildung 7.3 zeigt das Ergebnis der Evaluation. Dabei ist jeweils der Treibstoff auf der Ordinate aufgetragen, der in Abhängigkeit verschiedener Parameter bestimmt ist.

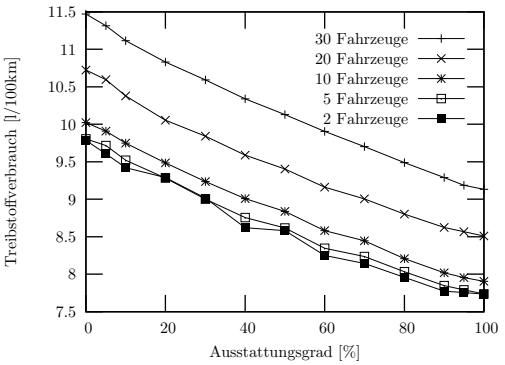
Die erste Auswertung in Abbildung 7.3a zeigt den Einfluss des Zeithorizonts, d. h. der Zeitspanne, für die eine Ampel zukünftige Phasenwechsel vorhersagen kann. Dazu ist der Zeithorizont auf der Abszisse aufgetragen und der Treibstoffverbrauch für zwei verschiedene Verkehrsdichten (5 und 30 Fahrzeuge) und jeweils einen Ausstattungsgrad von 0 % und 100 % eingezeichnet.

Die waagerechten Linien zeigen den Treibstoffverbrauch bei einem Ausstattungsgrad von 0 % und somit ohne Beeinflussung der Fahrzeuge durch die *Ampelassistenz*-Anwendung. Erwartungsgemäß spielt der Zeithorizont einer Ampel keine Rolle, solange kein Fahrzeug die gesendeten Daten auswertet. Bei 30 Fahrzeugen ergibt sich aufgrund der erhöhten Interaktion der Fahrzeuge untereinander ein höherer Treibstoffverbrauch (ca. 11,5 l/100km) als wenn nur 5 Fahrzeuge auf der Strecke unterwegs sind (ca. 9,8 l/100km).

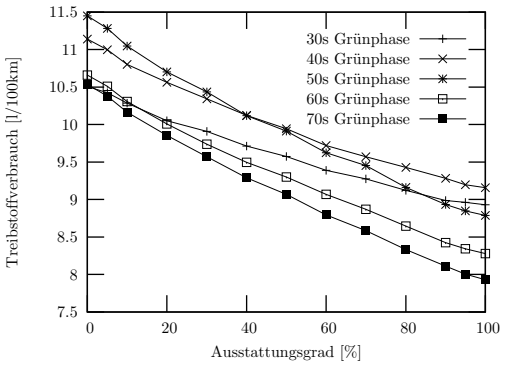
Bei einem Zeithorizont unter 20 s werden die Motoren niemals ausgeschaltet, da die verbleibenden Rotphasen immer kürzer sind als $t_{minEngineOff}$, wenn die Datenelemente bei den (wartenden) Fahrzeugen eintreffen. Mit darüber hinaus ansteigendem Zeithorizont, werden die Motoren für eine immer längere Zeit ausgeschaltet. Folglich sinkt der Treibstoffverbrauch bis ein Zeithorizont von ca. 60 s erreicht ist. Ein höherer Zeithorizont erscheint nicht sinnvoll, da Informationen zu Phasenwechseln bereits versendet werden, wenn diese für die Anpassung des Fahrverhaltens noch nicht relevant sind. Im Schubbetrieb ($a_{fuelCutOff} = -0,38\,\mathrm{m/s^2}$) dauert ein Anhaltevorgang bei einer Höchstge-



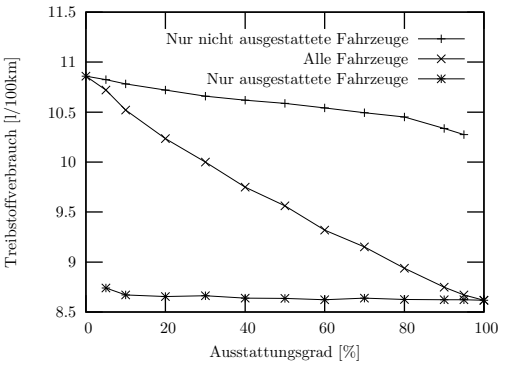
(a) Einfluss des Zeithorizonts auf den Treibstoffverbrauch (Treibstoffverbrauch gemittelt über alle Phasendauern)



(b) Treibstoffverbrauch bei unterschiedlicher Verkehrsdichte (60 s Zeithorizont, Treibstoffverbrauch gemittelt über alle Phasendauern)



(c) Treibstoffverbrauch bei unterschiedlicher Phasendauer (60 s Zeithorizont, Treibstoffverbrauch gemittelt über alle Verkehrsdichten)



(d) Treibstoffverbrauch für ausgestattete und nicht ausgestattete Fahrzeuge (60 s Zeithorizont, Treibstoffverbrauch gemittelt über alle Verkehrsdichten und Phasendauern)

Abbildung 7.3: Evaluationsergebnisse der VANET-Anwendung *Ampelassistenz*.

schwindigkeit von 50 km/h ca. 37 s. Ein darüber hinausgehender Zeithorizont verbessert den Treibstoffverbrauch nicht weiter. Für die weitere Evaluation wird somit ein Zeithorizont von 60 s gewählt.

Bei den Abbildungen 7.3b, 7.3c und 7.3d wird auf der Abszisse der Ausstattungsgrad aufgetragen und somit der Treibstoffverbrauch in Abhängigkeit vom Anteil der an der VANET-Anwendung teilnehmenden Fahrzeuge betrachtet. Der Treibstoffverbrauch ist über alle Fahrzeuge, d. h. über ausgestattete und nicht ausgestattete, gemittelt.

In Abbildung 7.3b wird der Treibstoffverbrauch über alle betrachteten Phasendauern gemittelt. Erwartungsgemäß liegt der Treibstoffverbrauch umso höher je mehr Fahrzeuge

sich auf der Strecke befinden. Unabhängig von der Verkehrsdichte sinkt der Treibstoffverbrauch in jedem Fall mit einem Anstieg des Ausstattungsgrads.

Der stetig sinkende Treibstoffverbrauch bei ansteigendem Ausstattungsgrad wird auch von Abbildung 7.3c bestätigt. Hier wird für den Treibstoffverbrauch jeweils der Mittelwert über alle Verkehrsdichten gebildet. Es zeigt sich, dass es in Bezug auf den Treibstoffverbrauch mehr oder weniger günstige Phasendauern gibt. Bei einem Ausstattungsgrad von 0 % stellt sich offenbar bei einer Phasendauer von 30 s eine grüne Welle ein, während eine Phasendauer von 50 s zu häufigem Anhalten und somit höherem Treibstoffverbrauch führt. Der unterschiedlich stark absinkende Treibstoffverbrauch bei ansteigendem Ausstattungsgrad wird durch die unterschiedlich langen mittleren Wartezeiten bedingt. So kann bei einer längeren Phasendauer der Motor für eine längere Zeit ausgeschaltet werden, was zu einer höheren Treibstoffeinsparung führt als bei einer kurzen Phasendauer.

Interessanterweise resultiert die Treibstoffeinsparung aus beiden Strategien, ist aber je nach Szenario unterschiedlich verteilt. Bei einer geringen Verkehrsdichte werden ca 70–75 % der Treibstoffeinsparung durch den Schubbetrieb erzielt, während bei einer hohen Verkehrsdichte das Ausschalten des Motors während der Wartezeit vor der Ampel 50–65 % der Treibstoffeinsparung ausmacht. Somit tragen beide Strategien wesentlich zur Treibstoffeinsparung bei.

Abbildung 7.3d zeigt den Treibstoffverbrauch gemittelt über alle Phasendauern und Verkehrsdichten. Die Kurve „Alle Fahrzeuge“ bezieht dabei sowohl die ausgestatteten als auch die nicht ausgestatteten Fahrzeuge mit ein und stellt so eine volkswirtschaftliche Perspektive dar. Aus Sicht individueller Verkehrsteilnehmer sind die Kurven „Nur nicht ausgestattete Fahrzeuge“ und „Nur ausgestattete Fahrzeuge“ aussagekräftiger. So profitiert ein Verkehrsteilnehmer auch schon dann in nahezu vollem Umfang, wenn der Ausstattungsgrad noch sehr gering ist. Die *Ampelassistenz*-Anwendung ist somit ein optimaler Kandidat, der auch schon während der Markteinführung vollen Nutzen bringt. Sogar nicht ausgestattete Fahrzeuge profitieren von der vorausschauenden Fahrweise der ausgestatteten Fahrzeuge.

Die von ns-2 durchgeführte drahtlose Kommunikation stellt sich als unkritisch dar. Das *AutoCast*-Protokoll ist sehr betriebssicher und robust gegen Kommunikationsfehler; zudem wird lediglich eine maximale Datenrate von 3 kbit/s im Szenario produziert. Da für die *Ampelassistenz* die Datenverteilung nur in einem kleinen Radius um die Ampel herum notwendig ist, skaliert diese Anwendung problemlos.

7.2.3 Zusammenfassung

Die *Ampelassistenz*-Anwendung zeigt das Potenzial, das VANET-Anwendungen haben. So können durch kleine Empfehlungen zur Anpassung des Fahrverhaltens bereits respektable Treibstoffeinsparungen erzielt werden. Es ist allerdings zu beachten, dass sich die in der Evaluation gezeigten Treibstoffverbräuche ausschließlich auf innerstädtische

Szenarien beziehen, bei denen praktisch alle Ampeln mit VANET-Modulen ausgestattet sind. In der Realität wird ein Verkehrsteilnehmer aber nicht so häufig an entsprechend ausgestatteten Ampeln vorbeikommen, was die Treibstoffeinsparung bezogen auf den Gesamtverbrauch relativiert.

In Bezug auf die verwendeten Algorithmen ist die *Ampelassistenz* eher schlicht gehalten. Als Erweiterung könnten beispielsweise die vorausfahrenden Fahrzeuge bei der Berechnung der Anhalteposition mit einbezogen werden. Im Moment wird immer von einer Anhalteposition direkt an der Haltelinie ausgegangen. Dazu könnten die ankommenden Fahrzeuge die HDC der Ampel mit Daten über ihre eigene Position anreichern.

Auch bei der Entscheidung, ob der Motor ausgeschaltet werden soll, wird die Position innerhalb der Warteschlange vor der Ampel zurzeit nicht berücksichtigt.

7.3 Adaptive Routenplanung

Die Anwendung *adaptive Routenplanung* erlaubt eine dynamische Routenwahl basierend auf mittleren Fahrtzeiten einzelner Streckenabschnitte. Die Streckenabschnitte führen immer von Kreuzung zu Kreuzung. Zur Ermittlung der mittleren Fahrtzeit, die für das Durchfahren eines Streckenabschnitts nötig ist, wird von jedem ausgestatteten Fahrzeug zunächst lokal die benötigte Zeit vom Anfang bis Ende des Streckenabschnitts gemessen. Am Ende des Streckenabschnitts werden diese Daten in einer HDC gespeichert. Eine solche HDC enthält somit Daten zu allen Streckenabschnitten, die zu ihr hinführen.

Die Wahl passender Streckenabschnitte auf Basis einer gemeinsamen Straßenkarte wird beispielsweise in [183] behandelt. Das hier präsentierte Beispiel konzentriert sich auf die Funktionsweise der externen HDC-Kommunikation, d. h. wie HDCs untereinander kommunizieren.

Eine HDC identifiziert sich bei dieser Anwendung über die Kennung der Kreuzung, in deren Bereich sie gespeichert wird, daher wird sie als Kreuzungs-HDC bezeichnet. Die in ihr gespeicherten Streckenabschnitte werden jeweils über die Kreuzungen am Anfang und Ende des Streckenabschnitts identifiziert. Ein Fahrtzeiten-Datenelement, das sowohl zur internen als auch zur externen Kommunikation der Kreuzungs-HDCs genutzt wird, enthält für jeden Streckenabschnitt die folgenden Datenfelder:

- Kennung der Kreuzungen am Anfang und Ende des Streckenabschnitts
- Zeit der letzten Messung, d. h. wann hat das letzte Mal ein Fahrzeug seine Fahrtzeit zur HDC beigetragen
- Anzahl der Fahrzeuge, deren Fahrtzeiten einbezogen wurden
- Mittlere Fahrtzeit

- Zeit der letzten Meldung, d. h. wann wurde die letzte Meldung vom Ende des Streckenabschnitts an dessen Anfang geschickt

Nähert sich ein Fahrzeug einer Kreuzung, an der bereits eine Kreuzungs-HDC vorhanden ist, so empfängt es diese Daten und speichert sie in einer lokalen HDC-Instanz. Sobald das Fahrzeug die Kreuzung erreicht hat, integriert es die eigene Fahrtzeit des gerade durchfahrenen Streckenabschnitts in die HDC. Ausgehend von der Zeit der letzten Messung erhalten die „alten“ Daten mehr oder weniger viel Gewicht bei der Bildung des Durchschnitts. Entsprechend der Gewichtung wird auch die Anzahl einbezogener Fahrzeuge angepasst.

Der so veränderte Inhalt der HDC wird wiederum in Form eines Datenelements als Broadcast gesendet und von den weiteren Fahrzeugen im Bereich der Kreuzungs-HDC ebenso integriert.

Ausgehend von der Zeit der letzten Meldung wird jeweils nach Ablauf der Zeit $t_{extract}$ ein Datenelement mit *AutoCast* an jede Kreuzung am Anfang der gespeicherten Streckenabschnitte gesendet. Die dort angekommenen Datenelemente werden ebenso in die Kreuzungs-HDC aufgenommen und integriert wie am Ende des Streckenabschnitts.

Somit erhalten alle ausgestatteten Fahrzeuge, die auf eine Kreuzung zufahren die aktuellen Fahrtzeiten der nachfolgenden Streckenabschnitte und können sich spontan für diejenige mit der kürzesten Fahrtzeit entscheiden.

Die weitere Aggregation der Streckenabschnitts-Fahrtzeiten im Sinne der Fischaugenperspektive ist in diesem Beispiel noch nicht umgesetzt. Stattdessen wurde das Verkehrsszenario der nachfolgend beschriebenen Evaluation so gewählt, dass es bereits zwischen zwei Kreuzungen einen alternativen Streckenabschnitt gibt.

7.3.1 Evaluation

Auch diese Evaluation wurde mithilfe einer Kombination des Netzwerksimulators ns-2 [38] und des Verkehrssimulators SUMO [93] durchgeführt, die über TraCI gekoppelt wurden (vgl. Kapitel 6).

TraCI sorgt für die synchrone Bewegung der Knoten im Netzwerksimulator auf Basis der Bewegung der ihnen assoziierten Fahrzeuge im Verkehrssimulator. Der Ausstattungsgrad wird für die Evaluation zwischen 0 % und 100 % variiert, so dass nur ein entsprechender Anteil der Fahrzeuge an den Netzwerksimulator gemeldet wird.

Zur Beeinflussung des Fahrverhaltens wird bei der *Adaptiven Routenplanung* die TraCI-Nachricht *Change Route* verwendet. Damit kann für ein ausgewähltes Fahrzeug eine individuelle Fahrtzeit für einen Streckenabschnitt an das Routenplanungsmodul des Verkehrssimulators gegeben werden. Unter Berücksichtigung dieser neuen (geschätzten) Fahrtzeit wird die Fahrtroute für das betroffene Fahrzeug erneut berechnet und gegebenenfalls angepasst.

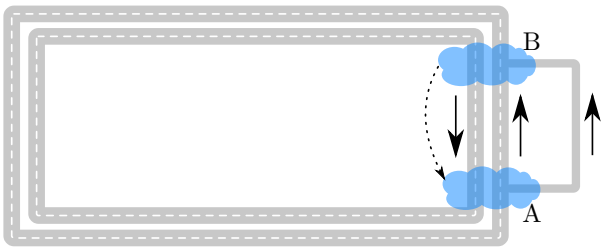


Abbildung 7.4: Straßenszenario zur Evaluierung der adaptiven Routenplanung.

Szenario

Das Verkehrsszenario, mit dem diese Evaluation durchgeführt wird, zeigt Abbildung 7.4. Es besteht aus einer Rundstrecke mit 78,5 km Länge. Beide Richtungsfahrbahnen verfügen über zwei Spuren und in jede Richung fahren jeweils 1500 Fahrzeuge, was im Mittel einer Verkehrsdichte von 9,5 Fahrzeugen/km/Fahrspur entspricht.

Zwischen den Kreuzungen A und B, die auf der Rundstrecke durch einen 7,5 km langen Streckenabschnitt verbunden sind, befindet sich eine zusätzliche Umleitungsstrecke von 15 km Länge. Somit wird die Umleitungsstrecke normalerweise nicht benutzt, kann aber dynamisch als Umleitung verwendet werden.

Die Simulation umfasst einen Zeitraum von 120 min. Nach 65 min wird mithilfe der TraCI-Nachricht *Set Maximum Speed* ein Fahrzeug kurz vor der Kreuzung B angehalten und so ein Unfall simuliert, bei dem eine Fahrspur blockiert ist. Dies führt wie erwartet zu einem Verkehrsstau, der die Fahrtzeit zwischen Kreuzung A und B deutlich erhöht.

Evaluationsparameter

Als Evaluationsparameter dient hier die mittlere Fahrtzeit aller Fahrzeuge, die vom Verkehrssimulator SUMO für jeden Streckenabschnitt im Minutentakt ermittelt wurde. Im Speziellen wird die mittlere Fahrtzeit zwischen den Kreuzungen A und B betrachtet. Die beiden Strecken wurden entsprechend der auf ihnen befindlichen Fahrzeuge in die Berechnung der mittleren Fahrtzeit einbezogen.

Ergebnis

Abbildung 7.5 zeigt auf der Abszisse die simulierte Zeit und auf der Ordinate die mittlere Fahrtzeit von Kreuzung A nach B. Für verschiedene Ausstattungsgrade ist jeweils eine Kurve in das Diagramm eingezeichnet.

Vor dem Unfall, also vor der 65. Minute beträgt die Fahrtzeit ca. 5 min, was einer durchschnittlichen Geschwindigkeit von 90 km/h entspricht. Nach dem Unfall nimmt die Fahrtzeit entsprechend der Länge des Rückstaus an der Unfallstelle zu.

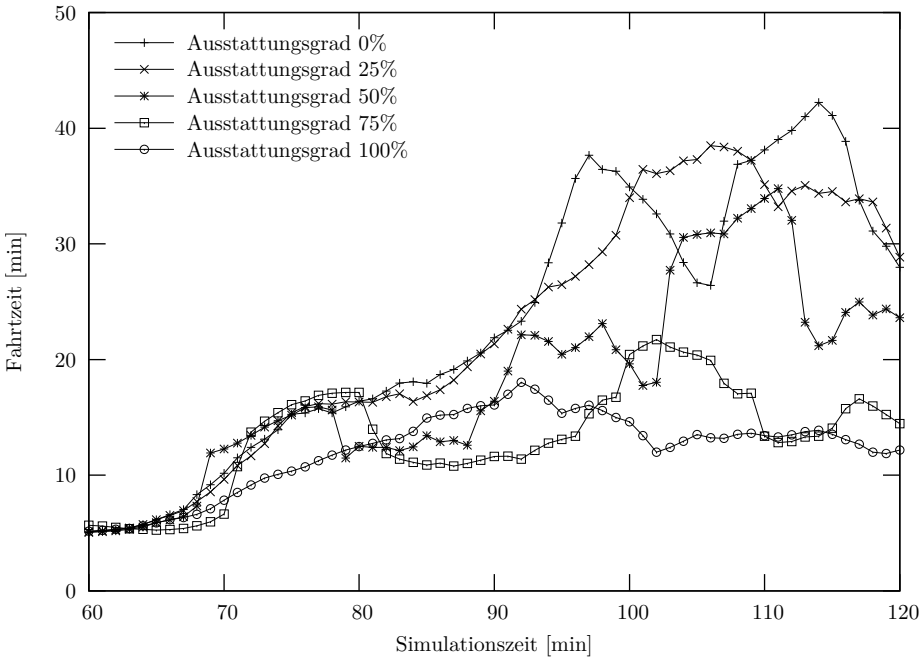


Abbildung 7.5: Mittlere Fahrtzeit von A nach B (vgl. Abbildung 7.4) bei verschiedenen Ausstattungsgraden.

Ohne diese Anwendung, d. h. bei einem Ausstattungsgrad von 0 %, steigt die Fahrtzeit nach dem Unfall zunächst kontinuierlich an. Da der Verkehrsfluss nicht vollkommen homogen ist, schwankt die mittlere Fahrtzeit ab der 95. Minute zwischen 26 min und 42 min.

Mit zunehmendem Ausstattungsgrad zeigt sich, dass die maximale Fahrtzeit immer weiter sinkt, bis bei einem Ausstattungsgrad von 100 % maximal 18 min für den Streckenabschnitt zwischen A und B benötigt werden. Die Kurvenverläufe und damit die mittleren Fahrtzeiten bei geringerem Ausstattungsgrad unterliegen jedoch relativ hohen Schwankungen, die teilweise sogar zu einer zeitweise höheren Fahrtzeit führen als ohne diese Anwendung.

7.3.2 Zusammenfassung

Auch wenn diese VANET-Anwendung im verkehrstechnischen Sinn noch weiterer Optimierung bedarf, zeigt sich doch das Potenzial, das in dieser Anwendung steckt.

Die Datenhaltung in HDCs hat sich bei der Umsetzung dieser Anwendung als sehr hilfreich erwiesen und die Komplexität der Anwendung deutlich gesenkt. Denn der Anwendungsentwickler muss sich keine Gedanken um die Weitergabe und Verwaltung der

Daten machen, sondern kann sich darauf konzentrieren, an welchen Positionen bzw. unter welchen Voraussetzungen Verkehrsdaten erhoben und wohin diese Verkehrsdaten geleitet werden sollen, um die Entscheidungen betroffener Verkehrsteilnehmer zu beeinflussen.

Die *adaptive Routenplanung* sollte weiter untersucht werden. Es sind einerseits Aggregationsregeln zu finden, mit denen Fahrtzeiten in beliebigen Straßennetzen ermittelt und auch auf längere Streckenabschnitte bezogen werden können, andererseits muss ein Algorithmus gefunden werden, der die Entscheidung, welche Route gewählt wird, optimiert, so dass – unabhängig vom Ausstattungsgrad – eine gleichmäßigere Verteilung der Fahrzeuge auf alternative Routen entsteht.

Hinsichtlich der angewandten Strategie, zeigte sich bei der vorhergehenden Evaluation, dass die Anzahl der Fahrzeuge, die die Umleitungsstrecke gewählt haben, über die Simulationszeit stark schwankte. Auch hier sollten weitere Untersuchungen angestellt werden insbesondere im Hinblick auf die Auswirkungen verschiedener Ausstattungsgrade.

7.4 Dezentrale Stauerkennung

Die dritte Anwendung nutzt das Konzept der HDCs und OICs, um Verkehrsstaus dezentral zu erkennen und über die Zeit zu verfolgen. Das Verfahren zur Stauerkennung wurde im Rahmen der Diplomarbeit von Nommensen [129] entwickelt und basiert auf hierarchischer Aggregation von Brems- und Beschleunigungsdaten. Dabei wird zunächst lokal das Bremsen bzw. Beschleunigen detektiert. Aus einer gewissen Anzahl korrelierender Brems- bzw. Beschleunigungsvorgänge wird auf ein Stauende bzw. einen Stauanfang geschlossen. Stauende und Stauanfang werden in HDCs gespeichert, die ihre Position aufgrund des Einflusses weiterer Brems- und Beschleunigungsvorgänge anpassen. Durch Aggregation der Positionsdaten zweier aufeinander folgender Stauende- und Stauanfang-HDCs lässt sich die Gesamtlänge des Staus berechnen. Das Verfahren wurde unter Verwendung der Simulatoren SUMO und ns-2 evaluiert und stellte sich als vielversprechend dar.

Diese Anwendung diente bereits in Abschnitt 3.2 zur Verdeutlichung der Konzepte HDC und OIC.

Im folgenden Abschnitt wird die Anwendung *Stauerkennung* allerdings nicht in einer Simulationsumgebung evaluiert, sondern mithilfe eines Demonstrators, bestehend aus Lego-Mindstorms-Fahrzeugen, die sich auf einer kreisförmigen Fahrbahn bewegen. Der Einsatz eines Demonstrators bietet gegenüber einer Simulation einige Vorzüge. So lassen sich einerseits die erzielten Forschungsergebnisse sehr anschaulich präsentieren, andererseits erlaubt der Demonstrator ein direktes Eingreifen ins Verkehrsszenario, wodurch das Verhalten von VANET-Anwendungen unter Einfluss verschiedener äußerer Faktoren evaluiert werden kann.

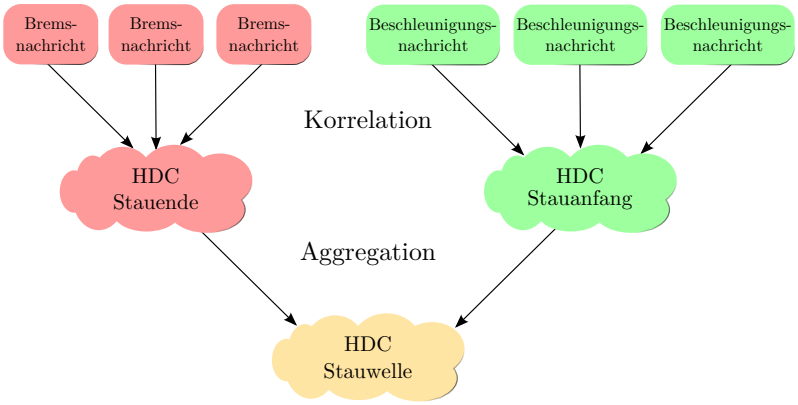


Abbildung 7.6: Hierarchische Korrelation und Aggregation von Daten zur dezentralen Erkennung eines Verkehrsstaus.

7.4.1 Algorithmus

Der Algorithmus zur Stauerkennung arbeitet in mehreren Stufen, wie in Abbildung 7.6 dargestellt.

Zunächst wird von der Funktionseinheit *Datenauswertung* (vgl. Abschnitt 4.3) das eigene Fahrverhalten hinsichtlich der aktuellen Geschwindigkeit v und der Beschleunigung a überwacht. Wird innerhalb einer Zeitspanne t_{brake} von einer relativ hohen Geschwindigkeit v_{brake}^{before} auf eine sehr niedrige Geschwindigkeit v_{brake}^{after} abgebremst, so wird ein *Brems*-Datenelement erzeugt, welches die aktuelle Position des Fahrzeugs enthält.

Über die Funktionseinheit *Korrelation & Integration* wird das so erzeugte Datenelement als *Stauende*-HDC in die lokale *HDC-Sammlung* aufgenommen, und zeitgleich als *Brems*-Datenelement an die Transportschicht weitergegeben, um per Broadcast an die benachbarten Fahrzeuge gesendet zu werden. Somit wird auch in den benachbarten Fahrzeugen eine *Stauende*-HDC etabliert.

In der *Stauende*-HDC werden die eingegangenen Bremsvorgänge gezählt und die Position des Stauendes jeweils an den letzten Bremsvorgang angepasst. Erst beim Überschreiten einer Anzahl c_{brake} an Bremsvorgängen wird die ermittelte Position tatsächlich als Stauende interpretiert und regelmäßig in Form eines *Stauende*-Datenelements in Fahrtrichtung gesendet.

Analog dazu wird der Stauanfang erkannt, wenn innerhalb einer Zeitspanne t_{accel} ausgehend von einer sehr niedrigen Geschwindigkeit v_{accel}^{before} die Geschwindigkeit v_{accel}^{after} erreicht wird. In diesem Fall wird ein *Beschleunigungs*-Datenelement erzeugt, das die Position des Beschleunigungsvorgangs enthält.

Aus diesen *Beschleunigungs*-Datenelementen wird, wie beim Stauende, eine *Stauanfang*-HDC gebildet und die Anzahl an Beschleunigungsvorgängen gezählt, deren Daten in

Abbildung 7.7: Lego-Mindstorms-Fahrzeug.

die *Stauanfang*-HDC eingeflossen sind. Sobald mehr als c_{accel} Beschleunigungsvorgänge gezählt wurden, wird regelmäßig ein *Stauanfang*-Datenelement extrahiert und entgegen der Fahrtrichtung gesendet.

Dort wo die Datenelemente *Stauanfang* und *Stauende* aufeinandertreffen, wird ihre weitere Verteilung gestoppt und die durch die beiden Datenelemente gebildeten *Stauanfang*- und *Stauende*-HDCs zu einer *Stauwelle*-HDC aggregiert.

7.4.2 Aufbau der Demonstrationsplattform

Der Demonstrator besteht aus insgesamt 18 Fahrzeugen, die aus Bauelementen der Lego-Mindstorms-NXT Produktreihe [1] konstruiert sind. Diese Fahrzeuge folgen einer farblich hervorgehobenen Fahrspur und können mithilfe eines Ultraschallsensors kollisionsfrei hintereinander herfahren. Die Kommunikation erfolgt über die Bluetooth-Schnittstelle der Fahrzeuge, wobei eine Java-Software auf einem PC zur Emulation einer Broadcast-Funkschnittstelle eingesetzt wird.

Die einzelnen Elemente dieses „Stau-Demonstrators“ werden nachfolgend erläutert.

Lego-Mindstorms-Fahrzeuge

Die Fahrzeuge sind jeweils auf Basis eines Lego-Mindstorms-NXT-Sets Nr. 8527 konstruiert. Abbildung 7.7 zeigt ein solches Fahrzeug.

Auf der Oberseite des Fahrzeugs ist deutlich der NXT-Hauptbaustein – im Folgenden kurz NXT genannt – zu erkennen, in dem sich die Recheneinheit befindet und der über

etliche Ein- und Ausgänge verfügt. Als Recheneinheit dient darin ein Atmel 32-bit ARM Prozessor mit 256 kB Flash ROM und 64 kB RAM, der mit 48 MHz getaktet ist. Diesem steht ein weiterer mit 8 MHz getakteter Atmel 8 bit Prozessor mit 4 kB Flash ROM und 512 Byte RAM zur Seite, der die Ein- und Ausgänge ansteuert.

Über vier Eingänge können sowohl analoge als auch digitale Signale gemessen werden. An jedem Eingang kann ein I^2C-Bus [130] genutzt werden, wobei der NXT die Rolle des I^2C-Masters einnimmt. Drei Ausgänge erlauben den Anschluss von Servomotoren. In den Servomotoren ist ein Rotationssensor integriert, so dass die Drehgeschwindigkeit und Winkelstellung des Motors ermittelt werden kann. Daneben vefügt der NXT über ein grafisches Display und vier Tasten an der Oberseite. Zur Kommunikation ist sowohl ein USB-Anschluss als auch ein Bluetooth-Baustein (CSR BlueCore 4) vorhanden. Eine technische Dokumentation zur NXT-Hardware findet sich in [98].

Das Fahrzeug ist von der Bauweise her einem echten Fahrzeug nachempfunden und verfügt über zwei Achsen mit insgesamt vier Rädern. Ein Motor steuert die Vorderradlenkung. An der Lenkung ist ein Lichtsensor befestigt (rechts in Abbildung 7.7), der sich mit der Lenkbewegung nach rechts und links bewegt. Ein Ultraschallsensor schwenkt ebenfalls mit der Lenkung und dient der Entfernungsmessung zum vorausfahrenden Fahrzeug. Ein weiterer Motor wird für den Hinterradantrieb eingesetzt; dabei ist der Motor über ein Differenzialgetriebe mit den Rädern verbunden.

Im hinteren Bereich des Fahrzeugs (links in Abbildung 7.7) befinden sich vier Leuchtdioden, die über einen I^2C-Bus an den NXT angeschlossen sind und zur Anzeige von Betriebszuständen dienen.

Zur Programmierung der NXTs wird die Programmiersprache *Not eXactly C* (NXC) [61] verwendet, die einen Sprachumfang anbietet, der bis auf die Unterstützung von Gleitkommazahlen und mehrdimensionalen Arrays, mit dem von Ansi-C vergleichbar ist. Obowhl die Syntax an einigen Stellen vom Ansi-C-Standard abweicht, ist mit NXC in Verbindung mit der zugehörigen IDE *Bricx Command Center* (BricxCC)[2] eine effiziente Programmierung für die NXT-Plattform möglich, die auch wissenschaftlichen Ansprüchen genügt.

Bewegungsmodell

Im Rahmen einer Studienarbeit [161] wurde ein Algorithmus entwickelt, der den Fahrzeugen das autonome Fahren ermöglicht. Die Aufgabe „Fahren“ ist dabei in zwei Teilprobleme unterteilt:

Lenken Mit Hilfe des Lichtsensors kann ein Fahrzeug entlang einer Hell-Dunkel-Kante fahren (vgl. Abbildung 7.7). Der dazu verwendete Algorithmus kommt mit zwei

[2] `http://bricxcc.sourceforge.net`

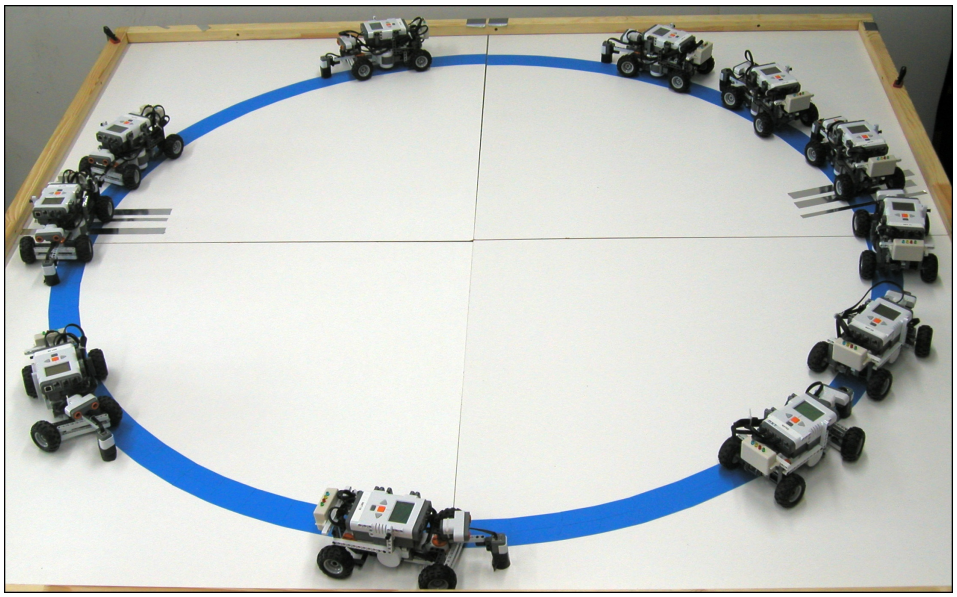


Abbildung 7.8: Verkehrsszenario des Demonstrators.

einfachen Regeln aus: Entspricht der vom Lichtsensor gemessene Wert der Fahrspur, dann lenke nach links; wenn der Helligkeitswert hingegen dem Untergrund der Fahrspur entspricht, dann lenke nach rechts. Durch die Anbringung des Lichtsensors am Lenkgestänge wird die Wirksamkeit des Verfahrens erhöht, da das Fahrzeug praktisch dem auf der Hell-Dunkel-Kante „vorauseilenden“ Lichtsensor folgt.

Abstand halten Die Regelung der Fahrgeschwindigkeit eines Fahrzeuges wurde in Anlehnung an das vom Verkehrssimulator SUMO verwendete Krauß-Modell [95] als Fahrzeug-Folgemodell ausgelegt; sie richtet sich also ausschließlich nach dem vorausfahrenden Fahrzeug. Jedes Fahrzeug hat eine Höchstgeschwindigkeit v_{max} und eine maximale Beschleunigung a_{max}, die von der Geschwindigkeitsregeluung berücksichtigt werden. Die im Krauß-Modell vorhandene zufällige Verzögerung ist hier nicht nötig, da der vom Ultraschallsensor zurückgelieferte Abstand zum vorausfahrenden Fahrzeug einer ausreichenden Messungenauigkeit unterliegt.

Fahrbahn

Von dem im vorhergehenden beschriebenen Fahrzeug wurden insgesamt 18 Exemplare angefertigt. Diese bewegen sich auf einer Fahrbahn, die durch einen dunklen Streifen auf hellem Hintergrund vorgegeben ist. Abbildung 7.8 zeigt das im Folgenden verwendete

Tabelle 7.1: Verkehrsmuster im Demonstrator in Abhängigkeit von der Fahrzeuganzahl (Rundstrecke mit ca. 5 m Umfang).

Anzahl Fahrzeuge	**Verkehrsmuster**
1–4	Fließender Verkehr
5–7	Kolonnenbildung
8–9	Stau mit mehreren Stauwellen
10–13	Stau mit einer Stauwelle
14–15	Verkehr kommt zum Erliegen

Verkehrsszenario, das aus einer ca. 5 m langen kreisförmigen Strecke besteht. Obwohl das Verkehrsszenario kaum einfacher sein könnte, bilden sich mithilfe des in den Fahrzeugen implementierten Fahrverhaltens je nach Verkehrsdichte verschiedene Verkehrsmuster (vgl. Tabelle 7.1). In Abbildung 7.8 fahren 11 Fahrzeuge auf der Rundstrecke, der entstandene Stau mit einer einzigen Stauwelle ist deutlich zu erkennen.

Aufgrund des eingeschränkten Horizonts eines einzelnen Fahrzeugs erkennt dieses nicht, dass es immer wieder in dieselbe Stauwelle hineinfährt.

Um eine Ortung der einzelnen Fahrzeuge auf der Strecke zu ermöglichen, sind orthogonal zur Fahrtrichtung reflektierende Streifen unterschiedlicher Breite angebracht. Diese „Barcodes“ werden von den Lichtsensoren der Fahrzeuge erkannt und jedem einzelnen Streifen entsprechend seiner Breite ein binärer Wert „0“ bzw. „1“ zugeordnet, so dass sich Zahlen im Bereich $[0; 2^{AnzahlStreifen} - 1]$ codieren lassen. Jeder Barcode leitet einen Streckenabschnitt ein, dessen Kennung der im Barcode codierten Zahl entspricht.

Am Anfang eines Streckenabschnitts wird der Rotationszähler des Fahrmotors zurückgesetzt, um nachfolgend die Position auf dem jeweiligen Streckenabschnitt zu enthalten.

Kommunikationsmodell

In einer weiteren Studienarbeit [113] wurde ein Kommunikationsmodell entwickelt, das auf Basis der in den NXTs vorhandenen Bluetooth-Schnittstelle eine Broadcast-Kommunikation emuliert.

Obwohl Bluetooth einen Funkstandard darstellt und sich somit alle Bluetooth-Geräte ein gemeinsames Medium teilen, ist Bluetooth ausschließlich für die Unicast-Kommunikation innerhalb eines *Piconetzes* ausgelegt. In einem Piconetz tritt immer ein Bluetooth-Gerät als Master auf, der mit maximal sieben Bluetooth-Slaves kommuniziert (vgl. Bluetooth Spezifikation [20]).

Da sich ein NXT als Bluetooth-Master aufgrund einer Einschränkung innerhalb der Firmware nur mit maximal drei weitern NXTs simultan per Bluetooth verbinden und auch nicht gleichzeitig als Bluetooth-Master und -Slave auftreten kann, ist der Einsatz eines externen Masters (PC) notwendig. Dieser wird als Bluetooth-Master eingesetzt,

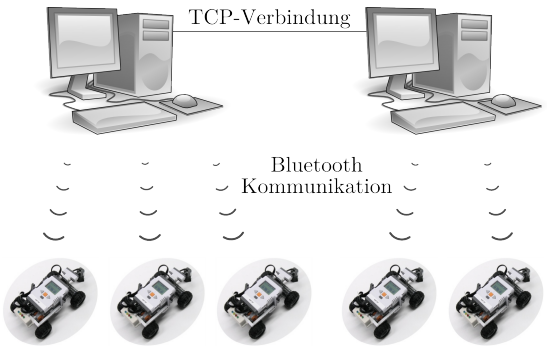


Abbildung 7.9: Emulation von Broadcast-Kommunikation auf Basis von Bluetooth.

der sich mit maximal sieben NXTs verbindet.

Da im Demonstrator mehr als sieben Fahrzeuge zum Einsatz kommen, wurde eine Java-Software entwickelt, die über mehrere Bluetooth-Schnittstellen, die sich auch auf mehreren Rechnern befinden können, jeweils bis zu sieben Verbindungen zu den NXTs herstellt. Die Architektur des Kommunikations-Systems zeigt Abbildung 7.9. Somit wird vom PC zu jedem NXT eine serielle Verbindung gemäß dem Bluetooth-Profil SPP [19] aufgebaut. Die Java-Software wird im Folgenden als Kommunikationsserver bezeichnet.

Über die Verbindungen zu den Fahrzeugen kann der Kommunikationsserver Steuerkommandos an jedes einzelne Fahrzeug senden, die direkt von der NXT-Firmware interpretiert werden (vgl [97]). Somit lassen sich beispielsweise Programme auf ein Fahrzeug übertragen, starten und auch wieder stoppen. In die Gegenrichtung senden die Fahrzeuge regelmäßig ihre Position in Form der Kennung des Streckenabschnitts und der Position auf dem Streckenabschnitt zum Kommunikationsserver.

Die Positionsdaten der Fahrzeuge werden vom eigentlichen Kommunikationsmodell verwendet, um die Empfänger einer Broadcast-Nachricht zu bestimmen. Gemäß dem Unit-Disk-Graph-Modell empfangen alle Fahrzeuge innerhalb des Kommunikationsradius r des Senders die gesendete Nachricht. Das Kommunikationsmodell erlaubt zudem die Einstellung einer Verlustrate p_{drop}, so dass die Fahrzeuge innerhalb des Kommunikationsradius eine Nachricht mit der Wahrscheinlichkeit $1 - p_{drop}$ empfangen.

Aufgrund der beschränkten Hardwareressourcen und der limitierten Anzahl miteinander kommunizierender Fahrzeuge wurde auf eine Implementierung des *AutoCast*-Protokolls (vgl. Kapitel 5) im Demonstrator verzichtet. Stattdessen kommt *einfaches Fluten* zum Einsatz, um ein Datenelement innerhalb seines Verbreitungsgebiets zu verteilen.

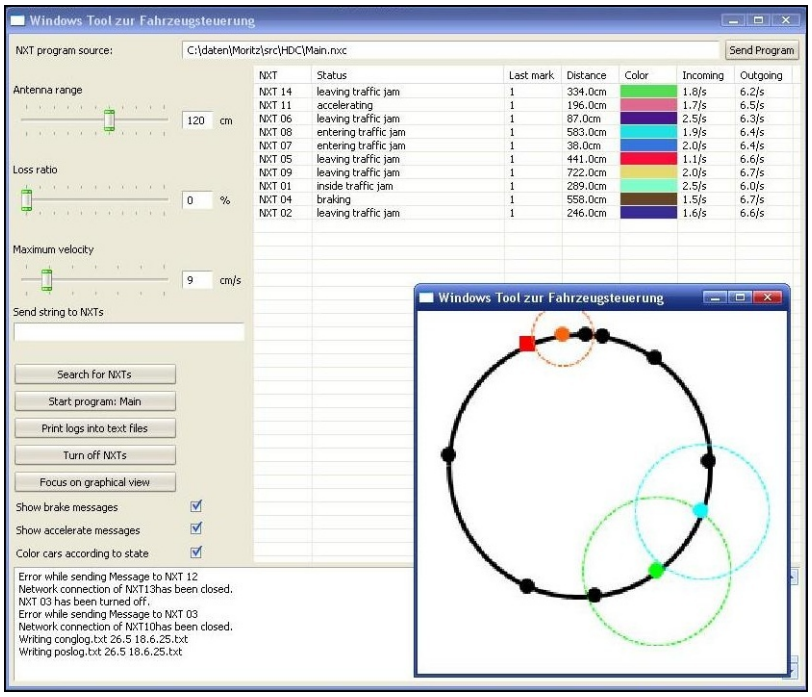


Abbildung 7.10: Grafische Benutzeroberfläche zur Steuerung des Demonstrators.

7.4.3 Evaluation

Im Rahmen einer weiteren Bachelorarbeit wurde der in Abschnitt 7.4.1 beschriebene Algorithmus in NXC implementiert (vgl. [15]). In der durchgeführten Evaluation wurden die Konstanten des Algorithmus gemäß Algorithmus 7.3 gesetzt.

1:	$t_{brake} \leftarrow 0,8\,\mathrm{s}$	▷ Maximale Beobachtungsdauer eines Bremsvorgangs
2:	$v_{brake}^{before} \leftarrow 120\,\mathrm{mm/s}$	▷ Mindest-Geschwindigkeit zu Beginn des Bremsvorgangs
3:	$v_{brake}^{after} \leftarrow 0\,\mathrm{mm/s}$	▷ Maximal-Geschwindigkeit nach dem Bremsvorgang
4:	$c_{brake} \leftarrow 3$	▷ Anzahl der nötigen Bremsvorgänge zur Etablierung einer *Stauende*-HDC
5:	$t_{accel} \leftarrow 1,2\,\mathrm{s}$	▷ Maximale Beobachtungsdauer eines Beschleunigungsvorgangs
6:	$v_{accel}^{before} \leftarrow 0\,\mathrm{mm/s}$	▷ Maximal-Geschwindigkeit zu Beginn des Beschleunigungsvorgangs
7:	$v_{accel}^{after} \leftarrow 120\,\mathrm{mm/s}$	▷ Minimal-Geschwindigkeit nach dem Beschleunigungsvorgang
8:	$c_{accel} \leftarrow 3$	▷ Anzahl der nötigen Beschleunigungsvorgänge zur Etablierung einer *Stauanfang*-HDC

Algorithmus 7.3: Konstanten des Algorithmus zur Stauerkennung.

Zur Durchführung der Experimente innerhalb des Demonstrators wurde in einer wei-

teren Bachelorarbeit [63] eine grafische Benutzeroberfläche in Java entwickelt, die den im vorhergehenden vorgestellten Kommunikationsserver kapselt und auch die Bluetooth-Verbindungen zu den Fahrzeugen aufbaut. Abbildung 7.10 zeigt einen Screenshot dieses Programms. Es werden alle Fahrzeuge, zu denen eine Bluetooth-Verbindung besteht, in einer Liste angezeigt. Darüber hinaus werden die Positionen der Fahrzeuge grafisch dargestellt. Gesendete Nachrichten sind in der grafischen Ansicht kurzzeitig als Kreise sichtbar.

Darüber hinaus lassen sich einige Parameter des Demonstrators variieren, z. B. der Kommunikationsradius, die Verlustrate für Nachrichten und auch die Höchstgeschwindigkeit der Fahrzeuge. NXC-Programme lassen sich kompilieren und über Bluetooth an die Fahrzeuge senden. Danach können die Programme gleichzeitig auf allen Fahrzeugen gestartet und auch wieder angehalten werden.

Abbildung 7.11 zeigt das Ergebnis eines Experiments, bei dem sich 10 Fahrzeuge mit einer Höchstgeschwindigkeit von $v_{max} = 300\,\mathrm{mm/s}$ auf der Rundstrecke bewegt haben. Der Kommunikationsradius jedes Fahrzeugs beträgt $r = 1\,\mathrm{m}$.

Die Fahrzeuge haben ihre Positionen ca. alle 150 ms an den Kommunikationsserver gesendet. In Abbildung 7.11 ist auf der Abszisse die Position der Fahrzeuge auf der Rundstrecke aufgetragen, während die Ordinate den Zeitpunkt der Messung wiedergibt. Die einzelnen Positionen der Fahrzeuge werden im Diagramm zu Trajektorien verbunden. Eine flache Trajektorie steht dabei für eine große Ortsänderung in kurzer Zeit, also für eine hohe Geschwindigkeit. Im Stau hingegen verändert sich die Position eines Fahrzeugs eine Zeit lang kaum, was in einer sehr steilen bis senkrechten Trajektorie resultiert. Abgebrochene Trajektorien, beispielsweise bei ca. 4 min und 80 cm sowie bei 6 min und 80 cm, weisen auf ein Kommunikationsproblem eines einzelnen Fahrzeugs hin; während des Experiments wurde kein Fahrzeug von der Strecke genommen.

Aus dieser „Vogelperspektive“ ist die Stauwelle sofort als Bereich mit senkrechten Trajektorien erkennbar. Gut sichtbar ist auch, dass sich die Stauwelle rückwärts bewegt, da Fahrzeuge vorne aus dem Stau herausfahren und hinten neue hinzukommen.

Die Ergebnisse der dezentralen Stauerkennung sind in Abbildung 7.11 den Trajektorien überlagert. Kreise und Dreiecke zeigen die Positionen der Stauanfang- bzw. Stauende-HDCs immer dann, wenn neue Daten in die jeweilige HDC integriert wurden, d. h. immer wenn ein neuer Beschleunigungs- oder Bremsvorgang erkannt wurde. Offenbar wurden das Stauende und der Stauanfang von fast jedem Fahrzeug erkannt. Darüber hinaus wurden außerhalb der Stauwelle nur selten falsche Stauenden und -anfänge detektiert.

Die Stau-HDCs sind als waagerechte Balken in die Grafik eingezeichnet. Die Lage der Balken zeigt die Position und die Länge des zum jeweiligen Zeitpunkt ermittelten Staus.

An dieser Stelle sei angemerkt, dass die in Algorithmus 7.3 gesetzten Konstanten an das Fahrverhalten, d. h. das Brems- und Beschleunigungsverhalten, der Fahrzeuge angepasst sind. Eine Änderung beispielsweise der maximalen Beschleunigung muss auch eine Anpassung von t_{accel} nach sich ziehen, da sonst Beschleunigungsvorgänge nicht mehr

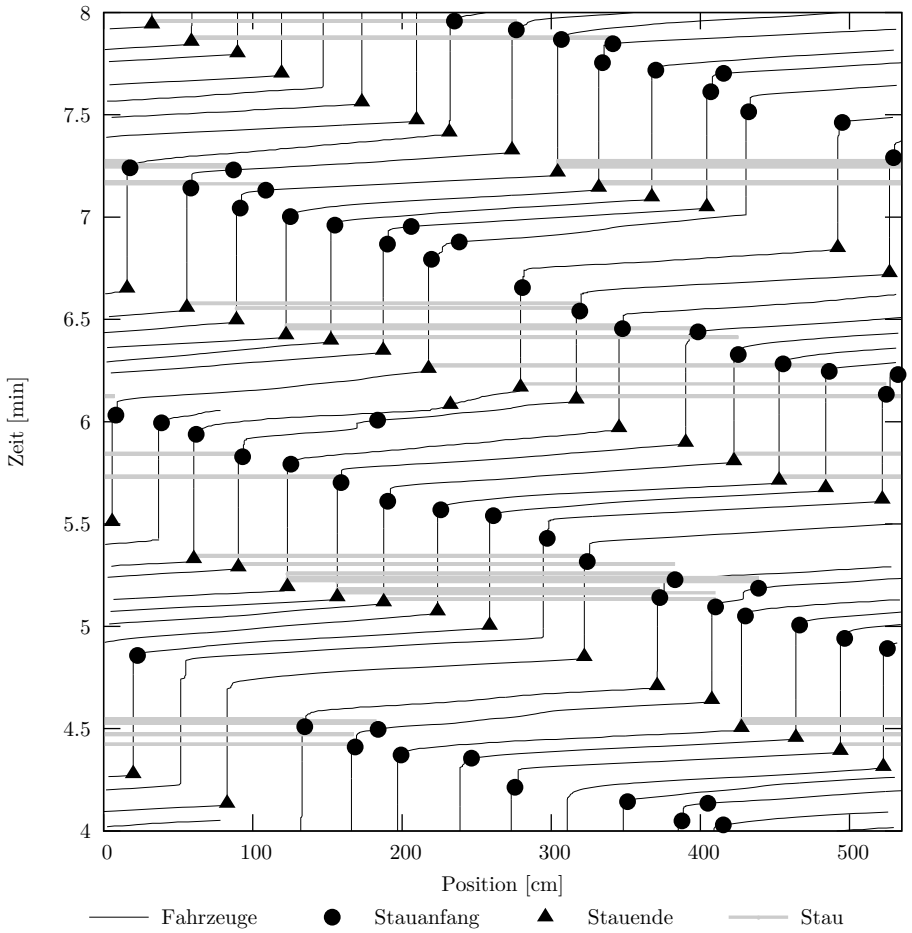


Abbildung 7.11: Erkennung eines Verkehrsstaus mittels hierarchischer Aggregation von HDCs.

erkannt werden. Die hier als Konstanten definierten Parameter des Stauerkennungs-Algorithmus sollten daher in einer realen VANET-Anwendung ständig an das momentane Fahrverhalten des jeweiligen Verkehrsteilnehmers angepasst werden.

Kapitel 8

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde die Verbesserung des Verkehrsflusses als Anwendungsgebiet adressiert. Dazu wurden bestehende zentralisierte Verkehrsinformationssysteme vorgestellt und der Vorteil einer dezentralen Lösung herausgearbeitet.

Dezentrale Anwendungen im Straßenverkehr, auch VANET-Anwendungen genannt, sind Gegenstand vieler Forschungsprojekte, bei denen Fahrzeuge über ein Ad-hoc-Netz (VANET) miteinander kommunizieren. Neu in dieser Arbeit ist die Anwendung von Organic-Computing-Grundsätzen zur Entwicklung solcher Anwendungen. Drei Organic-Computing-Konzepte bilden dabei die Basis für das *AutoNomos*-System, das ein Programmiermodell zur Entwicklung dezentraler Anwendungen darstellt.

Mit der *Hovering Data Cloud* (HDC) wurde eine Datenstruktur entwickelt, die eine Abstraktion der Daten von den einzelnen Fahrzeugen erlaubt. Eine HDC wird von allen Fahrzeugen innerhalb eines Kommunikationsradius redundant gehalten und strebt durch eine fortlaufende Integration neuer (Mess-)Daten einen Konsens an. Die neuen Daten können auch die Position der HDC verändern, so dass diese immer in der Nähe des beobachteten Phänomens bleiben kann. Durch die Bewegung der Fahrzeuge und auch durch die Bewegung der HDC selbst, migriert die HDC selbständig zu den Fahrzeugen, die sich in ihrem Bereich aufhalten.

Nachdem lokale Phänomene somit durch HDCs erfasst und verfolgt werden, sorgt eine hierarchische Aggregation der Daten mehrerer – auch räumlich verteilter – HDCs zur Bildung von *Organic Information Complexes* (OICs). Dazu senden HDCs Datenelemente mit Informationen zu den beobachteten Phänomenen aus; sobald passende Datenelemente aufeinandertreffen, werden diese aggregiert und wiederum als HDC gespeichert. Die hierarchische Aggregation erlaubt einerseits den Aufbau von nachvollziehbaren Strukturen und andererseits ermöglicht sie die Umsetzung einer „Fischaugenperspektive“, bei der Verkehrsinformationen umso mehr zusammengefasst werden, je weiter sie sich von ihrem Ursprung entfernen.

Die *Adaptable Distributed Strategies* (ADS) gehen als drittes Konzept über die Datenerfassung hinaus und zielen auf eine Verbesserung des Verkehrsflusses. Sie werden zurzeit von den *AutoNomos*-Projektpartnern eingehend untersucht und spielten daher in der vorliegenden Arbeit nur eine untergeordnete Rolle. Grundgedanke ist, auf Basis

lokaler Informationen mit möglichst kleinen lokalen Eingriffen in das Fahrverhalten eine Verbesserung des globalen Verkehrsflusses zu erzielen.

Die Konzepte wurden in Form eines objektorientierten Modells als *AutoNomos*-System umgesetzt. Dieses erlaubt die Entwicklung dezentraler Anwendungen, welche die vorgenannten Konzepte nutzen. Die Bedeutung eines einzelnen Fahrzeugs wird dabei reduziert auf das Liefern von Messdaten und die Umsetzung von Fahrstrategien. Die auf HDCs und OICs basierende Anwendungslogik kann sich darauf konzentrieren, unter welchen Voraussetzungen welche Phänomene beobachtet werden sollen, wie durch hierarchische Aggregation höherwertige Informationen gewonnen werden und wo diese Informationen im Straßennetz benötigt werden, um als Entscheidungsgrundlage für verteilte Strategien zu dienen. Welches einzelne Fahrzeug in diesem Prozess an welcher Stelle involviert ist, ist für den Anwendungsentwickler irrelevant.

Da sich bei der Entwicklung des *AutoNomos*-Systems das Verteilen von Daten als wichtiger Dienst herausstellte, wurde mit dem *AutoCast*-Protokoll ein Protokoll entwickelt, das Daten sehr effizient in einem vorgegebenen Verbreitungsgebiet verteilen kann. Im Gegensatz zu vielen Ansätzen aus der Literatur arbeitet *AutoCast* in sehr unterschiedlichen Netztopologien und kombiniert dazu die Datenverteilung über die Funkschnittstelle mit der Datenverteilung über das Mitnehmen der Daten und ihr späteres erneutes Aussenden. Durch die detaillierte Betrachtung des Protokollaufbaus und einer ausführlichen Evaluation stellt *AutoCast* einen Hauptteil dieser Arbeit dar. Es zeigte sich, dass *AutoCast* Daten sehr zuverlässig an praktisch alle Fahrzeuge im gewünschten Verbreitungsgebiet verteilt. Dabei werden Daten in gut konnektierten Netzen innerhalb von 80 s über 100 km befördert. Bei einem Austausch des zur Evaluation genutzten relativ einfachen Funkausbreitungsmodells durch das realitätsnähere *Radio Irregularity Model* (RIM) bewies *AutoCast* seine Robustheit gegenüber einer weniger zuverlässigen Nachrichtenübertragung.

Zudem wurde die Skalierbarkeit einer Netzwerksimulation unter Verwendung des *AutoCast*-Protokolls untersucht und festgestellt, dass der etablierte Netzwerksimulator ns-2 nicht zur Simulation von Verkehrsszenarien mit mehreren Tausend Fahrzeugen geeignet ist. Als Alternative wurde der Netzwerksimulator Shawn getestet, der statt der vollständigen physikalischen Modelle lediglich deren Auswirkungen simuliert. Tatsächlich ließ sich eine Verbesserung der Laufzeit um den Faktor 10 erzielen, während der Hauptspeicherverbrauch um den Faktor 50 sank. Bei einer qualitativen Betrachtung der *AutoCast*-Leistungsparameter konnten keine nennenswerten Abweichungen zwischen beiden Simulatoren festgestellt werden, so dass größere Verkehrsszenarien zukünftig mit Shawn simuliert werden können.

Bevor VANET-Anwendungen in realen Fahrzeugen und Verkehrsszenarien getestet werden, müssen sie vorher in einer VANET-Simulationsumgebung ihre Funktionsweise unter Beweis gestellt haben. Eine solche VANET-Simulationsumgebung stellt einen weiteren Schwerpunkt dieser Arbeit dar. Es wurde gezeigt, wie ein Verkehrssimulator

und ein Netzwerksimulator über eine neuartige Schnittstelle namens *Traffic Control Interface* (TraCI) gekoppelt werden. Dabei werden einerseits die vom Verkehrssimulator erzeugten Bewegungsdaten im Netzwerksimulator zur Bewegung der Knoten im Ad-hoc-Netz genutzt und andererseits werden Anpassungen des Fahrverhaltens, die aus der VANET-Anwendung stammen, vom Verkehrssimulator während der Simulation berücksichtigt. Somit ist, soweit bekannt, erstmals eine systematische Evaluation der Auswirkungen von VANET-Anwendungen mit frei verfügbarer Software möglich.

Abschließend wurden die erarbeiteten Konzepte in drei beispielhafte Anwendungen umgesetzt.

Die erste Anwendung stellt eine *Ampelassistenz* dar. Durch Informationen über zukünftige Phasenwechsel sind ankommende Fahrzeuge in der Lage durch Abschalten der Treibstoffzufuhr zum Motor sanft abzubremsen, um vor der Ampel zum Stehen zu kommen. Verbleibt genügend Wartezeit, so kann durch Ausschalten des Motors weiterer Treibstoff eingespart werden. Die nötigen Informationen stellt die Ampel im Rahmen einer HDC zur Verfügung. Es wurde gezeigt, dass in dieser Anwendung ein erhebliches Einsparpotenzial steckt und zudem schon das erste ausgestattete Fahrzeug den vollen Nutzen erhält, vorausgesetzt auch die Ampeln sind mit entsprechenden VANET-Modulen ausgestattet.

Die zweite Anwendung demonstriert die Kommunikation zwischen zwei HDCs, die jeweils an einer Kreuzung am Anfang und am Ende eines Streckenabschnitts positioniert sind. Sie erfassen die zum Durchfahren des Streckenabschnitts benötigte Fahrtzeit und stellen diese Information ankommenden Fahrzeugen zur Verfügung. Es wurde evaluiert, wie diese zusätzlichen Informationen bei einem simulierten Unfall genutzt werden, um auf eine längere Umleitungsstrecke auszuweichen.

Die dritte Anwendung erkennt einen Stau als Organic Information Complex durch die hierarchische Aggregation lokal gewonnener und in HDCs gespeicherter Daten. Die Evaluation der *dezentralen Stauerkennung* wurde anhand eines Demonstrators durchgeführt, bei dem sich Lego-Mindstorms-Fahrzeuge auf einem vorgegebenen Rundkurs bewegen. Somit wurde nicht nur das Konzept der HDC und der hierarchischen Aggregation an diesem Beispiel verdeutlicht, sondern auch eine Plattform geschaffen, mit deren Hilfe die Forschungsergebnisse sowohl wissenschaftlichem als auch nichtwissenschaftlichem Publikum auf anschauliche Weise nähergebracht werden können.

Raum für zukünftige Arbeiten ergibt sich vor allem im Bereich der Untersuchung einzelner VANET-Anwendungen. Hier wurden bislang nur beispielhafte Szenarien umgesetzt, um die vorgestellten Organic-Computing-Konzepte zu verdeutlichen. Auf der Basis dieser Arbeit ist im Rahmen der 3. Projektphase von AutoNomos [40] die Vorhersage zukünftiger Verkehrslagen geplant. Unter Einbeziehung von Rückkopplungs- und spieltheoretischen Effekten wird eine Verkehrsbeeinflussung angestrebt, die eine zuvor vorhergesagte ungünstige Verkehrslage vermeidet oder zumindest abschwächt.

Hinsichtlich des *AutoCast*-Protokolls könnte ein Bloom-Filter [115] für eine kom-

paktere Übergabe der Hash-Werte sorgen. Auch die Integration einiger Infrastruktur-Knoten, die kabelgebunden größere Entfernungen überbrücken, wäre eine nähere Untersuchung wert (vgl. Abschnitt 5.5). Beispielsweise könnten auch die aufgrund der *Ampelassistenz*-Anwendung in Ampeln vorhandenen VANET-Module als *AutoCast*-Infrastrukturknoten eingesetzt werden.

Auch für den in Abschnitt 7.4.2 vorgestellten Demonstrator ist eine Erweiterung in Form einer äußeren Umgehungsstraße geplant. Diese Erweiterung ist im Rahmen einer Diplomarbeit teilweise schon umgesetzt (vgl. [162]). Analog zur Anwendung *adaptive Routenplanung* sollen die Lego-Mindstorms-Fahrzeuge jeweils die schnellere innere oder äußere Straße verwenden.

Anhang A

Bestimmung von $ratio_{newNeighbors}$ für verschiedene Netztopologien

Beim einfachen Fluten in einem Ad-hoc-Netz leitet jeder Knoten, der eine Nachricht zum ersten Mal empfängt, diese umgehend und genau einmal weiter. Im Zuge der Optimierung des einfachen Flutens zum selektiven Fluten (vgl. Abschnitt 5.2.1) stellt sich die Frage, wie viele Nachbarn eines sendenden Knotens eine Nachricht zum ersten Mal erhalten und sich somit am weiteren Verbreiten beteiligen. Abbildung 5.4 veranschaulicht die Fragestellung. Die Nachricht von Knoten G erhalten lediglich 15 seiner insgesamt 43 Nachbarn zum ersten Mal. In diesem Beispiel ergibt sich somit $ratio_{newNeigbors} \approx 35\,\%$.

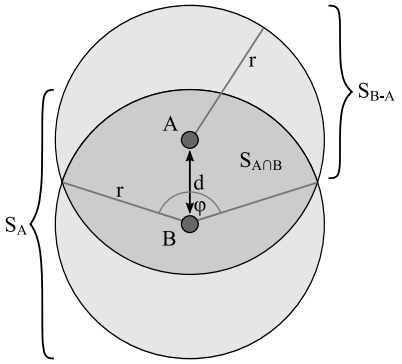
Für die analytische Betrachtung wird im Folgenden das Unit Disk Graph-Modell verwendet. Es besagt, dass eine gesendete Nachricht genau innerhalb einer Kreisfläche mit Radius r um den Sender empfangen wird. Abbildung A.1a stellt zwei Knoten dar die eine Nachricht weiterleiten. Knoten A überdeckt beim Senden einer Nachricht die Kreisfläche S_A mit Radius r. Ohne Beschränkung der Allgemeinheit sei $r = 1$. Der Knoten B gewinnt beim Weiterleiten der Nachricht die Fläche $S_{B-A} = S_B - S_{A \cap B}$ hinzu. Die Schnittfläche der beiden Kreise lässt sich berechnen mit $S_{A \cap B} = \varphi - \sin(\varphi)$, wobei $\varphi(d) = 2 \arccos(\frac{d}{2})$ ist. Somit ergibt sich

$$ratio_{newNeighbors}(d) = \frac{S_{B-A}(d)}{S_B} = \frac{S_B - S_{A \cap B}(d)}{S_B} = \frac{\pi - \varphi(d) + \sin(\varphi(d))}{\pi} \tag{A.1}$$

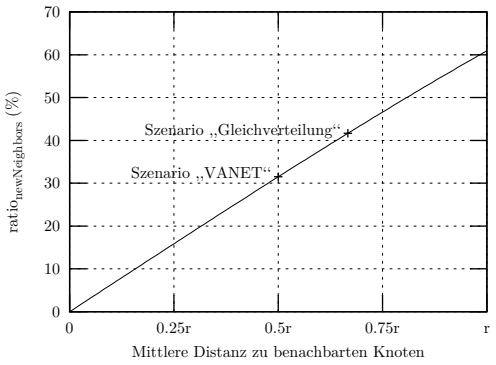
als Anteil der beim Weiterleiten neu informierten Nachbarn in Abhängigkeit vom Abstand zwischen den weiterleitenden Knoten (Hop-Distanz).

Das Problem reduziert sich somit auf die Bestimmung des mittleren Abstands $\bar{d}$ aller benachbarten Knoten. Dieser ist wiederum abhängig vom Szenario und der daraus folgenden Knotenverteilung. Im Allgemeinen zeigt Abbildung A.1b den mittleren Anteil neuer Nachbarn in Abhängigkeit von $\bar{d}$.

Im Folgenden werden zwei typische Knotenverteilungen betrachtet. Im ersten Fall sind die Knoten gleichmäßig in einer Ebene angeordnet (Szenario „Gleichverteilung“). Im zweiten Fall wird ein typisches Verkehrsszenario (Szenario „VANET“) angenommen, in dem die Knoten in gleichmäßigen Abständen auf einer Straße angeordnet sind.



(a) Analyse der hinzugewonnenen Fläche S_{B-A} wenn B eine von A gesendete Nachricht weiterleitet.



(b) Anteil neuer Knoten in Abhängigkeit vom mittleren Abstand der benachbarten Knoten.

Abbildung A.1: Bestimmung des Anteils unterschiedlicher Nachbarn in Abhängigkeit vom Abstand zweier Knoten.

A.1 Szenario „Gleichverteilung“

In diesem Szenario sind die Knoten – zumindest lokal – statistisch gleichverteilt auf einer Ebene angeordnet. Innerhalb der kreisförmigen Kommunikationsfläche eines Knotens A mit dem Radius r kann die Position benachbarter Knoten in Polarkoordinaten (ρ, ϕ) mit $\rho \in [0, r]$ und $\phi \in [0, 2\pi[$ relativ zu A beschrieben werden. Um eine Gleichverteilung der Knoten innerhalb der Kreisfläche zu erreichen, sei ϕ gleichverteilt im Bereich $[0..2\pi[$; für $\rho = r \cdot \sqrt{z}$ sei z gleichverteilt im Bereich $[0..1]$. Da der Winkel ϕ keinen Einfluss auf den Abstand eines Knotens zum Mittelpunkt des Kreises hat, ergibt sich für den mittleren Abstand

$$\bar{d}_{uniform} = \int_0^1 r \cdot \sqrt{z} dz = \frac{2}{3} r. \tag{A.2}$$

Das Einsetzen von $\bar{d}_{uniform}$ in Gleichung A.1 ergibt $ratio_{newNeighbors}(\bar{d}_{uniform}) = 41.6\,\%$. Dieses Ergebnis entspricht dem mittleren Anteil neuer Nachbarn von 41 %, wie er auch in [177] angegeben wird.

A.2 Szenario „VANET“

In einem VANET sind die miteinander kommunizierenden Knoten in der Regel auf einer Straße angeordnet. Angenommen die Straße sei eine Gerade ohne Kreuzungen. Dann ist der Abstand der Fahrspuren bzw. die Breite b einer Straße sehr viel kleiner als der Kommunikationsradius r ($b \ll r$). Für diese Betrachtung können somit die unterschied-

lichen Fahrspuren vernachlässigt werden; die Knotenanordnung reduziert sich auf eine Dimension. Die Nachbarn eines Knoten A seien gleichverteilt im Bereich $-r..r$ relativ zu A. Als mittlerer Abstand ergibt sich

$$\bar{d}_{VANET} = \int_0^1 r \cdot x dx = \frac{1}{2} r. \tag{A.3}$$

Mit Hilfe von Gleichung A.1 errechnet sich der mittlere Anteil neuer Knoten im VANET-Szenario zu $ratio_{newNeighbors}(\bar{d}_{VANET}) = 31.5\,\%$.

Anhang B

Optimierung des probabilistischen Flutens

Beim probabilistischen Fluten, wie es in Abschnitt 5.2.1 hergeleitet wurde, entscheidet ein Knoten jedes Mal beim Empfang eines neuen Datenelements anhand von Gleichung 5.1, ob er dieses Datenelement weiterleiten wird. Die Entscheidung basiert auf der Anzahl der benachbarten Knoten und der Konstante $\#\,forwarders$. In diesem Abschnitt wird $\#\,forwarders$ simulativ bestimmt als Kompromiss zwischen übertragener Datenmenge und dem Anteil der Knoten, die allein durch probabilistisches Fluten erreicht werden.

Für diese Simulationsreihe wird der in Abschnitt 5.4.6 eingeführte Netzwerksimulator Shawn verwendet. Das Übertragungsmodell ist zuverlässig, d. h. es gibt weder eine Begrenzung der verfügbaren Datenrate, noch gehen Pakete verloren. Gesendete Nachrichten werden von allen Knoten empfangen, die sich innerhalb des Kommunikationsradius $r = 250\,\mathrm{m}$ befinden. Zu Beginn der Simulation werden so viele Knoten auf einer Fläche von $10\,\mathrm{km} \times 10\,\mathrm{km}$ verteilt, dass sich die gewünschten mittleren Nachbarschaftsgrößen ergeben, die in Abbildung B.1 dargestellt sind. Für jede mittlere Nachbarschaftsgröße wird die Weiterleitungswahrscheinlichkeit von 0 % bis 100 % variiert und jeweils das probabilistische Fluten von 200 Nachrichten simuliert. Für jede geflutete Nachricht werden die Knoten erneut zufällig gleichverteilt auf der Fläche angeordnet und bewegen sich nicht während eine Nachricht per probabilistischem Fluten verteilt wird. Der initiale Sender wird zufällig bestimmt.

Abbildung B.1 zeigt den Anteil im Mittel erreichter Knoten für verschiedene Weiterleitungswahrscheinlichkeiten und verschiedene Knotendichten. Bei einer geringen Knotendichte, bei der jeder Knoten im Mittel weniger als sechs Nachbarn hat, ist ein Verteilen einer Nachricht an alle Knoten im Netz selbst bei einer Weiterleitungswahrscheinlichkeit von 100 % nicht möglich. Bei ansteigender Knotendichte stellt sich ein Phasenwechsel (vgl. Perkolationstheorie [56]) ein, so dass bei einer zu geringen Weiterleitungswahrscheinlichkeit nur ein sehr geringer Anteil der Knoten erreicht wird, während bei einem Anstieg der Weiterleitungswahrscheinlichkeit sehr schnell nahezu alle Knoten erreicht werden. Mit ansteigender Knotendichte stellt sich der Phasenwechsel einerseits bei immer kleiner werdender Weiterleitungswahrscheinlickeit ein und andererseits steigt die Kurve immer steiler an. Bei einer mittleren Nachbarschaftsgröße von $\#\,neighbors = 100$

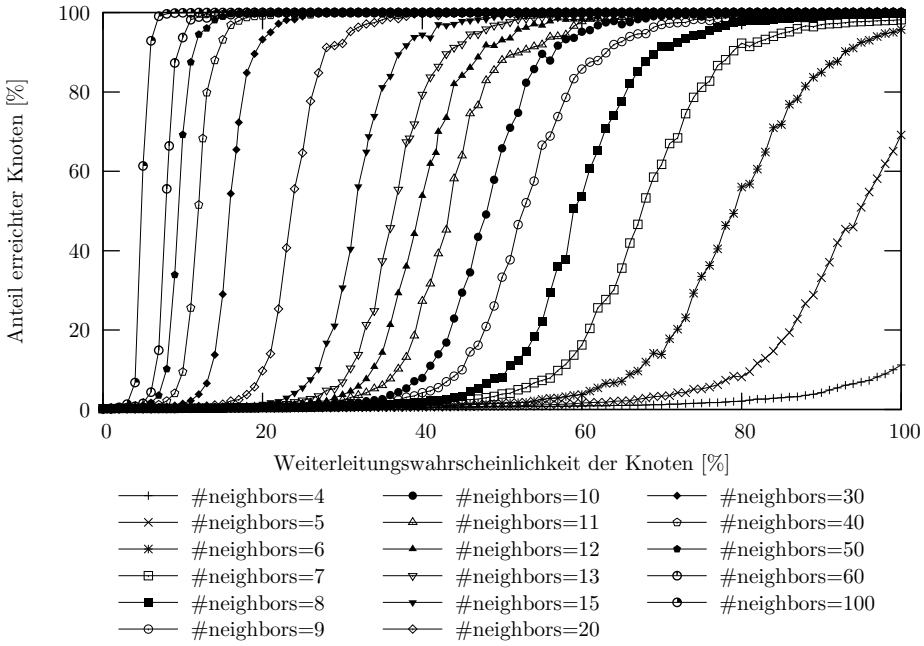


Abbildung B.1: Auswirkung der Weiterleitungswahrscheinlichkeit auf den Anteil erreichter Knoten.

werden beispielsweise bei einer Weiterleitungswahrscheinlichkeit von 3 % nur 1 % der Knoten erreicht, während bei einer Weiterleitungswahrscheinlichkeit von 7 % bereits 99 % der Knoten erreicht werden.

Die nötige Weiterleitungswahrscheinlichkeit ist somit offensichtlich von der Knotendichte abhängig. Jetzt gilt es $\#\,forwarder$ aus Gleichung 5.1 so zu wählen, dass immer genau der Bereich des Phasenwechsels getroffen wird. In Abbildung B.2 sind die Entscheidungen dargestellt, die *AutoCast* bei unterschiedlicher Wahl von $\#\,forwarder$ treffen würde. Um den Erfolg der jeweiligen Entscheidung abzuschätzen sind diese auf die aus Abbildung B.1 bekannten Kurven projiziert. Bemerkenswert ist der nahezu waagerechte Verlauf der einzelnen „Entscheidungskurven“; die Adaption an der Knotendichte führt offenbar zum gewünschten konstanten Erfolg in unterschiedlichen Netztopologien. Das für *AutoCast* postulierte Vorgehen, die Weiterleitungswahrscheinlichkeit ausschließlich an die Anzahl der Nachbarn eines Knoten anzupassen, kann somit als bestätigt angesehen werden.

Es zeigt sich, dass für $\#\,forwarder \leq 1,75$ nur ein geringer Anteil der Knoten erreicht wird ($< 30\,\%$ für $\#\,forwarder = 1,75$). Andererseits erweist sich $\#\,forwarder > 2,5$ als unnötig, da bereits fast alle Knoten erreicht werden ($\approx 95\,\%$ für $\#\,forwarder = 2,5$). Da die Datenrate proportional zu $\#\,forwarder$ ansteigt, gilt es einen möglichst

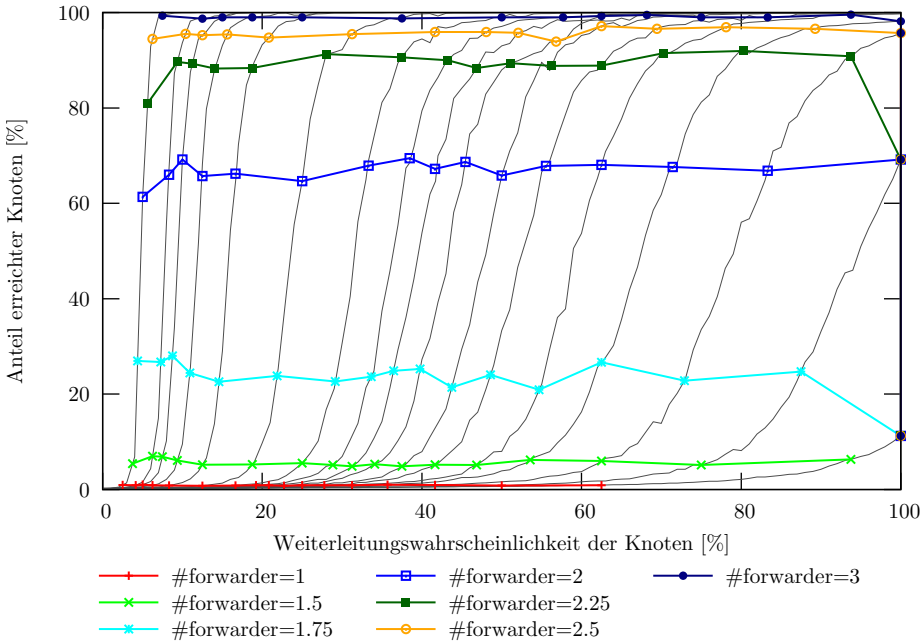


Abbildung B.2: Auswirkung von $\#\,forwarder$ auf den Anteil erreichter Knoten.

niedrigen Wert für $\#\,forwarder$ zu wählen. Die Wahl fällt auf $\#\,forwarder = 2$, da hier der Zugewinn an erreichten Knoten maximal ist und bereits knapp 70 % der Knoten allein durch probabilistisches Fluten erreicht werden. Die Wahl eines höheren Werts ist unter Berücksichtigung des in Abschnitt 5.2.3 beschriebenen wiederholten Sendens von Datenelementen nicht nötig.

Anhang C

Implementierungsdetails zu AutoCast

Die Konzepte des Datenverteilungsprotokolls *AutoCast* wurden in Kapitel 5 beschrieben. Hier folgt die ausführliche Beschreibung der *AutoCast*-Implementierung.

C.1 Nachrichtenformat

AutoCast baut auf der Sicherungsschicht auf und geht somit davon aus, dass die Datenintegrität während der Übertragung gesichert ist. Eine Broadcast-Nachricht wird also von benachbarten Knoten entweder korrekt oder gar nicht empfangen.

Abbildung C.1 zeigt das von *AutoCast* verwendete Nachrichtenformat, welches für jeglichen Nachrichtenaustausch verwendet wird. Eine *AutoCast*-Nachricht besteht immer aus einem Nachrichtenkopf (Header) gefolgt von drei Listen: Hash-Werte uninteressanter Datenelemente (Set of stale Data Hashes), Hash-Werte aller bekannten Datenelemente (Set of known Data Hashes) sowie vollständige Datenelemente (Data Units). Die Größe der jeweiligen Liste ist im Nachrichtenkopf in den Feldern $\#\,stale\,Hashes$, $\#\,Hashes$ bzw. $\#\,Data\,Units$ vermerkt.

Zur Verwaltung der Nachbarschaften enthält der Nachrichtenkopf zudem die Felder $Sender\,Id$, $next\,Beacon\,Msg\,in\,ms$ und $\#\,Neighbors$, die neben der Identität des sendenden Knotens angeben, wann dieser Knoten spätestens die nächste (Beacon-)Nachricht senden wird, und wie viele Nachbarn er hat.

Jede *AutoCast*-Nachricht enthält die Hash-Werte aller uninteressanten sowie lokal bekannten Datenelemente. Wenn es sich ausschließlich um eine Beacon-Nachricht handelt, sind keine vollständigen Datenelemente angehängt, ansonsten folgen den Hash-Werten ein oder mehrere Datenelemente, die geflutet werden, oder als *Antwort* auf eine (implizite) *Anfrage* verschickt werden. Für jedes Datenelement wird die Länge der Nutzdaten sowie die eigentlichen Nutzdaten übermittelt. Jedes Datenelement bzw. sein Hash-Wert findet sich in genau einer der drei Listen.

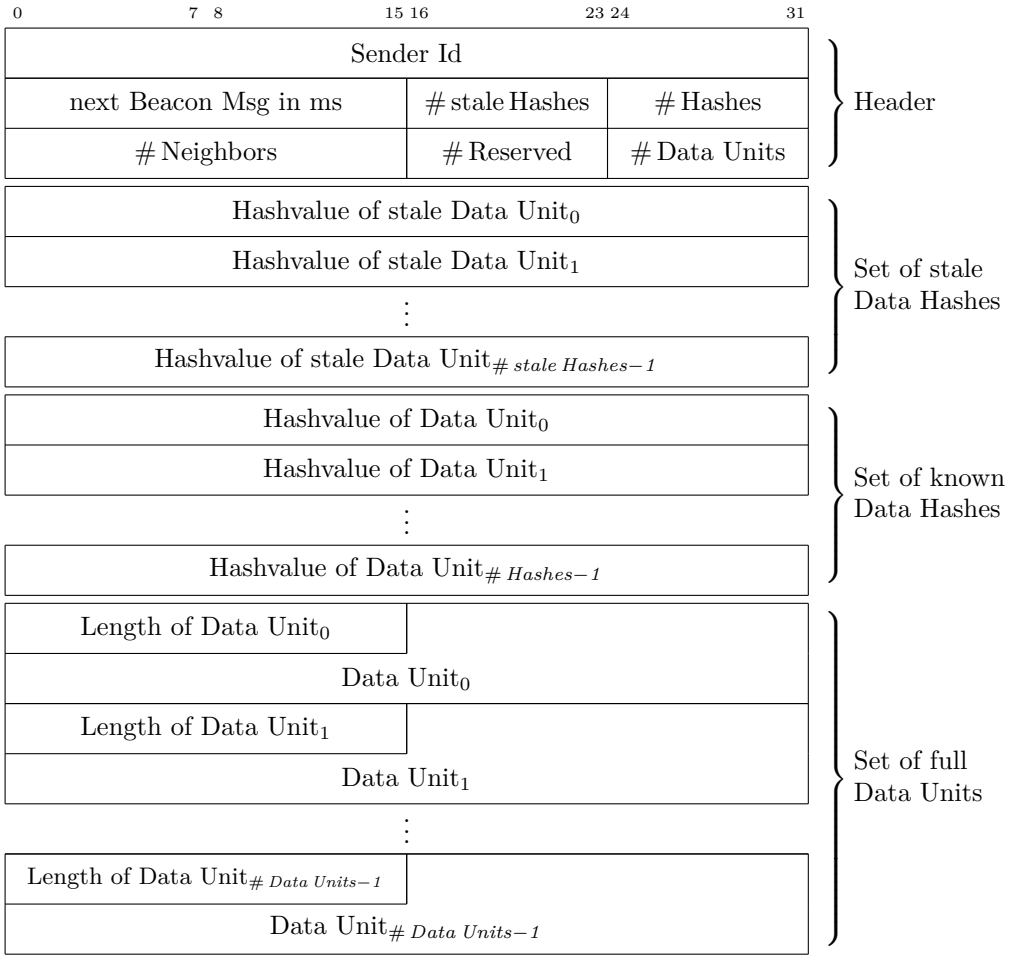


Abbildung C.1: *AutoCast*-Nachrichtenformat.

C.2 Protokollablauf

Mit dem vorgestellten Nachrichtenformat und einigen lokalen Variablen lassen sich alle Protokollbausteine wie in Abschnitt 5.2 vorgestellt umsetzen. Die zur Umsetzung des Protokollablaufs verwendeten Algorithmen werden im Folgenden vorgestellt.

```
1: setDataUnits                                              ▷ Gültige Datenelemente
2: setStaleDataUnits                                         ▷ „Uninteressante“ Datenelemente
3: setDUsForSend                  ▷ Datenelemente, die in der nächsten Nachricht versendet werden
4: setReqHashes                   ▷ Unbekannte Hash-Werte zur Anfrage in der nächsten Nachricht
5: setNeighbors                                              ▷ Nachbarschaftsliste
6: answerTimer, floodTimer, requestTimer, beaconTimer        ▷ Aktions-Zeitgeber
```

Algorithmus C.1: Lokale Variablen.

Jeder Knoten besitzt einige für *AutoCast* relevante lokale Variablen, die Algorithmus C.1 zeigt. In zwei Listen werden Datenelemente längerfristig gespeichert: Solche, die zeitlich gültig sind und in deren Verbreitungsgebiet sich der Knoten bewegt, befinden sich in *setDataUnits*; Datenelemente, deren Lebenszeit abläuft oder deren Verbreitungsgebiet ein Knoten verlässt, werden für eine gewisse Dauer in *setStaleDataUnits* gespeichert.

Datenelemente werden kurzfristig in die Liste *setDUsForSend* aufgenommen, wenn sie zum Senden in der nächsten Nachricht vorgemerkt sind; unbekannte Hash-Werte werden bis zum Senden der nächsten Nachricht in der Liste *setReqHashes* abgelegt.

Darüber hinaus speichert *setNeighbors* die Nachbarn eines Knotens, und vier Zeitgeber sind den Aktionen *Antworten*, *Fluten*, *Anfragen* bzw. Senden einer *Beacon*-Nachricht zugeordnet. Beim Ablauf eines Zeitgebers wird jeweils die Funktion onTimerExpired() aufgerufen.

Algorithmus C.2 zeigt die Funktion onTimerExpired(). Sie lässt sich in die folgenden vier Blöcke unterteilen:

- In den Zeilen 1–15 wird jedes Datenelement auf die Zugehörigkeit zu *setDataUnits* bzw. *setStaleDataUnits* geprüft. Datenelemente aus *setDataUnits*, deren Verbreitungsgebiet ein Knoten verlassen hat oder deren Lebenszeit abgelaufen ist, werden in die Liste *setStaleDataUnits* verschoben. Umgekehrt werden Datenelemente aus *staleDataUnits* wieder in die Liste *setDataUnits* aufgenommen, wenn sich der Knoten innerhalb der Lebenszeit eines Datenelements in sein Verbreitungsgebiet bewegt. Entfernt sich ein Knoten um mehr als den doppelten Kommunikationsradius R von einem Datenelement oder ist dessen Lebenszeit seit mehr als 10 s abgelaufen, dann wird es auch aus *setStaleDataUnits* gelöscht.

- Im zweiten Schritt (Zeilen 16–20) werden alle Zeitgeber abgebrochen und der Zeit-

```
 1: for each DU ∈ setDataUnits do                          ▷ Datenelemente uninteressant?
 2:     if (not in DU's area) ∨ (DU.EXPIRES( ) < now) then
 3:         setDataUnits.REMOVE(DU)
 4:         setStaleDataUnits.ADD(DU)
 5:     end if
 6: end for
 7: for each staleDU ∈ setStaleDataUnits do                ▷ Uninteressante Datenelemente ...
 8:     if (in staleDU's area) ∧ (staleDU.EXPIRES( ) > now) then
 9:         setStaleDataUnits.REMOVE(staleDU)              ▷ ... wieder interessant
10:         setDataUnits.ADD(staleDU)
11:         RECEIVE(staleDU)
12:     else if (staleDU.DISTANCE( ) > 2r) ∨ (staleDU.EXPIRES( )+10 < now) then   ▷
    ... endgültig löschen
13:         setStaleDataUnits.REMOVE(staleDU)
14:     end if
15: end for

16: floodTimer.CANCEL( )                                   ▷ nächsten Aufruf festlegen
17: answerTimer.CANCEL( )
18: requestTimer.CANCEL( )
19: t_beacon ← MAX(0.5, MIN(0.1 r/v, 20.0))                ▷ vgl. Gleichung 5.2
20: beaconTimer.EXPIREIN(t_beacon)

21: msg.nextBeaconTime ← t_beacon                          ▷ Nachricht erstellen ...
22: msg.#staleHashes ← setStaleDataUnits.SIZE( )
23: msg.#Hashes ← setDataUnits.SIZE( )
24: msg.#Neighbors ← local neighborhood size;
25: msg.setStaleHashes ← HASH(setStaleDataUnits)
26: msg.setDataUnits ← as many data units from setDUsForSend as fit into the packet
27: msg.#DataUnits ← msg.setDataUnits.SIZE( )
28: msg.setDataHashes ← HASH(setDataUnits \ msg.setDataUnits)
29: SEND(msg)                                              ▷ ... und senden

30: setDUsForSend ← ∅
31: setReqHashes ← ∅                                       ▷ Die gesendete Nachricht hat ggf. als Anfrage gedient
```

Algorithmus C.2: onTimerExpired( )

punkt zum Senden der nächsten Beacon-Nachricht berechnet (vgl. Abschnitt 5.2.2). Der Zeitgeber *beaconTimer* sorgt in diesem Fall für den Aufruf dieser Funktion. Die übrigen Zeitgeber werden bei Bedarf für die jeweilige Aktion Fluten, Antworten bzw. Anfragen verwendet.

- Daraufhin wird in den Zeilen 21–29 eine Nachricht erstellt und versendet. Das Nachrichtenformat entspricht Abbildung C.1. Sollten mehr Datenelemente zum Senden vorgemerkt sein als in die Nachricht passen, werden zufällig so viele ausgewählt wie die Nachricht aufnehmen kann.

- Zuletzt werden noch zwei temporäre Listen geleert. Da potenziell mehrere Knoten auf fehlende Datenelemente reagieren, wird $setDUsForSend$ nach jeder gesendeten Nachricht geleert. Falls einem Knoten weiterhin Datenelemente fehlen, so wird dies spätestens festgestellt, wenn er die nächste Beacon-Nachricht sendet. Da jede gesendete Nachricht als implizite *Anfrage* fungiert, kann $setReqHashes$ ebenfalls nach dem Senden geleert werden.

Wenn *AutoCast* ein neues Datenelement von der darüber liegenden Schicht erhält, wird dieses neue Datenelement den Listen $setDataUnits$ und $setDUsForSend$ hinzugefügt und direkt die Funktion onTimerExpired() aufgerufen.

Für jede Nachricht, die ein Knoten empfängt, wird die in Algorithmus C.3 gezeigte Funktion onReceive aufgerufen.

Zunächst werden alle geplanten Aktionen um ihre spezifische Verzögerungszeit verschoben, um eine Überlastung des Funkkanals zu vermeiden (Zeile 1–9). Lediglich der für das Versenden der Beacon-Nachrichten zuständige $beaconTimer$ wird nicht verschoben, da beim Ausbleiben der Beacon-Nachricht zum geplanten Zeitpunkt alle Nachbarn den entsprechenden Knoten aus ihrer Nachbarschaftsliste entfernen würden.

In Zeile 10–16 werden alle in der empfangenen Nachricht msg enthaltenen Datenelemente, die dem Knoten noch nicht bekannt sind, in neue und uninteressante Datenelemente unterteilt und entsprechend in die Listen $setNewDataUnits$ bzw. $setStaleDataUnits$ aufgenommen. Alle neuen Datenelemente aus $setNewDataUnits$ werden anschließend (Zeile 17) in $setDataUnits$ gespeichert.

In Zeile 18 wird der Sender der Nachricht in die Nachbarschaftsliste aufgenommen. Der Eintrag wird aus der Liste entfernt, wenn nach $msg.nextBeaconTime$ Sekunden keine weitere Nachricht vom entsprechenden Knoten empfangen wird. Die Überprüfung auf abgelaufene Einträge findet implizit beim Aufruf der Methode $setNeighbors$.size() statt. Aufgrund der so ermittelten Nachbarschaftsgröße entscheidet der Knoten in Zeile 19, ob er – falls nötig – aktiv die Aktionen Fluten, Antworten und Anfragen ausführen wird.

Der folgende Block (Zeilen 20–35) initiiert gegebenenfalls die Aktionen *Fluten*, *Antworten* und *Anfragen*. Geflutet wird mit einer Verzögerung von 2δ, falls die empfangene Nachricht neue Datenelemente enthält. Alle Datenelemente aus $setDataUnits$, deren

```
 1: if answerTimer.isPending( ) then                          ▷ Verzögere geplante Aktionen
 2:     answerTimer.expireIn(δ)
 3: end if
 4: if floodTimer.isPending( ) then
 5:     floodTimer.expireIn(2δ)
 6: end if
 7: if requestTimer.isPending( ) then
 8:     requestTimer.expireIn(3δ)
 9: end if

10: for each DU ∈ msg.setDataUnits \ [setDataUnits ∪ setStaleDataUnits] do
11:     if (in DU's area) ∧ (DU.expires( ) > now) then   ▷ Empfangene Datenelemente sortieren ...
12:         setNewDataUnits.add(DU)                                  ▷ ... in neue ...
13:     else
14:         setStaleDataUnits.add(DU)                ▷ ... und uninteressante Datenelemente
15:     end if
16: end for
17: setDataUnits.add(setNewDataUnits)                ▷ Neue Datenelemente lokal speichern

18: Add msg.sender to setNeighbors for msg.nextBeaconTime seconds      ▷ vgl. Abschnitt 5.2.2
19: active ← (rnd(0,1) < #forwarder / (setNeighbors.size()·0.4))       ▷ vgl. Abschnitt 5.2.1

20: if active then
21:     if setNewDataUnits ≠ ∅ then                                    ▷ Fluten
22:         setDUsForSend.add(setNewDataUnits)
23:         floodTimer.expireIn(2δ)
24:     end if
25:     setHashesForAnswer ← hash(setDataUnits) ↩                      ▷ Antworten
          ↪ \ [hash(msg.setDataUnits) ∪ msg.setHashes ∪ msg.setStaleHashes]
26:     if setHashesForAnswer ≠ ∅ then
27:         setDUsForSend.add(unhash(setHashesForAnswer))
28:         answerTimer.expireIn(1δ)
29:     end if
30:     setHashesForRequest ← msg.setHashes ↩                          ▷ Anfragen
          ↪ \ [hash(setDataUnits) ∪ hash(setStaleDataUnits)]
31:     if setHashesForRequest ≠ ∅ then
32:         setReqHashes.add(setHashesForRequest)
33:         requestTimer.expireIn(3δ)
34:     end if
35: end if

36: setReqHashes ← setReqHashes \ Hash(msg.setDataUnits)       ▷ Anfrage abbrechen, ...
37: if setReqHashes = ∅ then                  ▷ ... wenn keine anzufragenden Hash-Werte übrig sind
38:     requestTimer.cancel( )
39: end if
```

Algorithmus C.3: onReceive(msg)

Hash-Wert nicht in der empfangenen Nachricht enthalten ist, werden zum Antworten mit einer Verzögerung von δ vorgesehen. Enthält die Nachricht Hash-Werte, die lokal nicht bekannt sind, so werden diese in die Liste $setReqHashes$ aufgenommen und das Senden einer Anfragenachricht nach 3δ geplant.

Zuletzt werden noch in den Zeilen 36–39 alle in der empfangenen Nachricht enthaltenen Datenelemente aus $setReqHashes$ entfernt, da wegen dieser Datenelemente keine Anfrage mehr gesendet werden muss. Falls infolgedessen keine anzufragenden Datenelemente mehr vorhanden sind, so wird der zugehörige Zeitgeber gestoppt.

Anhang D

Implementierungsdetails des Traffic Control Interfaces

In diesem Abschnitt werden die Implementierungsdetails des in Kapitel 6 vorgestellten *Traffic Control Interfaces* (TraCI) vorgestellt. Dabei wird zunächst auf das grundlegende Format des von TraCI verwendeten Nachrichten-Containers eingegangen und die von TraCI verwendeten Datentypen werden spezifiziert. Dem folgt eine Beschreibung der einzelnen TraCI-Nachrichten inklusive ihrer Syntax. Abschließend wird auf die Implementierung in den Simulatoren SUMO, ns-2 und Shawn eingegangen und das Vorgehen beim Starten einer Simulation erläutert.

D.1 Format des Nachrichten-Containers

Wie in Abschnitt 6.3 beschrieben, baut TraCI eine TCP-Verbindung zwischen einem Verkehrs- und einem Netzwerksimulator auf. Ein Datenaustausch geht immer vom Netzwerksimulator aus, der eine *Anfrage* an den Verkehrssimulator sendet. Dieser wiederum antwortet nach Abarbeitung der *Anfrage* mit einer *Antwort*. Nachdem der Netzwerksimulator eine *Anfrage* gesendet hat, wartet er bis die *Antwort* des Verkehrssimulators vollständig empfangen wurde. Der Verkehrssimulator hingegen wird nur dann aktiv, wenn er eine *Anfrage* empfangen hat. Nach dem Absenden der *Antwort* wartet er bis zur nächsten *Anfrage* des Netzwerksimulators.

Sowohl *Anfragen* als auch *Antworten* werden in *Nachrichten-Containern* gekapselt, wie in Abbildung D.1 gezeigt. Diese sind in einem eigens entwickelten Binärformat codiert, da die Verarbeitung von beispielsweise XML-Daten aufgrund der enthaltenen Strukturdaten deutlich längere Zeit in Anspruch nimmt als die Verarbeitung des für TraCI entwickelten Binärformats.

Der *Header* eines Nachrichten-Containers umfasst lediglich ein Feld, das die Länge des vollständigen Containers in Bytes angibt. Für das Feld Nachrichtenlänge sind 32 bit vorgesehen; ein Nachrichten-Container kann somit eine maximale Länge von 4 GB haben. TCP kümmert sich dabei um die Fragmentierung. In der Realität sind die einzelnen Nachrichten-Container allerdings deutlich kleiner.

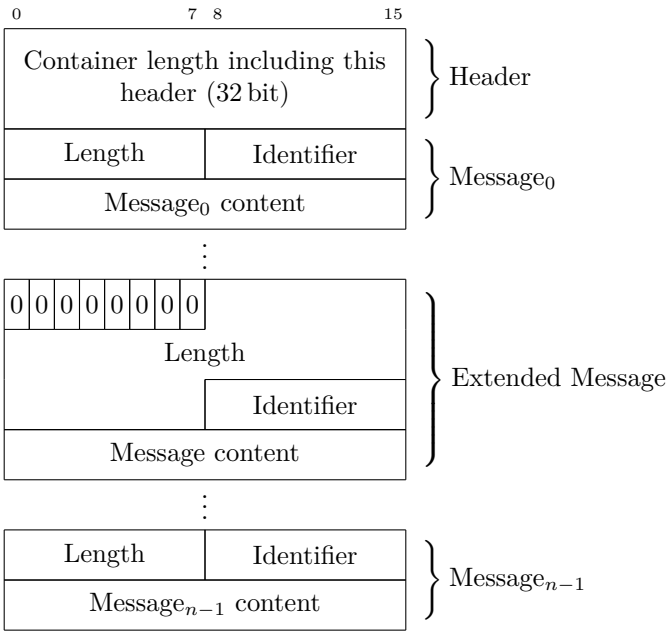


Abbildung D.1: Format des von TraCI verwendeten Nachrichten-Containers der die in *Anfragen* und *Antworten* enthaltenen TraCI-Nachrichten kapselt.

Dem *Header* folgen ein oder mehrere *TraCI-Nachrichten*, wie sie im folgenden Abschnitt D.3 beschrieben werden. Aus der Containerlänge und den Längen der einzelnen TraCI-Nachrichten ergibt sich, welches die letzte TraCI-Nachricht ist.

Jede TraCI-Nachricht startet mit den `ubyte`-Feldern *Length* und *Identifier*, die jeweils ein Byte belegen (Datentypen vgl. Tabelle D.1). Die Länge einer einzelnen TraCI-Nachricht ist daher inklusive der Felder *Length* und *Identifier* auf 255 Byte beschränkt. Um auch längere TraCI-Nachrichten zuzulassen, beispielsweise für die Abfrage der Fahrtroute, wurde eine *erweiterte TraCI-Nachricht* (*Extended Message*, vgl. Abbildung D.1) eingeführt, bei der die Länge als `integer` angegeben wird. Eine *erweiterte TraCI-Nachricht* wird eingeleitet, indem das `ubyte`-Feld für die Länge auf `0x00` gesetzt wird.

Bei allen versendeten Nachrichten richtet sich die Byte-Reihenfolge und die Anordnung der Bits innerhalb eines Bytes nach der in RFC 791 [142] beschriebenen „Data Transmission Order“.

D.2 Datentypen

Wie oben bereits erwähnt, setzt TraCI im Gegensatz zu XML nicht auf die Verwendung struktureller Informationen innerhalb der Nachrichten-Container. Stattdessen setzen sich

Tabelle D.1: Elementare Datentypen in TraCI-Nachrichten

ID	Datentyp	Länge	Beschreibung
0x07	ubyte	8 bit	Ganzzahlen (0 to 255)
0x08	byte	8 bit	Ganzzahlen (-128 to 127)
0x09	integer	32 bit	Ganzzahlen (-2^{31} to 2^{31}-1)
0x0A	float	32 bit	Gleitkommazahlen gemäß IEEE 754 [73]
0x0B	double	64 bit	Gleitkommazahlen gemäß IEEE 754 [73]
0x0C	string	variable	Text im ASCII-Format gemäß RFC 20 [28]

die Nachrichten-Container und die enthaltenen Nachrichten aus einer Folge elementarer Datentypen zusammen. Jeder Datentyp verfügt über eine *ID* (`ubyte`), die im Protokoll verwendet wird, falls verschiedene Datentypen für die Parameter einer TraCI-Nachricht infrage kommen.

Die elementaren Datentypen sind in Tabelle D.1 angegeben und umfassen `ubyte`, `byte`, `integer`, `float`, `double` und `string`. Alle numerischen Datentypen sind vorzeichenbehaftet, lediglich für den Datentyp `byte` existiert ein entsprechender vorzeichenloser Datentyp (`ubyte`).

Zur Codierung von Entfernungsangaben werden `float`-Variablen unter Verwendung der Einheit Meter genutzt. Zeitangaben werden in Sekunden als `double`-Variablen angegeben, um eine höhere Auflösung zu erzielen. Geschwindigkeiten werden als `float`-Variablen übermittelt und in der Einheit m/s interpretiert.

Der Datentyp `string` setzt sich zusammen aus einem `integer`, der die Länge des nachfolgenden Textes in Byte angibt, sowie einer entsprechenden Anzahl `ubyte`-Werte, die den Text als ASCII-codierte Zeichen enthalten (ASCII-Codierung vgl. RFC 20 [28]). `Strings` sind nicht null-terminiert.

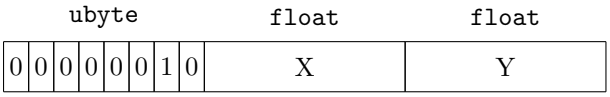
Neben den elementaren Datentypen werden einige zusammengesetzte Datentypen verwendet, die auf den vorgenannten elementaren Datentypen basieren. Jeder zusammengesetzte Datentyp beginnt mit seiner *ID*, die den Datentyp eindeutig kennzeichnet.

Zur Repräsentation von Fahrzeugpositionen existieren zwei verschiedene Datentypen, die Positionen auf unterschiedliche Art wiedergeben. *2D-Positionen* beziehen sich auf ein kartesisches Koordinatensystem und stellen die übliche Positionsdarstellung in Netzwerksimulatoren dar. Demgegenüber beziehen sich *Straßennetz-Positionen* auf einen Streckenabschnitt und die Position eines Fahrzeugs relativ zum Anfang des Streckenabschnitts. In VANET-Anwendungen kann diese Positionsdarstellung nützlich sein, um beispielsweise Navigationsentscheidungen zu treffen, oder das Passieren von Kreuzungen zu erkennen.

Durch die Verwendung einer `ubyte`-ID zur Identifizierung der jeweiligen Darstellungsart von Positionen, ist eine einfache Erweiterbarkeit gegeben. So könnten zusätzlich zu den beiden zurzeit implementierten Varianten auch Positionsangaben gemäß dem ASCII-

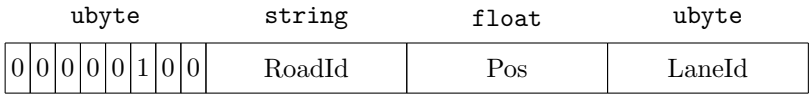
basierten NMEA-0183-Protokoll [125] eines GPS-Empfängers angegeben oder geografische Koordinaten mit Längen- und Breitengrad verwendet werden.

2D Position (ID: `0x02`)



Eine *2D Position* beginnt mit der `ubyte`-Kennung `0x02`. Dem folgen zwei `float`-Variablen, welche die X-, und Y- Position im kartesischen Koordinatensystem festlegen. Die Lage und Orientierung des Koordinatensystems ergibt sich aus der Definition des Straßennetzes im Verkehrssimulator.

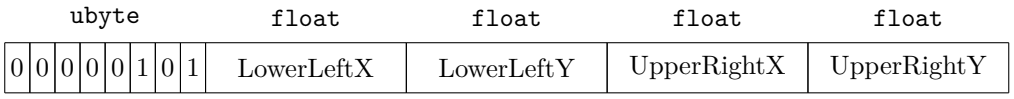
Roadmap Position (ID: `0x04`)



Die *Roadmap Position* liefert Positionen, die sich immer auf eine exakte Stelle im Straßennetz beziehen.

Die *RoadId* kennzeichnet einen Streckenabschnitt im Straßennetz. Dieser bezieht sich immer nur auf eine Fahrtrichtung, ist also gerichtet. Das Feld *pos* gibt die Position auf dem Streckenabschnitt im Bereich $[0, Length_{RoadId})$ wieder. Die Fahrspuren sind von rechts nach links beginnend bei Null durchnummeriert; entsprechend ist eine Fahrspur im Feld *LaneId* angegeben.

Bounding Box (ID: `0x05`)



Eine *Bounding Box* wird genutzt zur Darstellung eines umgebenden Rechtecks für verschiedene Objekte innerhalb des Verkehrsszenarios, wie beispielsweise, Streckenabschnitte, Polygone oder das gesamte Streckennetz. Ein umgebendes Rechteck wird angegeben durch die kartesischen Koordinaten von zwei diagonal gegenüberliegenden Eckpunkten des Rechtecks.

Polyline (ID: 0x06)

ubyte								integer	float	float	float	float	
0	0	0	0	0	1	1	0	#Vertices	x_1	y_1	x_2	y_2	···

Zur Darstellung von Streckenabschnitten und Polygonen wird der Datentyp *Polyline* (Linienzug) verwendet. Der Linienzug wird von einer Sequenz von Knotenpunkten, bzw. deren kartesischen Koordinaten, angegeben. Die Anzahl der Knotenpunkte enthält das Feld *#Vertices*.

Soll der Linienzug ein Polygon angeben, dann muss der erste und letzte Knotenpunkt identisch sein.

D.3 TraCI-Nachrichten

In diesem Abschnitt werden die einzelnen TraCI-Nachrichten definiert. Die Struktur folgt der Darstellung in Abschnitt 6.3.1 bis 6.3.3.

Bei der Einführung des Nachrichtenformats in Abschnitt D.1 wurde bereits der *Identifier* erwähnt, der jede TraCI-Nachricht einleitet. Dieser Identifier ist im Folgenden jeweils in der Überschrift einer TraCI-Nachricht angegeben. Bei der Darstellung des Inhalts der TraCI-Nachrichten wird auf die in jedem Fall vorhandenen einleitenden Datenfelder *Length* und *Identifier* verzichtet.

D.3.1 Simulationsablauf

Simulation Step (Identifier: 0x01)

double	ubyte
TargetTime	PositionType

Diese TraCI-Nachricht wird vom Netzwerksimulator an den Verkehrssimulator gesendet, um neue Bewegungsdaten für den nächsten Simulationsschritt zu erhalten. Sie weist den Verkehrssimulator an, seine Simulation auszuführen, bis die übergebene *TargetTime* erreicht ist.

Danach sendet der Verkehrssimulator eine *Antwort* bestehend aus einer *Status*-Nachricht und je einer *Move Node*-Nachricht für jedes Fahrzeug, das sich zur *TargetTime* auf dem Straßennetz befindet. Beide TraCI-Nachrichten werden nachfolgend beschrieben. Über die Variable *PositionType* lässt sich der Datentyp der zurückgegebenen Fahrzeugpositionen festlegen (`2D Position` oder `Roadmap Position`).

Status (Identifier: variabel)

ubyte	string
Result	Description

Der Verkehrssimulator beantwortet jede in der *Anfrage* enthaltene TraCI-Nachricht zunächst mit einer *Status*-Nachricht, bevor weitere spezifische TraCI-Nachrichten als Antwort folgen. Die Reihenfolge der TraCI-Nachrichten innerhalb des als Antwort gesendeten Nachrichten-Containers richtet sich nach der Reihenfolge der TraCI-Nachrichten innerhalb der Anfrage.

Im Gegensatz zu allen übrigen TraCI-Nachrichten ist der *Identifier* der *Status*-Nachricht nicht konstant, sondern entspricht dem *Identifier* der TraCI-Nachricht, auf das sich die *Status*-Nachricht bezieht.

Die *Status*-Nachricht gibt eine Rückmeldung, ob die Verarbeitung der eingegangenen TraCI-Nachricht erfolgreich war oder nicht. Dazu wird das *Result*-Feld verwendet; im Falle einer erfolgreichen Verarbeitung enthält dieses Feld `0x00`, bei einem Verarbeitungsfehler wird `0xFF` zurückgegeben. Sollte die angefragte TraCI-Nachricht nicht implementiert sein, wird *Result* auf `0x01` gesetzt. In jedem Fall wird zusätzlich ein Text (*Description*) angehängt, der beispielsweise Hinweise zum aufgetretenen Fehler enthalten kann; im einfachsten Fall kann ein leerer `string` verwendet werden.

Tritt ein Fehler innerhalb der Verkehrssimulation auf, wird diese nach dem Senden der Antwort an den Netzwerksimulator abgebrochen. Bei einer nicht implementierten TraCI-Nachricht entscheidet hingegen die Netzwerksimulation, ob sie ohne die angefragte Funktionalität weiterarbeiten kann.

Move Node (Identifier: `0x80`)

integer	double	position
NodeId	TargetTime	Position

Der Verkehrssimulator sendet als *Antwort* auf eine *Simulation Step*-Nachricht neben der *Status*- eine Reihe von *Move Node*-Nachrichten an den Netzwerksimulator.

Jede *Move Node*-Nachricht enthält die *Position* eines Fahrzeugs zu einer bestimmten Simulationszeit (*TargetTime*), die der *TargetTime* in der *Simulation Step*-Nachricht entspricht. Jedes Fahrzeug wird über eine *NodeId* identifiziert.

D.3.2 Elementare Fahrmanöver

Set Maximum Speed (Identifier: `0x11`)

integer	float
NodeId	MaxSpeed

Diese TraCI-Nachricht beschränkt die Geschwindigkeit des durch *NodeId* bestimmten Fahrzeugs auf *MaxSpeed*. Die Geschwindigkeit des Fahrzeugs soll vom Verkehrssimulator entsprechend der maximal möglichen Bremsverzögerung des Fahrzeugs angepasst werden. Um eine individuelle Geschwindigkeitsbeschränkung aufzuheben, wird die *Set Maximum Speed*-Nachricht mit einem negativen Wert für *MaxSpeed* aufgerufen.

Stop Node (Identifier: `0x12`)

integer	position	float	double
NodeId	StopPosition	Radius	WaitTime

Mit der *Set Maximum Speed*-Nachricht kann ein Fahrzeug unmittelbar zum Anhalten veranlasst werden, indem die Geschwindigkeit auf Null gesetzt wird. Die *Stop Node*-Nachricht erlaubt hingegen das Anhalten an einem definierten Ort entlang der Fahrtroute.

Dazu wird neben der *NodeId*, die das Fahrzeug kennzeichnet, eine Anhalteposition (*StopPosition*) übergeben. Als Anhalteposition kann entweder eine `2D Position` oder eine `Roadmap Position` angegeben werden. Um die Anhalteposition herum bestimmt ein *Radius* ein Gebiet vor und hinter der Anhalteposition, das z. B. einer Bushaltestelle entsprechen kann. Der Verkehrssimulator sorgt dafür, dass das Fahrzeug innerhalb des so definierten kreisförmigen Gebiets anhält. Die Variable *WaitTime* definiert die Zeit, die das Fahrzeug an der Anhalteposition wartet. Liegt die Anhalteposition nicht auf der Fahrtroute des Fahrzeugs, so wird sie nicht berücksichtigt. Das heißt im Umkehrschluss, dass diese TraCI-Nachricht keinerlei Einfluss auf die Fahrtroute hat.

Change Lane (Identifier: `0x13`)

integer	byte	float
NodeId	Lane	Time

Die *Change Lane*-Nachricht legt die Fahrspur (*Lane*) eines Fahrzeugs (*NodeId*) fest. Das Fahrzeug wechselt so schnell wie möglich auf die angegebene Fahrspur und verbleibt dort für eine gewisse Zeit (*Time*). Beim Übergang des Fahrzeugs von einem Streckenabschnitt zum nächsten, wird die Einschränkung auf dem nächsten Streckenabschnitt

fortgesetzt, sofern die entsprechende Spur weiterhin vorhanden ist. Ansonsten wird die Beschränkung aufgehoben.

Change Route (Identifier: 0x30)

integer	string	double
NodeId	RoadId	TravelTime

Zur Anpassung der Fahrtroute einzelner Fahrzeuge kann die erwartete Fahrtzeit für einzelne Streckenabschnitte verändert werden, wodurch die Nutzung des Streckenabschnitts mehr oder weniger attraktiv wird. Für Streckenabschnitte, für die keine individuelle Fahrtzeit angegeben ist, wird standardmäßig der Quotient aus der Länge des Streckenabschnitts und der zulässigen Höchstgeschwindigkeit als Fahrtzeit verwendet. Die hier angegebene Fahrtzeit hat nur Einfluss auf die Streckenberechnung, die Mikrosimulation der Fahrzeuge wird hierdurch nicht beeinflusst.

Die *Change Route*-Nachricht benötigt dafür die *NodeId* des betroffenen Fahrzeugs, die *RoadId* des Streckenabschnitts und die neue erwartete Fahrtzeit für diesen Streckenabschnitt (*TravelTime*). Ist die übergebene Fahrtzeit negativ, so wird die individuelle Gewichtung für den Streckenabschnitt aufgehoben. Um hingegen einen Streckenabschnitt zu blockieren, kann eine sehr hohe Fahrtzeit übergeben werden.

Bei jedem Aufruf dieser TraCI-Nachricht wird die Fahrtroute des betroffenen Fahrzeugs unter Berücksichtigung der neuen Gegebenheiten erneut berechnet.

Change Destination (Identifier: 0x31)

integer	string
NodeId	RoadId

Die *Change Destination*-Nachricht erlaubt eine Anpassung des Fahrtziels (*RoadId*) für ein Fahrzeug (*NodeId*). Falls bei der Neuberechnung der Fahrtroute festgestellt wird, dass eine Fahrtroute zum angegebenen Streckenabschnitt nicht existiert, so wird in der zugehörigen *Status*-Nachricht ein Fehler zurückgegeben und das Fahrtziel des Fahrzeugs nicht verändert.

D.3.3 Zugriff auf das Verkehrsszenario

Position Conversion (Identifier: 0x71)

position	ubyte
Position	PositionType

Diese TraCI-Nachricht dient zur Umrechnung zwischen verschiedenen Darstellungsformen von Positionen. Dazu wird eine Position in einem der in Abschnitt D.2 vorgestellten Positions-Datentypen angegeben, sowie die Datentyp-Kennung (*ID*) des Zielformats im Feld *PositionType*. Bei der Umrechnung einer beliebigen `2D Position` in eine `Roadmap Position` wird die am nächsten liegende Position innerhalb des Straßennetzes berechnet.

Die *Antwort* wird ebenfalls als *Position Conversion*-Nachricht gesendet. Dabei wird der Wert *PositionType* entsprechend der Angabe in der *Anfrage* wiederholt.

Scenario (Identifier: `0x73`)

ubyte	ubyte	integer	ubyte	ubyte	Datatype
read/write	DomainId	ObjectId	Property	Identifier	Value

Die *Scenario*-Nachricht wird zum Zugriff auf diverse Parameter der Verkehrssimulation verwendet. Über das Feld *read/write* wird festgelegt, ob lesend (`0x00`) oder schreibend (`0x01`) auf den Verkehrssimulator zugegriffen wird. Dem folgt die Auswahl der Domäne (*DomainId*), die festlegt, ob sich diese Nachricht auf das Straßennetz (`0x00`), die Fahrzeuge (`0x01`), die Ampeln (`0x02`), die Sehenswürdigkeiten (`0x03`) oder die Polygone (`0x04`) im Verkehrsszenario bezieht. Die *ObjectId* legt den Bezug auf ein bestimmtes Objekt aus einer der vorgenannten Domänen fest. Über das Feld *Property* wird die Eigenschaft des ausgewählten Objekts gewählt, die gelesen oder geschrieben werden soll. *Identifier* legt den Datentyp fest, der bei einem schreibenden Zugriff als *Value* übergeben wird, bzw. bei einem lesenden Zugriff als Rückgabewert erwartet wird.

Als *Antwort* wird ebenfalls eine *Scenario*-Nachricht verwendet. Die Felder *read/write*, *DomainId*, *ObjectId*, *Property* und *Identifier* bleiben unverändert. Eine Antwort enthält aber in jedem Fall das Feld *Value*, das bei einem lesenden Zugriff den Wert der angefragten Eigenschaft enthält. Bei einem schreibenden Zugriff wird ebenfalls der Wert der Eigenschaft zurückgeliefert, der bei einer erfolgreichen Zuweisung dem *Value* der *Anfrage* entspricht.

Tabelle D.2 enthält eine Auswahl der Eigenschaften, auf die in den einzelnen Domänen zugegriffen werden kann. Für eine detaillierte Übersicht wird auf [4] verwiesen.

Get Traffic Light Status (Identifier: `0x41`)

integer	double	double
TrafficLightId	TimeFrom	TimeUntil

Diese TraCI-Nachricht dient zur Anfrage der Phasenwechsel, die eine Ampel (*TrafficLightId*) innerhalb des von *TimeFrom* und *TimeUntil* eingeschlossenen Zeitraums ausführen wird, bzw. ausgeführt hat. Dabei ist zu beachten, dass je nach Schaltungslogik einer Ampel die Phasenwechsel nicht für beliebige zukünftige Zeiten feststehen.

Tabelle D.2: Zugriff auf das Verkehrsszenario mithilfe der *Scenario*-Nachricht.

Domain / **Property**	**Roadmap** (0x00)	**Vehicle** (0x01)	**Traffic Light** (0x02)	**Point of Interest** (0x03)	**Polygon** (0x04)
Count (0x01)	✓	✓	✓	✓	✓
Name (0x0D)	✓	✓	✓	✓	✓
Position (0x02)		✓	✓	✓	✓
Shape (0x09)	✓				✓
Bounding Box (0x03)	✓				✓
Equipped Count (0x0B)		✓			
Speed (0x04)		✓			
Route (0x0E)		✓			
Actual Phase (0x05)			✓		
Next phase change (0x06)			✓		

Traffic Light Switching Information (Identifier: 0x91)

double	string	float	string	ubyte
SwitchTime	FromRoad	FromRoadPos	ToRoad	NewStatus

Als *Antwort* auf die *Get Traffic Light Status*-Nachricht wird für jeden Phasenwechsel und jede beteiligte Fahrtrichtung je eine *Traffic Light Switching Information*-Nachricht generiert. Den Umschaltzeitpunkt enthält das Feld *SwitchTime*. Die Felder *FromRoad* und *ToRoad* identifizieren die vom Phasenwechsel betroffenen ein- und ausgehenden Streckenabschnitte. *FromRoadPos* gibt die Position der Haltelinie auf dem Streckenabschnitt *FromRoad* an. Das neue Signalbild ist in *NewStatus* angegeben (0x01 für Rot und 0x03 für Grün).

D.4 Implementierung

Das *Traffic Control Interface* (TraCI) wurde so generisch wie möglich entworfen, um eine möglichst große Anzahl an Verkehrs- und Netzwerksimulatoren unterstützen zu können. Im Rahmen dieser Arbeit wurden *TraCI-Module* für den Verkehrssimulator SUMO sowie die Netzwerksimulatoren ns-2 und Shawn entwickelt, die im Folgenden aus Anwendersicht vorgestellt werden.

D.4.1 Verkehrssimulator SUMO

Die Änderungen am Verkehrssimulator SUMO sind nach außen sichtbar über zwei neue Kommandozeilenparameter:

`-port <int>` Legt den Netzwerkport fest, an dem SUMO auf eine eingehende Verbindung eines TraCI-Clients wartet. Beim Vorhandensein mehrerer Netzwerkschnittstellen wird an allen Schnittstellen auf eine eingehende Verbindung gewartet.

`-penetration <float>` Bestimmt den Ausstattungsgrad, d. h. den Anteil der Fahrzeuge, die über TraCI an den Netzwerksimulator gemeldet werden sollen. Der Wert liegt im Bereich [0, 1]. Wenn dieser Parameter nicht angegeben wird, werden alle Fahrzeuge an den Netzwerksimulator gemeldet. Sollte eine genauere Angabe der ausgestatteten Fahrzeuge notwendig sein, z. B. je nach Fahrzeugtyp, dann kann diese Angabe in den SUMO-Konfigurationsdateien vorgenommen werden. Die Angabe in den Konfigurationsdateien wird dann mit dem Ausstattungsgrad multipliziert, der auf der Kommandozeile übergeben wurde.

Beim Start von SUMO stehen alle Optionen des Verkehrssimulators zur Verfügung. Ist der Parameter `-port <int>` angegeben, wird die Simulation allerdings nicht sofort ausgeführt, sondern SUMO wartet nach dem Laden des Verkehrsszenarios auf dem angegebenen Port auf eine eingehende Verbindung. Ein TraCI-Client, der sich daraufhin mit SUMO verbindet, steuert die Simulation wie in Abschnitt 6.3.1 angegeben.

D.4.2 Netzwerksimulator ns-2

Der Netzwerksimulator ns-2 ist in der Programmiersprache C++ implementiert. Um ohne Neukompilierung eine Komposition des Simulationsszenarios auf Basis der in C++ implementierten Algorithmen, Protokolle und Modelle zu erlauben, wird über die Skriptsprache TCL [189] auf die C++-Objekte zugegriffen.

Unter Berücksichtigung dieser Architektur wurde der TraCI-Client für ns-2 in C++ als Schnittstellenklasse implementiert. Der Zugriff auf die *TraCIClient*-Instanz kann sowohl aus der C++- als auch der TCL-Umgebung erfolgen. Üblicherweise wird TraCI durch einige Zeilen TCL-Code initialisiert:

```
set traci_client [new TraCIClient]
$traci_client set-remoteHost localhost
$traci_client set-remotePort 8888
$traci_client set-timeInterval 1.0
$traci_client startSimStepHandler
```

Dabei wird zunächst mittels `new TraCIClient` eine Instanz der TraCI-Client-Klasse angelegt. Daraufhin werden einige Parameter gesetzt, die die Netzadresse des TraCI-Servers (`set-remoteHost` und `set-remotePort`) sowie die Dauer eines Simulationsschritts (`set-timeInterval`) festlegen. Der Aufruf von `startSimStepHandler` startet die Steuerung des Simulationsablaufs gemäß Abschnitt 6.3.1. Dafür werden nach jedem Ablauf eines Simulationsschritts neue Bewegungsdaten vom Verkehrssimulator erfragt und die Knoten im Netzwerksimulator entsprechend bewegt.

Da ns-2 das Instanziieren von Knoten nur zu Beginn der Simulation zulässt, kann über die *Scenario*-Nachricht die maximale Anzahl an Fahrzeugen in der Simulation erfragt werden und eine entsprechende Anzahl an Knoten schon vor Beginn der Simulation angelegt werden. Damit der TraCI-Client die Knoten automatisch nutzt, werden sie ihm mittels seiner Methode `add-node` bekannt gemacht. Um die VANET-Anwendung auf einem Knoten erst in dem Moment zu aktivieren, in dem er in der Verkehrssimulation auftaucht, wird die Methode `start-on-move` verwendet. Analog sorgt `stop-on-halt` für eine Deaktivierung der VANET-Anwendung auf einem Knoten, sobald das assoziierte Fahrzeug die Verkehrssimulation verlassen hat.

```
set number_of_nodes [$traci_client command-scenario
                     DOM_VEHICLE -1 DOMVAR_EQUIPPEDCOUNTMAX]
for {set i 0} {$i < number_of_nodes } {incr i} {
   set node($i) [$ns node]
   $traci_client add-node $node($i)
   $traci_client start-on-move $node($i) $agent($i)
   $traci_client stop-on-halt $node($i) $agent($i)
}
```

Alle im vorhergehenden vorgestellten TraCI-Nachrichten können sowohl von TCL als auch von C++ aus aufgerufen werden. Der folgende Aufruf nutzt beispielsweise die *Set Maximum Speed*-Nachricht, um ein Fahrzeug (`$node($i)`) anhalten zu lassen:

```
$traci_client command-setMaximumSpeed $node($i) 0
```

In C++ wird zum Zugriff auf den TraCI-Client die statische Methode `TraCIClient::instance()` verwendet, die einen Zeiger auf die in TCL angelegte *TraCIClient*-Instanz zurückliefert. Der gleiche Befehl zum Anhalten eines Fahrzeugs ergibt sich wie folgt:

```
TraCIClient::instance()->commandSetMaximumSpeed(node[i], 0);
```

Der TraCI-Client für ns-2 wird im TraCI-Wiki [4] als Patch zum Download angeboten.

D.4.3 Netzwerksimulator Shawn

Ebenso wie ns-2 ist auch Shawn in C++ implementiert. Die Implementierung des TraCI-Clients fällt daher grundsätzlich sehr ähnlich aus, ist allerdings an die Architektur von Shawn angepasst. So wird die Simulation in Shawn über eine Konfigurationsdatei gesteuert. Über den Aufruf verschiedener *SimulationTasks* kann das Simulationsszenario aus Knoten, Protokollen und Algorithmen zusammengestellt werden.

Ein solcher *SimulationTask* wird von der Klasse *SimulationTaskNodeMovement* bereitgestellt. Diese enthält verschiedene Bewegungsmodelle, welche die Bewegung aller Knoten steuern kann. Um die Bewegungsdaten über TraCI zu beziehen, ist folgender Aufruf innerhalb der Konfigurationsdatei nötig:

```
node_movement mode=TraCI \
              remote_host=localhost \
              remote_port=8888 \
              time_interval=1
```

Der *SimulationTaskNodeMovement* wird über das Schlüsselwort `node_movement` eingeleitet. Der nachfolgende Parameter `mode` bestimmt das verwendete Bewegungsmodell. Die weiteren Parameter bestimmen die Netzadresse des TraCI-Servers (`remote-host` und `remote-port`) sowie die Dauer eines Simulationsschritts (`time_interval`). Nach diesem Aufruf steuert der *SimulationTaskNodeMovement* den Simulationsablauf gemäß Abschnitt 6.3.1.

Die Methoden zur Beeinflussung des Fahrverhaltens sind in Shawn nur direkt über C++ aufrufbar. Abgesehen von der abweichenden Namenskonvention sind die Aufrufe identisch mit denen für ns-2.

So lässt der folgende C++-Ausdruck das Fahrzeug, das dem Knoten `node[i]` zugeordnet ist, anhalten:

```
TraCIClient::instance()->command_set_maximum_speed(node[i], 0);
```

Der TraCI-Client ist fester Bestandteil des Netzwerksimulators Shawn. Auf den Seiten des Shawn-Wikis [3] befinden sich eine TraCI-Referenz sowie eine Möglichkeit zum Download.

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ABS	Antiblockiersystem
ADS	Adaptable Distributed Strategy
AODV	Ad-hoc On-Demand Distance Vector Routing
ASCII	American Standard Code for Information Interchange
AVMN	Autonomous Virtual Mobile Node [36]
C2C-CC	CAR-2-CAR Communication Consortium (Initiative europäischer Automobilhersteller zur Entwicklung eines offenen VANET Kommunikationsstandards)
API	Programmierschnittstelle (engl. Application Programming Interface)
BMBF	Bundesministerium für Bildung und Forschung
BMWI	Bundesministerium für Wirtschaft und Technologie
BricxCC	Bricx Command Center IDE für NXC
CSMA	Carrier Sense Multiple Access
DFG	Deutsche Forschungsgemeinschaft
DHT	Distributed Hash Table
DSDV	Destination-Sequenced Distance Vector Routing
DSRC	Dedicated Short Range Communication
DVSIS	Distributed Virtual Shared Information Space
ESP	Elektronisches Stabilitäts Programm (Fahrerassistenzsystem)
FCD	Floating Car Data
GCC	GNU Compiler Collection
GHT	Geographical Hash Table
GPS	Global Positioning System
GPSR	Greedy Perimeter Stateless Routing
GSM	Global System for Mobile Communications (Mobilfunkstandard der 2. Generation)
HBEFA	Handbuch für Emissionsfaktoren des Strassenverkehrs [85]
HDC	Hovering Data Cloud

IDE	Integrated Development Environment (engl für Integrierte Entwicklungsumgebung)
LAR	Location-Aided Routing
MANET	Mobile Ad-hoc Network
NEMO	Protokoll-Stack für Network Mobility [11, 12]
NMEA	National Marine Electronics Association
ns-2	Network Simulator 2 [38]
NXC	Not eXactly C ist eine Programmiersprache für das Lego-Mindstorms-NXT-System
OIC	Organic Information Complex
PBSM	Parameterless Broadcasting from Static to Mobile [87]
RIM	Radio Irregularity Model (auch RI-Modell) [197]
SOTIS	Self Organizing Traffic Information System [194]
SPP	Serial Port Profile (Bluetooth-Profil) [19]
SUMO	Simulator of Urban Mobility (Verkehrssimulator) [93]
TCL	Tool Command Language [189]
TCP	Transmission Control Protocol
TMC	Traffic Message Channel
TraCI	Traffic Control Interface (Schnittstelle zur Interaktion mit Verkehrssimulatoren) [186, 187]
TraNS	Traffic and Network Simulation Environment [141]
WWW	World Wide Web
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System (Mobilfunkstandard der 3. Generation)
VANET	Vehicular Ad-hoc Network (engl. für Fahrzeug-Fahrzeug Netz)
XML	Extensible Markup Language

Literaturverzeichnis

[1] *Offizielle Internetseite der Lego Mindstorms Produktreihe.* – URL `http://mindstorms.lego.com/eng/overview`

[2] *Verbundprojekt ADVEST: Adaptive Verkehrssteuerung.* BMBF-Programm „Mathematik für Innovationen in Industrie und Dienstleistungen“. – URL `http://www.math.tu-berlin.de/coga/projects/traffic/advest/`

[3] *Wiki for the Network Simulator Shawn.* – URL `http://shawn.sf.net`

[4] *Wiki for the **Tra**ffic **C**ontrol **I**nterface (TraCI).* – URL `http://apps.sf.net/mediawiki/sumo/index.php?title=TraCI`

[5] *Organic Computing.* DFG Schwerpunktprogramm 1183. 2005–2011. – URL `http://www.organic-computing.de/spp`

[6] *AKTIV – Adaptive und Kooperative Technologien für den Intelligenten Verkehr.* Bundesministerium für Wirtschaft und Technologie. 2006–2010. – URL `http://www.aktiv-online.org`

[7] Aberer, Karl ; Hauswirth, Manfred: An Overview on Peer-to-Peer Information Systems. In: *Proceedings of the 4th International Workshop on Distributed Data & Structures (WDAS).* Paris, France : Carleton Scientific, März 2002, S. 171–188

[8] Abolhasan, Mehran ; Wysocki, Tadeusz ; Dutkiewicz, Eryk: A Review of Current On-demand Routing Protocols. In: *Proceedings of the First International Conference on Networking (ICN)* Bd. 2. London, UK : Springer-Verlag, 2001, S. 186–195. – ISBN 3-540-42303-6

[9] Ahn, Sungjoon ; Shankar, A. U.: Adapting to Route-demand and Mobility (ARM) in Ad hoc Network Routing. In: *International Journal of Computer and Telecommunications Networking* 38 (2002), Nr. 6, S. 745–764. – ISSN 1389-1286

[10] Baldessari, Roberto ; Bödekker, Bert ; Deegener, Matthias ; Festag, Andreas ; Franz, Walter ; Kellum, C. C. ; Kosch, Timo ; Kovacs, Andras ; Lenardi, Massimiliano ; Menig, Cornelius ; Peichl, Timo ; Röckl, Matthias ; Seeberger, Dieter ; Strassberger, Markus ; Stratil, Hannes ; Vögel, Hans-Jörg ;

Weyl, Benjamin ; Zhang, Wenhui ; Car-2-Car Communication Consortium (Hrsg.): *Manifesto (Version 1.1)*. August 2007

[11] Baldessari, Roberto ; Festag, Andreas ; Abeillé, Julien: NEMO meets VANET: A Deployability Analysis of Network Mobility in Vehicular Communication. In: *7th International Conference on ITS Telecommunications (ITST)*. Sophia Antipolis, France, Juni 2007, S. 375–380

[12] Baldessari, Roberto ; Festag, Andreas ; Lenardi, Massimiliano: *C2C-Communication Consortium Requirements for Usage of NEMO in VANETs*. Internet Draft. Februar 2007. – URL `http://www.ietf.org/internet-drafts/draft-baldessari-c2ccc-nemo-req-00.txt`

[13] Bani-Yassein, Muneer M. ; Ould-Khaoua, Mohamed ; Mackenzie, Lewis M. ; Papanastasiou, Stylianos: Performance Analysis of Adjusted Probabilistic Broadcasting in Mobile Ad Hoc Networks. In: *International Journal of Wireless Information Networks* 13 (2006), April, Nr. 2, S. 127–140. – ISSN 1068-9605

[14] Bassett, Danielle S. ; Bullmore, Ed: Small-World Brain Networks. In: *The Neuroscientist* 12 (2006), Nr. 6, S. 512–523. – ISSN 1073-8584

[15] Baum, Moritz: *Stauerkennung mittels Hovering Data Clouds für den Mindstorms Stau Demonstrator*. Bachelorarbeit, Institut für Telematik, Universität zu Lübeck. Juni 2008

[16] Bechler, Marc: *Internet Integration of Vehicular Ad Hoc Networks*, Technische Universität Braunschweig, Dissertation, 2004

[17] Berns, Hans-Gerd ; Burnett, Toby H. ; Gran, Richard ; Wilkes, R. J.: GPS Time Synchronization in School-Network Cosmic Ray Detectors. In: *IEEE Transactions on Nuclear Science* 51 (2004), Juni, Nr. 3, Part 3, S. 848–853. – ISSN 0018-9499

[18] Bloom, Burton H.: Space/time trade-offs in hash coding with allowable errors. In: *Communications of the ACM* 13 (1970), Nr. 7, S. 422–426. – ISSN 0001-0782

[19] Bluetooth Special Interest Group: *Bluetooth Specification – Version 1.1*. Kap. K:5 Serial Port Profile, S. 171–196, 2001

[20] Bluetooth Special Interest Group: *Specification of the Bluetooth System – Covered Core Package version: 2.0 + EDR*. Volume 0–4. November 2004. – 1230 S

[21] Braess, Dietrich: On a Paradox of Traffic Planning. In: *Transportation Science* 39 (2005), November, Nr. 4, S. 446–450. – Translation from the original German:

Braess, Dietrich. 1968. Über ein Paradoxon aus der Verkehrsplanung. Unternehmensforschung 12 258-268

[22] Brockfeld, Elmar ; Kühne, Reinhart ; Wagner, Peter: Calibration and Validation of Microscopic Traffic Flow Models. In: *Transportation Research Record: Journal of the Transportation Research Board* 1934 (2005), Januar, S. 179–187. – ISSN 0361-1981

[23] Brutschy, Arne ; Scheidler, Alexander ; Merkle, Daniel ; Middendorf, Martin: Learning from House-Hunting Ants: Collective Decision-Making in Organic Computing Systems. In: *Proceedings of the 6th International Conference on Ant Colony Optimization and Swarm Intelligence (ANTS)*, 2008, S. 96–107. – ISBN 978-3-540-87526-0

[24] Buschmann, Carsten ; Hellbrück, Horst ; Fischer, Stefan ; Kröller, Alexander ; Fekete, Sándor P.: Radio Propagation-Aware Distance Estimation Based on Neighborhood Comparison. In: *Proceedings of the 4th European Conference on Wireless Sensor Networks*, 2007

[25] Buschmann, Carsten ; Pfisterer, Dennis: iSense: A Modular Hardware and Software Platform for Wireless Sensor Networks. In: *Tagungsband der 6. Fachgespräche „Drahtlose Sensornetze“ der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme*, Juli 2007, S. 15–18

[26] Camazine, Scott ; Deneubourg, Jean-Louis ; Franks, Nigel R. ; Sneyd, James ; Theraulaz, Guy ; Bonabeau, Eric: *Self-Organization in Biological Systems*. Princeton University Press, 2003. – 560 S. – ISBN 0-691-11624-5

[27] Cartigny, Julien ; Simplot, David: Border Node Retransmission Based Probabilistic Broadcast Protocols in Ad-Hoc Networks. In: *Telecommunication Systems* 22 (2003), Januar, Nr. 1, S. 189–204

[28] Cerf, V.G.: *ASCII format for network interchange*. RFC 20. Oktober 1969. – URL `http://www.ietf.org/rfc/rfc20.txt`

[29] Choffnes, David R. ; Bustamante, Fabián E.: An Integrated Mobility and Traffic Model for Vehicular Wireless Networks. In: *Proceedings of the 2nd ACM International Workshop on Vehicular Ad-hoc Networks (VANET)*. Cologne, Germany : ACM, 2005, S. 69–78. – ISBN 1-59593-141-4

[30] Chowdhury, Debashish ; Santen, Ludger ; Schadschneider, Andreas: Statistical Physics of Vehicular Traffic and Some Related Systems. In: *Physics Reports* 329 (2000), Mai, Nr. 4–6, S. 199–329

[31] Cottingham, David N. ; Davies, Jonathan J.: A Vision for Wireless Access on the Road Network. In: *Proceedings of the 4th International Workshop on Intelligent Transportation (WIT)*. Hamburg, Germany, März 2007, S. 25–30

[32] Cottingham, David N. ; Davies, Jonathan J. ; Beresford, Alastair R.: Congestion-Aware Vehicular Traffic Routing using WiFi Hotspots. In: *Proceedings of the Communications Innovation Institute Workshop*. Cambridge, UK, Mai 2005

[33] Cseh, Christian ; Eberhardt, Reinhold ; Franz, Walter J.: Mobile Ad-Hoc Netzwerke für die Fahrzeug-Fahrzeug Kommunikation. In: *Tagungsband des 1. deutschen Workshops über Mobile Ad-Hoc Netzwerke (WMAN)*. Ulm, Germany, 2002, S. 109–119

[34] De Wolf, Tom ; Holvoet, Tom: Emergence Versus Self-Organisation: Different Concepts but Promising When Combined. In: *Engineering Self-Organising Systems* 3464 (2005), S. 1–15

[35] Dittrich, Peter: Chemical Computing. In: *Proceedings of the International Workshop on Unconventional Programming Paradigms*, 2005, S. 19–32. – ISBN 978-3-540-27884-9

[36] Dolev, Shlomi ; Gilbert, Seth ; Schiller, Elad ; Shvartsman, Alex A. ; Welch, Jennifer L.: Autonomous Virtual Mobile Nodes. In: *Proceedings of the 2005 joint workshop on Foundations of mobile computing (DIALM-POMC)*. Cologne, Germany, Juni 2005, S. 62–69. – ISBN 1-59593-092-2

[37] Eichler, Stephan ; Ostermaier, Benedikt ; Schroth, Christoph ; Kosch, Timo: Simulation of Car-to-Car Messaging: Analyzing the Impact on Road Traffic. In: *Proceedings of the 13th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, IEEE Computer Society, September 2005, S. 507–510

[38] Fall, Kevin ; Varadhan, Kannan: *The ns Manual*. The VINT Project - a collaboration between researchers. 1989–2001. – URL `http://www.isi.edu/nsnam/ns/doc/`

[39] Farooq, Muddassar: *From the Wisdom of the Hive to Intelligent Routing in Telecommunication Networks - A Step towards Intelligent Network Management through Natural Engineering*, Universität Dortmund, Dissertation, 2006

[40] Fekete, Sándor P. ; Fischer, Stefan ; Hellbrück, Horst ; Hendriks, Björn ; Wegener, Axel: *AutoNomos: A Distributed and Self-Regulating Approach for Organizing a Large System of Mobile Objects*. DFG SPP 1183. – URL `http://www.auto-nomos.de/`

[41] Fekete, Sándor P. ; Gray, Chris ; Hendriks, Björn: *Verbundprojekt ADVEST: Teilprojekt Dezentrale Optimierung von Verkehrsflüssen.* BMBF

[42] Fekete, Sándor P. ; Kröller, Alexander ; Fischer, Stefan ; Pfisterer, Dennis: Shawn: The fast, highly customizable sensor network simulator. In: *Proceedings of the 4th International Conference on Networked Sensing Systems (INSS)* Bd. 6-8, Juni 2007, S. 299

[43] Fekete, Sándor P. ; Schmidt, Christiane ; Wegener, Axel ; Fischer, Stefan: Hovering Data Clouds for Recognizing Traffic Jams. In: *Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA).* Paphos, Cyprus, November 2006, S. 198–203. – ISBN 978-0-7695-3071-0

[44] Fekete, Sándor P. ; Tessars, Christopher ; Schmidt, Christiane ; Wegener, Axel ; Fischer, Stefan ; Hellbrück, Horst: *Verfahren und Vorrichtung zur Ermittlung einer Fahrstrategie.* 2008. – Patentanmeldung, Aktenzeichen UNV-TUB-102-DE

[45] Festag, Andreas ; Noecker, Gerhard ; Strassberger, Markus ; Lübke, Andreas ; Bochow, Bernd ; Torrent-Moreno, Marc ; Schnaufer, Sascha ; Eigner, Robert ; Catrinescu, Catrinel ; Kunisch, Jürgen: NoW – Network on Wheels: Project Objectives, Technology and Achievements. In: *Proceedings of the 5th International Workshop on Intelligent Transportation (WIT).* Hamburg, Germany, März 2008, S. 211–216

[46] Feuerstein, M. J. ; Blackard, K. L. ; Rappaport, T. S. ; Seidel, S. Y. ; Xia, H. H.: Path loss, delay spread, and outage models as functions of antenna height for microcellular system design. In: *IEEE Transactions on Vehicular Technology* 43 (1994), August, Nr. 3, S. 487–498. – ISSN 0018-9545

[47] Fey, Dietmar ; Gaede, C. ; Loos, Andreas ; Komann, Marcus: A New Marching Pixels Algorithm for Application-Specific Vision Chips for Fast Detection of Objects' Centroids. In: *Proceedings of the 20th International Conference on Parallel and Distributed Computing and Systems (PDCS).* Orlando, Florida, USA, 2008

[48] Forschungsgesellschaft für Strassen- und Verkehrswesen / Kommission Bemessung von Strassenverkehrsanlagen: *Handbuch für die Bemessung von Straßenverkehrsanlagen.* Köln, Germany : FGSV, Ausgabe 2001, Fassung 2005

[49] Frank, Christian ; Römer, Kay: Algorithms for Generic Role Assignment in Wireless Sensor Networks. In: *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys)*, 2005, S. 230–242

[50] Franz, Walter (Hrsg.) ; Hartenstein, Hannes (Hrsg.) ; Mauve, Martin (Hrsg.): *Inter-Vehicle-Communications Based on Ad Hoc Networking Principles - The FleetNet Project*. Universitätsverlag Karlsruhe, 2005. – ISBN 3-937300-88-0

[51] Ganesan, Deepak ; Krishnamachari, Bhaskar ; Woo, Alec ; Culler, David ; Estrin, Deborah ; Wicker, Stephen: Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks / UCLA Computer Science Technical Report. 2002 (UCLA/CSD-TR 02-0013). – Forschungsbericht

[52] Gardner, Martin: MATHEMATICAL GAMES: The fantastic combinations of John Conway's new solitaire game "life". In: *Scientific American* 223 (1970), Oktober, S. 120–123

[53] Ghose, Abhishek ; Grossklags, Jens ; Chuang, John: Resilient Data-Centric Storage in Wireless Ad-Hoc Sensor Networks. In: *Proceedings of the 4th International Conference on Mobile Data Management (MDM)*. Melbourne, Australia : Springer-Verlag, 2003, S. 45–62. – ISBN 3-540-00393-2

[54] Goldberg, David E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989. – ISBN 0-201-15767-5

[55] Gorgen, Daniel ; Frey, Hannes ; Hiedels, Christian: JANE – The Java Ad Hoc Network Development Environment. In: *Proceedings of the 40th Annual Simulation Symposium*. Los Alamitos, CA, USA : IEEE Computer Society, 2007, S. 163–176

[56] Grimmett, Geoffrey R.: *Percolation*. 2nd Edition. Springer, 1999. – ISBN 978-3-540-64902-1

[57] Grossglauser, Matthias ; Tse, David N. C.: Mobility Increases the Capacity of Ad-hoc Wireless Networks. In: *IEEE/ACM Transactions on Networking* 10 (2002), August, Nr. 4, S. 477–486

[58] Haas, Zygmunt J. ; Halpern, Joseph Y. ; Li, Li: Gossip-Based Ad Hoc Routing. In: *IEEE/ACM Transactions on Networking* 14 (2006), Juni, Nr. 3, S. 479–491. – ISSN 1063-6692

[59] Hamilton, Scott: Taking Moore's law into the next century. In: *IEEE Computer* 32 (1999), Januar, Nr. 1, S. 43–48. – ISSN 0018-9162

[60] Hanashi, Abdalla M. ; Siddique, Aamir ; Awan, Irfan ; Woodward, Mike: Dynamic Probabilistic Flooding Performance Evaluation of On-Demand Routing Protocols in MANETs. In: *Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems*. Los Alamitos, CA, USA : IEEE Computer Society, 2008, S. 200–204

[61] Hansen, John: *NXT Power Programming: Robotics in C*. Variant Press, 2007. – 544 S. – ISBN 978-0-973864-922

[62] Härri, J. ; Filali, F. ; Bonnet, C. ; Fiore, Marco: VanetMobiSim: Generating Realistic Mobility Patterns for VANETs. In: *Proceedings of the 3rd International Workshop on Vehicular Ad-hoc Networks (VANET)*, 2006, S. 96–97

[63] Heckel, Sergej: *Entwicklung einer grafischen Oberfläche für den Mindstorms Stau Demonstrator*. Bachelorarbeit, Institut für Telematik, Universität zu Lübeck. Juli 2008

[64] Hedetniemi, Sandra M. ; Hedetniemi, Stephen T. ; Liestman, Arthur L.: A Survey of Gossiping and Broadcasting in Communication Networks. In: *Networks* 18 (1988), Nr. 4, S. 319–349

[65] Heinzelman, Wendi R. ; Kulik, Joanna ; Balakrishnan, Hari: Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In: *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, August 1999, S. 174–185. – ISBN 1-58113-142-9

[66] Hellbrück, Horst: *Analytische und simulationsbasierte Verfahren zur Konnektivitätsbestimmung und -verbesserung von Ad-Hoc-Netzen*, Technische Universität Braunschweig, Dissertation, September 2004

[67] Hellbrück, Horst ; Fischer, Stefan: MINE and MILE: Improving Connectivity in Mobile Ad-Hoc Networks. In: *ACM SIGMOBILE Mobile Computing and Communications Review* 8 (2004), Oktober, Nr. 4, S. 19–36

[68] Hellbrück, Horst ; Wegener, Axel ; Fischer, Stefan: AutoCast: A General-Purpose Data Dissemination Protocol and its Application in Vehicular Networks. In: *Ad Hoc & Sensor Wireless Networks Journal (AHSWN)* 6 (2008), Nr. 1–2, S. 91–122. – ISSN 1551-9899

[69] Henricksen, Karen ; Robinson, Ricky: A Survey of Middleware for Sensor Networks: State-of-the-Art and Future Directions. In: *Proceedings of the International Workshop on Middleware for Sensor Networks (MidSens)*. Melbourne, Australia : ACM, 2006, S. 60–65. – ISBN 1-59593-424-3

[70] Holland, John: *Emergence: From Chaos to Order.* Addison-Wesley, 1997. – 258 S. – ISBN 0-201-14943-5

[71] IEEE 802.11 Working Group: *802.11: IEEE Standard for Information technology – Telecommunication and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11 Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* – URL `http://www.ieee802.org/11/`

[72] IEEE 802.11 Working Group, Task Group P: *„IEEE 802.11p Wireless Access for Vehicular Environments“ Draft Standard.* – URL `http://grouper.ieee.org/groups/802/11/`

[73] IEEE Task P754: *IEEE 754-2008, Standard for Floating-Point Arithmetic.* August 2008. – 58 S. – ISBN 978-0-7381-5753-5

[74] Intanagonwiwat, Chalermek ; Govindan, Ramesh ; Estrin, Deborah ; Heidemann, John ; Silva, Fabio: Directed Diffusion for Wireless Sensor Networking. In: *ACM/IEEE Transactions on Networking* 11 (2002), Februar, Nr. 1, S. 2–16. – ISSN 1063-6692

[75] International Organization for Standardization and International Electrotechnical Commision: *Information Technology – Open Systems Interconnection – Basic Reference Model, Part 1: The Basic Model.* International Standard. 1994

[76] Jiang, Daniel ; Delgrossi, Luca: IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments. In: *Proceedings of the 67th IEEE Vehicular Technology Conference (VTC2008-Spring).* Marina Bay, Singapore, Mai 2008, S. 2036–2040

[77] Jiang, Daniel ; Taliwal, Vikas ; Meier, Andreas ; Holfelder, Wieland ; Herrtwich, Ralf: Design of 5.9 GHz DSRC-based Vehicular Safety Communication. In: *IEEE Wireless Communications* 13 (2006), Oktober, Nr. 5, S. 36–43

[78] Johnson, D. ; Hu, Y. ; Maltz, D.: *The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4.* RFC 4728 (Experimental). Februar 2007. – URL `http://www.ietf.org/rfc/rfc4728.txt`

[79] Johnson, Steven: *Emergence: The Connected Lives of Ants, Brains, Cities and Software.* Scribner Book Company, 2002. – 288 S. – ISBN 0-684-86876-8

[80] Junhai, Luo ; Liu, Xue ; Danxia, Ye: Research on multicast routing protocols for mobile ad-hoc networks. In: *International Journal of Computer and Telecommunications Networking* 52 (2008), April, Nr. 5, S. 988–997. – ISSN 1389-1286

[81] Kan, Masataka ; Pande, Rahul ; Vinograd, Patrick ; Garcia-Molina, Hector: Event Dissemination in High-Mobility Ad-Hoc Networks. 2005. – Forschungsbericht

[82] Kargl, Frank ; Papadimitratos, Panagiotis ; Buttyan, Levente ; Müter, Michael ; Schoch, Elmar ; Wiedersheim, Björn ; Schoch, Elmar ; Thong, Ta-Vinh ; Calandriello, Giorgio ; Held, Albert ; Kung, Antonio ; Hubaux, Jean-Pierre: Secure Vehicular Communication Systems: Implementation, Performance, and Research Challenges. In: *IEEE Communications Magazine* 46 (2008), November, Nr. 11, S. 110–118

[83] Karnadi, Feliz K. ; Mo, Zhi H. ; Lan, Kun chan: Rapid Generation of Realistic Mobility Models for VANET. In: *Proceedings of the IEEE Wireless Communications and Networking Conference*, März 2007, S. 2506–2511

[84] Karp, Brad ; Kung, H. T.: GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In: *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*. Boston, Massachusetts, USA, August 2000, S. 243–254. – ISBN 1-58113-197-6

[85] Keller, Mario ; Haan, Peter de ; Knörr, Wolfram ; Hausberger, Stefan ; Steven, Heinz: *Handbuch Emissionsfaktoren des Strassenverkehrs, Version 2.1 – Dokumentation*. INFRAS, im Auftrag von Umweltbundesamt Berlin, Umweltbundesamt / Lebensministerium / Bundesministerium für Verkehr, Innovation und Technologie, Wien und Bundesamt für Umwelt, Wald und Landschaft, Bern. August 2004

[86] Kephart, Jeffrey O. ; Chess, David M.: The Vision of Autonomic Computing. In: *IEEE Computer* 36 (2003), Januar, Nr. 1, S. 41–50. – ISSN 0018-9162

[87] Khan, Adnan A. ; Stojmenovic, Ivan ; Zaguia, Nejib: Parameterless Broadcasting in Static to Highly Mobile Wireless Ad Hoc, Sensor and Actuator Networks. In: *Proceedings of the 22nd IEEE International Conference on Advanced Information Networking and Applications (AINA)*. Ginowan, Okinawa, Japan : IEEE Computer Society, 2008, S. 620–627. – ISSN 1550-445X

[88] Khelil, Abdelmajid ; Marrón, Pedro J. ; Becker, Christian ; Rothermel, Kurt: Hypergossiping: A Generalized Broadcast Strategy for Mobile Ad Hoc Networks. In: *Ad Hoc Networks* 5 (2007), Nr. 5, S. 531–546. – ISSN 1570-8705

[89] Knuth, Donald E.: *The Art of Computer Programming*. Bd. Volume 3 – Sorting and Searching. Reading, Massachusetts : Addison-Wesley, 1973. – ISBN 0-201-03803-X

[90] Ko, Young-Bae ; Vaidya, Nitin H.: Location-Aided Routing (LAR) in mobile ad hoc networks. In: *Wireless Networks* 6 (2000), September, Nr. 4, S. 307–321. – ISSN 1022-0038

[91] Koberstein, Jochen ; Luttenberger, Norbert ; Buschmann, Carsten ; Fischer, Stefan: Shared Information Spaces for Small Devices. The Swarms Software Concept. In: *Proceedings of the Workshop on Sensor Networks at Informatik 2004*. Ulm, Germany, 2004, S. 375–379. – ISBN 3-88579-380-6

[92] Kraftfahrt-Bundesamt: *Pressemitteilung Nr. 7/2009 – Der Fahrzeugbestand am 1. Januar 2009*. 2009

[93] Krajzewicz, Daniel ; Bonert, Michael ; Wagner, Peter: The Open Source Traffic Simulation Package SUMO. In: *RoboCup 2006 Infrastructure Simulation Competition*. Bremen, Germany, 2006

[94] Krajzewicz, Daniel ; Hertkorn, Georg ; Rössel, Christian ; Wagner, Peter: SUMO (Simulation of Urban MObility); An open-source traffic simulation. In: *Proceedings of the 4th Middle East Symposium on Simulation and Modelling*, SCS European Publishing House, September 2002, S. 183–187

[95] Krauss, Stefan: *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics*, Universität zu Köln, Dissertation, April 1998

[96] Kröller, A. ; Pfisterer, D. ; Buschmann, C. ; Fekete, S. P. ; Fischer, S.: Shawn: A new approach to simulating wireless sensor networks. In: *Design, Analysis, and Simulation of Distributed Systems 2005, Part of the SpringSim 2005*, April 2005

[97] Lego Group: *LEGO MINDSTORMS NXT – Bluetooth Developer Kit*. Version 1.00, 2006

[98] Lego Group: *LEGO MINDSTORMS NXT – Hardware Developer Kit*. Version 1.00, 2006

[99] Li, Jinyang ; Jannotti, John ; Couto, Douglas S. J. D. ; Karger, David R. ; Morris, Robert: A Scalable Location Service for Geographic Ad Hoc Routing. In: *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*. Boston, Massachusetts, USA : ACM, 2000, S. 120–130. – ISBN 1-58113-197-6

[100] Li, Shuoqi ; Son, Sang H. ; Stankovic, John A.: Event Detection Services Using Data Service Middleware in Distributed Sensor Networks. In: *Proceedings of the 2nd International Workshop on Information Processing In Sensor Networks:*, April 2003, S. 502–517

[101] Li, Xiang-Yang ; Moaveninejad, Kousha ; Frieder, Ophir: Regional gossip routing for wireless ad hoc networks. In: *Mobile Networks and Applications* 10 (2005), Februar, Nr. 1-2, S. 61–77. – ISSN 1383-469X

[102] Liggins, Martin E. ; Llinas, James ; Hall, David L.: *Multisensor Data Fusion.* 2nd Edition. Boca Raton, FL, USA : CRC Press, Inc., 2008. – ISBN 978-1-4200-5306-7

[103] Lochert, Christian ; Caliskan, Murat ; Scheuermann, Björn ; Barthels, Andreas ; Cervantes, Alfonso ; Mauve, Martin: Multiple Simulator Interlinking Environment for Inter Vehicle Communication. In: *Proceedings of the 2nd International Workshop on Vehicular Ad Hoc Networks (VANET)*, ACM, September 2005, S. 87–88

[104] Lohse, Dieter ; Lätzsch, Lothar ; Schnabel, Werner ; Windolph, Joachim: *Grundlagen der Straßenverkehrstechnik und der Verkehrsplanung.* Bd. 1: Verkehrstechnik, Bd. 2: Verkehrsplanung. 2. Auflage. Beuth, 1997. – 1040 S. – ISBN 3-410-16445-6

[105] Lownes, Nicholas E. ; Machemehl, Randy B.: VISSIM: A Multi-Parameter Sensitivity Analysis. In: *Proceedings of the 38th Winter Simulation Conference (WSC).* Monterey, California, 2006, S. 1406–1413. – ISBN 1-4244-0501-7

[106] Lua, Eng K. ; Crowcroft, Jon ; Pias, Marcelo ; Sharma, Ravi ; Lim, Steven: A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. In: *IEEE Communications Surveys & Tutorials* 7 (2005), 2nd Quarter, Nr. 2, S. 72–93. – ISSN 1553-877X

[107] Lum, V. Y. ; Yuen, P. S. T. ; Dodd, M.: Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files. In: *Communications of the ACM* 14 (1971), April, Nr. 4, S. 228–239. – ISSN 0001-0782

[108] Luttenberger, Norbert ; Reuter, Florian ; Koberstein, Jochen: The XCast Approach for Content-based Flooding Control in Distributed Virtual Shared Information Spaces - Design and Evaluation. In: *Proceedings of the 1st European Workshop on Wireless Sensor Networks*, 2004

[109] Madden, Samuel R. ; Franklin, Michael J. ; Hellerstein, Joseph M. ; Hong, Wei: TinyDB: An Acquisitional Query Processing System for Sensor Networks. In: *ACM Transactions on Database Systems (TODS)* 30 (2005), März, Nr. 1, S. 122–173. – ISSN 0362-5915

[110] Mangharam, Rahul ; Weller, Daniel S. ; Stancil, Daniel D. ; Rajkumar, Ragunathan: GrooveSim: A Topography-Accurate Simulator for Geographic Routing in Vehicular Networks. In: *Proceedings of the 2nd International Workshop on Vehicular Ad-hoc Networks (VANET)*. Cologne, Germany : ACM, 2005, S. 59–68. – ISBN 1-59593-141-4

[111] Mauve, Martin ; Widmer, Jörg ; Hartenstein, Hannes: A Survey on Position-Based Routing in Mobile Ad Hoc Networks. In: *IEEE Network Magazine* 15 (2001), Nr. 6, S. 30–39

[112] Mazur, Florian ; Chrobok, Roland ; Hafstein, Sigurdur F. ; Pottmeier, Andreas ; Schreckenberg, Michael: Future of Traffic Information – Online-Simulation of a Large Scale Freeway Network. In: *Proceedings of the IADIS International Conference WWW/Internet* Bd. 1. Madrid, Spain, Oktober 2004, S. 665–672

[113] Mentler, Tilo: *Umsetzung eines Kommunikationsmodells für Lego Mindstorms Fahrzeuge*. Studienarbeit, Institut für Telematik, Universität zu Lübeck. Januar 2008

[114] Messelodi, Stefano ; Modena, Carla M. ; Zanin, Michele ; De Natale, Francesco G. B. ; Granelli, Fabrizio ; Betterle, Enrico ; Guarise, Andrea: Intelligent extended floating car data collection. In: *Expert Systems with Applications: An International Journal* 36 (2009), Nr. 3, S. 4213–4227. – ISSN 0957-4174

[115] Mitzenmacher, Michael: Compressed Bloom Filters. In: *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. Newport, Rhode Island, United States, 2001, S. 144–150. – ISBN 1-58113-383-9

[116] Müller-Schloer, Christian ; Sick, Bernhard: Emergence in Organic Computing Systems: Discussion of a Controversial Concept. In: *Proceedings of the Third International Conference on Autonomic and Trusted Computing*. Wuhan, China, September 2006, S. 1–16. – ISBN 978-3-540-38619-3

[117] Müller-Schloer, Christian ; von der Malsburg, Christoph ; Würtz, Rolf P.: Organic Computing. Aktuelles Schlagwort. In: *Informatik Spektrum* 27 (2004), August, Nr. 4, S. 332–336

[118] Molla, Mohammad M. ; Ahamed, Sheikh I.: A Survey of Middleware for Sensor Network and Challenges. In: *Proceedings of the International Conference Workshops on Parallel Processing (ICPPW).* Melbourne, Australia : IEEE Computer Society, 2006, S. 223–228

[119] Moore, Gordon E.: Cramming more components onto integrated circuits. In: *Electronics* 19 (1965), April, Nr. 3, S. 114–117

[120] Morenz, Tino: *iTranSIM – Simulation based Vehicle Location*, University of Dublin, Diplomarbeit, 2007

[121] Morris, Robert ; Jannotti, John ; Kaashoek, Frans ; Li, Jinyang ; Decouto, Douglas: CarNet: A Scalable Ad Hoc Wireless Network System. In: *Proceedings of the 9th ACM SIGOPS European Workshop.* Kolding, Denmark : ACM, 2000, S. 61–65. – ISBN 1-23456-789-0

[122] Nadeem, Tamer ; Dashtinezhad, Sasan ; Liao, Chunyuan ; Iftode, Liviu: TrafficView: A Scalable Traffic Monitoring System. In: *Proceedings of IEEE International Conference on Mobile Data Management.* Los Alamitos, CA, USA : IEEE Computer Society, 2004, S. 13–26. – ISBN 0-7695-2070-7

[123] Nagel, Kai ; Schreckenberg, Michael: A cellular automaton model for freeway traffic. In: *Journal de Physique I* 2 (1992), Dezember, S. 2221–2229

[124] Nathan, Andre ; Barbosa, Valmir C.: V-like Formations in Flocks of Artificial Birds. In: *Artificial Life* 14 (2008), März, Nr. 2, S. 179–188. – ISSN 1064-5462

[125] National Marine Electronics Association: *NMEA 0183 Interface Standard Version 3.01.* 2002. – URL `http://www.nmea.org`

[126] Naumov, Valery ; Baumann, Rainer ; Gross, Thomas: An Evaluation of Inter-Vehicle Ad Hoc Networks Based on Realistic Vehicular Traces. In: *Proceedings of the 7th ACM International Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc).* Florence, Italy : ACM Press, 2006, S. 108–119. – ISBN 1-59593-368-9

[127] Neumann, John von: *Theory and Organization of Complicated Automata.* Bd. Part One. Kap. Theory and Organization of Complicated Automata, S. 29–87. In: Burks, Arthur W. (Hrsg.): *Theory of Self-reproducing automata* Bd. Part One, University of Illinois Press, 1966. – ISBN 0-598-37798-0

[128] Nikoletseas, Sotiris ; Spirakis, Paul: Efficient Information Propagation Algorithms in Smart Dust and NanoPeer Networks. In: *Proceedings of the IST/FET*

International Workshop on Global Computing. Rovereto, Italy : Springer, März 2004, S. 127–145

[129] Nommensen, Sönke N.: *Erkennung von Verkehrsstrukturen mithilfe von Hovering Data Clouds*. Masterarbeit, Institut für Telematik, Universität zu Lübeck. Dezember 2006

[130] NXP Semiconductors: *I^2C-bus specification and user manual*. Revision 03, Juni 2007. – URL `http://www.nxp.com/acrobat/usermanuals/UM10204_3.pdf`

[131] Ou, Chia-Ho ; Ssu, Kuo-Feng ; Jiau, Hewijin C.: Connecting Network Partitions with Location-Assisted Forwarding Nodes in Mobile Ad Hoc Environments. In: *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*. Los Alamitos, CA, USA : IEEE Computer Society, 2004, S. 239–247. – ISBN 0-7695-2076-6

[132] Papadimitratos, Panagiotis ; Buttyan, Levente ; Holczer, Tamás ; Schoch, Elmar ; Freudiger, Julien ; Raya, Maxim ; Ma, Zhendong ; Kargl, Frank ; Kung, Antonio ; Hubaux, Jean-Pierre: Secure Vehicular Communication Systems: Design and Architecture. In: *IEEE Communications Magazine* 46 (2008), November, Nr. 11, S. 100–109

[133] Perkins, C. ; Belding-Royer, E. ; Das, S.: *Ad hoc On-Demand Distance Vector (AODV) Routing*. RFC 3561 (Experimental). Juli 2003. – URL `http://www.ietf.org/rfc/rfc3561.txt`

[134] Perkins, Charles E. ; Bhagwat, Pravin: Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In: *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*. London, United Kingdom : ACM, 1994, S. 234–244. – ISBN 0-89791-682-4

[135] Perkins, Charles E. ; Royer, Elizabeth M.: Ad-hoc On-Demand Distance Vector Routing. In: *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*. New Orleans, LA, USA, 1999, S. 90–100. – ISBN 0-7695-0025-0

[136] Peters, Karsten ; Johansson, Anders ; Helbing, Dirk: Swarm Intelligence beyond Stigmergy: Traffic Optimization in Ants. In: *Künstliche Intelligenz (KI)* 19 (2005), Nr. 4, S. 11–17. – ISSN 0933-1875

[137] Pfisterer, Dennis: *Comprehensive Development Support for Wireless Sensor Networks*, University of Lübeck, Dissertation, Oktober 2007

[138] Pfisterer, Dennis ; Buschmann, Carsten: Coalescing simulation and embedded WSN application development. In: *Proceedings of the International Workshop on Sensor Network Engineering (IWSNE)*. Santorini Island, Greece, Juni 2008

[139] Pfisterer, Dennis ; Buschmann, Carsten ; Hellbrück, Horst ; Fischer, Stefan: Data-type centric middleware synthesis for wireless sensor network application development. In: *Proceedings of the 5th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, 2006

[140] Pfisterer, Dennis ; Fischer, Stefan ; Kröller, Alexander ; Fekete, Sandor: Shawn: Ein alternativer Ansatz zur Simulation von Sensornetzwerken. In: *Tagungsband des 4. Fachgesprächs „Drahtlose Sensornetze“ der GI/ITG-Fachgruppe „Kommunikation und Verteilte Systeme“*. Zürich, Schweiz, März 2005

[141] Piórkowski, Michal ; Raya, Maxim ; Lugo, Ada ; Papadimitratos, Panos ; Grossglauser, Matthias ; Hubaux, Jean-Pierre: TraNS: Realistic Joint Traffic and Network Simulator for VANETs. In: *SIGMOBILE Mobile Computing and Communications Review* 12 (2008), Januar, Nr. 1, S. 31–33. – ISSN 1559-1662

[142] Postel, J.: *Internet Protocol*. RFC 791 (Standard). September 1981 (Request for Comments). – URL `http://www.ietf.org/rfc/rfc791.txt`. – Updated by RFC 1349

[143] Rappaport, Theodore S.: *Wireless Communications: Principles and Practice*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2001. – ISBN 978-0-13-042232-0

[144] Ratnasamy, Sylvia ; Karp, Brad ; Shenker, Scott ; Estrin, Deborah ; Govindan, Ramesh ; Yin, Li ; Yu, Fang: Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table. In: *Mobile Networks and Applications* 8 (2003), August, Nr. 4, S. 427–442. – ISSN 1383-469X

[145] Röbke-Doerr, Peter: Am Stau vorbei. Die Unterschiede zwischen TMC und TMCpro. In: *c't* 13 (2006), S. 218

[146] Rechenberg, Ingo: *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973. – 176 S. – ISBN 3-7728-0374-1

[147] Reichardt, D. ; Miglietta, M. ; Moretti, L. ; Morsink, P. ; Schulz, W.: CarTALK 2000: Safe and Comfortable Driving based upon Inter-Vehicle-Communication. In: *Proceedings of the IEEE Intelligent Vehicle Symposium (IV)* Bd. 2, Juni 2002, S. 545–550

[148] Richter, Andreas: *Geschwindigkeitsvorgabe an Lichtsignalanlagen*, Helmut-Schmidt-Universität / Universität der Bundeswehr, Dissertation, 2004

[149] Richter, Urban ; Mnif, Moez ; Branke, Jürgen ; Müller-Schloer, Christian ; Schmeck, Hartmut: Towards a generic observer/controller architecture for Organic Computing. In: *Tagungsband der GI-Jahrestagung (Informatik)* Bd. 1. Dresden, Germany, Oktober 2006, S. 112–119. – ISBN 973-3-88579-187-4

[150] Römer, Kay: Programming Paradigms and Middleware for Sensor Networks. In: *Tagungsband der GI/ITG Fachgespräche „Sensornetze“*. Karlsruhe, Germany, Februar 2004

[151] Saha, Amit K. ; Johnson, David B.: Modeling Mobility for Vehicular Ad Hoc Networks. In: *Proceedings of the 1st ACM International Workshop on Vehicular Ad-hoc Networks (VANET)*. Philadelphia, PA, USA : ACM, 2004, S. 91–92. – ISBN 1-58113-922-5

[152] Salzmann, Jakob ; Behnke, Ralf ; Timmermann, Dirk: A Self-Organized Localization-Free Clustering Approach for Redundancy Exploitation in Large Wireless Sensor Networks. In: *Tagungsband der GI-Jahrestagung (Informatik)*. München, Germany, September 2008, S. 747–754. – ISBN 978-3-88579-228-4

[153] Sasson, Yoav ; Cavin, David ; Schiper, André: Probabilistic Broadcast for Flooding in Wireless Mobile Ad hoc Networks. In: *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, 2003

[154] Schaller, Robert R.: Moore's law: past, present and future. In: *IEEE Spectrum* 34 (1997), Juni, Nr. 6, S. 52–59. – ISSN 0018-9235

[155] Schmeck, Hartmut: Organic Computing – Vision and Challenge for System Design. In: *Proceedings of the International Conference on Parallel Computing in Electrical Engineering*. Los Alamitos, CA, USA : IEEE Computer Society, 2004, S. 3. – ISBN 978-0-7695-2080-3

[156] Schmeck, Hartmut: Organic Computing - A New Vision for Distributed Embedded Systems. In: *Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*. Los Alamitos, CA, USA : IEEE Computer Society, 2005, S. 201–203. – ISBN 0-7695-2356-0

[157] Schmidt, Tobias ; Schlender, Dirk: Untersuchung zum saisonalen Reifenwechsel unter Berücksichtigung technischer und klimatischer Aspekte / Bergische Universität Wuppertal. 2003. – Forschungsbericht

[158] Schreckenberg, Michael ; Pottmeier, Andreas ; Hafstein, Sigurdur F. ; Chrobok, Roland ; Wahle, Joachim: *Simulation Approaches in Transportation Analysis – Recent Advances and Challenges*. Bd. 31. Kap. Simulation of the Autobahn Traffic in North Rhine-Westphalia, S. 205–233, Springer US, 2005

[159] Schulze, Matthias ; Mäkinen, Tapani ; Irion, Joachim ; Flament, Maxime ; Kessel, Tanja: *Preventive and Active Safety Applications Integrated Project PReVENT – Final Report*. Mai 2008

[160] Shenker, Scott ; Ratnasamy, Sylvia ; Karp, Brad ; Govindan, Ramesh ; Estrin, Deborah: Data-Centric Storage in Sensornets. In: *ACM SIGCOMM Computer Communication Review* 33 (2003), Januar, Nr. 1, S. 137–142. – ISSN 0146-4833

[161] Sikorski, Zbigniew: *Umsetzung realistischer Fahrzeugbewegung mit Lego Mindstorms Fahrzeugen*. Studienarbeit, Institut für Telematik, Universität zu Lübeck. November 2007

[162] Sikorski, Zbigniew: *Demonstrator für dynamische Verkehrsführung mittels Car-to-Car Kommunikation*. Diplomarbeit, Institut für Telematik, Universität zu Lübeck. Dezember 2008

[163] Singh, J. P. ; Bambos, N. ; Srinivasan, B. ; Clawin, D.: Wireless LAN performance under varied stress conditions in vehicular traffic scenarios. In: *Proceedings of the 56th IEEE Vehicular Technology Conference (VTC2002-Fall)* Bd. 2, 2002, S. 743–747. – ISSN 1090-3038

[164] Sommer, Christoph ; Yao, Zheng ; German, Reinhard ; Dressler, Falko: On the Need for Bidirectional Coupling of Road Traffic Microsimulation and Network Simulation. In: *9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM Mobihoc 2008): 1st ACM International Workshop on Mobility Models for Networking Research (MobilityModels'08)*. Hong Kong, China : ACM, May 2008

[165] Sommer, Christoph ; Yao, Zheng ; German, Reinhard ; Dressler, Falko: Simulating the Influence of IVC on Road Traffic using Bidirectionally Coupled Simulators. In: *27th IEEE Conference on Computer Communications (IEEE INFOCOM 2008): Mobile Networking for Vehicular Environments (MOVE 2008)*. Phoenix, AZ, USA : IEEE, April 2008

[166] Stibor, Lothar ; Zang, Yunpeng ; Reumerman, Hans-Jürgen: Neighborhood Evaluation of Vehicular Ad-Hoc Network Using IEEE 802.11p / RWTH Aachen, Germany. April 2007. – Forschungsbericht

[167] Suntum, Ulrich van ; Hartwig, Karl-Hans ; Holznagel, Bernd ; Ströbele, Wolfgang ; Armbrecht, Henrik ; Deckers, Sebastian ; Uhde, Nicole ; Westermeier, Andreas: *Bedeutung der Infrastrukturen im internationalen Standortwettbewerb und ihre Lage in Deutschland - Gutachten im Auftrag des Bundesverbandes der Deutschen Industrie (BDI).* Mai 2008

[168] Tanenbaum, Andrew S. ; Steen, Marten van: *Verteilte Systeme: Grundlagen und Paradigmen.* Pearson Studium, 2003. – ISBN 3-8273-7057-4

[169] Tauschek, Stefan: Car Talk – Fahrzeuge kommunizieren. In: *Automobil-Elektronik* 02 (2006), April, S. 18–21

[170] Terfloth, Kirsten ; Wittenburg, Georg ; Schiller, Jochen: FACTS - A Rule-Based Middleware Architecture for Wireless Sensor Networks. In: *Proceedings of the 1st IEEE International Conference on Communication System Software and Middleware (COMSWARE)*, 2006

[171] Teubler, Torsten: *Implementierung und Validierung des Daten-Verteilungs-Protokolls AutoCast in Shawn.* Studienarbeit, Institut für Telematik, Universität zu Lübeck. Januar 2008

[172] Toh, Chai-Keong: *Ad Hoc Mobile Wireless Networks: Protocols and Systems.* Prentice Hall PTR, 2002. – 302 S. – ISBN 0-13-007817-4

[173] Torrent-Moreno, Marc: *Inter-Vehicle Communications: Achieving Safety in a Distributed Wireless Environment: Challenges, Systems and Protocols*, Universität Karlsruhe, Dissertation, 2007

[174] Torrent-Moreno, Marc: Inter-Vehicle Communications: Assessing Information Dissemination under Safety Constraints. In: *Proceedings of the 4th Annual IEEE/IFIP Conference on Wireless on Demand Network Systems and Services (WONS).* Obergurgl, Austria, Januar 2007, S. 59–64

[175] *TraceExporter: Exports SUMO dumps as ns2 traces.* – URL `http://apps.sourceforge.net/mediawiki/sumo/index.php?title=TraceExporter`

[176] Treiber, Martin ; Helbing, Dirk: Microsimulations of Freeway Traffic Including Control Measures. In: *AUTOMATISIERUNGSTECHNIK* 49 (2001), S. 478

[177] Tseng, Yu-Chee ; Ni, Sze-Yao ; Chen, Yuh-Shyan ; Sheu, Jang-Ping: The Broadcast Storm Problem in a Mobile Ad Hoc Network. In: *Wireless Networks* 8 (2002), März, Nr. 2/3, S. 153–167. – ISSN 1022-0038

[178] Varga, András: The OMNeT++ Discrete Event Simulation System. In: *Proceedings of the European Simulation Multiconference (ESM)*. Prague, Czech Republic, Juni 2001

[179] VDE/ITG/GI-Arbeitsgruppe Organic Computing: *Positionspapier: Organic Computing – Computer- und Systemarchitektur im Jahr 2010*. Juli 2003

[180] Viswanath, Kumar ; Obrazcka, Katia: An Adaptive Approach to Group Communications in Multi Hop Ad Hoc Networks. In: *Proceedings of the 7th International Symposium on Computers and Communications (ISCC)*. Washington, DC, USA : IEEE Computer Society, 2002, S. 559–566. – ISBN 0-7695-1671-8

[181] Wang, S.Y. ; Chou, C.L. ; Chiu, Y.H. ; Tseng, Y.S. ; Hsu, M.S. ; Cheng, Y.W. ; Liu, W.L. ; Ho, T.W.: NCTUns 4.0: An Integrated Simulation Platform for Vehicular Traffic, Communication, and Network Researches. In: *Proceedings of the 1st IEEE International Symposium on Wireless Vehicular Communications (WiVec)*. Baltimore, MD, USA, 2007, S. 2081–2085. – ISBN 978-1-4244-0263-2

[182] Wang, Ziyuan ; Kulik, Lars ; Ramamohanarao, Kotagiri: Proactive traffic merging strategies for sensor-enabled cars. In: *Proceedings of the 4th ACM International Workshop on Vehicular Ad-hoc Networks (VANET)*. New York, NY, USA : ACM, 2007, S. 39–48. – ISBN 978-1-59593-739-1

[183] Wedde, Horst F. ; Lehnhoff, Sebastian ; Bonn, Bernhard van ; Bay, Z. ; Becker, S. ; Böttcher, S. ; Brunner, C. ; Büscher, A. ; Fürst, T. ; Lazarescu, A.M. ; Rotaru, E. ; Senge, S. ; Steinbach, B. ; Yilmaz, F. ; Zimmermann, T.: Highly Dynamic and Adaptive Traffic Congestion Avoidance in Real-Time Inspired by Honey Bee Behavior. In: *Informatik Aktuell – Mobilität und Echtzeit*. Boppard, Germany, Dezember 2008, S. 21–31

[184] Wegener, Axel ; Hellbrück, Horst ; Fischer, Stefan ; Hendriks, Björn ; Schmidt, Christiane ; Fekete, Sándor P.: Designing a Decentralized Traffic Information System – AutoNomos. In: *Proceedings of the 16th ITG/GI - Fachtagung Kommunikation in Verteilten Systemen (KiVS)*. Kassel, Germany, März 2009, S. 309–315. – ISBN 978-3-540-92665-8

[185] Wegener, Axel ; Hellbrück, Horst ; Fischer, Stefan ; Schmidt, Christiane ; Fekete, Sándor P.: AutoCast: An Adaptive Data Dissemination Protocol for Traffic Information Systems. In: *Proceedings of the 66th IEEE Vehicular Technology Conference (VTC2007-Fall)*. Baltimore, USA, Oktober 2007, S. 1947–1951

[186] Wegener, Axel ; Hellbrück, Horst ; Wewetzer, Christian ; Lübke, Andreas: VANET Simulation Environment with Feedback Loop and its Application to

Traffic Light Assistance. In: *Proceedings of the 3rd IEEE Workshop on Automotive Networking and Applications*. New Orleans, LA, USA, Dezember 2008, S. 1–7. – ISBN 978-1-4244-3061-1

[187] Wegener, Axel ; Piórkowski, Michał ; Raya, Maxim ; Hellbrück, Horst ; Fischer, Stefan ; Hubaux, Jean-Pierre: TraCI: An Interface for Coupling Road Traffic and Network Simulators. In: *Proceedings of the 11th Communications and Networking Simulation Symposium (CNS)*. Ottawa, Canada, April 2008, S. 155–163. – ISBN 1-56555-318-7

[188] Wegener, Axel ; Schiller, Elad M. ; Hellbrück, Horst ; Fekete, Sándor P. ; Fischer, Stefan: Hovering Data Clouds: A Decentralized and Self-Organizing Information System. In: *Proceedings of the 1st International Workshop on Self-Organizing Systems*. Passau, Germany, September 2006, S. 243–247. – ISBN 978-3-540-37658-3

[189] Welch, Brent ; Jones, Ken: *Practical Programming in Tcl and Tk*. 4th Edition. Prentice Hall PTR, 2003. – ISBN 0-13-038560-3

[190] Williams, Brad ; Camp, Tracy: Comparison of broadcasting techniques for mobile ad hoc networks. In: *Proceedings of the 3rd ACM International Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc)*. Lausanne, Switzerland : ACM, 2002, S. 194–205. – ISBN 1-58113-501-7

[191] Wischhof, Lars ; Ebner, André ; Rohling, Hermann: Information dissemination in self-organizing intervehicle networks. In: *IEEE Transactions on Intelligent Transportation Systems* 6 (2005), März, Nr. 1, S. 30–101

[192] Wischhof, Lars ; Ebner, André ; Rohling, Hermann: Self-Organizing Traffic Information System based on Car-to-Car Communication: Prototype Implementation. In: *Proceedings of the 1st International Workshop on Intelligent Transportation (WIT)*. Hamburg, Germany, März 2004

[193] Wischhof, Lars ; Ebner, André ; Rohling, Hermann ; Lott, Matthias ; Halfmann, Rüdiger: Adaptive Broadcast for Travel and Traffic Information Distribution Based on Inter-Vehicle Communication. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, Juni 2003, S. 6–11

[194] Wischhof, Lars ; Ebner, André ; Rohling, Hermann ; Lott, Matthias ; Halfmann, Rüdiger: SOTIS - A Self-Organizing Traffic Information System. In: *Proceedings of the 57th IEEE Vehicular Technology Conference (VTC2003-Spring)*. Jeju, South Korea, April 2003

[195] Wischhof, Lars ; Rohling, Hermann: Congestion Control in Vehicular Ad Hoc Networks. In: *Proceedings of IEEE International Conference on Vehicular Electronics and Safety*. Xi'an, Shaanxi, China, Oktober 2005, S. 58–63

[196] Wu, Jie ; Yang, Shuhui ; Dai, Fei: Logarithmic Store-Carry-Forward Routing in Mobile Ad Hoc Networks. In: *IEEE Transactions on Parallel and Distributed Systems* 18 (2007), Nr. 6, S. 735–748. – ISSN 1045-9219

[197] Zhou, Gang ; He, Tian ; Krishnamurthy, Sudha ; Stankovic, John A.: Impact of Radio Irregularity on Wireless Sensor Networks. In: *Proceedings of the 2nd International Conference on Mobile systems, applications, and services (MobiSys)*. Boston, MA, USA : ACM, 2004, S. 125–138. – ISBN 1-58113-793-1

Persönliche Informationen

Lebenslauf

Persönliche Daten

Name	Axel Wegener
Geburtsdatum	17.06.1979
Geburtsort	Holzminden
Anschrift	Adolf-Ehrtmann-Str. 1 23564 Lübeck
Familienstand	Verheiratet, ein Kind

Ausbildung

1991 – 1998	Gymnasium Uslar (Abitur mit der Note 2,6)
07/1998 – 08/1999	Zivildienst beim DRK Krankentransport Northeim, Rettungswache Uslar; mit Ausbildung zum staatlich geprüften Rettungssanitäter
10/1999 – 06/2005	Studium der Informatik an der Technischen Universität Braunschweig mit Abschluss als Diplom-Informatiker. Hauptfächer Robotik, Verteilte Systeme und Chipentwurf, Nebenfach Mechatronik. Benotung der Diplomarbeit 1,0 und Gesamtnote 1,1

Beruflicher Werdegang

05/1995 – 12/2000	Nebentätigkeit bei Apollo, Licht- und Tontechnik (Veranstaltungstechnik)
seit 06/2000	Nebentätigkeit als Softwareentwickler / IT-Consultant bei der EDV-Unternehmensberatung SOL-Computer, Uslar
seit 07/2005	Wissenschaftlicher Mitarbeiter am Institut für Telematik der Universität zu Lübeck

Eigene Publikationen

[1] Wegener, Axel ; Schiller, Elad M. ; Hellbrück, Horst ; Fekete, Sándor P. ; Fischer, Stefan: Hovering Data Clouds: A Decentralized and Self-Organizing Information System. In: *Proceedings of the 1st International Workshop on Self-Organizing Systems*. Passau, Germany, September 2006, S. 243–247. – ISBN 978-3-540-37658-3

[2] Fekete, Sándor P. ; Schmidt, Christiane ; Wegener, Axel ; Fischer, Stefan: Hovering Data Clouds for Recognizing Traffic Jams. In: *Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA)*, 2006, S. 198–203

[3] Wegener, Axel ; Hellbrück, Horst ; Fischer, Stefan ; Schmidt, Christiane ; Fekete, Sándor P. : AutoCast: An Adaptive Data Dissemination Protocol for Traffic Information Systems. In: *Proceedings of the 66th IEEE Vehicular Technology Conference Fall 2007 (VTC2007-Fall)*. Baltimore, USA, Oktober 2007, S. 1947–1951

[4] Lipphardt, Martin ; Hellbrück, Horst ; Wegener, Axel ; Fischer, Stefan: GRAPE - Gradient based Routing for All PurposE. In: *Proceedings of the 4th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*. Pisa, Italy, Oktober 2007, S. 1–6. – ISBN 978-1-4244-1455-0

[5] Wegener, Axel ; Piórkowski, Michał ; Raya, Maxim ; Hellbrück, Horst ; Fischer, Stefan ; Hubaux, Jean-Pierre: TraCI: An Interface for Coupling Road Traffic and Network Simulators. In: *Proceedings of the 11th Communications and Networking Simulation Symposium (CNS'08)*. Ottawa, Canada, April 2008, S. 155–163. – ISBN 1-56555-318-7

[6] Hellbrück, Horst ; Wegener, Axel ; Fischer, Stefan: AutoCast: A General-Purpose Data Dissemination Protocol and its Application in Vehicular Networks. In: *Ad Hoc & Sensor Wireless Networks journal (AHSWN)* 6 (2008), Nr. 1–2, S. 91–122. – ISSN 1551-9899

[7] Wegener, Axel ; Hellbrück, Horst ; Wewetzer, Christian ; Lübke, Andreas: VANET Simulation Environment with Feedback Loop and its Application to

Traffic Light Assistance. In: *Proceedings of the 3rd IEEE Workshop on Automotive Networking and Applications*. New Orleans, LA, USA, Dezember 2008, S. 1–7. – ISBN 978-1-4244-3061-1

[8] Fekete, Sándor P. ; Tessars, Christopher ; Schmidt, Christiane ; Wegener, Axel ; Fischer, Stefan ; Hellbrück, Horst: *Verfahren und Vorrichtung zur Ermittlung einer Fahrstrategie*. 2008. – Patentanmeldung, Aktenzeichen UNV-TUB-102-DE

[9] Wegener, Axel ; Hellbrück, Horst ; Fischer, Stefan ; Hendriks, Björn ; Schmidt, Christiane ; Fekete, Sándor P.: Designing a Decentralized Traffic Information System – AutoNomos. In: *Proceedings of the 16th ITG/GI - Fachtagung Kommunikation in Verteilten Systemen (KiVS)*. Kassel, Germany, März 2009, S. 309–315. – ISBN 978-3-540-92665-8

[10] Hellbrück, Horst ; Wegener, Axel ; Cao, Junjian ; Zuo, Tian: Fast Prototyping for VANET Applications with PDAs. In: *Proceedings of the 1st International Conference on Wireless Communication Society, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE)*. Aalborg, Denmark, Mai 2009