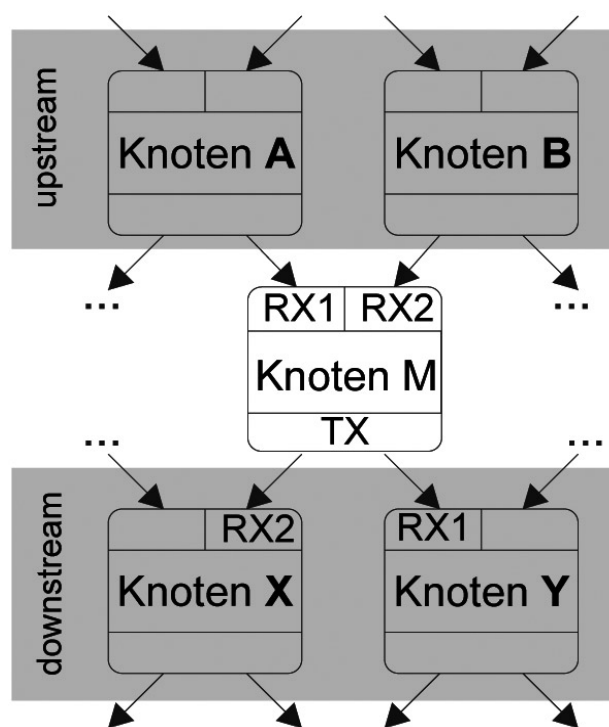


Philipp Nenninger

Vernetzung verteilter sicherheitsrelevanter Systeme im Kraftfahrzeug

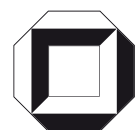


Philipp Nenninger

**Vernetzung verteilter sicherheitsrelevanter Systeme im
Kraftfahrzeug**

Vernetzung verteilter sicherheits- relevanter Systeme im Kraftfahrzeug

von
Philipp Nenninger



universitätsverlag karlsruhe

Dissertation, Universität Karlsruhe (TH)
Fakultät für Elektrotechnik und Informationstechnik, 2007

Impressum

Universitätsverlag Karlsruhe
c/o Universitätsbibliothek
Straße am Forum 2
D-76131 Karlsruhe
www.uvka.de



Dieses Werk ist unter folgender Creative Commons-Lizenz
lizenziert: <http://creativecommons.org/licenses/by-nc-nd/2.0/de/>

Universitätsverlag Karlsruhe 2007
Print on Demand

ISBN: 978-3-86644-134-7

Vernetzung verteilter sicherheitsrelevanter Systeme im Kraftfahrzeug

Zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS

von der Fakultät für

Elektrotechnik und Informationstechnik

an der Universität Karlsruhe (TH)

genehmigte

DISSERTATION

von

Dipl.-Ing. Philipp Nenninger

aus Lahr/Schwarzwald

Tag der mündl. Prüfung: 20. Februar 2007
Hauptreferent: Prof. Dr.-Ing. U. Kiencke, Universität Karlsruhe (TH)
Korreferent: Prof. Dr.-Ing. J. Becker, Universität Karlsruhe (TH)

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Angestellter am Institut für industrielle Informationstechnik (IIIT) der Universität Karlsruhe. Ich bedanke mich bei Herrn Professor Kiencke, dem Leiter des IIIT, für die Initiierung dieser Arbeit sowie für die wertvollen Anregungen, aus denen nicht zu letzt auch SafeNet hervorging. Herrn Professor Becker möchte ich für sein Interesse an der Arbeit und die Übernahme des Korreferats danken.

Eine Arbeit wie diese entsteht nicht im fachlichen Vakuum. Ich möchte mich bei allen aktuellen und ehemaligen Kollegen am IIIT für die konstruktiven fachlichen Diskussionen und den regen Austausch von Ideen bedanken. Ganz besonderer Dank gilt hier Marko Babic, Jörg Barrho, Stephan Brummund und Benedikt Merz für die sorgfältige fachliche Korrektur der Arbeit sowie Timo Kistner, der mit der von ihm Projekt entwickelten FPGA-Platine maßgeblich zum Gelingen des SafeNet-Projekts beigetragen hat. Auch meinen Studenten, die mit ihrer engagierten Arbeit wertvolle Beiträge zu dieser Arbeit geliefert haben, möchte ich danken.

Eine Arbeit wie diese entsteht aber auch nicht im sozialen Vakuum. Auch hier möchte ich meinen Kollegen und den Mitarbeitern des Instituts danken, diesmal dafür, dass sie meine Marotten vier Jahre lang (meist) klaglos aushielten. Ohne meine Familie und besonders meine Frau Kanako, deren Rücksicht und Unterstützung ich oft über Gebühr strapaziert habe, wäre diese Arbeit wohl nach diesem Vorwort zu Ende. Ein angemessener Dank würde den Rest des Buches füllen.

Alles Leben ist Problemlösen. Alle Organismen sind Erfinder und Techniker, gute oder weniger gute, erfolgreich oder weniger erfolgreich im Lösen von technischen Problemen.

Karl R. Popper

Karlsruhe, im Februar 2007

Philipp Nenninger

Inhaltsverzeichnis

1	Einleitung	1
1.1	Stand der Technik	5
1.2	Gliederung der Arbeit	6
2	Kommunikation verteilter Automobilanwendungen	7
2.1	Grundlagen fehlertoleranter Echtzeitsysteme	7
2.1.1	Klassifizierung von Ausfällen, Fehlern und Fehlerursachen .	9
2.1.2	Fehlerkapselung und Redundanzstrukturen	13
2.2	Anforderungen an sicherheitsrelevante Automobilelektronik	20
2.2.1	Vergleich von fly-by-wire und drive-by-wire Systemen	20
2.2.2	Umsetzung der Anforderungen für Automobilanwendungen	25
2.2.3	Kommunikation in verteilten Elektroniksystemen	29
2.3	Grundlagen zu Kommunikationsmedien	30
2.3.1	Nachrichtenübertragung	30
2.3.2	Kanalcodierung	33
2.4	Stand der Technik	43
2.4.1	CAN	43
2.4.2	FlexRay	48
2.4.3	Systeme aus anderen Anwendungsbereichen	50
2.4.4	Zusammenfassende Betrachtung	59
3	Netzwerkbasiertes Kommunikationsmedium (<i>SafeNet</i>)	61
3.1	Zielsetzung	61
3.2	Funktionsprinzip	63
3.3	Netzwerktopologie	63
3.4	Verteilte Algorithmen	69
3.4.1	Sendealgorithmus	69
3.4.2	Fehlererkennung	71
3.5	Modellierung mit Warteschlangen	74
3.6	Erweiterung des lokalen Sendealgorithmus	78
3.7	Analyse der Fehlerursachen und -auswirkungen	89
3.7.1	Ausfall einer Verbindung	90
3.7.2	Passiver Ausfall eines Knotens	91
3.7.3	Byzantinischer Fehler	93

Inhaltsverzeichnis

3.7.4	Fehlerhafte Nachricht	95
3.7.5	Babbling Idiot	95
3.7.6	Masquerading	96
3.7.7	Unterschlagen einer Nachricht	97
3.7.8	Ungültige Header-Daten	97
3.7.9	Ungültige Nachricht	98
3.7.10	Nachrichtenformat	98
3.7.11	Kanalcodierung in SafeNet	101
3.8	Fehlerbehandlungsstrategien	103
3.9	Aufbau eines Knotens	107
4	Spezifikation	111
4.1	Topologie	111
4.2	Nachrichtenformat	111
4.3	Fehlerbehandlung	113
4.3.1	Syntaktisch inkorrekte Nachricht	114
4.3.2	Ausfall einer Verbindung	115
4.3.3	Passiver Ausfall eines Knotens	115
4.3.4	Babbling Idiot	116
4.3.5	Byzantinischer Fehler	116
4.3.6	Versenden einer kontextuell falschen Nachricht	119
4.3.7	Masquerading Fehler	119
4.3.8	Tabelle der Fehlercodes	121
4.4	Bitübertragungsschicht	121
4.5	Aufbau eines SafeNet-Knotens	122
4.5.1	Globale Variablen	123
4.5.2	Lokale Variablen in der Operation Control	124
4.6	Controller-Host Interface (CHI)	125
4.7	DECODE	125
4.8	ENCODE	127
4.9	Operation Control (OC)	128
4.10	Dynamischer Start-up	136
4.10.1	Verifizieren der vorliegenden Daten	136
4.10.2	Vollständig dynamischer Start-up	137
4.10.3	Bewertung des dynamischen Start-ups	138
5	Simulation und Implementierung	139
5.1	Simulation	139
5.1.1	Modell	139
5.1.2	Simulationsergebnisse	140
5.2	Implementierung	149

5.3	Vergleichende Betrachtung	151
5.3.1	Netzwerktopologie	152
5.3.2	Datenrate	157
5.3.3	Fehlerbehandlung	166
6	Zusammenfassung und Ausblick	181
A	Erweiterung des Systemkonzepts auf dynamische Verteilung	185
A.1	Der <i>State-Server</i> -Ansatz	185
A.2	Implementierung und Verifikation	190
A.3	State-Server in SafeNet	193
B	Weitere Simulationsergebnisse	197
B.1	Leitertopolgie mit sechs Knoten	197
B.2	Chordaler Ring mit zwölf Knoten, Skiplänge vier	200
B.3	Chordaler Ring mit zwölf Knoten in sechs Paaren	203
C	Nomenklatur	207
C.1	Formelzeichen	207
C.2	Abkürzungen	209
	Literatur	211
	Lebenslauf	225

Inhaltsverzeichnis

1 Einleitung

Elektronik wird ein immer wichtigerer Bestandteil von Kraftfahrzeugen. War der Anteil der Kosten für die elektronische Ausstattung bei der Einführung des mikroprozessorgesteuerten ABS (Anti-Blockier-System) 1984 noch vernachlässigbar, so betrug er 2002 bereits 23 % der Gesamtkosten. Es wird erwartet, dass dieser Trend weiter anhält und bei einer jährlichen Wachstumsrate von circa 8 % bei einem Anteil an den Gesamtkosten von etwa 35 % im Jahre 2010 Sättigungseffekte eintreten [SV05]. Experten schätzen den Anteil der Innovationen im Automobilbereich, der durch Elektronik ermöglicht oder getrieben wird auf 80 % bis 90 % [LH02], [Heb05].

Die Gründe hierfür sind vielfältig und lassen sich in zwei Hauptgruppen unterteilen. Von Seiten der Automobilhersteller und deren Zulieferern kommen die Angebotsfaktoren, die durch Fortschritte in Forschung und Entwicklung ermöglicht werden. Typisch für diese Produkte ist es, dass sie oft im täglichen Betrieb nicht oder nur geringfügig wahrgenommen werden, wie zum Beispiel ABS oder der Airbag. Die Forderungsfaktoren, also Anforderungen, die von außen an die Automobilhersteller gestellt werden, haben ihren Ursprung oft in neuer Gesetzgebung. Beispielhaft sind hier die Abgasregelungen zu nennen, die eine treibende Kraft hinter der Entwicklung moderner Motorelektronik ist. So wurden die Kraftfahrzeugemissionen bezogen auf CO, HC und NOx seit 1970 vom Kunden fast unbemerkt um 99 % verringert [SV05]. Aber auch Kundenwünsche tragen zur Entwicklung bei. Diese Innovationen liegen oft im Komfortbereich, also bei Unterhaltungs- und Navigationssystemen, die im Vergleich zu ABS und vergleichbaren Systemen einen laxen Prozess für die Zulassung im Kraftfahrzeug durchlaufen müssen.

Große Chancen für den Einsatz von Elektronik im Kraftfahrzeug liegen jedoch in einem anderen, nämlich dem sicherheitsrelevanten Bereich. Systeme, die außerhalb des Einflussbereichs normaler Fahrer agieren, eine manuelle ABS-Bremse ist zum Beispiel für die meisten Fahrer aufgrund der geforderten Reaktionszeit nicht zu bewerkstelligen, haben sich längst von teuren Zusatz- zu Standardfunktionen gewandelt. Der positive Effekt ist an den in Deutschland im Straßenverkehr getöteten Personen deutlich zu erkennen, siehe auch Abbildung 1.1.

Die Halbierung innerhalb der letzten 20 Jahre ist natürlich nicht ausschließlich auf Fortschritte in der Automobilelektronik zurückzuführen, da in diese

1 Einleitung

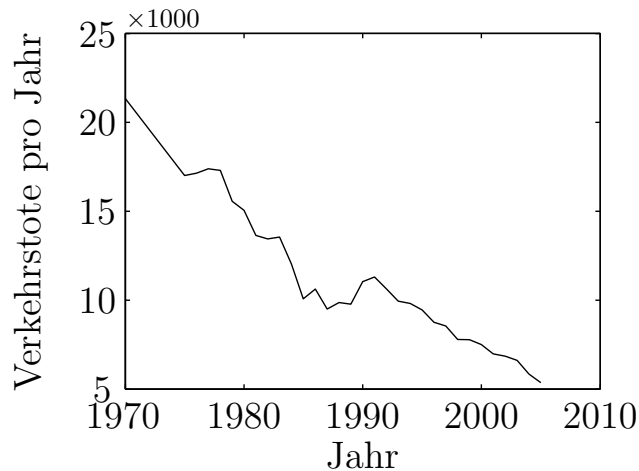


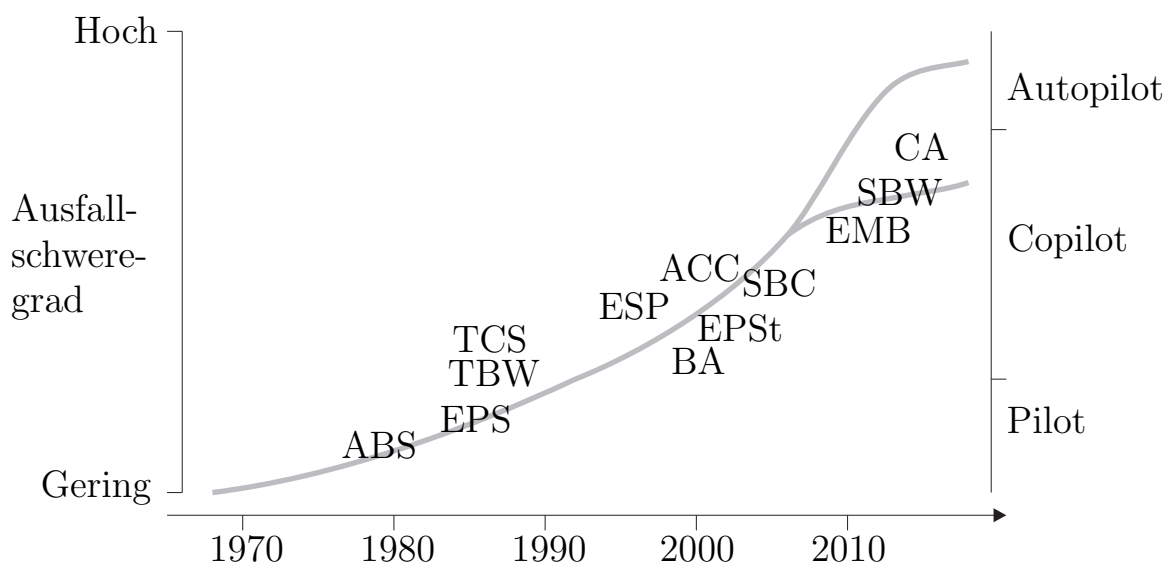
Abbildung 1.1: Anzahl der Verkehrstoten in Deutschland pro Jahr (bis 1990 BRD [Sta06]).

Zeit unter anderem auch die Einführung der Gurtpflicht fällt. Es ist aber davon auszugehen, dass elektronische Sicherheitssysteme durchaus einen Anteil an dieser positiven Entwicklung haben. Trotz der Unterstützung des Fahrers bei schnellen Entscheidungen, zum Beispiel durch ESP (electronic stability program) oder Pre-Safe [ATZ05] und seine Entlastung durch Funktionen wie Klimaautomatik und Navigationssysteme werden heutzutage noch immer circa 97% aller Unfälle durch den Fahrer verursacht [Spi04], siehe auch [SM97]. Vergleicht man Unfälle mit Personenschaden, die durch technische Mängel hervorgerufen werden, mit denen, die durch Fahrerfehler verursacht werden, so erhält man ein Verhältnis von circa 100:0.8 [Sta06].

Es ist also deutlich, dass Potential besteht, den Straßenverkehr durch den Einsatz von Fahrerassistenzsystemen sicherer zu machen. Um effektiv arbeiten zu können, müssen diese Systeme jedoch direkt in die Fahrzeugdynamik eingreifen können. Hier sind so genannte X-by-wire Fahrzeuge, bei denen keine direkte Verbindung zwischen Fahrer und Aktoren existiert im Vorteil, da es hier ohne Umwege möglich ist, die Fahrervorgabe in physikalische Stellgrößen umzuwandeln. Diese Systeme sind, in Vorgriff auf Abschnitt 2.2, sicherheitsrelevant.

Sicherheitsrelevante Automobilelektronik findet sich in aktuellen Fahrzeugen in Form von Systemen, bei deren Fehlfunktion lediglich eine Zusatzfunktion ausfällt, die Grundfunktion aber nicht beeinträchtigt wird. Als Beispiele sind hier ESP und aktive Lenkung [KPF⁺03] zu nennen. Typischerweise sind diese Systeme aus nicht-redundant ausgelegten Steuergeräten mit mechanischer Rückfallebene aufgebaut. Ein weit fortgeschrittenes System dieser Art ist SBC (Sensotronic Brake Control) von Bosch [Bos04]. Dieses akti-

ve Bremssteuerungssystem, welches von Mercedes-Benz in der W211-Serie¹ verbaut wurde, nimmt die Stellung der Bremspedals sowohl elektronisch als auch hydraulisch auf. Der elektronische Messwert dient dabei als Grundlage für Zusatzfunktionen wie Anfahrassistent und Trockenbremsen, wobei bei Nässe durch regelmäßiges leichtes Bremsen die Bremsscheiben und -beläge trocken gehalten werden. Bei einer Fehlfunktion wie einer Unterbrechung der Stromversorgung stellen die Trennventile, die im Normalfall den elektronisch geregelten hydraulischen Kreis mit den Bremsen verbinden und den direkt vom Fahrer betätigten Kreis trennen, einen direkten hydraulischen Durchgriff zu den Bremsen an den Vorderrädern her. Hierdurch ist der sichere Betrieb der Grundfunktion gewährleistet.



ABS	Anti-Blockier-System	EPSt	Elektrische Servolenkung
ACC	Adaptive Cruise Control	ESP	Electronic Stability Program
BA	Bremsassistent	SBC	Sensotronic Brake Control
CA	Collision Avoidance	SBW	Steer-by-Wire
EMB	Elektromechanisches Bremssystem	TBW	Throttle-by-Wire
EPS	Elektropneumatische Schaltung	TCS	Traction Control System

Abbildung 1.2: Entwicklung der sicherheitsrelevanten Automobilelektronik im Überblick (mit Veränderungen aus [ISS02])

An diesem Beispiel lassen sich zwei zentrale Punkte zum Stand der Technik verdeutlichen. Zum einen werden sicherheitsrelevante Funktionen in der Serie nicht ohne mechanische Rückfallebene verwendet. Die Gründe hierfür

¹Aufgrund von Qualitätsproblemen, die unter anderem einen Rückruf verursachten, wurde die SBC im Rahmen einer Modellpflege aus der Serie entfernt und durch ein System ersetzt, das ebenfalls elektronische Eingriffe in die Hydraulik zulässt.

1 Einleitung

liegen eher im juristischen Bereich und bei der Akzeptanz durch den Kunden [Spi04], da eine softwarebasierte Implementierung von Sicherheitsfunktionen technisch realisierbar ist, wie Erfahrungen aus der Luft- und Raumfahrt (*fly-by-wire*) und Forschungsergebnisse im Automobilbereich [Roo05] sowie mehrere Prototypen zeigen. Zum anderen werden Funktionen als möglichst abgeschlossene Systeme verbaut. Dies hat den Vorteil, dass Information meist lokal verarbeitet wird und die oft unzureichend vorhandene Ressource Netzwerkkapazität geschont wird, siehe auch Abschnitt 2.4. Andererseits stieg durch diesen Ansatz aber die Zahl der in einem Auto verbauten Steuergeräte auf über hundert, da jedes System auf separate Hardware aufbaut. Diese Geräte müssen, wenn auch wenig, so doch regelmäßig miteinander kommunizieren und sind daher über ein komplexes Netzwerk von Nachrichten miteinander verbunden.

Ein vielversprechender Ansatz zur Lösung der Komplexitätssteigerung ist die dynamische Verteilung von Funktionen zwischen den Steuergeräten. Hierdurch können nicht nur mehrere Funktionen auf einem Steuergerät vereinigt werden, es besteht auch die Möglichkeit die Art der ausgeführten Funktionen an die momentane Fahrsituation anzupassen und hierdurch Ressourcen einzusparen. So ist es denkbar, verschiedene Funktionen nur im Bedarfsfall auszuführen und die frei werdende Rechenzeit anderen Funktionen, zum Beispiel aus dem Komfortbereich, zur Verfügung zu stellen. Aus technischer Sicht scheitert dieser integrierte Ansatz momentan hauptsächlich an dem erhöhten Kommunikationsaufwand, der unter den im Kraftfahrzeug herrschenden Bedingungen nicht befriedigt werden kann.

In der vorliegenden Arbeit wird untersucht, welche Anforderungen das Einsatzgebiet Automobilelektronik an verteilte sicherheitsrelevante Systeme und speziell an die dynamische Verteilung von Funktionen stellt. Besonderes Augenmerk liegt dabei auf:

- der Vernetzung der einzelnen Teilsysteme untereinander und der Kommunikation zwischen den Geräten. Da die heute auf dem Markt beziehungsweise in Entwicklung befindlichen Kommunikationssysteme aus unterschiedlichen Gründen für den Einsatz in sicherheitsrelevanter Automobilelektronik nur eingeschränkt geeignet sind, wird ein hochdatenratiger netzwerkbasierter Kommunikationsansatz (*SafeNet*²) vorgestellt.
- Möglichkeiten und Rahmenbedingungen, die sich durch die dynamische Verteilung, die eine Änderung des laufenden Systems mit sich bringt, ergeben. Hierzu wird der *State-Server*-Ansatz vorgestellt, der eine Umkonfigurierung von Systemen im laufenden Betrieb ermöglicht.

²Der Begriff *SafeNet* ist markenrechtlich geschützt. Das hier vorgestellte System steht zu dem geschützten in keinerlei Relation.

1.1 Stand der Technik

Ziel dieses Abschnittes ist es, einen Überblick über den Stand der Technik im Bereich der dynamischen Verteilung sicherheitsrelevanter Anwendungen zu geben. Da die Bewertung des netzwerkbasierten Kommunikationsmediums SafeNet einen detaillierten Vergleich mit existierenden Lösungen im Automobilbereich und anderen Anwendungsgebieten, sowie deren Ansätze zur Lösung der an sie gestellten Anforderungen beinhalten muss, wird der Stand der Technik in diesem Bereich ausführlich in Abschnitt 2.4 behandelt, nachdem zuvor in Abschnitt 2.2 die Anforderungen definiert und in Abschnitt 2.3 die Rahmenbedingungen dargestellt werden.

Wie eingangs bereits erwähnt, ist heute eine gekapselte Architektur bestehend aus größtenteils selbständigen Einheiten Stand der Technik in der Automobilbranche. Daher ist auch die Architektur existierender X-by-wire Prototypen monolithisch, also mit fest integrierten Modulen, die losgelöst vom ursprünglichen System nicht funktionsfähig sind [Roo05]. Es existieren Ansätze wie zum Beispiel OSEK und AUTOSAR (siehe auch [BBJ99], [HSF⁺04]), Funktionen unabhängig von der Umgebung lauffähig zu machen. Diese Ansätze beruhen aber auf der Allokation von Funktionen zu Steuergeräten zur Entwurfszeit und bilden keine ausreichende Grundlage für eine dynamische Verteilung der Funktionen zur Laufzeit, siehe auch [Ram06].

Dynamische Verteilung von Funktionen wird in nicht sicherheitsrelevanten Bereichen über Middleware wie zum Beispiel CORBA und SOAP bewerkstelligt [Sch04]. Die im Automobilbereich für Steuergeräte zur Verfügung stehenden Ressourcen, beispielhaft sei hier das SAPS/RC-Board [Tor03] genannt, lassen eine Verwendung dieser Ansätze, die Speicher in der Größenordnung von 200 kByte bis zu mehreren MByte benötigen, nicht zu. Darüber hinaus sind Echtzeitfähigkeit und geeignete Ausfallsicherheit nicht gewährleistet.

In [SKA04] wird eine rekonfigurierbare Architektur für sicherheitsrelevante Luftfahrtsysteme (*SCRAM*, Subsystem Control Reconfiguration Analysis and Management) vorgestellt, die parallel zu der vorliegenden Arbeit entwickelt wurde, vgl. [NRK04b]. Im Unterschied zu dem in der vorliegenden Arbeit entwickelten State-Server greift SCRAM auf eine fail-stop Architektur der Teilsysteme zurück, geht also davon aus, dass fehlerhafte Steuergeräte ihren Ausfall den anderen Steuergeräten aktiv mitteilen können, während der State-Server lediglich den allgemeineren Fall des fail-silent Verhaltens voraussetzt. Da Ausfallarten, zum Beispiel ein Ausfall der Energieversorgung, vorstellbar sind, bei denen ein Gerät nicht in der Lage ist, seinen Ausfall korrekt aktiv zu signalisieren, ist fail-silent Verhalten in Bezug auf Verifikation des Gesamtsystems als vorteilhaft anzusehen.

1.2 Gliederung der Arbeit

Die Arbeit gliedert sich dabei wie folgt: Kapitel 2 geht auf die Anforderungen sicherheitsrelevante Elektronik im Kraftfahrzeug ein. Des Weiteren werden grundlegende Prinzipien der Fehlertoleranz, die die Grundlage für die folgenden Ausführungen darstellen, diskutiert. Anschließend werden diese Prinzipien auf Kommunikationsmedien übertragen und Grundlagen von Kommunikationsmedien behandelt. Vor diesem Hintergrund wird abschließend der Stand der Technik für Kommunikationsmedien im Kraftfahrzeug dargestellt.

Kapitel 3 stellt den neu entwickelten hochdatenratigen Netzwerkansatz *SafeNet* vor, geht auf die Funktionsweise und Fehlerbehandlung ein und präsentiert Modellierung und Simulationsergebnisse.

In Kapitel 5 wird eine Implementierung dieses Ansatzes vorgestellt und es werden Simulations- und Messergebnisse präsentiert und im Vergleich mit anderen Kommunikationssystemen bewertet.

Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 6.

Im Anschluss wird in Anhang A der *State-Server* Ansatz vorgestellt, der eine dynamische Verteilung von Funktionen ermöglicht.

2 Kommunikation verteilter Automobilanwendungen

2.1 Grundlagen fehlertoleranter Echtzeitsysteme

Bei Automobilelektronik im Allgemeinen und besonders bei drive-by-wire Anwendungen ist die Zuverlässigkeit des Systems einer der zentralen Punkte, der bei der Entwicklung berücksichtigt werden muss.

Definition 2.1 (Zuverlässigkeit) *Als Zuverlässigkeit, reliability eines Systems wird die Fähigkeit des Systems bezeichnet, einen Dienst vertrauenswürdig zu leisten.*

Aufgrund ihrer Interaktion mit physikalischen Größen sind sicherheitsrelevante Kfz-Elektroniksysteme sogenannte *Echtzeitanwendungen*, das heißt, bei der Berechnung eines Datums ist nicht nur der Wert des Ergebnisses relevant, sondern auch der Zeitpunkt, zu dem dieser Wert zur Verfügung steht. Auch bei sorgfältig entworfenen Systemen kann es in der Anwendung vorkommen, dass ein nicht vorhergesehener Zustand eintritt. Aufgrund der schwerwiegenden Konsequenzen, die ein Ausfall oder eine Fehlfunktion eines solchen Systems nach sich ziehen kann, ist es erforderlich, im System eine Strategie zur Behandlung solcher Situationen zu verankern. Zum Entwurf einer solchen Strategie wird eine systematische Betrachtung von Fehlerursachen, Fehlern und deren Konsequenzen sowie der Gegenmaßnahmen benötigt [Lap92], [Kop97], [ALR01].

Definition 2.2 (Ausfall, failure) *Als Ausfall eines Systems wird eine Abweichung des von dem System geleisteten Dienstes von seinen spezifizierten oder beabsichtigten Diensten bezeichnet. Der Ausfall eines Systems ist ein Ereignis und wird von der Umgebung des Systems wahrgenommen.*

Definition 2.3 (Fehler, error) *Ein inkorrekt oder nicht vorhergesehener innerer Zustand eines Systems wird als Fehler bezeichnet. Ein Fehler, der die Grenze des Systems erreicht und den vom System geleisteten Dienst beeinflusst, ist die Ursache für einen Ausfall.*

2 Kommunikation verteilter Automobilanwendungen

Definition 2.4 (Fehlerursache, *fault*) *Der Auslöser, durch den ein System in einen unvorhergesehenen Zustand überführt werden kann, wird als Fehlerursache bezeichnet. Eine Fehlerursache, die einen Fehler verursacht, wird als aktiv bezeichnet, eine Fehlerursache die keinen Fehler verursacht wird als inaktiv bezeichnet.*

Bemerkung 2.1 Im deutschen Sprachgebrauch wird, im Unterschied zum Englischen, häufig nicht zwischen Fehler und Fehlerursache unterschieden, sondern der Begriff *Fehler* für beide Fälle benutzt. Dies ist oft gerechtfertigt, da sowohl Fehler als auch Fehlerursachen, im Gegensatz zu dem *Ereignis* Ausfall des Systems, *Zustände des Systems* sind, siehe auch [Kop97]. Meist ist jedoch die Unterscheidung zwischen Fehler und Fehlerursache für die weiteren Betrachtungen hilfreich und wird deshalb in dieser Arbeit verwendet. Insbesondere ist diese Unterscheidung für die Betrachtung von inaktiven Fehlerursachen hilfreich, da sie durch eine unter Umständen durchaus zulässige Änderung der Randbedingungen aktiviert werden und so einen Fehler verursachen können.

In der Nachrichtentechnik hat sich der Begriff *Fehler*, *engl. error* für falsche Übertragung von Nachrichten eingebürgert. Es ist daher in diesem Zusammenhang von fehlerentdeckenden Codes und Forward Error Correction die Rede, auch wenn die Übertragung einer fehlerhaften Nachricht als *Ausfall* des Kommunikationssystems interpretiert werden kann. Die gebräuchlichen Formulierungen werden an den gegebenen Stellen, also im besonderen im Rahmen von Abschnitt 2.3 verwendet.

Aus den Definitionen 2.2 bis 2.4 wird deutlich, dass zwischen dem Ausfall eines Systems, einem Fehler und der Fehlerursache ein kausaler Zusammenhang besteht, wenngleich auch nicht von einer direkten bijektiven Beziehung ausgegangen werden kann. So ist es durchaus möglich, dass Fehler die Grenzen des Systems nicht verlassen und somit zu keinem Ausfall des Systems führen. Andererseits ist der Ausfall eines Systems immer auf einen Fehler zurückzuführen. In der Regel kann jedoch davon ausgegangen werden, dass ein Fehler ohne geeignete Gegenmaßnahmen die Schnittstelle des Systems nach außen erreicht und einen Ausfall verursacht. Diese Betrachtung wird im Zusammenhang mit Fehlerkapselung und Redundanzstrategien in diesem Abschnitt genauer betrachtet. Die hierzu nötige Klassifikation von Ausfällen, Fehlern und Fehlerursachen geschieht in Anlehnung an [ALR01], [ISO02] und [Lap92].

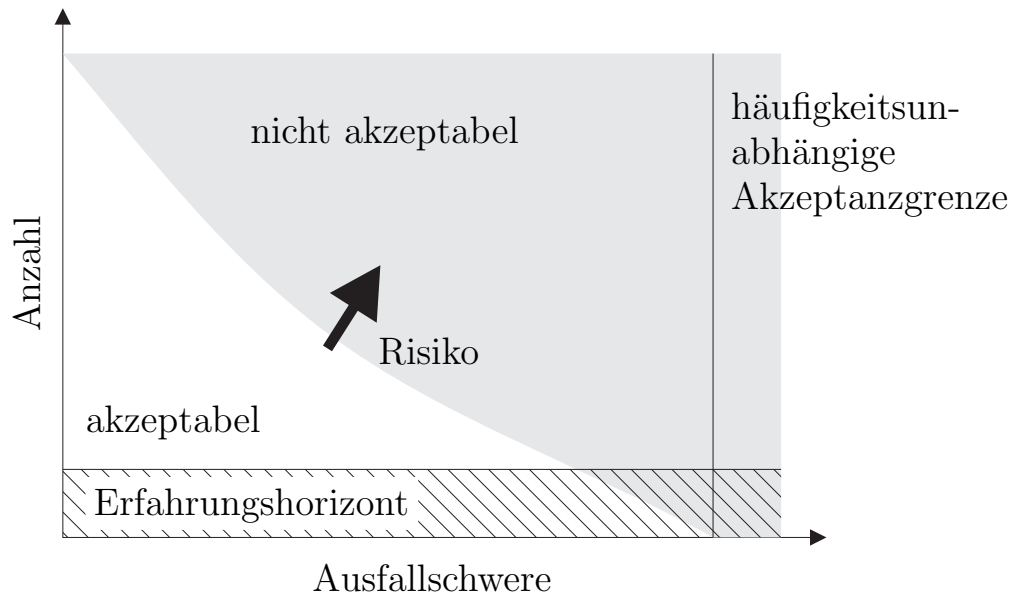


Abbildung 2.1: Einfluss von Ausfallschwere und Ausfallhäufigkeit auf das von einem System ausgehenden Risiko

2.1.1 Klassifizierung von Ausfällen, Fehlern und Fehlerursachen

Eine weitere, für die Diskussion von Systemzuverlässigkeit hilfreiche Definition ist die des sicheren Zustandes:

Definition 2.5 (Sicherer Zustand) *Als sicher wird ein Zustand des Systems definiert, in dem von ihm keine unvermeidbaren Risiken ausgehen.*

Die Grenze, ab der ein Risiko als nicht akzeptabel gilt, ist dabei von der Anwendung abhängig. Parallel zu der in Abbildung 2.1 eingezeichneten Grenze zwischen akzeptablem und nicht akzeptablem Risiko werden in [ISO02] die Safety Integrity Level (SIL) definiert. Je höher das Risiko ist, das von einem System ausgeht, desto höher sind die Anforderungen an die Zuverlässigkeit dieses Systems. Gängig sind in technischen Anwendungen SIL 1 bis SIL 4, wobei die Systeme mit steigendem SIL höheren Anforderungen genügen müssen. Unterteilt man die Ausfälle von Teilsystemen in *ungefährliche* Ausfälle, deren Auswirkungen auf das Gesamtsystem beherrschbar sind, und *gefährliche* Ausfälle, die den Dienst des Systems beeinträchtigen, so erhält man die in Tabelle 2.2 als *Safe Failure Fraction* (SFF) angegebene Klassifizierung, siehe auch [Bör02], [ISO02].

Daneben kann auch die Ausfallrate eines Systems zur Klassifizierung verwendet werden, siehe Tabelle 2.1. In [ISO02] wird hierbei zwischen Systemen

2 Kommunikation verteilter Automobilanwendungen

Tabelle 2.1: Klassifizierung von Systemen in SIL (*Safety Integrity Level*) anhand der Ausfallrate für Systeme mit hoher Anforderungsrate

SIL	mittlere Wahrscheinlichkeit eines gefährlichen Ausfalls /h
1	$\geq 10^{-6}$ bis $< 10^{-5}$
2	$\geq 10^{-7}$ bis $< 10^{-6}$
3	$\geq 10^{-8}$ bis $< 10^{-7}$
4	$\geq 10^{-9}$ bis $< 10^{-8}$

Tabelle 2.2: Klassifizierung von Systemen in SIL anhand der Fehlertoleranz und der Safe Failure Fraction (SFF)

SFF	definiertes Ausfallverhalten			nicht definiertes Ausfallverhalten		
	0 Fehler	1 Fehler	2 Fehler	0 Fehler	1 Fehler	2 Fehler
< 60 %	SIL 1	SIL 2	SIL 3	–	SIL 1	SIL 2
60 % – < 90 %	SIL 2	SIL 3	SIL 4	SIL 1	SIL 2	SIL 3
90 % – < 99 %	SIL 3	SIL 4	SIL 4	SIL 2	SIL 3	SIL 4
> 99 %	SIL 3	SIL 4	SIL 4	SIL 3	SIL 4	SIL 4

mit niedriger Anforderungsrate, bei denen die mittlere Ausfallwahrscheinlichkeit bei Anforderung ausschlaggebend ist, und solchen mit hoher Anforderungsrate, die nach der mittleren Ausfallwahrscheinlichkeit pro Stunde klassifiziert werden, unterschieden. Wird Automobilelektronik betrachtet, so liegt meist eine hohe Anforderungsrate vor, weshalb sich die nachfolgende Betrachtung auf diese Klasse beschränkt.

Wie eingangs erwähnt, handelt es sich bei sicherheitsrelevanter Automobilelektronik um Echtzeitsysteme. Der *Geltungsbereich* eines Fehlers kann also sowohl in der Zeit- als auch in der Wertedimension liegen. Ein Datum wird als falsch in der Zeitdimension bezeichnet, wenn es nicht zu einem a priori festgelegten Zeitpunkt zur Verfügung steht. Typischerweise sind diese Zeitpunkte bei zeitorientierten Systemen periodisch und bei ereignisorientierten Systemen ab dem Auftretenszeitpunkt eines Ereignisses, welches Eingangsdaten zur Verfügung stellt oder eine Berechnung anstößt, gemessen. Zur Bewertung, ob ein Datum in der Zeitdimension falsch ist, muss eine verlässliche und bei verteilten Systemen hinreichend synchronisierte lokale Zeitbasis vorliegen. Ebenso muss für die Beurteilung des Wertes eines Datums eine Referenz vor-

2.1 Grundlagen fehlertoleranter Echtzeitsysteme

liegen. Da der wahre Wert als Vergleich im Allgemeinen nicht verfügbar ist, kann es sich dabei zum Beispiel um einen Luenberger-Beobachter oder ein parallel rechnendes, redundantes Gerät handeln, siehe [Roo05].

Ein weiteres Merkmal zur Klassifizierung von Ausfällen ist die Art, wie fehlerfreie Geräte den Ausfall wahrnehmen. Der einfachste und bei weitem häufigste Fall ist natürlich, dass die gesamte Umgebung den Ausfall gleich wahrnimmt. Alle fehlerfreien Geräte können also den Ausfall eines fehlerhaften Geräts eindeutig diagnostizieren oder registrieren zumindest die gleichen Symptome und können sich, falls dies gefordert ist, auch auf diese Tatsache einigen. Diese Fehler werden als *symmetrisch* bezeichnet. Ein *asymmetrischer* oder *byzantinischer* Ausfall hingegen zeigt unterschiedlichen Betrachtern unterschiedliche Symptome¹. Dies schließt ein, dass einige Geräte ein fehlerfreies Datum erhalten und so keinen Ausfall diagnostizieren. Byzantinische Ausfälle sind in den meisten Systemen selten. Bei dem Beschuss eines TTP/C Controllers mit schweren Ionen zum Beispiel traten bei mehreren tausend Versuchen lediglich einige byzantinische Fehler auf, siehe [DHSZ03]. Da byzantinische Fehler in der ersten Version des TTP/C Bausteins aber nicht berücksichtigt wurden, war ein Ausfall der Kommunikation die Folge. Aufgrund ihres seltenen Auftretens werden byzantinische Fehler oft beim Systementwurf nicht explizit berücksichtigt und als zufällig oder nicht erklärbar abgetan. Daher liegen in vielen Systemen keine geeigneten Gegenmaßnahmen vor. Die möglichen Fehlerursachen für byzantinische Ausfälle sind vielfältig. Oft liegen so genannte *slightly off specification* (SOS)-Fehlerursachen vor, was bedeutet, dass der tatsächliche Wert nur leicht von dem beabsichtigten abweicht. Durch Fertigungsstreuungen, zum Beispiel in Treiberbausteinen für ein Kommunikationssystem, kann ein Wert von einem Gerät als korrekt, von einem anderen als falsch beurteilt werden. Gleiches kann auch durch die Drift lokaler Uhren in Steuergeräten oder so genannte *race-conditions*, bei denen mehrere Teilnehmer gleichzeitig versuchen, auf eine Ressource zuzugreifen, auftreten, siehe Abbildung 2.2. Aber auch zufällige Fehlerursachen können byzantinische Ausfälle verursachen.

Die Folge byzantinischer Ausfälle kann die Bildung von so genannten *Cliquen* sein. Eine Clique ist dabei eine Menge von Geräten, die eine einheitliche Sicht über die Funktionsfähigkeit der Menge aller Geräte hat. Diese Partitionierung des Gesamtsystems kann durch *Clique-Avoidance* Algorithmen entdeckt und somit vermieden werden, siehe auch Abschnitt 3.7 und Unterab-

¹Diese Nomenklatur wird auch in [Kop97] verwendet, während [DHSZ03] byzantinische Fehlerursachen definiert. Dies lässt sich wie folgt vereinbaren: Während [Kop97] bereits das Versenden verschiedener Daten als einen Ausfall des Kommunikationssystems oder des sendenden Geräts sieht, ist für [DHSZ03] der Empfang unterschiedlicher Eingangsdaten für den Ausfall ursächlich, da hier mit Schwerpunkt auf das Gesamtsystem argumentiert wird, während [Kop97] besonderes Augenmerk auf das Kommunikationssystem legt.

2 Kommunikation verteilter Automobilanwendungen

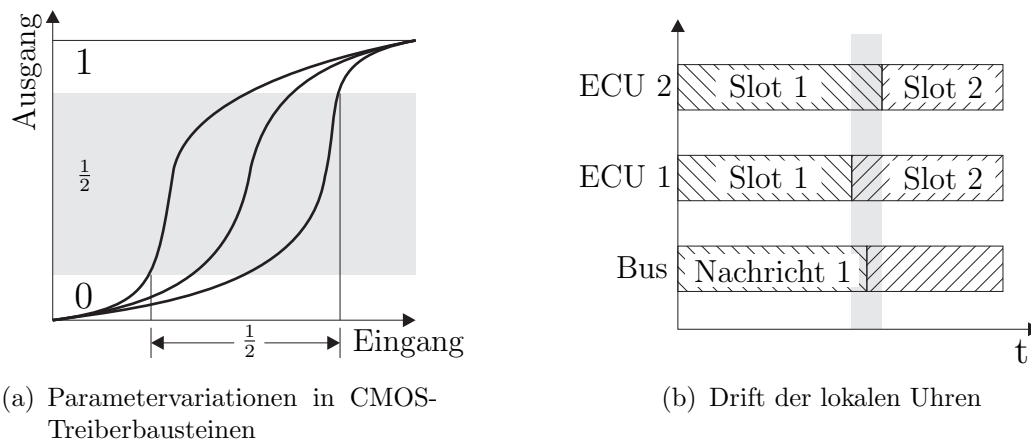


Abbildung 2.2: Beispiele für Fehlerursachen byzantinischer Ausfälle in der Werte- und Zeitdimension [DHSZ03]

schnitt 5.3.3. Im Gegensatz zu symmetrischen Fehlern ist die Verständigung auf die Fehlerursache in Gegenwart eines byzantinischen Fehlers (*byzantine agreement*) ungleich schwerer, siehe auch [LSP82], und benötigt im Allgemeinen mehr Kommunikation zwischen den Geräten.

Ausfälle lassen sich auch anhand ihrer Entstehung unterscheiden. Die so genannten *systematischen* Ausfälle treten unter gegebenen Randbedingungen reproduzierbar auf und haben ihre Ursache in Entwurfs- oder Implementierungsfehlern. Sie treten daher fast ausschließlich in seltenen Systemzuständen auf, da sie sonst durch Testprozeduren entdeckt und beseitigt worden wären, [LPS01]. Die Klasse der *zufälligen* Ausfälle hingegen wird meist durch äußere, aber immer durch nicht oder schwer reproduzierbare Fehlerursachen, wie zum Beispiel elektromagnetische Störungen oder α -Teilchenbeschuss ausgelöst. Da zufällige Fehlerursachen oft lokal begrenzt auftreten, kann ihnen durch eine redundante Auslegung und gegebenenfalls räumliche Trennung von Systemen begegnet werden. Diese Gegenmaßnahme ist bei systematischen Fehlern, wie zum Beispiel in [ESA96] beschrieben, hingegen oft wirkungslos, da in diesem Fall Common-Mode Fehler vorliegen und lediglich das falsche Ergebnis redundant berechnet wird. Hier kann auf diversitäre Hard- beziehungsweise Software, zurückgegriffen werden. Hierzu werden die verschiedenen redundanten Teilsysteme unabhängig voneinander und mit unterschiedlichen Hilfsmitteln entwickelt um die Wahrscheinlichkeit von Common-Mode Fehlern zu minimieren. Eine andere Strategie ist es, bei einem diagnostizierten Ausfall das betroffene System in einen bekannt fehlerfreien Zustand zu versetzen, zum Beispiel durch ein Reset. Hierbei muss jedoch sichergestellt werden, dass der fehlerfreie Zustand des Teilsystems auch einen fehlerfreien Zustand des Gesamtsystems zur Folge hat. In [ALR01] wird eine ähnliche Unterteilung vorge-

2.1 Grundlagen fehlertoleranter Echtzeitsysteme

nommen: Hier werden Fehlerursachen anhand des Auftrittszeitpunktes und der Systemgrenze klassifiziert. Auf Ausfälle weitergeführt kann daraus die hier dargestellte Klassifizierung gewonnen werden.

Auch ihre Dauer ist ein Unterscheidungskriterium für Ausfälle. Dauerhafte Ausfälle werden als *permanent* bezeichnet, während kurzzeitige Ausfälle *transient* genannt werden. Hierbei ist zu berücksichtigen, dass ein transienter Ausfall eines Teilsystems durchaus den permanenten Ausfall des Gesamtsystems bedingen kann, wie zum Beispiel eine kurzzeitige Störung auf dem Kommunikationssystem, die durch eine fehlerhafte Meldung das Gesamtsystem zum Absturz bringt, verdeutlicht. Unter Umständen ist es möglich, trotz transienter Störungen die Echtzeitbedingungen des Systems durch Wiederholungen von Operationen noch einzuhalten; dies kann jedoch ohne genaue Analyse der Fehlerursachen nicht mit Bestimmtheit behauptet werden.

Eine allgemeinere Betrachtung müsste an dieser Stelle noch andere Kriterien aufführen. Zu nennen sind hier besonders die Absichtlichkeit, ob die Fehlerursache also ein Unfall oder eine „böartige“ Fehlbedienung ist, oder nach der Entstehung um Benutzerfehler von natürlichen Ursachen unterscheiden zu können. Diese Merkmale zielen jedoch auf Sicherheit im Sinne von Datensicherheit oder Vertraulichkeit (engl. *security*) ab oder sind im Bereich der Mensch-Maschine-Schnittstelle anzusiedeln. Diese Themenbereiche gehen über das Zielgebiet dieser Arbeit hinaus und werden daher nicht gesondert behandelt.

2.1.2 Fehlerkapselung und Redundanzstrukturen

Zur Vermeidung von Ausfällen oder zur Begrenzung des Ausfallschweregrades ist es hilfreich, Fehler in dem zu untersuchenden System zu entdecken, bevor es zu einem Ausfall kommt. Dies ist jedoch nicht für alle Fehler möglich. Laut [Ham02] können circa 95 % aller Fehler durch built-in self-tests (BIST), also Tests in dem betreffenden (Teil-)system selbst, diagnostiziert werden und es kann ein definierter Fehlerzustand eingenommen werden. Für die anderen Fälle ist es notwendig, den Ausfall des Teilsystems durch Gegenmaßnahmen wie zum Beispiel Redundanz zu entdecken und gegebenenfalls zu kompensieren, bevor dieser Ausfall als Fehler die Fähigkeit des Gesamtsystems, seinen Dienst wie geplant auszuführen, beeinflusst. Dieses Vorgehen wird als *Fehlerkapselung* oder *Fehlermaskierung* bezeichnet. Hierzu existieren eine Reihe von Ansätzen, die sich grob anhand von drei Merkmalen unterscheiden lassen.

Art des Vergleichs

Bevor auf eine Abweichung reagiert werden kann, muss zuerst festgestellt werden, ob sich die Eingangsdaten überhaupt unterscheiden. Hierfür kann bei digitalen Eingangsdaten entweder ein bitgenauer Vergleich benutzt oder ein Bereich definiert werden, in dem die Eingangsdaten als gleich angesehen werden. Während sich letzteres vor allem dadurch auszeichnet, dass auch digitalisierte Eingangsdaten von redundanten analogen Sensoren, die vor allem in den unteren Bit abweichen können, wenig Probleme bereiten, ist der bitgenaue Vergleich vorzuziehen, wenn das System eine schnelle Fehleraufdeckung benötigt. Besonders bei der Verwendung von Daten, die nach dem Vergleich integriert werden, wie zum Beispiel Beschleunigungswerte, ist dies von Bedeutung, da nicht mittelwertfreie Abweichungen im Endergebnis enthalten bleiben.

Fehlerbehandlungsstrategie

Grundsätzlich muss bei der Behandlung von Fehlern zwischen zwei Zielsetzungen unterschieden werden. Einerseits ist es möglich, mit Hilfe der redundanten Auslegung Abweichungen zwischen den Systemen zu detektieren. Dies hat den großen Vorteil, dass der Vergleich zwischen den Ergebnissen technisch nicht aufwändig ist; im Extremfall ist zum Beispiel eine Anordnung denkbar, bei der der Vergleich der Buspegel, die von zwei Geräten, welche die gleiche Funktion ausführen, angelegt werden, ausreicht, um einen Ausfall aufzudecken. Diese Lösung ist technisch robust und einfach ohne komplexe Elemente wie Mikrocontroller implementierbar und daher aller Voraussicht nach auch zuverlässig im Einsatz. Nachteilig ist hier, dass durch die Beschränkung auf die Detektion von abweichenden Ausgangsdaten auch bei der Verwendung von mehr als zwei Teilsystemen nicht auf das korrekte Datum geschlossen werden kann und das System durch ein funktionierendes ersetzt werden muss. Hier erhöht der Einsatz von mehr sich gegenseitig überwachenden Teilsystemen die Zuverlässigkeit des Systems nicht².

Wird hingegen versucht, aus den von den Teilsystemen zur Verfügung gestellten Daten das korrekte Datum zu gewinnen, so wird eine normalerweise externe Einheit, ein so genannter *Voter*, benötigt, der nach einer bestimmten Strategie die Ergebnisse der Teilsysteme aggregiert³. Durch den Einsatz

²Es ist viel mehr das Gegenteil der Fall: Mehr Redundanz führt bei dieser Strategie zu einer höheren Ausfallwahrscheinlichkeit.

³Die Verwendung eines Voters ist nötig, wenn die Eingangsdaten interpretiert werden müssen, sei es aufgrund der verwendeten Votingstrategie oder durch die Verwendung von digitalen Eingangsdaten. Ein in der Luftfahrt häufig anzutreffendes Beispiel für einen Fall, in dem kein expliziter Voter benötigt wird, ist die Kraftmittelung von Aktoren an Stellflächen. Hier wird die Position der Stellflächen von mehreren

2.1 Grundlagen fehlertoleranter Echtzeitsysteme

dieser externen Einheit wird aber auch die Ausfallwahrscheinlichkeit des Gesamtsystems erhöht, da der Aufbau dieser Voter im Allgemeinen nicht trivial ist. Zudem muss darauf geachtet werden, dass der Ausfall des Voters nicht direkt den Ausfall des gesamten Systems zur Folge hat, der Voter also kein *single point of failure* (SPOF) ist. Des Weiteren wird eine Strategie benötigt, wie aus den Eingangsdaten das Ausgangsdatum gewonnen werden kann. Drei häufig anzutreffende sind:

Mittelwertbildung: Bei einfacher Mittelwertbildung werden die Eingangswerte lediglich gemittelt. Hierbei muss davon ausgegangen werden, dass falsche Werte nur wenig von den korrekten abweichen und damit das Endergebnis nur leicht verfälschen.

Middle Value Selection: Die Middle Value Selection bietet sich bei einer ungeraden Anzahl von Eingangsgrößen an. Hierbei werden die Eingangsdaten nach ihrem Wert sortiert und der mittlere Werte als Ausgangsdatum verwendet.

Fault Tolerant Midpoint: Dieses auch bei TTP/C anzutreffende Verfahren, siehe Abschnitt 2.4.3, toleriert n fehlerhafte Eingangswerte, indem die n kleinsten und n größten Eingangswerte nicht mit in die Berechnung des Ausgangswertes mit einbezogen werden. Hierdurch ist sichergestellt, dass nur korrekte Eingangswerte Einfluss auf das Ergebnis haben. Um das tatsächliche Endergebnis zu erhalten, bietet sich eine Mittelwertbildung an, da ja durch den Ausschluss aller n möglichen fehlerhaften Werte von der Korrektheit der verwendeten Eingangsdaten ausgegangen werden kann, und so eine Filterwirkung gegen kleine mittelwertfreie Störungen implementiert werden kann.

Nachteilig ist bei Verwendung von Abstimmungsmechanismen zusätzlich, dass eine Verletzung der maximalen Fehlerhypothese nicht ohne weiteres entdeckt werden kann. Werden zum Beispiel bei Verwendung der Middle Value Selection drei Eingangswerte verwendet, ist nicht ersichtlich, ob nur ein oder schon zwei Teilsysteme ausgefallen sind. Letzteres würde die Annahme über die maximal tolerierbaren Fehler verletzen und zu einem Ausfall des Systems führen. Eine Aufdeckung dieser Fehler ist zwar unter Umständen, zum Beispiel mit zusätzlichem Wissen über die Teilsysteme, möglich, erhöht dann aber die Komplexität des Voters erheblich.

Aktoren unabhängig eingestellt. Dabei wird davon ausgegangen, dass die korrekt arbeitenden Aktoren, die in der Überzahl sind, die fehlerhaften überstimmen können, da ihnen in der Summe mehr Kraft zur Verfügung steht [Col99].

2 Kommunikation verteilter Automobilanwendungen

Diversität

So genannte *common-mode failures*, also Ausfälle, die ihre Ursache in dem gleichen Aufbau mehrerer Teilsysteme haben, können durch die oben beschriebenen Redundanzstrukturen nicht entdeckt werden, da potentiell alle Teilsysteme gleichzeitig ausfallen. Um sich dagegen abzusichern, kann diversitäre Hard- und/oder Software verwendet werden: Der Einsatz von mehreren unabhängigen Programmierern, die Software für unterschiedliche Steuergeräte mit unterschiedlichen Tools erstellen, führt im Idealfall zu zwei oder mehr Systemen, die keine Common-Mode Ausfälle mehr aufweisen. Die Diversität kann allerdings dazu führen, dass sich die Ausgangsdaten der Rechner geringfügig unterscheiden und der Vergleich nicht mehr bitweise durchgeführt werden kann.

Gängige Redundanzstrukturen

Für die folgende Betrachtung gängiger Redundanzmechanismen werden einige Einschränkungen gemacht. Zum einen sei es ausgeschlossen, dass Fehler gleichzeitig auftreten, das heißt, der zweite Fehler tritt erst ein, wenn das System auf den ersten reagiert hat. Dabei ist es unerheblich, welches der Teilsysteme ausfällt. Common-Mode Ausfälle und Ausfälle in den Votern werden hierbei nicht berücksichtigt. Für die Analyse der Strukturen wird die Ausfallrate des Gesamtsystems λ_{sys} betrachtet. Die Ausfallrate eines einzelnen Teilsystems sei mit λ_{ECU} , (Electronic Control Unit) gegeben. Die einzelnen Redundanzstrukturen sind in Abbildung 2.3 zu sehen.

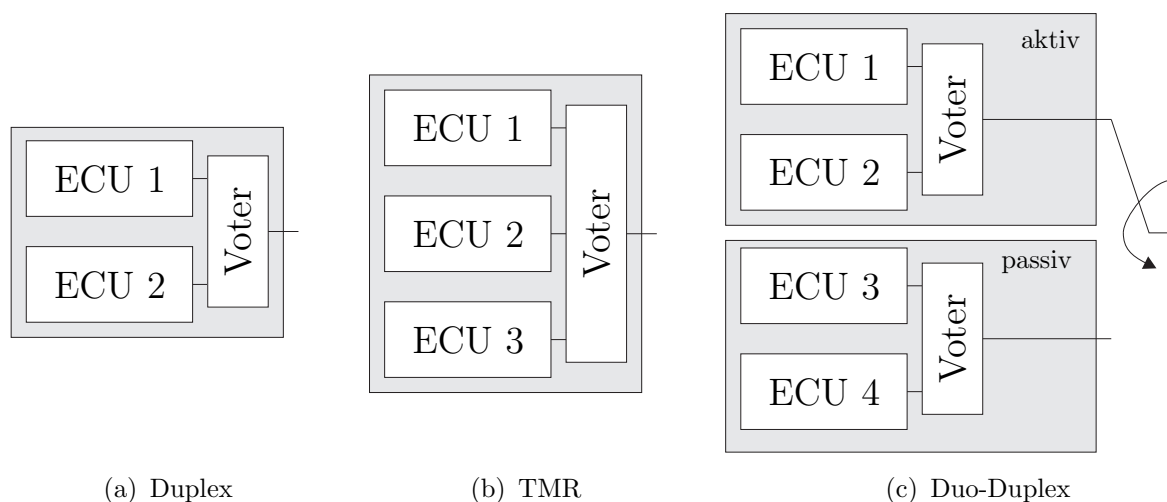


Abbildung 2.3: Verschiedene Redundanzstrukturen

2.1 Grundlagen fehlertoleranter Echtzeitsysteme

Simplex Bei einem in Abbildung 2.3 nicht dargestellten Simplex System liegt keine Redundanz vor. Daher gilt:

$$\lambda_{sys} = \lambda_{ECU}.$$

Duplex Bei einem Duplex System überwachen sich zwei Teilsysteme gegenseitig, indem sie die gleichen Berechnungen durchführen. Weichen die Ergebnisse der Teilsysteme ab, so kann davon ausgegangen werden, dass eines der Teilsysteme fehlerhaft ist. Da aber im Allgemeinen nicht entschieden werden kann, welches der vorliegenden Ergebnisse, und damit welches der Teilsysteme, fehlerhaft ist, stellen beide Teilsysteme den Betrieb ein und das System fällt aus. Mathematisch betrachtet entspricht dies einer Reihenschaltung der Teilsysteme. Die Ausfallrate des Gesamtsystems ergibt sich zu

$$\lambda_{sys} = 2\lambda_{ECU}$$

und ist damit trotz des nötigen Mehraufwandes höher als die des Simplex Systems. Der Vorteil des Duplex Systems liegt aber in seiner deutlich höheren Fehlerabdeckung. Anders als bei dem Simplex System, welches nur bei Fehlern, die durch BIST entdeckbar sind, definiert ausfällt, ist bei einem Duplex System ohne Common-Mode Fehler und gleichzeitige Ausfälle ein definiertes Ausfallverhalten mit geringem Mehraufwand gewährleistet. Dies erlaubt dem Rest des Systems eine angemessene Reaktion, wie zum Beispiel einen „*limp-home*“ Modus.

Triple Modular Redundancy (TMR) Während der Vergleich von zwei Eingangsdaten lediglich das Feststellen einer Abweichung ermöglicht, lässt sich bei TMR durch die Hinzunahme eines dritten Teilsystems über einen Mehrheitsentscheid ein fehlerhaftes Eingangsdatum tolerieren. Der Mehrheitsentscheid im Voter entspricht einer Parallelschaltung der Systeme. Nach Ausfall eines Systems kann durch Vergleich der verbleibenden zwei Eingangsdaten ein weiterer Fehler aufgedeckt und somit, wie bei einem Duplex System, ein definiertes Ausfallverhalten sichergestellt werden. Die Ausfallrate eines TMR System berechnet sich zu

$$\lambda_{sys} = 6\lambda_{ECU}^2.$$

Nach dem ersten Ausfall muss in einem TMR System die Voting-Strategie geändert werden. Dies erhöht den Aufwand für den Voter verglichen mit dem bei Duplex Systemen eingesetzten Vergleich. Wird der Voter mit der gleichen Ausfallrate wie ein Teilsystem und als SPOF

2 Kommunikation verteilter Automobilanwendungen

modelliert, so rückt die Ausfallrate eines TMR Systems in dem in Abbildung 2.4 verwendeten Wertebereich für λ_{ECU} in die Größenordnung des Simplex Systems.

Duo-Duplex Für ein Duo-Duplex System werden zwei Duplex Systeme so verschaltet, dass bei dem Ausfall des aktiven Duplex-Paares (aktiver Kanal) das passive übernehmen kann. Für die Betrachtung der Systemausfallrate ist es daher unerheblich, ob der Fehler im aktiven oder passiven Kanal zuerst ausfällt, da das Ergebnis in beiden Fällen ein funktionierendes Duplex System ist. Die Ausfallrate des Systems ist

$$\lambda_{sys} = 8\lambda_{ECU}^2$$

und damit höher als die des TMR Systems, welches im Gegensatz zu Duo-Duplex nur drei statt vier Teilsysteme benötigt. Der Hauptvorteil des Duo-Duplex Systems gegenüber einem TMR Systems ist, dass die Rekonfiguration des Voters im Fehlerfall entfällt. Darüber hinaus wird, wie zuvor beschrieben, kein spezielles System für das Voting benötigt, da ein Vergleich der Ausgangsdaten unter Umständen ausreicht.

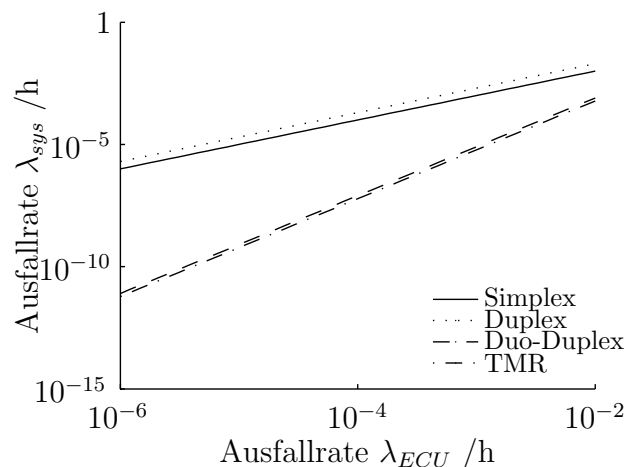


Abbildung 2.4: Ausfallraten verschiedener Redundanzarchitekturen

Die Ausfallraten der oben erwähnten Systeme sind in Abbildung 2.4 dargestellt. Hierbei ist jedoch zu beachten, dass Ausfallraten von $\lambda_{sys} \leq 10^{-9}$ /h in realen Systemen nur schwer zu realisieren sind, siehe auch [ISO02].

Wie bereits erwähnt, ist neben der Ausfallwahrscheinlichkeit das Verhalten des Systems im Fehlerfall ein weiteres wichtiges Auswahlkriterium für eine den Anforderungen angepasste Redundanzstruktur. Im günstigsten Fall ist

2.1 Grundlagen fehlertoleranter Echtzeitsysteme

das System trotz des Fehlers, also des Ausfalls eines Teilsystems, weiter korrekt funktionsfähig. Dieses Verhalten wird als *fail-operational* bezeichnet. Ein System sollte für alle Fehlerursachen, die von der maximalen Fehlerursachenhypothese abgedeckt werden, fail-operational sein. Diese Hypothese wird anhand der Randbedingungen, unter denen das System arbeitet, aufgestellt, und dient als Grundlage für den Entwurf des Systems. Ist dies nicht der Fall ist, je nach Zielsetzung, entweder Analyse eines existierenden oder Konstruktion eines neuen Systems, entsprechend die Hypothese oder die Systemarchitektur zu ändern. Wird die maximale Fehlerursachenhypothese verletzt, treten also entweder nicht abgedeckte Fehlerursachen auf oder wird die maximale Ankunftsrate überschritten, so ist zu erwarten, dass das System, wenn auch nicht zwingend, ausfällt. Zu vermeiden ist in diesem Fall ein undefinierter Ausfall, das so genannte *fail-active* Verhalten, bei dem das System weiterhin versucht, seinen Dienst aufrecht zu erhalten, ohne dass die Korrektheit gewährleistet werden kann. Zwar ist es oft wünschenswert, dass statt eines totalen Systemausfalls Daten geliefert werden, die wichtige Einschränkung bei fail-active Verhalten ist jedoch, dass nicht zwischen möglicherweise falschen und korrekten Daten unterschieden werden kann. Vorteilhafter ist ein definierter Ausfall, der dem restlichen System gegenüber den Ausfall signalisiert und so die Fehlerausbreitung verhindert. Das so genannte *fail-stop* Verhalten signalisiert hierbei den Ausfall aktiv, zum Beispiel durch eine Fehlermeldung, siehe auch [SKA04]. Nachteilig ist hier, dass zum Beispiel bei Ausfall der Stromversorgung in einem Steuergerät oder einer Trennung der Kommunikation es nicht möglich ist, den Ausfall aktiv zu signalisieren, die Fehlerabdeckung ist also beschränkt. Dieser Mangel wird durch das *fail-silent* Verhalten behoben, bei dem der Ausfall durch Nichtversenden von Nachrichten angezeigt wird. Durch das Ausbleiben einer Nachricht kann sicher auf den Ausfall des Teilsystems geschlossen werden. Andererseits stellt sich bei definierten fail-silent Ausfällen die Problematik, ein defektes Gerät sicher von dem gemeinsamen Kommunikationsmedium zu trennen⁴.

Das Verhalten der oben vorgestellten Strukturen im Fehlerfall ist in Tabelle 2.3 dargestellt. Dabei wird lediglich zwischen fail-operational, fail-active und fail-silent Verhalten unterschieden und fail-active angegeben, falls dieses Verhalten möglich ist.

Bemerkung 2.2 Der aktive Ausfall des Duplex Systems kann zum Beispiel durch den Ausfall der Trenneinheit entstehen. Dadurch kann der aktive Ausfall eines Teilsystems unter Umständen nach außen propagieren. Bei dem TMR System wird eine Strategie angenommen, die das TMR System nach dem ersten Ausfall auf ein Duplex System reduziert.

⁴Dieses Problem ist eine der zentralen Triebfedern hinter der Entwicklung von SafeNet, siehe Kapitel 3

Tabelle 2.3: Ausfallverhalten ausgewählter Redundanzstrukturen

Architektur	Verhalten nach dem ersten Fehler	Verhalten nach dem zweiten Fehler
Simplex	fail-active	fail-active
Duplex	fail-silent	fail-active
TMR	fail-operational	fail-silent
Duo-Duplex	fail-operational	fail-silent

2.2 Anforderungen an sicherheitsrelevante Automobilelektronik

2.2.1 Vergleich von fly-by-wire und drive-by-wire Systemen

Die Verwendung von by-wire ohne mechanische oder hydraulische Rückfall-ebene ist sowohl in der militärischen als auch in der zivilen Luftfahrt Stand der Technik. Der erste Prototyp im militärischen Bereich, eine von der NASA modifizierte F-8C, wurde bereits 1972 gebaut, wobei zu dieser Zeit bereits auf Know-how aus der Raketentechnik zurückgegriffen werden konnte. Diese Entwicklung brachte schließlich 1984 den Airbus A340 hervor, das erste zivile Flugzeug, das vollständig über fly-by-wire geflogen wird. Vor diesem Hintergrund stellt sich die Frage, warum die Technologie der Luftfahrtindustrie nicht in Kraftfahrzeuge übernommen wird. Zur Beantwortung dieser Frage müssen die Anforderungen, die beide Bereiche an sicherheitsrelevante Elektronik stellen, genauer untersucht werden. Die Hauptunterschiede zwischen Luftfahrt- und Automobilindustrie lassen sich folgendermaßen zusammenfassen, siehe auch [Roo05]:

Ausfallsschweregrad: Der Ausfallsschweregrad, der die Auswirkung des Ausfalls eines System auf seine Umgebung bewertet, ist bei Flugzeugelektronik höher zu beurteilen, da bei einem Ausfall erstens mehr Menschen betroffen sind und zweitens die Folgen für die einzelnen Personen wahrscheinlich schwerwiegender sind.

Zeit zur Überführung in den sicheren Zustand: Als sicherer Zustand kann beim Automobil das Parken am Straßenrand angesehen werden, da hier weder Gefahr vom Fahrzeug noch von dem fließenden Verkehr ausgeht. Beim Flugzeug kann durch Lotsen der Verkehr von einem geparkten Flugzeug weggeleitet werden, weshalb hier keine spezielle Parkposition gefordert werden muss und daher das Flugzeug nur gelandet werden

2.2 Anforderungen an sicherheitsrelevante Automobilelektronik

muss, um einen sicheren Zustand einzunehmen. Während die Zeit aus dem Betriebszustand in den sicheren Zustand bei einem Flugzeug sehr lang sein kann, man bedenke die Reiseflughöhe, ist es dem Fahrer eines Kraftfahrzeugs innerhalb kürzester Zeit möglich, einen sicheren Zustand zu erreichen. In [Mah00] wird hierfür eine Zeitspanne von 5 min angegeben, gemessen ab dem Aufleuchten einer roten Warnlampe. Durch angemessene optische, akustische und haptische Informationen an den Fahrer kann diese Zeit aber deutlich verringert werden.

Stichprobenumfang: Während Flugzeuge normalerweise in Stückzahlen von bis zu 2000 Stück (Boeing 737/300 /-400 /-500) gebaut werden, liegen die Stückzahlen bei Kraftfahrzeugen ungleich höher (bis zu circa 5 Millionen Fahrzeugen, zum Beispiel VW Golf I). Hierdurch steigt zwar einerseits die Wahrscheinlichkeit, dass ein System ausfällt; andererseits können durch die höhere Stichprobengröße systematische Fehler früher erkannt und behoben werden, ohne dass es zu kritischen Ausfällen kommt. Beispielhaft sei hier der Rückruf der W211-Serie von Mercedes-Benz aufgrund eines Fehlers in der SBC genannt.

Missionsdauer: Als Missionsdauer sicherheitsrelevanter Elektronik kann die Zeitspanne zwischen Ein- und Ausschalten des Systems gesehen werden. Während der Missionsdauer muss das System mit einer gewissen Zuverlässigkeit seinen Dienst erfüllen. Ein Einfluß der Missionsdauer auf die Zuverlässigkeit kann durch die Annahme begründet werden, dass die Wahrscheinlichkeit, mit der das System seltene fehlerhafte Zustände annimmt, mit steigender Betriebszeit steigt. Diese Zustände sind aber für die oben beschriebenen systematischen Ausfälle verantwortlich, da sie in der Testphase des Systems nicht entdeckt wurden. Bei einem Neustart befindet sich das System wieder in einem definierten Zustand. Die Missionsdauer liegt bei Flugzeugen im Mittel mit 113 min mehr als doppelt so hoch wie die durchschnittliche tägliche Nutzungsdauer eines Pkws.

Wartung: Während Flugzeuge vor jedem Flug durchgecheckt werden, ist die regelmäßige Wartung von Automobilen meist auf Benzin- und Ölstand sowie Luftdruck beschränkt und bezieht sicherheitsrelevante Elektronik, Sensorik und Aktorik nicht mit ein. Hierdurch besteht die Gefahr, dass drohende Ausfälle durch Verschleiß, zum Beispiel im Kabelbaum in den Türen, oder durch Alterung nicht erkannt werden und zu einem Ausfall des Systems führen.

Kostenorientierung: Bedingt durch die hohen Stückzahlen liegt der Fokus

2 Kommunikation verteilter Automobilanwendungen

bei Kostenoptimierung von Kraftfahrzeugen auf der Reduktion der Kosten pro Stück, während bei Flugzeugen trotz des hohen Anschaffungspreises auch die laufenden Betriebskosten stark berücksichtigt werden müssen. So sind beispielsweise Wartungskosten und mittelbar über den Verbrauch auch das Gewicht ausschlaggebend. Daher können in der Luftfahrtindustrie Mehrausgaben für die redundante Auslegung von sicherheitsrelevanten Systemen gerechtfertigt werden, wenn dadurch Einsparungen in den Betriebskosten entstehen.

Komplexität des Verkehrs: Wie bereits in der Einleitung erwähnt, ist ein Hauptvorteil von by-wire Systemen gegenüber herkömmlichen Varianten, dass Copilot- und Autopilotfunktionen leichter in das Fahrzeug integriert werden können. Im Bereich der Fahrzeugdynamik wird dies erfolgreich getestet. Soll das Fahrzeug jedoch zeitweise oder dauerhaft autonom agieren, ist es im Straßenverkehr, im Gegensatz zum Einsatz in dem von Lotsen kontrollierten Luftverkehr, notwendig die momentane Fahrsituation korrekt zu erfassen. Hierzu genügt es nicht, die eigene Position, zum Beispiel per GPS (Global Positioning System), zu kennen, es müssen auch Informationen über Hindernisse, Fahrbahnbeschaffenheit, genaue Route und andere Faktoren vorliegen. In der Praxis werden hierfür eine Vielzahl verschiedener Sensoren und anderer Informationsquellen benötigt, deren Daten zu einem Gesamtbild der momentanen Fahrsituation fusioniert werden müssen. Darüber hinaus ist eine komplexe Interaktion mit anderen Verkehrsteilnehmern, also anderen eventuell auch autonom gesteuerten Fahrzeugen, Fußgängern und Fahrradfahrern nötig, was die Entwicklung solcher Systeme im Kraftfahrzeug weiter erschwert. Bedingt durch die geforderte Echtzeitfähigkeit des Prozesses werden hierdurch ungleich höhere Anforderungen an die Fahrzeugelektronik gestellt als dies bei Flugzeugautopiloten der Fall ist. An dieser Stelle sei auch auf das Wiener Übereinkommen über den Straßenverkehr, Kapitel II, Art. 8, §5 [StV77] verwiesen. Dort ist die Rolle des Fahrers geregelt: „*Jeder Führer muss dauernd sein Fahrzeug beherrschen . . . können*“. Dies verhindert eine Einführung von vollständig autonomen Systemen aus juristischer Sicht. Bei aktuellen Assistentensystemen wird dieses Problem umgangen, indem das automatisch geregelte Verhalten vom Fahrer überstimmt werden kann. Hinzu kommt die ungeklärte Frage der Haftung bei Ausfall von Copilot- und Autopilotfunktionen.

Gewicht: Während im Flugzeugbau das Gewicht eine entscheidende Rolle spielt und fly-by-wire Steuerungen den mechanischen oder hydraulischen

2.2 Anforderungen an sicherheitsrelevante Automobilelektronik

schen Varianten in diesem Punkt deutlich überlegen sind, ist dieser Punkt im Automobilbau nur zweitrangig. Ein Indiz hierfür ist die Zunahme des Gewichts in Österreich neu zugelassener Kraftfahrzeuge von ca. 1400 kg auf ca. 1600 kg zwischen 2000 und 2005. Wie zuvor bereits erwähnt, ist das Gewicht, beziehungsweise die Einsparung, nicht direkt ausschlaggebend für Kaufentscheidungen der Kunden. Vielmehr sind die Auswirkungen des Mehrgewichts wie höherer Verbrauch und eventuell verbessertes Crashverhalten wichtig. Da in dem oben genannten Zeitraum auch die Effizienz der Motoren gesteigert und die passive Sicherheit der Fahrzeuge erhöht wurde, ist zu erklären, dass die Steigerung des Gewichts im Automobilbereich kaum eine Rolle im Kaufverhalten der Kunden spielte, da sie einerseits größtenteils kompensiert wurde und andererseits die Folge einer sich positiv auf das Kaufverhalten auswirkenden Änderung war.

Training der Benutzer: Piloten in der kommerziellen Luftfahrt werden für jeden Flugzeugtyp einzeln geschult und sind dadurch in der Lage, ein verändertes Mensch-Maschine-Interface, wie zum Beispiel den Übergang vom Steuerhorn zum Sidestick, und ein leicht verändertes Systemverhalten wie durch das Ausregeln von pilot induced oscillations zu tolerieren. Im Gegensatz hierzu ist im Kraftfahrzeug nicht davon auszugehen, dass der Fahrer speziell für dieses Fahrzeug ausgebildet wurde. Vielmehr muss die Bedienung insbesondere in Gefahrensituationen eindeutig und intuitiv sein. Dies gilt besonders für neuartige Systeme.

Eine qualitative Übersicht über die oben genannten Einflussfaktoren ist in Abbildung 2.5 gegeben.

Die geforderten Ausfallraten, die sich aus diesen Anforderungen ergeben, unterscheiden sich innerhalb der Luftfahrtindustrie deutlich. Wird im militärischen Bereich eine Ausfallrate von 10^{-7} /h für katastrophale Ausfälle in fly-by-wire System toleriert, so wird im zivilen Bereich eine Ausfallrate von 10^{-9} /h gefordert. Diese Raten lassen sich nicht mit Einzelgeräten erreichen, sondern müssen über fehlertolerante Redundanzstrukturen erreicht werden. Im Regelfall wird von den Systemen verlangt, mindestens zwei beliebige Fehler tolerieren zu können, weshalb im militärischen Bereich oft diversitäre Quadruplex Systeme verwendet werden [Col99]. Im zivilen Bereich wird teilweise mehr Redundanz eingesetzt, wie im Folgenden am Beispiel der Boeing 777 gezeigt werden soll, siehe auch [Yeh96].

Herzstück der fly-by-wire Elektronik der 777 sind die *actuator control electronics* (ACE), siehe Abbildung 2.6. Die quadruplex ausgelegten ACEs sind die Schnittstelle zwischen den analogen Teilen, den Piloten auf der einen und

2 Kommunikation verteilter Automobilanwendungen

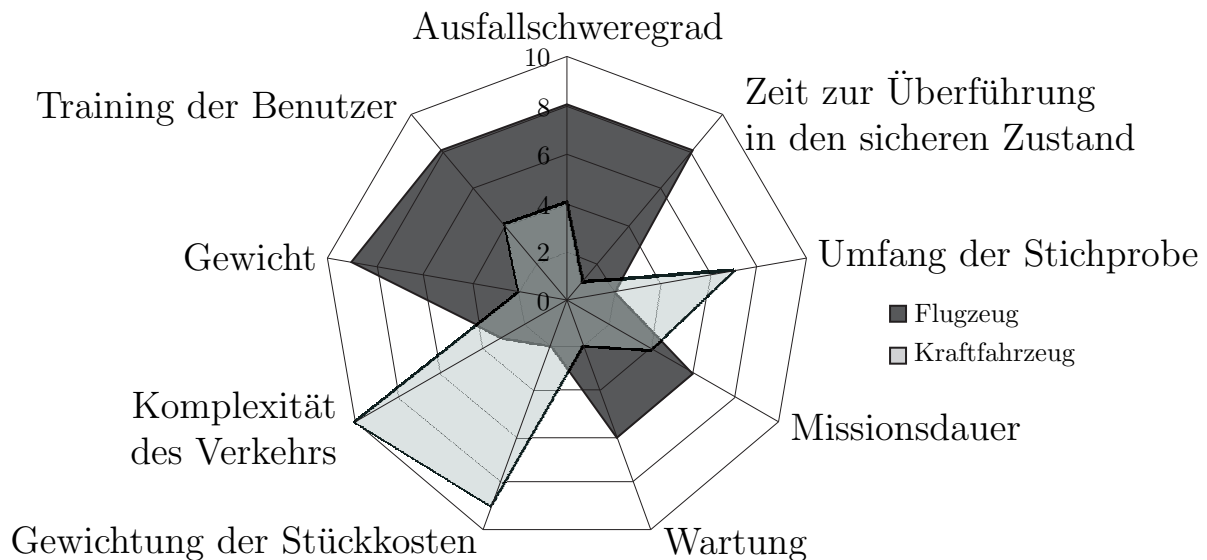


Abbildung 2.5: Qualitative Bewertung charakteristischer Größen der Automobil- und der Luftfahrtindustrie im Vergleich

den *power control units* (PCU), also den Aktoren an den Stellflächen und den zugehörigen Sensoren, auf der anderen Seite und dem digitalen Teil. Die ACEs dienen dazu, die Regelkreise des fly-by-wire Systems zu schließen. Hierzu stehen, abhängig von dem Funktionieren der anderen Komponenten mehrere Strategien zur Verfügung. Im Normalfall werden hauptsächlich die Daten der *primary flight computer* (PFC) verwendet, die wie die unterstützenden Systeme (*airplane information management system* AIMS, etc.) über einen dreifach ausgelegten ARINC 629 Bus mit den ACEs verbunden sind. Aufgabe der PFCs ist es, die Eingaben der Piloten beziehungsweise des Autopiloten in gültige Steuersignale umzusetzen, also zu hohe Geschwindigkeit oder zu hohe Anstellwinkel zu vermeiden.

Als Architektur der PFCs kommt eine sogenannte *triple-triple* Struktur zum Einsatz. Jede der drei PFCs besteht aus drei Rechnern (*lanes*) mit unterschiedlichen Prozessoren (AMD 29050, Intel 80486 und Motorola 68040). Um Common-Mode Ausfälle zu vermeiden, dürfen zwar alle drei Lanes alle drei Busse, die mit der PFC verbunden sind, lesen und damit die Eingangsdaten verarbeiten, aber nur je eine Lane pro Bus ist sendeberechtigt. So wird verhindert, dass eine defekte Lane die funktionierenden Lanes davon abhält, miteinander zu kommunizieren.

Aufgrund von Studien, [Yeh96] zitiert hier [ALS95], und Erfahrungen der Entwickler wurde entschieden, zwar diversitäre Hardware und als notwendige Konsequenz auch unterschiedliche Compiler zu verwenden, sich jedoch auf die Entwicklung und den Test einer Software Version zu beschränken. Die Entscheidung für diversitäre Hardware und Entwicklungswerkzeuge wird

2.2 Anforderungen an sicherheitsrelevante Automobilelektronik

verständlich, wenn man berücksichtigt, dass die ECUs aus *commercial off-the-shelf* (COTS) Komponenten bestehen und die Errata des Pentium II, einem Nachfolger des verwendeten Intel 80486, beispielsweise im Zeitraum von Mai 1997 bis November 1998 bereits 68 Designfehler enthielten. Durch die TMR Anordnung der Lanes innerhalb der PFC ist es möglich, mindestens einen dieser Designfehler zu tolerieren. Des Weiteren sind die PFCs an unterschiedlichen Orten im Flugzeug angebracht, sodass eine örtlich begrenzte Fehlerursache wie zum Beispiel ein Feuer in einem Schaltschrank nicht alle drei PFCs gleichzeitig betrifft.

Durch die dreifach Auslegung der wichtigen Komponenten ist es möglich, ein fail-silent Verhalten für die PFCs zu gewährleisten, welches frühestens nach dem zweiten sicherheitsrelevanten Fehler eintritt. Sollten die PFCs dennoch ausfallen, so schalten die ACEs in einen analogen Modus, in dem Zusatzfunktionen wie die Trajektorienbegrenzung wegfallen und die Piloten das Flugzeug direkt steuern.

Wichtig für die folgenden Betrachtungen ist, dass an diesem Beispiel ersichtlich ist, dass fly-by-wire Anwendungen bedingt hauptsächlich durch die lange Zeit, die für die Überführung in den sicheren Zustand benötigt wird, und den hohen Ausfallschweregrad in der Lage sein müssen, mehrere sicherheitsrelevante Fehler zu tolerieren. Aufgrund der relativ langen Missionsdauer ist es wahrscheinlich, dass auch seltene fehlerhafte Zustände in den Systemen angenommen werden. Dem kann durch die Verwendung von diversitären Teilsystemen, besonders für komplexe Komponenten wie Prozessoren und zugehörige Compiler, vorgebeugt werden, während sich für andere Teilsysteme, hier die Software, ein strukturierter Designprozess sowie ausreichendes Testen empfiehlt. Die Entstehung von SPOFs ist in der Kommunikation zu vermeiden, indem keines der an der Kommunikation beteiligten Systeme in der Lage ist, im Fehlerfall die Kommunikation der anderen zu stören. Dies wird durch eine physikalische Trennung erreicht.

2.2.2 Umsetzung der Anforderungen für Automobilanwendungen

Wie bereits erwähnt ist die Zeit, die für eine Überführung in den sicheren Zustand benötigt wird, beim Kraftfahrzeug deutlich geringer als beim Flugzeug und die Wahrscheinlichkeit, dass während dieser Zeit ein weiterer sicherheitsrelevanter Ausfall auftritt, entsprechend gering. Es wird für Kraftfahrzeuganwendungen daher meist von einer Ein-Fehler-Hypothese ausgegangen, was bedeutet, dass das System lediglich den Ausfall eines Teilsystems tolerieren kann, ohne dass die Zuverlässigkeit des geleisteten Dienstes beeinträchtigt

2 Kommunikation verteilter Automobilanwendungen

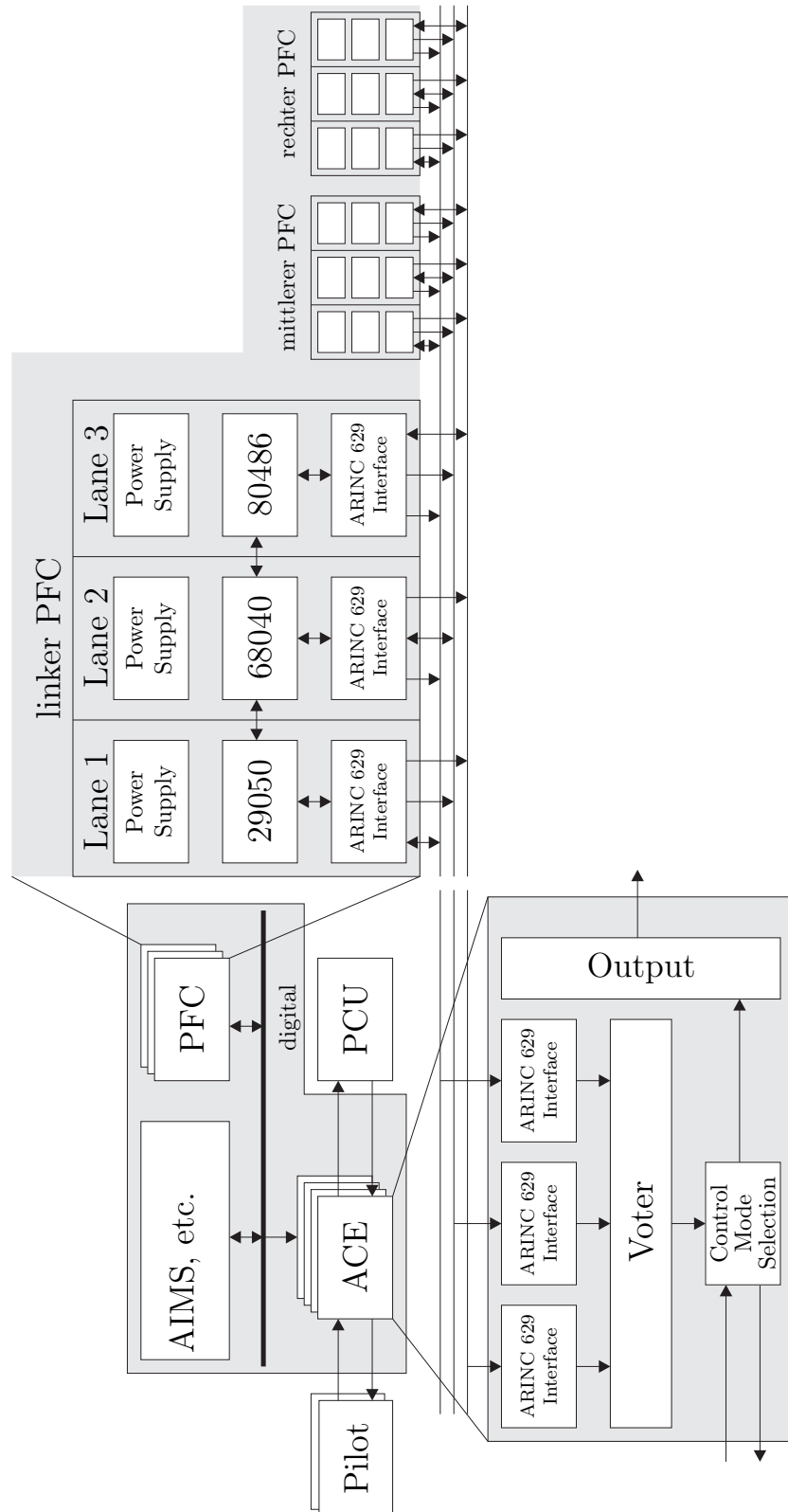


Abbildung 2.6: Vereinfachter Überblick über das fly-by-wire System der Boeing 777. Eine detailliertere Betrachtung ist in [Yeh96] zu finden.

2.2 Anforderungen an sicherheitsrelevante Automobilelektronik

wird [ISS02], [Roo05].

In diesem Zusammenhang ist auch die Reduktion der Komplexität der Komponenten als sinnvoll zu betrachten, da dies unter Umständen die vollständige Verifikation eines Teilsystems ermöglicht. Hierzu wird in [Roo05] der Ansatz gewählt, fail-silent als einziges Fehlverhalten zuzulassen und dies durch Duo-Duplex Anordnung der Teilsysteme zu gewährleisten, wodurch eine Umkonfigurierung des Voters im Fehlerfall vermieden wird. Auch wird der bitweise Vergleich einem tolerierenden vorgezogen. Hierbei verschiebt sich der Schwerpunkt der Betrachtung auf das Sicherstellen konsistenter Eingangsdatensätze, was sich mit einfachen Mitteln realisieren lässt, siehe auch Beispiel 1.

Durch die Verwendung von bewährten COTS Komponenten kann eine Reife der verwendeten Entwicklungswerkzeuge gewährleistet werden, wodurch eine gute Fehlerabdeckung der verwendeten Tests und der Erfahrungen aus dem Einsatz nahegelegt wird. Unter diesen Annahmen ist die Verwendung von diversitärer Hardware und Software nach heutigem Erkenntnisstand nicht nötig. Für die akzeptable Ausfallrate von sicherheitsrelevanter Automobilelektronik gibt es mehrere Ansätze, die von $\lambda_{sys} < 10^{-9}$ /h bis 10^{-6} /h reichen, [Rie03] beziehungsweise [ATJ01].

An dieser Stelle ist zu erwähnen, dass eine Anpassung der ISO 61508 [ISO02] an die Anforderungen der Automobilindustrie durch die ISO im Gange ist. Bei Inkrafttreten der Norm sind die in den vorigen Abschnitten hergeleiteten Schlussfolgerungen gegebenenfalls der Norm anzupassen.

Beispiel 1 (Softwarebasiertes Duo-Duplex System) Ein System, das den oben genannten Anforderungen genügt, ist das softwarebasierte Duo-Duplex System mit switching signal path BUSPWR-Block, welches in [Roo05] vorgestellt wird, siehe auch Abbildung 2.7. Dieses für einen drive-by-wire Hardware-in-the-Loop Prüfstand konzipierte System verwendet redundant ausgelegte CAN Busse zur Kommunikation mit dem übrigen Fahrzeug sowie je einen CAN Bus zwischen Master und Slave jedes Kanals. Dieser dient dazu, die Buslast auf den externen globalen Bussen nicht durch Zustands- und Synchronisationsachrichten, die innerhalb eines Kanals versendet werden müssen, zu blockieren. Der BUSPWR Block, der in Abbildung 2.7 mit vier Teilsystemen dargestellt ist, ist in der Lage, die Geräte eines Kanals im Fehlerfall sicher vom Bus zu trennen. Um möglichst viele Fehler aufdecken zu können, wird die so genannte *switching signal path* Struktur verwendet. Hierbei werden zyklisch Teilsysteme innerhalb des BUSPWR Blocks von den Leitungen getrennt und die Auswirkungen dieses Trennens beobachtet, wodurch sich auch *stuck-at* Fehler erkennen lassen. Die einzelnen Teilsysteme des BUSPWR-Blocks sind dabei, wie die Steuergeräte auch, aus COTS Kom-

2 Kommunikation verteilter Automobilanwendungen

ponenten aufgebaut, um einerseits Kosten zu sparen und andererseits auf die Erfahrungen aus Test und bisherigem Betrieb in anderen Anwendungen zurückgreifen zu können.

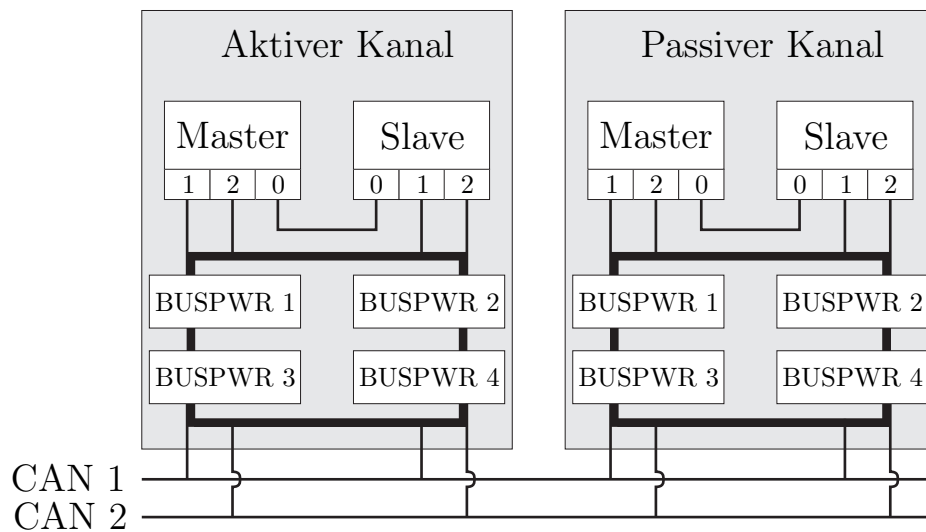


Abbildung 2.7: Ein softwarebasiertes Duo-Duplex System

Um sicher gehen zu können, dass keine falschen Ausgangsdaten das System verlassen, wird der BUSPWR Block von dem Output Management der ECUs angesteuert. Da stuck-at Ausfälle von Teilsystemen erkennbar bleiben müssen, werden von den Geräten eines Kanals orthogonale Rechtecksignale verwendet, was die Detektion eines in einem Zustand verharrenden Teilsystems erlaubt.

Vergleiche innerhalb des BUSPWR Blocks sowie der Daten, die von den Steuergeräten durch den BUSPWR auf den Bus verschickt werden sollen, erfolgen aufgrund der besseren Fehlerkapselung bitweise und nicht über einen tolerierenden Vergleich. Hieraus ergibt sich, dass besonderer Wert auf übereinstimmende Eingangsdatensätze im gesamten Duo-Duplex System gelegt werden muss. Durch die übereinstimmenden Eingangsdatensätze können die redundanten Berechnungen im fehlerfreien Fall zuverlässig durchgeführt werden.

Es zeigt sich auch in diesem Beispiel deutlich, dass eine Schwachstelle des Gesamtsystems die sichere Trennung von fehlerhaften Komponenten von dem gemeinsamen Kommunikationsmedium ist. Hierfür ist in Form der BUSPWR Blocks ein hoher Mehraufwand nötig. Obwohl dieses Gesamtsystem sowohl auf der Basis von COTS Steuergeräten aufgebaut wurde als auch die für die Fehlertoleranz nötigen Ergänzungen wo möglich in Software realisiert wurden, werden durch den BUSPWR Block nicht unerhebliche Mehrkosten verursacht. Wie bereits in Abschnitt 2.2.1 erwähnt, ist gerade die Automobil-

2.2 Anforderungen an sicherheitsrelevante Automobilelektronik

industrie sensibel gegenüber höheren Stückkosten, was eine Lösung, wie sie in diesem Beispiel vorgestellt wurde, für den Serieneinsatz untauglich macht. Prinzipiell ist eine weitere Integration des BUSPWR-Blocks in CAN jedoch ein vielversprechender Lösungsansatz.

2.2.3 Kommunikation in verteilten Elektroniksystemen

Wie in den vorherigen Abschnitten gezeigt, ist es, mit den erwähnten Einschränkungen, im automobilen Umfeld ausreichend, einen Fehler zu tolerieren. Wie sich ebenfalls gezeigt hat, ist eine der Hauptherausforderungen das sichere Trennen eines fehlerhaften Gerätes von dem gemeinsamen Kommunikationsmedium. Dies kann, in Vorgriff auf Abbildung 2.8, nur in den unteren Schichten geschehen, da sonst bestimmte Fehlerfälle, zum Beispiel stuck-at Fehler der höheren Schichten, nicht abgedeckt werden können. Zudem sollte es möglich sein, Fehler innerhalb des Kommunikationssystems aufzudecken um so die eventuelle Verwendung fehlerhafter Eingangsdaten zu vermeiden. Zusammenfassen lassen sich diese Voraussetzungen an Kommunikationsmedien im Kraftfahrzeug wie folgt:

Redundante Kommunikationswege Zwischen zwei Teilsystemen müssen mindestens zwei unabhängige Wege existieren, sodass kein Einzelfehler die Kommunikation zwischen den Teilsystemen verhindern kann.

Fehleraufdeckung Das Kommunikationssystem muss es ermöglichen, fehlerhafte Übertragungen zu erkennen.

Sichere Trennung Nach der Erkennung eines Ausfalls muss das Kommunikationssystem den übrigen Teilsystemen eine ungestörte Kommunikation in der Präsenz des fehlerhaften Teilsystems ermöglichen.

Streng genommen lässt sich die dritte Forderung, also die nach der sicheren Trennung fehlerhafter Geräte vom gemeinsamen Kommunikationsmedium aus der ersten, also der Ein-Fehler-Toleranz, ableiten. Wäre die effektive Trennung von dem gemeinsamen Kommunikationsmedium nicht möglich, so könnte ein Fehler zum Ausfall des gesamten Systems führen. Da die sichere Trennung fehlerhafter Geräte jedoch bei dem Entwurf von sicherheitsrelevanten Kommunikationsmedien eine entscheidende Rolle spielt, siehe auch Kapitel 3, ist die gesonderte Aufführung an dieser Stelle für die im Folgenden angestellten Überlegungen hilfreich.

Die Fehlererkennung wird in den meisten existierenden Systemen zuverlässig mit Hilfe von Prüfsummen gewährleistet. Diese lassen die Identifizierung einer fehlerhaften Übertragung zu, ermöglichen jedoch nicht direkt

2 Kommunikation verteilter Automobilanwendungen

die Lokalisation des Fehlers. Dies muss meist in den höheren Schichten sichergestellt werden, was in der Gegenwart von fehlerhaften Teilnehmern durchaus aufwändig werden kann, siehe auch [LSP82].

Bei der folgenden Betrachtung der Kommunikationssysteme soll davon ausgegangen werden, dass Common-Mode Ausfälle des Kommunikationssystems nicht berücksichtigt werden müssen. Erfahrungen in der Luftfahrtindustrie, siehe Unterabschnitt 2.4.3 sowie der Automobilindustrie, siehe Abschnitt 2.4.1, zeigen, dass es möglich ist, diese Systeme mit ausreichender Sicherheit zu testen und durch geeignete Maßnahmen im Entwurfsprozess die Funktionalität zu gewährleisten. Auf diversitäre Auslegung des Kommunikationssystems kann also verzichtet werden.

2.3 Grundlagen zu Kommunikationsmedien

2.3.1 Nachrichtenübertragung

Ziel der Nachrichtenübertragung ist es, Nachrichten von einem Sender zu einem Empfänger zu transportieren. Hierzu wird ein Kommunikationssystem oder Kommunikationsmedium verwendet. Dieses Gesamtsystem besteht üblicherweise aus mehreren Teilsystemen, die hierarchisch gegliedert sind. Gängig ist die Einteilung dieser Teilsysteme anhand des in Abbildung 2.8 dargestellten OSI-Schichtenmodells.

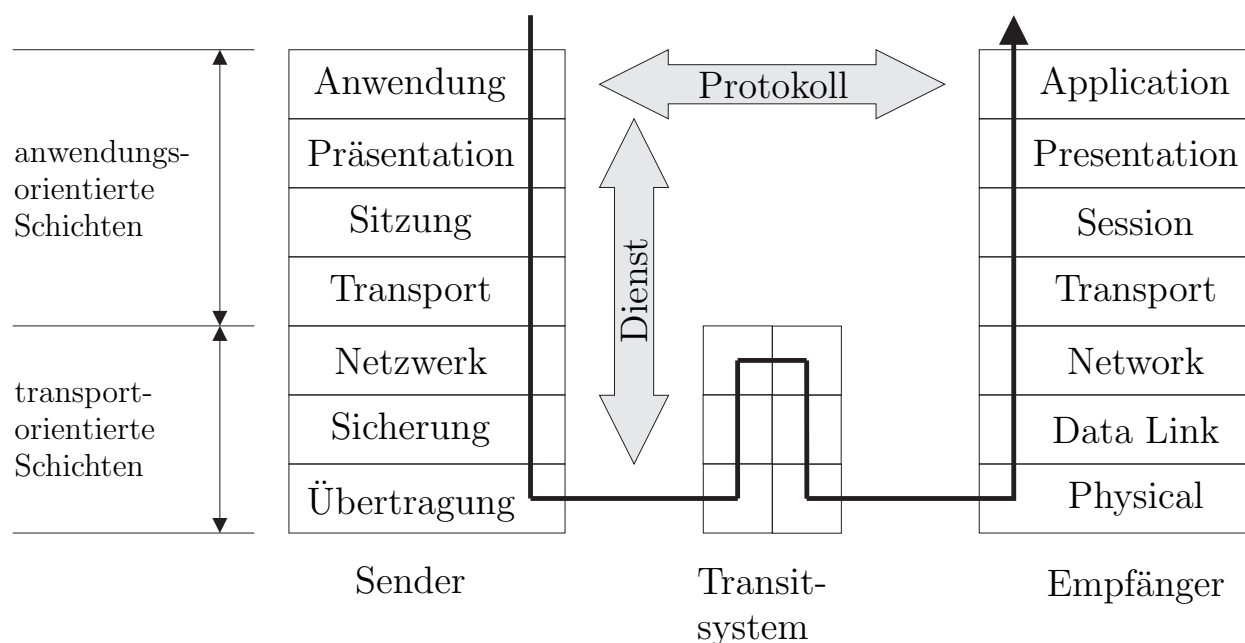


Abbildung 2.8: Das OSI-Schichtenmodell [Jon01]

2.3 Grundlagen zu Kommunikationsmedien

Die einzelnen Schichten (*layer*) des Kommunikationssystems sind dabei abgeschlossene Teilsysteme, die mit ihrer Umwelt, also den anderen Schichten eines Systems, über definierte Schnittstellen, die so genannten *Dienste* kommunizieren. Hierdurch ist es möglich, einzelne Schichten in einem Kommunikationssystem auszutauschen, ohne dass die darüber oder darunter liegenden Schichten angepasst werden müssen, solange die Konventionen des Dienstes eingehalten werden. Andererseits können Teilsysteme so auch in anderen Kommunikationssystemen eingesetzt, also wiederverwendet, werden.

Eine Schicht kommuniziert mit der entsprechenden Schicht eines anderen Systems durch ein so genanntes *Protokoll*. Dies ermöglicht es einer Schicht, Daten zu verarbeiten, ohne andere Schichten berücksichtigen zu müssen. Hierfür werden die Daten, die eine Schicht von der darüber liegenden Schicht zum Versand erhält, dem Protokoll gemäß verändert. Im Normalfall erhöht sich hierdurch die Länge der Nachricht in Richtung des physikalischen Mediums. Für die folgenden Betrachtungen sind hauptsächlich die unteren, transportorientierten Schichten von Bedeutung.

Erwähnenswert ist, dass nur selten Systeme strikt nach dem OSI-Schichtenmodell implementiert werden. Vielmehr werden häufig mehrere Schichten in einem Teilsystem zusammengefasst.

Betrachtet man ein Kommunikationssystem mit Fokus weniger aus informationstechnischer sondern aus nachrichtentechnischer Perspektive, so ist die in Abbildung 2.9 dargestellte Interpretation hilfreich. Im Gegensatz zu dem OSI-Schichtenmodell sind hier nur die Aufgaben der Sicherungsschicht detailliert betrachtet, während die Bitübertragungsschicht als Kanalmodell eingeht.

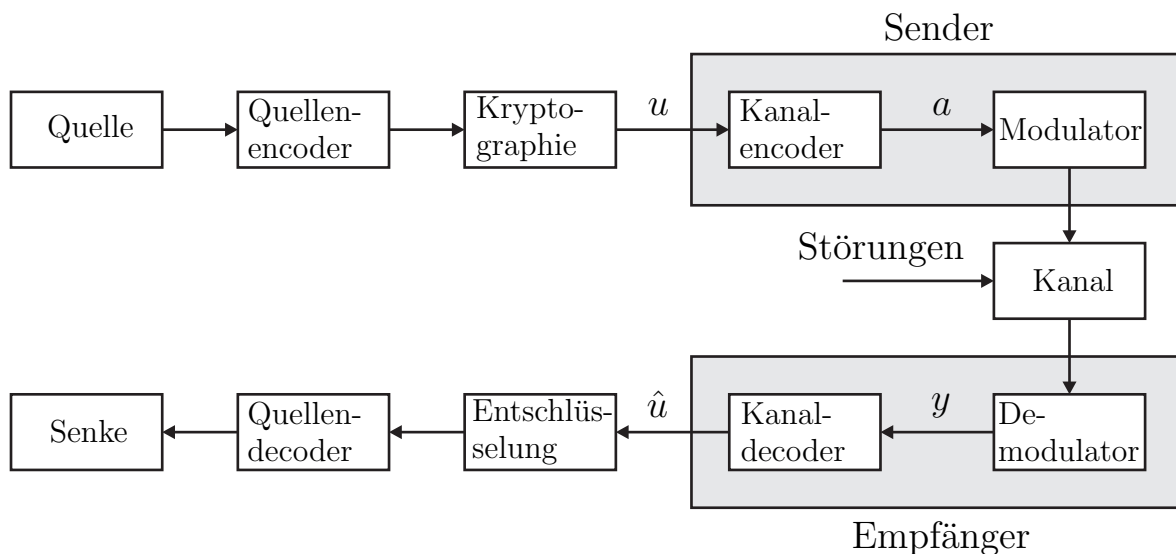


Abbildung 2.9: Nachrichtenübertragung ([Fri96] mit Ergänzungen)

2 Kommunikation verteilter Automobilanwendungen

Die *Quellencodierung* entzieht der Nachricht Redundanz und versucht die Nachrichtenlänge zu minimieren. Die Nachrichten werden so komprimiert, dass zwar keine Information verloren geht und somit eine perfekte Wiedergewinnung der Nachrichten möglich ist, aber dafür wird die Anzahl der zu übertragenden Symbole reduziert. Durch Quellencodierung wird also überflüssige Redundanz eliminiert und das Übertragungssystem entlastet. Dazu werden sogenannte optimale Codes, wie zum Beispiel Shannon-Fano- oder Huffman-Codes, verwendet.

Die *Kryptographie* ist die Codierung zur Verschlüsselung von Nachrichten, um sie für Unberechtigte unlesbar zu machen bzw. um zu verhindern, dass Nachrichten gefälscht oder vorgetäuscht werden können. Während durch Kanalcodierung Nachrichten auch im Fall von Störungen lesbar bleiben, sollen die verschlüsselten Nachrichten auch bei ungestörter Übertragung ohne Kenntnis des Schlüssels unlesbar sein. Ein gängiges Verfahren zum Verschlüsseln von Informationen ist das RSA-Verfahren.

Die *Kanalcodierung* fügt, im Gegensatz zur Quellencodierung, der Nachricht gezielt Redundanz hinzu, um so die Nachricht vor Störungen zu schützen und außerdem Übertragungsfehler zu korrigieren. Durch Störung der Übertragung entstandene Fehler im empfangenen Wort können erkannt beziehungsweise korrigiert werden. Für die Umwandlung der Zeichen aus der einen Menge in die andere Menge existiert eine Vorschrift, ein so genannter Code.

Die Kanalcodierung stellt Methoden und Verfahren zur Verfügung, mit denen Informationen mit einem Minimum an Fehlern von einer Quelle zu einer Senke übertragen werden können. Den eigentlichen Informationen wird sendeseitig kontrolliert Redundanz hinzugefügt, so dass bei der Übertragung entstandene Fehler empfangsseitig erkannt und korrigiert werden können. Damit lässt sich unter Umständen eine, verglichen mit uncodierter Übertragung, erhöhte Zuverlässigkeit der versendeten Daten erreichen. Ferner sind Störungen kompensierbar, die durch andere Maßnahmen, wie beispielsweise durch eine Erhöhung der Sendeleistung, prinzipiell nicht zu verhindern wären. Da die Kanalcodierung eine zentrale Eigenschaft sowohl bei der Entwicklung als auch bei der Bewertung von Kommunikationssystemen ist, wird sie im folgenden Abschnitt detaillierter beschrieben. Hierbei wird weitgehend auf eine vollständige, mathematisch exakte Darstellung, wie sie in [Fri96], [PS04] und [LC83] zu finden ist, verzichtet. Stattdessen werden die zentralen Zusammenhänge, die für das Verständnis der im Nachfolgenden präsentierten Kommunikationssysteme nötig sind, aus eben diesen Quellen zitiert.

2.3.2 Kanalcodierung

Fehlererkennung und -korrektur

Wird im Sender und im Empfänger das gleiche Nachrichtenalphabet verwendet, so kann die Übertragung als Überlagerung des Sendewortes mit einem *Fehlerwort* interpretiert werden:

$$y = a + e$$

Das Empfangswort y ist also die Summe aus dem gesendeten Codewort a und dem Fehlerwort (Fehlermuster) e .

Wenn die fehlerhaften Bits, also die Einsen im Fehlerwort e , unabhängig voneinander auftreten – was bei dem gedächtnislosen Kanal der Fall ist – liegen *Einzelfehler* vor. Treten die fehlerhaften Bits geballt auf – sie sind dann statistisch abhängig – so spricht man von Bündelfehlern. Ein Bündelfehler $e(x)$ der Länge t ist dadurch gekennzeichnet, dass höchstens t aufeinanderfolgende Stellen ungleich Null sind, also

$$e(x) = x_i b(x) \text{ mit Grad } b(x) < t, 0 \leq i \leq n - t$$

gilt.

Um die Erkennung oder Korrektur von Fehlern möglich zu machen wird ein Abstandsmaß zwischen Codewörtern eingeführt, das eine Aussage darüber erlaubt, wie stark sich zwei Codewörter voneinander unterscheiden.

Definition 2.6 (Hammingdistanz [Fri96]) Die Hammingdistanz $d_H(x, y)$ ist definiert als die Anzahl der Abweichungen zwischen den Komponenten von x und y . Sofern eine Null definiert ist, wird als Hamminggewicht $w_H(x)$ die Anzahl der Komponenten von x bezeichnet, die ungleich Null sind.

Automatic Repeat Request (ARQ)

Es ist zwischen zwei grundsätzlichen Prinzipien der Kanalcodierung zu unterscheiden, die sich für unterschiedliche Systeme eignen, abhängig davon, ob eine Information sehr schnell übertragen werden muss und ob ein Rückkanal verfügbar ist. Es handelt sich hierbei um Automatic Repeat Request (ARQ) und Forward Error Correction (FEC). In den folgenden Unterabschnitten wird auf die zentralen Eigenschaften dieser Verfahren, sowie auf ihre Eignung unter speziellen Randbedingungen eingegangen.

Bei Automatic Repeat Request-Verfahren werden die Übertragungsfehler nicht korrigiert, sondern es erfolgt empfangsseitig eine Beschränkung auf die

2 Kommunikation verteilter Automobilanwendungen

Erkennung von Fehlern. Wird hierbei ein Fehler in der übertragenen Nachricht entdeckt, so wird der Sender aufgefordert, die Nachricht erneut zu senden. Hierbei kann die Nachricht gegebenenfalls mit zusätzlicher Redundanz ausgestattet werden. Voraussetzung für die Verwendung von ARQ-Verfahren ist, dass ein Rückkanal existiert und dieser auch in der Lage ist, die an das System gestellten Echtzeitanforderungen einzuhalten, da sich durch das erneute, gegebenenfalls mehrfache, Versenden der Nachricht natürlich die Latenzzeit der Nachricht deutlich erhöht. Bei ARQ-Verfahren werden, im Vorgriff auf die im Folgenden gegebenen Definitionen und Diskussion, hauptsächlich Blockcodes verwendet, da sich Faltungscodes für die Fehlererkennung schlecht eignen und daher fast ausschließlich zur Fehlerkorrektur eingesetzt werden, siehe auch [Fri96].

Man unterscheidet drei grundsätzliche Prinzipien von ARQ-Verfahren, siehe auch [LC83]:

Stop-and-wait Bei einem stop-and-wait ARQ Verfahren wartet der Sender eine Bestätigung vom Empfänger ab, mit der dieser den korrekten oder fehlerhaften Empfang der Nachricht quittiert. Eine positive Rückmeldung (ACK) vom Empfänger signalisiert, dass der Codevektor erfolgreich empfangen wurde und der Sender überträgt den nächsten Codevektor. Eine negative Rückmeldung (NAK) vom Empfänger gibt an, dass der Empfangsvektor als fehlerhaft erkannt wurde und veranlasst den Sender zu einer erneuten Übertragung. Abbildung 2.10 zeigt ein solches stop-and-wait ARQ Verfahren.

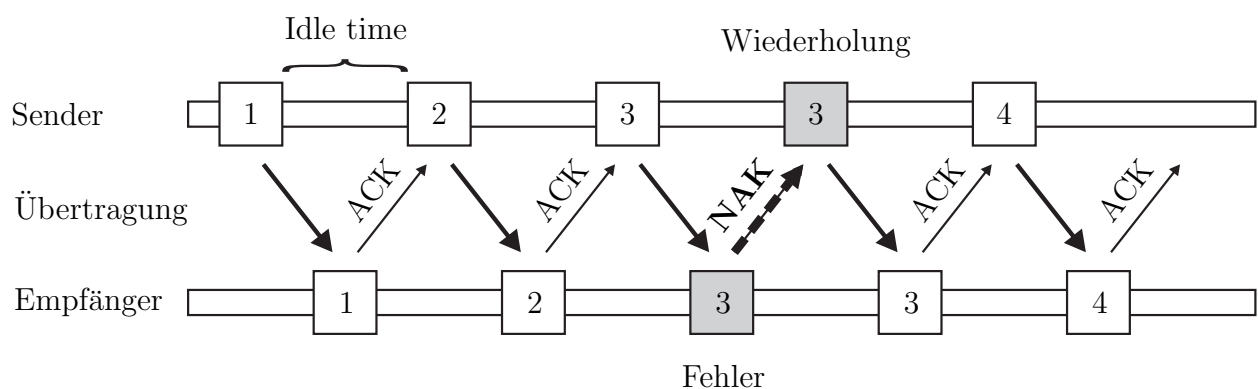


Abbildung 2.10: Schematische Darstellung eines stop-and-wait ARQ-Systems [LC83]

Go-back-N In einem go-back-N ARQ System werden die Codevektoren kontinuierlich übertragen. Der Sender wartet nach dem Senden eines Codevektors nicht die Bestätigung ab; sobald er einen Codevektor übertragen

2.3 Grundlagen zu Kommunikationsmedien

hat, beginnt er mit der Übertragung des nächsten Codevektors. Die Bestätigung für einen Codevektor erreicht den Sender nach einer bestimmten Zeitverzögerung. Dieses „round-trip-delay“ ist definiert als die Zeitspanne zwischen dem Senden eines Vektors und dem Empfang der entsprechenden Bestätigung. Während dieser Zeit wurden $N-1$ weitere Codevektoren übertragen. Wenn eine Fehlermeldung (NAK) empfangen wird, wiederholt der Sender die Übertragung des fehlerhaften Codevektors und aller folgenden. Abbildung 2.11 zeigt ein go-back-N ARQ System mit $N = 4$.

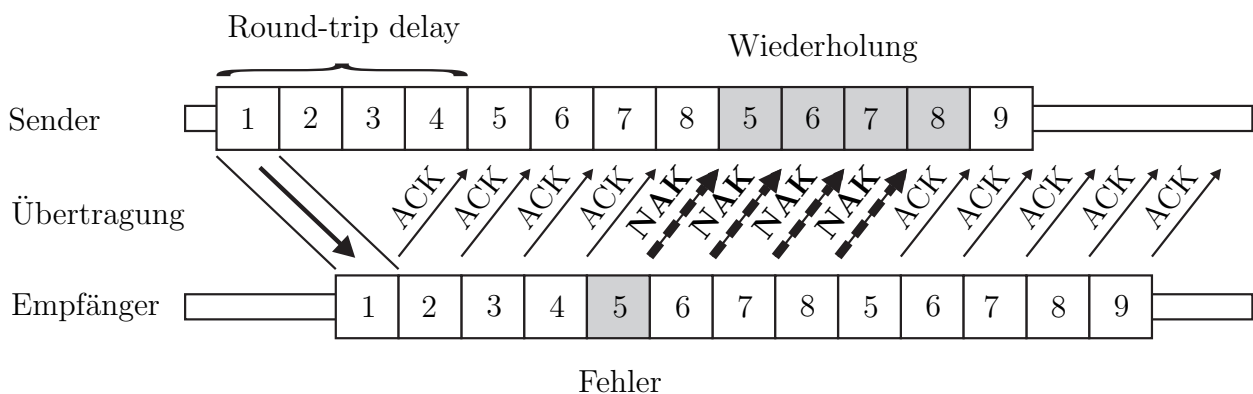


Abbildung 2.11: Schematische Darstellung eines go-back-N ARQ-Systems [LC83]

Selective-repeat Ein go-back-N ARQ Verfahren wird für großes round-trip-delay ineffektiv, da außer dem fehlerhaften Codevektor auch viele fehlerfreie Codevektoren erneut übertragen werden müssen. Dieser Nachteil kann mit einer selective-repeat ARQ Strategie überwunden werden. In einem selective-repeat ARQ System werden die Codevektoren ebenfalls kontinuierlich übertragen. Der Sender wiederholt nur den Codevektor, der als fehlerhaft rückgemeldet wurde. Im Empfänger müssen die Codevektoren anschließend wieder in ihre ursprüngliche Reihenfolge gebracht werden.

Forward Error Correction (FEC)

Ist die für den Einsatz von ARQ notwendige Voraussetzung, also das Vorhandensein eines schnellen Rückkanals nicht erfüllt, so kann mit Hilfe eines Forward Error Correction Verfahrens unter Umständen dennoch eine Verbesserung des Systemverhaltens erzielt werden. Im Gegensatz zu ARQ wird bei FEC die den Nutzdaten hinzugefügte Redundanz nicht dazu verwendet,

2 Kommunikation verteilter Automobilanwendungen

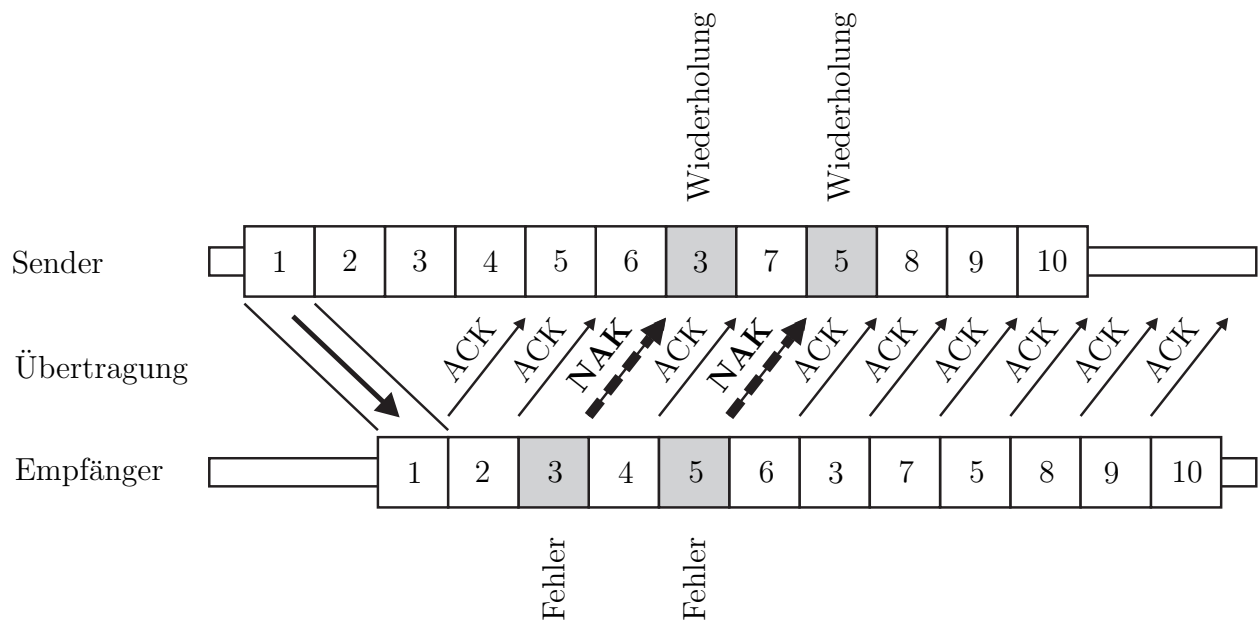


Abbildung 2.12: Schematische Darstellung eines selective-repeat ARQ-Systems [LC83]

Fehler zu entdecken, sondern diese zu korrigieren. Hierzu bildet der FEC-Algorithmus normalerweise ein fehlerhaftes Codewort auf das Codewort mit der geringsten Distanz ab. Diese Abbildung erfolgt aufgrund von Laufzeitabwägungen häufig heuristisch, zum Beispiel mit dem Berlekamp-Massey- oder dem Viterbi-Algorithmus. Bedingt durch diese Zuordnung kann jedoch auch der Fall eintreten, dass ein verfälschtes Codewort auf ein gültiges Codewort abgebildet wird, welches nicht der ursprünglich gesendeten Nachricht entspricht. In diesem Fall ist die Korrekturfähigkeit des Codes überschritten.

Vergleich ARQ und FEC

Bei ARQ-Verfahren ist die Fehlerrate unabhängig von der Kanalkapazität, während der Durchsatz von der Kanalkapazität abhängt. Bei FEC-Verfahren dagegen bestimmt die Kanalkapazität die Fehlerrate, wobei der Durchsatz davon unabhängig ist ([Fri96]). Die Restfehlerrate bei ARQ-Verfahren ergibt sich aus der Wahrscheinlichkeit, dass ein Fehler auftritt, der dennoch eine gültige CRC-Prüfsumme, bzw. ein gültiges Codewort liefert. Je mehr Redundanz für eine Fehlererkennung eingesetzt wird, desto geringer ist diese Restfehlerrate. Mit der Vergrößerung der Redundanz sinkt allerdings im umgekehrten Maß der Durchsatz der Datenübertragung. Einfluss auf den

Datendurchsatz hat außerdem die Kanalkapazität, beziehungsweise die Qualität der Übertragung. Verschlechtert sich der Kanal, wird bei ARQ-Verfahren häufiger die Wiederholung von Nachrichten notwendig. Damit sinkt der Datendurchsatz ebenfalls.

Hybride ARQ Verfahren

Vergleicht man die beiden Verfahren, kann man erkennen, dass ARQ auf sehr einfachen Prinzipien beruht und hohe Verlässlichkeit bietet. Allerdings haben ARQ-Systeme einen schwerwiegenden Nachteil. Steigt die Kanalfehlerrate an, werden bei allen drei ARQ-Verfahren die nötigen Wiederholungen immer häufiger. Der Datendurchsatz der Übertragung fällt bei steigender Kanalfehlerrate immer mehr ab. Dies kommt dadurch zustande, dass ab einer bestimmten Häufigkeit der Fehler die zunehmenden Wiederholungen die Auslieferung der Nachrichten bremsen. Der Effekt kommt beim stop-and-wait ARQ durch das Abwarten der round-trip-time ganz besonders zum Tragen.

FEC-Verfahren bieten einen konstanten Durchsatz unabhängig von der Fehlerrate des Kanals. Nach [LC83] haben FEC-Verfahren jedoch zwei Nachteile. Wird ein empfangener Codevektor als fehlerhaft erkannt, muss er dennoch decodiert und dem Empfänger ausgeliefert werden, ohne dass überprüft werden kann, ob die Decodierung das korrekte oder ein fehlerhaftes Codewort lieferte. Wird mit derselben Codierung eine Fehlererkennung sowie eine Fehlerkorrektur realisiert, ist die Wahrscheinlichkeit eines Decodierfehlers bei FEC viel größer als die Wahrscheinlichkeit eines unentdeckten Fehlers bei ARQ. Vergleichbare Systemsicherheit erreicht man bei FEC, indem man mehr Redundanz einsetzt.

Um eine hohe Zuverlässigkeit des Gesamtsystems zu erreichen, müssen lange leistungsfähige Codes eingesetzt werden und eine große Menge Fehlermuster muss korrigierbar sein. Dies macht die Implementierung der Decodierung schwierig und teuer⁵. Aus den genannten Gründen werden in Kommunikationssystemen die ARQ-Verfahren oft den FEC-Verfahren als Fehlerkontrolle vorgezogen. In Datenübertragungssystemen in denen kein Rückkanal zur Verfügung steht, oder in Systemen zur Datenspeicherung sind FEC-Verfahren jedoch trotz dieser Eigenschaften im Vorteil, da die Wiederbeschaffung der als fehlerhaft erkannten Daten nicht möglich ist.

Den Nachteilen beider Verfahren, ARQ und FEC, kann dadurch begegnet werden, das beide Methoden richtig kombiniert werden. Eine solche Kombination der beiden grundlegenden Methoden wird als hybrides ARQ Verfahren bezeichnet. Ein hybrides ARQ System besteht aus einem FEC-Verfahren, das

⁵An dieser Stelle ist herauszuheben, dass diese Aussage aus [LC83] stammt, und somit die Kriterien Kosten und Implementierungsaufwand nicht eins-zu-eins auf heutige Systeme übertragen werden können.

2 Kommunikation verteilter Automobilanwendungen

in ein ARQ System als Subsystem eingebettet ist. Die Funktion der Fehlererkennung (FEC) dient dazu, die häufigsten Fehler zu beseitigen und damit die Anzahl der Wiederholungen zu reduzieren. Eine solche Kombination von FEC und ARQ wird eine höhere Zuverlässigkeit liefern, als ein reines FEC-Verfahren und wird außerdem einen höheren Datendurchsatz ermöglichen als ein reines ARQ-Verfahren.

Block- und Faltungscodes

Die Implementierung von Fehlererkennung beziehungsweise Fehlerkorrektur bedeutet, wie zuvor bereits angedeutet, dass der zu versendenden Information gezielt Redundanz hinzugefügt wird. Hierfür werden in praktischen Anwendungen oft *Block-* oder *Faltungscodes* eingesetzt.

Definition 2.7 (Blockcode [Fri96]) Ein (n, k) -Blockcode ist die umkehrbar eindeutige, zeitinvariante und gedächtnislose Zuordnung zwischen den Infowörtern $u = (u_0, \dots, u_{k-1})$ und den Codewörtern $a = (a_0, \dots, a_{n-1})$. Die Menge der Codewörter heißt Code Γ .

Anwendung finden vor allem die linearen, zyklischen Blockcodes, da sich hier durch die algebraische Struktur des Codes die Implementierung vereinfacht. Eindeutig beschrieben werden diese Codes durch ihr Generatorpolynom. Die Qualität eines Codes hängt maßgeblich davon ab, wie stark sich alle Codewörter untereinander unterscheiden. Je größer die Minimaldistanz ist, desto besser ist ein Code.

Definition 2.8 (Minimaldistanz [Fri96]) Die Minimaldistanz d_{min} eines (n, k, d_{min}) -Blockcodes Γ ist definiert als die minimale Hammingdistanz zwischen allen Codewörtern:

$$d_{min} = \min\{d_H(a, b) \mid a, b \in \Gamma, a \neq b\}.$$

Anhand des Fehlermusters e , der Abweichung des Empfangswortes y der Länge n zu einem gültigen Codewort a der Länge k kann die Fähigkeit eines Blockcodes zur *Erkennung* sowie zur *Korrektur* von Fehlern in Anlehnung an [Fri96] definiert werden:

Definition 2.9 (Fehlererkennung [Fri96]) Ein (n, k) -Blockcode Γ erkennt höchstens t' Fehler, wenn für jedes Fehlermuster $e \neq 0$ mit $w_h(e) \leq t'$ das Empfangswort $y = a + e$ kein Codewort ist (error detection).

Zur Fehlererkennung wird ausschließlich überprüft, ob ein gültiges Codewort vorliegt.

Definition 2.10 (Fehlerkorrektur [Fri96]) Ein (n, k) -Blockcode Γ korrigiert t Fehler, wenn für jedes Fehlermuster e mit $w_h(e) \leq t$ die Maximum-Likelihood-Decodierung das richtige Codewort liefert (error correction).

Bei der Fehlererkennung wird die Entscheidung getroffen ob ein gültiges oder ungültiges Codewort vorliegt. Zur Fehlerkorrektur soll bei Empfang eines ungültigen Codewortes (Fehlermuster $e \neq 0$) nach einem der Decodierprinzipien die Entscheidung für ein gültiges Codewort getroffen werden. Die Fehlerkorrekturfähigkeit t' desselben Codes Γ ist sicher kleiner als seine Fähigkeit zur Fehlererkennung t .

Abhängig von der minimalen Hammingdistanz d_{min} lassen sich folgende Aussagen über Erkennungs- und Korrekturfähigkeit von Blockcodes treffen:

Satz 2.1 (Fehlererkennung [Fri96]) Ein (n, k, d_{min}) -Code Γ erkennt höchstens $t' = d_{min} - 1$ Fehler.

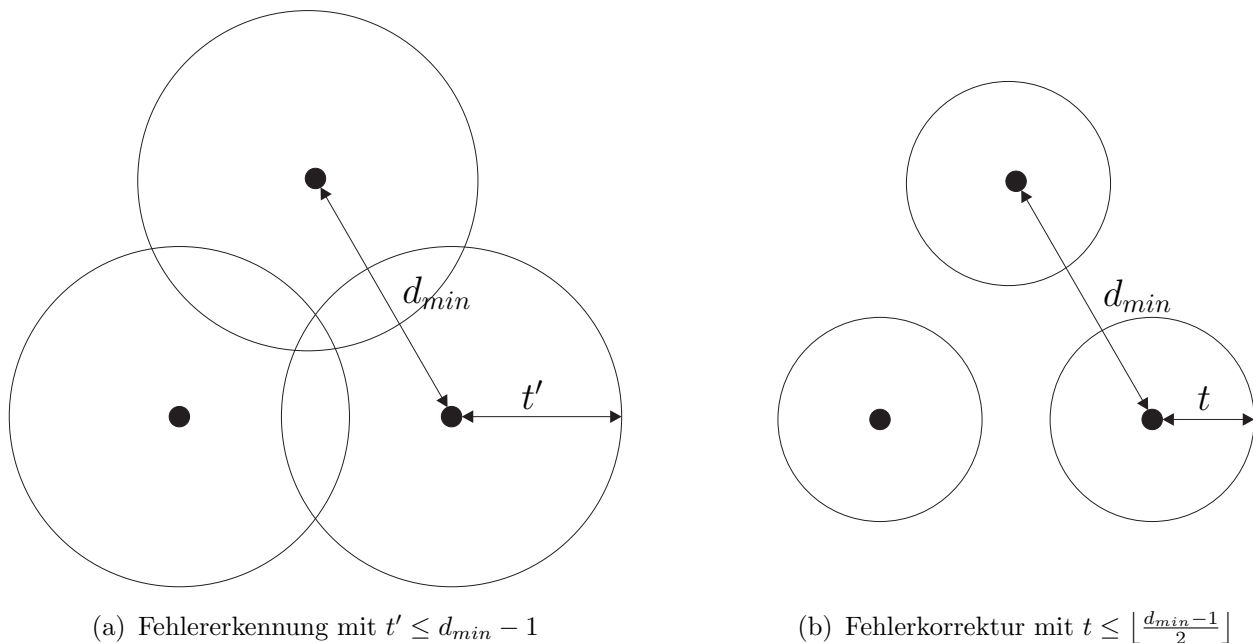


Abbildung 2.13: Fehlererkennung und Fehlerkorrektur im Signalraum

Satz 2.2 (Fehlerkorrektur [Fri96]) Ein (n, k, d_{min}) -Code Γ korrigiert t Fehler, sofern $2t + 1 \leq d_{min}$ ist.

Es lassen sich Fehlererkennung und -korrektur auch verbinden. Bis zu einer bestimmten Entscheidungsgrenze werden Fehler korrigiert, Abweichungen von gültigen Codewörtern die darüber hinausgehen, werden noch erkannt.

Um Fehlererkennung und -korrektur zu verbinden, müssen Korrekturfähigkeit t und Erkennungsfähigkeit t' in der Summe noch unterhalb der minimalen Hammingdistanz d_{min} liegen, siehe auch [Fri96]:

2 Kommunikation verteilter Automobilanwendungen

Satz 2.3 (Fehlererkennung und -korrektur) *Ein (n, k, d_{min}) -Code Γ kann gleichzeitig t Fehler korrigieren und t' Fehler erkennen ($t' \leq t$), sofern $t + t' + 1 \leq d_{min}$ ist.*

Von praktischer Bedeutung sind nur einige wenige Codeklassen, nämlich die Reed-Solomon (RS)- und Bose-Chaudhuri-Hocquenghem- (BCH)-Codes. Die RS- und BCH-Blockcodes sind Klassen von Blockcodes, die anhand von bestimmten Entwurfparametern konstruiert werden können. RS- und BCH-Codes können als Verallgemeinerung des jeweils anderen aufgefasst werden und haben daher ähnliche Eigenschaften. Der Hauptunterschied besteht darin, dass RS-Codes auf Folgen von Symbolen definiert sind, die aus mehreren Bit bestehen. Treten Burstfehler auf, so führen diese bei RS-Codes nur zu relativ einfach zu korrigierenden Symbolfehlern, wenn die Symbole nur ausreichend lang sind. Hinzu kommt, dass sich diese Symbole gut für Modulationsverfahren eignen. Treten hauptsächlich zufällige Fehler auf und findet die Übertragung unmoduliert statt, so sind die binären BCH-Codes im Vorteil, da RS-Codes ihre Vorteile nicht ausspielen können. Auch die Länge der zu übertragenden Nachricht geht in die Betrachtung ein. So sind RS-Codes für längere Nachrichten, BCH-Codes für kurze Nachrichten geeignet, da sich die Zusammenfassung von Bit zu Symbolen für lange Nachrichten effektiver bewerkstelligen lässt. Beispiele sind die Verwendung von RS-Codes in ATM- (124 Byte pro Nachricht) [KPS03] oder Powerline-Netzen (bis 200 kByte) [Bab07] während BCH-Codes in CAN (112 Datenbit)⁶ verwendet werden.

Im Gegensatz zu Blockcodes sind Faltungscodes gedächtnisbehaftet, was bedeutet, dass der von einem Infowort erzeugte Code nicht nur von diesem Wort, sondern auch von einer Anzahl von zuvor codierten Infowörtern abhängt. Zur Ermittlung des Codewortes wird diese Sequenz mit dem Generatorpolynom gefaltet. Diese Generatorpolynome werden nicht, wie zum Beispiel BCH-Codes, über analytische Verfahren konstruiert, sondern gesucht. Faltungscodes eignen sich besonders zur Verarbeitung von Zuverlässigkeitsinformationen, die der Demodulator zur Verfügung stellt. Die Decodierung von Faltungscodes erfolgt üblicherweise mit Hilfe des Viterbi-Algorithmus in Trellis-Diagrammen. Faltungscodes eignen sich sehr gut für den Einsatz in Systemen, die eine quasi-kontinuierliche Übertragung, also lange Sende- und Empfangsfolgen, benötigen und über einen Kanal versenden, der hauptsächlich Einzelfehler erzeugt. Treten Bündelfehler auf, so entstehen bei der Decodierung mit dem Viterbi-Algorithmus längere Bündelfehler [Fri96].

⁶Hier ist die Anzahl der Datenbit auf den Code bezogen, nicht auf die eigentliche Nutzdatenlänge der Nachricht, vergleiche Unterabschnitt 2.4.1.

Interleaving

Die Behandlung von Einzelfehlern ist mit den zuvor vorgestellten Codes gut möglich, mit RS-Codes sogar die Behandlung von Bündelfehlern, sofern sie innerhalb eines Symbols bleiben. Allgemein ist es aber möglich, die Sende- und Empfangsreihenfolge der Bit zu ändern und Bündelfehler damit zu entzerren. Für dieses *Interleaving* oder Codespreizung genannte Verfahren stehen mehrere Methoden zur Verfügung, beispielhaft soll hier das einfache Block-Interleaving vorgestellt werden.

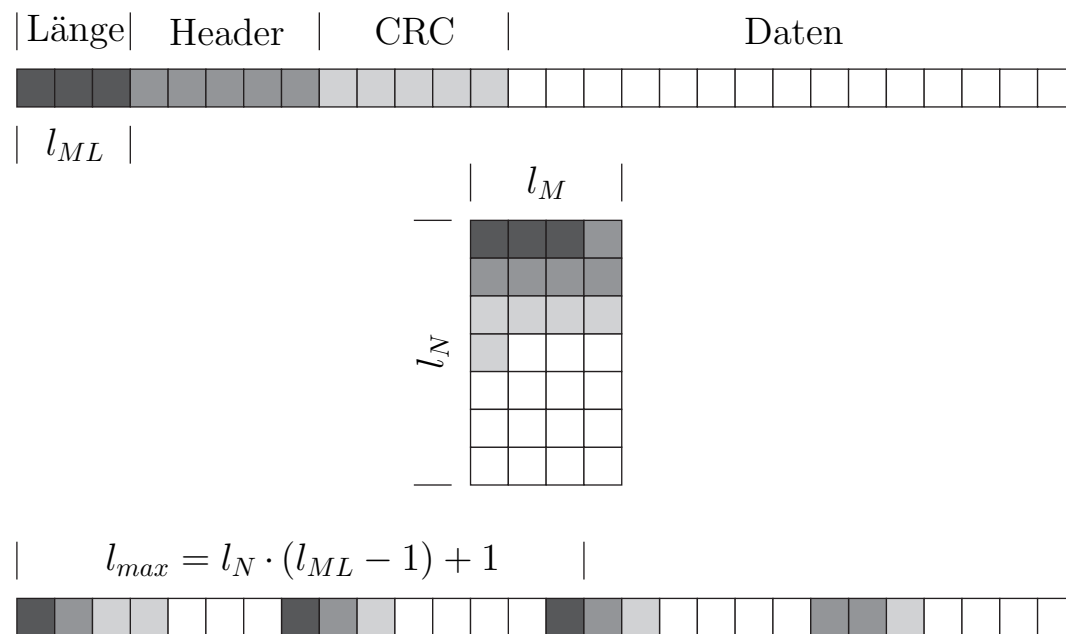


Abbildung 2.14: Block-Interleaver

In Abbildung 2.14 ist eine typische Nachricht in einem Kfz-Kommunikationssystem abgebildet. Diese Nachricht wird nun im Sender in Richtung von l_M in den Interleaver ein- und in Richtung von l_N aus dem Interleaver ausgelesen. Hierdurch werden Bit, die in der ursprünglichen Nachricht nebeneinander liegen, über die Nachricht verteilt. Im Empfänger läuft dieser Prozess in umgekehrte Richtung, um wieder die ursprüngliche Nachricht zu erhalten. Werden nun während der Übertragung mehrere auf dem Kanal aufeinanderfolgende Bit gestört, so wird dieser Bündelfehler gespreizt, falls seine Länge unter l_N liegt. Es liegen also nach dem Interleaving effektiv mehrere Einzelfehler vor.

Problematisch ist der Einsatz von Blockinterleavern bei Nachrichten variabler Länge. Ist die Nachrichtenlänge Teil der Nachricht selbst, in Abbildung 2.14 das Länge-Feld, so muss dieses Feld während dem Empfang der

2 Kommunikation verteilter Automobilanwendungen

Nachrichte ausgewertet werden. Daher ist es nicht möglich, die Länge des Interleavers a priori zu bestimmen.

Ist jedoch l_N fest, so kann der Zeitpunkt, zu dem das gesamte ML-Feld im Empfänger ausgewertet bestimmt werden, wie in Abbildung 2.15 angedeutet ist. Durch Variieren von l_M kann der Interleaver, wenn auch nicht optimal, an die Nachricht angepasst werden.

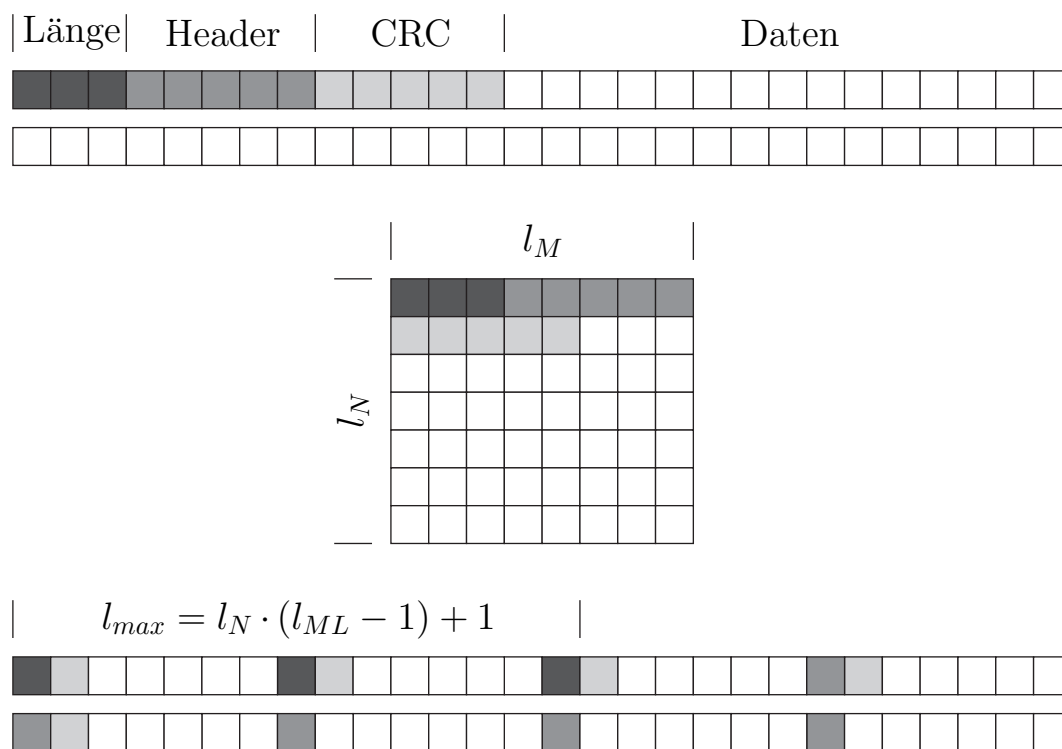


Abbildung 2.15: Anpassung eines Block-Interleavers an variable Nachrichtenlänge

Definition 2.11 (Effizienz eines Kommunikationssystems) Fügt ein Kommunikationssystem einer Nachricht N der Länge n_{netto} Bit zur Übertragung $n_{overhead}$ Bit hinzu, so beträgt die Effizienz des Kommunikationssystems

$$\eta(N) = \frac{n_{netto}}{n_{netto} + n_{overhead}}.$$

Bemerkung 2.3 In der Regel ist η eine Funktion von n_{netto} . Von den in Abschnitt 2.4 vorgestellten Kommunikationsmedien spielt der *Inhalt* der Nachricht für die Berechnung von η lediglich bei CAN eine Rolle, da hier Bitstuffing verwendet wird, siehe auch Unterabschnitt 2.4.1.

Der *automotive baseband* Kanal

Im Gegensatz zu dem in der Literatur oft betrachteten *additive white Gaussian noise* (AWGN)-Kanal, dessen Störungen über die Rauschleistungsdichte, beziehungsweise das Verhältnis zwischen Bit-Signalenergie und Rauschleistungsdichte, vollständig definiert sind, ist bei dem Einsatz im Kraftfahrzeug zusätzlich mit sogenannten Bündelstörern zu rechnen, die nicht nur ein Bit sondern eine Folge von Bit innerhalb einer Nachricht stören. Gründe für das Auftreten der Bündelstörer sind beispielsweise Einspritzung oder auch das Einschalten von Komfortelektronik.

Eine genaue Analyse der Kanaleigenschaften für die Verwendung von Powerline im Kraftfahrzeug findet sich in [HSH05]. Einfache Messungen bestätigten, dass für den im Rahmen dieser Arbeit verwendeten Kanal die Störungen bei Verwendung von *shielded twisted pair* (STP) Leitungen die Hauptenergie der Impulse auf circa 30 ns konzentriert ist, was bei einer Übertragungsgeschwindigkeit von 100 MBit/s der Länge von 3 Bit entspricht. Verursacht wurden diese durch die Einspritzung in Dieselmotoren, weshalb sie sowohl bei der Messung an einem Versuchsfahrzeug als auch bei der Messung an dem Injektionsprüfstand des Instituts⁷ gemessen wurden.

2.4 Stand der Technik

2.4.1 CAN

CAN (Controller Area Network) wurde bereits Mitte der Achtziger Jahre von Bosch in Kooperation mit Intel entwickelt und 1986 im Rahmen des SAE World Congress vorgestellt [KDL86]. CAN ist mittlerweile ein ISO Standard [ISO94] und wird außer in seinem ursprünglichen Einsatzbereich in der Automobilindustrie, wo er für Motorsteuerung und als Komfortbus Verwendung findet, in den verschiedensten Bereichen wie Industrieautomation und wenig sicherheitsrelevanten Flugzeuganwendungen (zum Beispiel im Airbus A380) eingesetzt. Für CAN existiert eine Fülle von einführender und weiterführender Literatur, zum Beispiel [Law99], [Ets94], aus denen auch Teile dieses Abschnittes entnommen sind.

CAN arbeitet nach dem Multi-Master Prinzip, in dem alle an den Bus angeschlossenen Geräte sendeberechtigt sind. Die Arbitrierung erfolgt dabei nachrichtenbasiert, was eine schnelle Übertragung wichtiger Nachrichten sicherstellen soll. Aus diesem Grund ist auch die maximale Nachrichtenlänge auf 8 Byte beschränkt. Die Geschwindigkeit (maximal 1 MBit/s) ermöglicht

⁷Eine genaue Beschreibung des Prüfstandes findet sich in [Bau06].

2 Kommunikation verteilter Automobilanwendungen

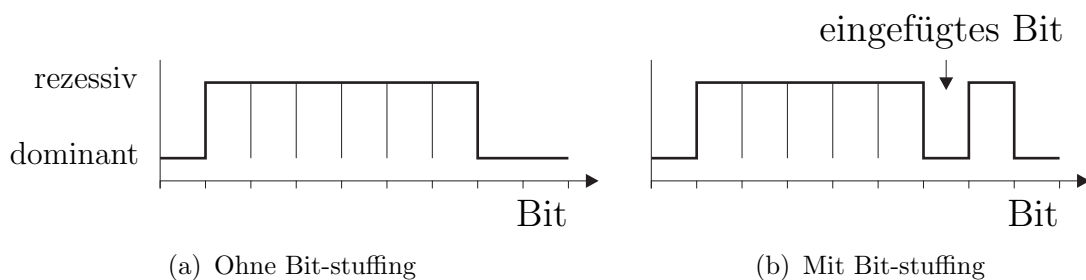


Abbildung 2.16: Bit-stuffing mit fünf Bit

den Einsatz in Systemen, die auf kurze Latenzzeiten angewiesen sind. An dieser Stelle sei jedoch angemerkt, dass aufgrund der beschränkten Ausbreitungsgeschwindigkeit die Länge des verwendeten physikalischen Mediums die maximale Geschwindigkeit auf dem Bus beschränkt. So ist die Verwendung eines 1 MBit/s CAN Busses bis maximal 40 m Buslänge zulässig. Im Automobilbereich wird daher oft ein 500 kBit/s CAN als High Speed Bus eingesetzt.

In der oben zitierten ISO-Norm ist ein elektrisches Übertragungsmedium für CAN definiert, das auf verdrehten Doppeladern basiert. Wichtigste Eigenschaft dieses Übertragungsmediums ist es, zwischen einem *dominanten* (in diesem Fall 0 V, logisch 0) und einem *rezessiven* (5 V, logisch 1) Pegel zu unterscheiden. Die Bezeichnungen werden im Zusammenhang mit der Realisierung als wired-AND Schaltung klar: Legt auch nur ein Teilnehmer am Bus 0 V an, so wird der Buspegel unabhängig von dem Pegel der anderen Teilnehmer auf 0 V gezogen. Dieser Pegel wird also als dominant bezeichnet. Diese Eigenschaft wird bei der Arbitrierung verwendet und ist daher eine Voraussetzung für alle CAN-Übertragungsmedien. Ein optisches Übertragungsmedium (dominant: hell, rezessiv: dunkel) ist ebenso denkbar, wird aber aus Kostengründen nicht im Kraftfahrzeug angewendet.

Nachrichtenformat

CAN verwendet zur Darstellung der Bits einen Non-Return-to-Zero Code, bei dem sich der Buspegel innerhalb eines Bits nicht ändert. Zur Synchronisierung muss daher sichergestellt werden, dass ausreichend viele Flanken in dem Signal vorkommen. Hierzu wird in den Teilen der Nachricht, deren Format nicht festgelegt ist, nach jedem fünften Bit mit gleichem Pegel ein entgegengesetztes in das Signal hinein-, „gestopft“ (*bit-stuffing*, siehe Abbildung 2.16). Für die nachfolgende Betrachtung des Nachrichtenformats wird Bit-stuffing jedoch nicht berücksichtigt.

Die Nachrichten, die in CAN versendet werden können, können in vier Gruppen unterteilt werden:

Die überwiegende Zahl der in einem CAN-Netzwerk versendeten Nachrichten sind *Datennachrichten*. Daher wird in den folgenden Betrachtungen Nachricht als Synonym für Datennachricht verwendet, soweit nicht anders angegeben. Sie enthalten neben dem unten näher beschriebenen Overhead auch Nutzdaten, siehe auch Abbildung 2.17.

Ein *Remote Frame* kann von einem beliebigen Empfänger einer Datennachricht gesendet werden. Durch ihn wird im Sender das Verschicken einer Datennachricht veranlasst. Es enthält alle Bestandteile einer Datennachricht bis auf das Datenfeld. Das RTR-Bit ermöglicht eine Unterscheidung zwischen Daten- und Datenanforderungstelegrammen.

Die *Fehlernachricht* dient zum gezielten Abbruch einer fehlerhaften Nachricht. Sie besteht aus sechs dominanten Bit, die eine gezielte Verletzung der Bit-stuffing Regel darstellen und acht rezessiven Bit. Durch die Verletzung der Bit-stuffing Regel wird die Übertragung der aktuellen Nachricht abgebrochen. Die rezessiven Bit am Ende ermöglichen es anderen Knoten, die die Störung später erkennen, ihren Error Frame ebenfalls komplett zu versenden. Error Frames dürfen nur von fehleraktiven Knoten versendet werden, siehe Unterabschnitt 5.3.3.

Die *Überlastnachricht* ermöglicht es einem Controller, das Senden der nächsten Nachricht um einige Bitzeiten zu verzögern, da die Berechnung der Daten noch andauert. In praktischen Anwendungen werden Overload Frames nicht verwendet.

Eine Darstellung einer CAN-Nachricht im so genannten Standard Format mit einem 11 Bit Identifier ist in Abbildung 2.17 gegeben. Dabei haben die einzelnen Felder die folgenden Bedeutungen:

Das *Start of Frame (SOF)-Bit* markiert den Beginn einer neuen Nachricht. Über die erste Flanke dieses Bits werden die Busteilnehmer synchronisiert.

Der *Identifier* ist eindeutig über den Inhalt der Nachricht bestimmt, womit auf Grund der eingangs erwähnten inhaltsbezogenen Priorisierung der Nachrichten auch die Priorität der Nachricht eindeutig festgelegt ist. Der Identifier ist im in Abbildung 2.17 dargestellten Standard-Format 11 Bit, im ebenfalls standardisierten *extended*-Format 29 Bit lang. Die Arbitrierung in CAN erfolgt mit Hilfe des Identifiers und des RTR-Bits. Die Übertragung des Identifiers erfolgt beginnend mit dem höchstwertigen Bit.

Das *Remote-Transmission-Request (RTR)-Bit* unterscheidet Daten- von Datenanforderungsnachrichten. Dabei ist das Bit für Datennachrichten dominant um diesen bei der Arbitrierung den Vorrang vor Datenanforderungsnachrichten zu geben.

Durch die vier niederwertigsten Bit des *Steuerfelds* wird die Länge des Datenfelds der Nachricht festgelegt. Das höchstwertige dieser Bit wird dazu verwendet, Standard- von Extended-Nachrichten zu unterscheiden.

2 Kommunikation verteilter Automobilanwendungen

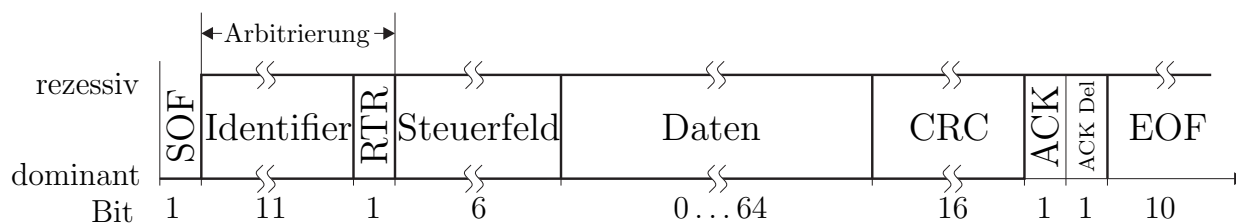


Abbildung 2.17: Aufbau einer Standard CAN-Nachricht mit den vom Sender angelegten Pegeln. Die Längen der Felder sind für den Fall angegeben, dass kein Bit-stuffing benötigt wird.

Das *Daten-Feld* enthält die eigentlichen Nutzdaten der Nachricht und kann zwischen null und acht Byte lang sein. Die Länge von eventuell kürzeren Daten ist dabei auf ganze Byte aufzufüllen. Die Zusammensetzung der Daten ist nicht vorgeschrieben. Es ist zum Beispiel möglich, Daten für verschiedene Empfänger, die oft gemeinsam auftreten, aus Effizienzgründen in einer Nachricht zu bündeln, wie zum Beispiel in [NRK04a] beschrieben.

In dem *CRC-Feld* kommt eine 15 Bit Prüfsumme zum Einsatz. Auf die Eigenschaften dieser Prüfsumme wird in dem Unterabschnitt zur Fehlererkennung genauer eingegangen. Abgeschlossen wird das CRC-Feld durch ein rezessives Bit.

Das *Bestätigungsfeld*, besteht aus ACK (*acknowledge*) und dem ACK-Delimiter (ACK Begrenzung). Das ACK-Bit wird von allen Knoten, die die Nachricht korrekt empfangen haben dominant gesendet, während der Sender selbst einen rezessiven Pegel anlegt. Ist der ACK-Pegel also dominant, so hat wenigstens ein Knoten die Nachricht richtig empfangen. Das rezessive Delimiter-Bit ist notwendig, um ein dominantes ACK-Bit von einer gleichzeitig beginnenden Fehlernachricht unterscheiden zu können.

Eine Nachricht endet mit 7 rezessiven Bit dem sogenannten *End-of-Frame (EOF)-Feld*, um allen Teilnehmern eine Interpretation der Nachricht zu ermöglichen, da diese Zeit ausreicht, um eine Fehlernachricht zu empfangen.

Nicht dargestellt ist der so genannte Inter-Frame-Space (drei Bit), eine Pause zwischen den Nachrichten, in der der Buspegel rezessiv sein muss. Zusammen mit dem EOF-Feld ergeben sich also mindestens 10 rezessive Bit am Ende einer Nachricht.

Arbitrierung

Da es sich bei CAN um ein Multi-Master Protokoll handelt, muss der Zugriff auf den Bus eindeutig geregelt werden, damit es nicht zu einer unbeabsichtigten Störung der Busteilnehmer untereinander kommt. Hierzu dient der

Identifiziert. Da der Identifier eindeutig über den Inhalt der Nachricht definiert ist und beim Entwurf des Systems sichergestellt werden kann, dass jede Nachricht im fehlerfreien Fall nur von einem Gerät gesendet werden kann, gibt es zu jeder Zeit genau eine zu versendende Nachricht mit der höchsten Priorität⁸. Jeder Knoten sendet den Identifier der von ihm zu verschickenden Nachricht beginnend mit dem höchstwertigen Bit und überprüft gleichzeitig, ob der gesendete und der tatsächlich auf dem Bus anliegende Pegel übereinstimmen. Sendet ein Knoten ein rezessives Bit während auf dem Bus ein dominantes anliegt, so weiß der Knoten, dass ein anderer Knoten eine Nachricht mit höherer Priorität versenden will und stellt die Übertragung ein. Da das RTR-Bit in die Arbitrierung miteinbezogen wird, besitzt eine Datenanforderungsnachricht eine höhere Priorität als eine Datenanforderungsnachricht gleichen Typs.

Fehlererkennung und -behandlung

Wie zuvor erwähnt wird bei CAN ein CRC-Code zur Fehlererkennung verwendet. Es handelt sich dabei um einen BCH Code mit $d_H = 6$, wie er in Unterabschnitt 2.3.2 vorgestellt wurde. Als Generatorpolynom wird dabei

$$g_{CAN} = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1 \quad (2.1)$$

verwendet. Da der Code lediglich zur Fehlerentdeckung verwendet wird, können also prinzipiell fünf beliebig verteilte Bitfehler und Burstfehler von bis zu 15 Bit Länge entdeckt werden. Durch die Verwendung von Bit-stuffing in Verbindung mit speziellen Fehlern kann die Hamming-Distanz aber in Ausnahmefällen auf HD 2 sinken.

Neben dem CRC werden auch eine Überwachung des Buspegels durch den sendenden Knoten (*bit-monitoring*) und die Einhaltung der Bit-stuffing Regel zur Fehlererkennung verwendet. Durch Bit-monitoring können globale Fehler, die den Buspegel auf den dominanten Pegel ziehen, erkannt werden. Des Weiteren können lokale Fehler im Sender erkannt werden, wobei Bit-monitoring hier als BIST zu behandeln ist und daher auch den in Abschnitt 2.2 genannten Einschränkungen unterliegt. Die Überwachung der Bit-stuffing Regel greift, wenn ein Knoten sechs gleichwertige Bit in Folge empfängt. Bei einer Verletzung dieser Regel durch sechs dominante Bit ist aber nicht klar, ob der Fehler bei dem Sender liegt oder eine Fehlermeldung von einem anderen Knoten gesendet wurde, die ja ebenfalls mit sechs dominanten Bit beginnt.

Eine genauere Beschreibung der Fehlerbehandlungsmechanismen wird in Unterabschnitt 5.3.3 gegeben.

⁸Der triviale Fall, dass kein Knoten eine Nachricht versenden will, muss hier nicht berücksichtigt werden. In diesem Fall bleibt der Bus leer, bis eine Nachricht in einem beliebigen Knoten vorliegt.

2 Kommunikation verteilter Automobilanwendungen

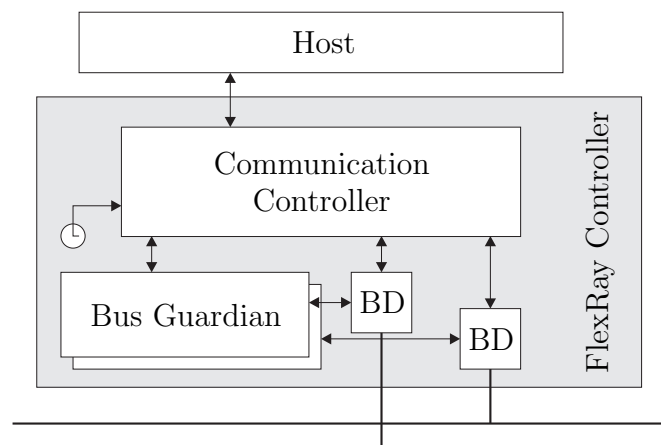


Abbildung 2.18: Vereinfachter Aufbau eines FlexRay-Controllers

2.4.2 FlexRay

FlexRay entstand ursprünglich aus einer Zusammenarbeit von Daimler-Chrysler und BMW. Die Tatsache, dass CAN für den Einsatz in sicherheitsrelevanter Kraftfahrzeugelektronik nur bedingt geeignet ist, veranlasste diese Firmen, gemeinsam mit Motorola und Philips das FlexRay Consortium zu gründen, dem mittlerweile eine große Zahl von Automobilherstellern und Zulieferern angehört. Auf der von dem FlexRay Consortium herausgegebenen Spezifikation des Protokolls in der Version 2.1 [Fle05b] beruhen die Aussagen dieses Unterabschnittes.

Im Gegensatz zu CAN basiert FlexRay auf *time-division multiple access* (TDMA) um den Zugriff auf das gemeinsame Medium zu regeln. Das hierzu nötige Protokoll wird größtenteils auf dem Communication Controller abgearbeitet, siehe auch Abbildung 2.18. Der optionale Bus Guardian gewährleistet die Einhaltung des Zeitschemas auch im Fehlerfall. Hierfür benötigt er lediglich Wissen über die Zeitschlitze, in denen der Communication Controller senden darf. Der Bus Guardian verfügt nicht zwingend über eine eigene Uhr und wird über den Communication Controller synchronisiert. Diese Synchronisation darf jedoch nur in engen Grenzen erfolgen, wodurch einerseits eine Fehlertrennung zwischen Communication Controller und Bus Guardian gewährleistet wird und andererseits die Implementierung des Bus Guardians vereinfacht wird [Fle04]. Der Buszugriff erfolgt über den Bustreiber (Bus Driver, BD). Diese Teile dürfen gemeinsam auf einem Chip untergebracht werden.

FlexRay lässt, neben der Busstruktur auch eine Sternstruktur mit aktiven und passiven Sternen, sowie eine Mischung aus Stern- und Busstruktur zu. Dabei dürfen maximal zwei aktive Sterne zwischen zwei beliebigen Steuer-

geräten des Netzwerks liegen [Fle05a]. Diese aktiven Sterne dürfen dabei wie normale Knoten auch, optional Bus Guardians enthalten [RLH05]. Da ohne Bus Guardian die FlexRay nicht besser für sicherheitsrelevante Anwendungen geeignet ist als CAN, wird im Folgenden davon ausgegangen, dass jeder Knoten in dem betrachteten Netzwerk im Fehlerfall durch einen Bus Guardian sicher vom funktionierenden Rest des Netzwerkes getrennt werden kann. Ob sich der Bus Guardian in dem Knoten oder in einem aktiven Stern befindet, spielt hierfür keine Rolle. FlexRay unterstützt auch eine redundante Auslegung des Kommunikationsmediums, wobei auf den beiden Kanälen andere Topologien verwendet werden können.

Um ein höheres Maß an Flexibilität zu erlauben, gestattet es FlexRay, neben dem statischen Segment, in dem die Sendezeit mit Hilfe des TDMA Verfahrens und einem zur Entwicklungszeit festgelegten Zeitplan zwischen den Geräten aufgeteilt wird, auch ein dynamisches Segment in der Kommunikation zuzulassen. In diesem Segment dürfen prinzipiell alle Geräte Nachrichten ereignisdiskret versenden. Im Unterschied zu dem statischen Segment besteht aber im dynamischen Segment kein Schutz durch den Bus Guardian, weshalb sicherheitsrelevante Kommunikation in dem dynamischen Segment vermieden werden sollte. Das dynamische Segment ist in einem FlexRay *Cycle* nach dem statischen Segment des gleichen Cycles angeordnet. Im Anschluss folgt wieder das statische Segment⁹, siehe auch Abbildung 2.19.

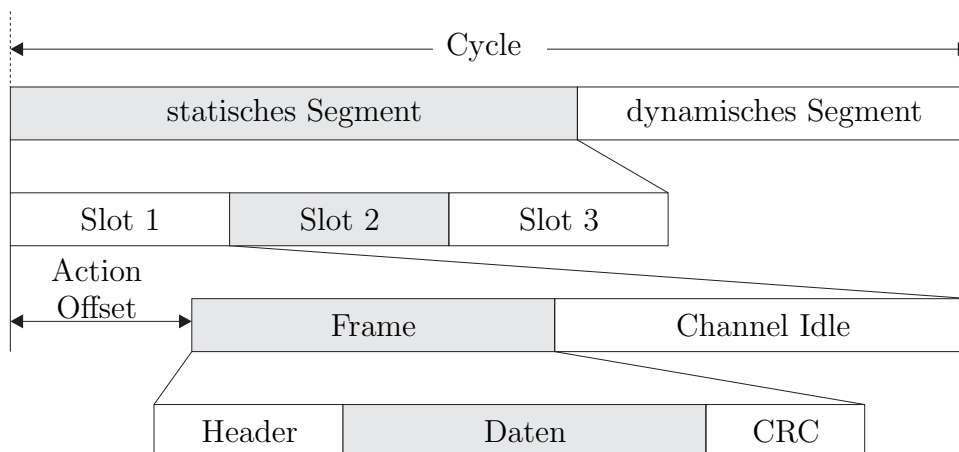


Abbildung 2.19: Vereinfachte Darstellung eines Cycles in FlexRay

Innerhalb des statischen Segments befinden sich die den einzelnen Geräten zugeordneten *Slots*. Ein Knoten darf hierbei mehrere Slot besitzen, wobei die Länge der Slots aber für alle Geräte gleich ist. Innerhalb des Slots beginnt nach dem so genannten *Action Offset* die eigentliche Nachricht (*Frame*). Der

⁹Symbol Window und Network Idle Time werden hier der Übersichtlichkeit halber nicht betrachtet.

2 Kommunikation verteilter Automobilanwendungen

Action Offset wird benötigt, um sicherzustellen, dass alle Communication Controller und Bus Guardians des Netzwerkes den Slot dem richtigen Knoten zuordnen. Wird nicht der gesamte Slot für die Übertragung der Nachricht benötigt, so folgt auf den Frame eine Channel Idle Phase, in der keine Daten versendet werden.

Der Frame beginnt mit dem 35 Bit langen Header. Dieser enthält neben 24 Bit Steuerdaten wie einem Nachrichten-Identifikationsfeld und der Länge des Datenfeldes auch einen 11 Bit CRC, der es ermöglicht, Veränderungen im Header, die während der Übertragung entstehen, zu erkennen. Auf den Header folgen die Daten, deren Länge in zwei-Byte Schritten von 0 Byte bis 254 Byte verändert werden kann. Die gesamte Nachricht wird durch den abschließenden 24 Bit CRC gesichert, der bis zu einer Datenlänge von 248 Byte eine Hammingdistanz von $d_H = 6$, darüber von $d_H = 4$ gewährleistet.

Die fundamentale Einheit für die Zeitrechnung innerhalb eines FlexRay Knotens ist der Microtick, der, eventuell skaliert, direkt aus dem Oszillator des Controllers übernommen wird. Da diese Microticks Controller-spezifischer Drift und anderen Einflüssen unterworfen sind, werden für die Berechnung der Cyclelänge die so genannten Macroticks verwendet. Macroticks stellen dabei die kleinste netzwerkweit synchronisierte Einheit dar. Für eine detaillierte Darstellung der in FlexRay verwendeten Synchronisationsmechanismen sei hier auf [Fle05b] verwiesen.

Eine Fehlerentdeckung findet in FlexRay nur in Bezug auf Synchronisierung und syntaktische Fehler statt. Die Fehlerbehandlung wird größtenteils auf die höheren Ebenen delegiert, siehe auch Unterabschnitt 5.3.3.

2.4.3 Systeme aus anderen Anwendungsbereichen

TTP/C

Die *time-triggered architecture* (TTA) setzt, ähnlich wie FlexRay auf TDMA um den Buszugriff zu regeln. Als Protokoll für sicherheitsrelevante Anwendungen kommt dabei das Time-Triggered Protocol / Class C (TTP/C) zur Anwendung. Neben TTP/C existiert auch TTP/A als einfacher Sensor/Aktorbus, auf den an dieser Stelle nicht näher eingegangen werden soll. Da der Einsatz von TTP/C vor allem im Rahmen von TTA sinnvoll ist, wird im Folgenden, falls nicht explizit anders formuliert, der Begriff „TTP“ als Synonym für „TTP/C als Bestandteil von TTA“ verwendet.

TTP wird bereits in mehreren Anwendungen im Bereich der Luftfahrt eingesetzt, wie zum Beispiel bei der Triebwerkssteuerung bei manchen F16 Modellen (General Electric) sowie der Kabinendrucksteuerung des Airbus A380 (Hamilton Sunstrand/Nord Micro). Da sich TTP prinzipiell auch für den Ein-

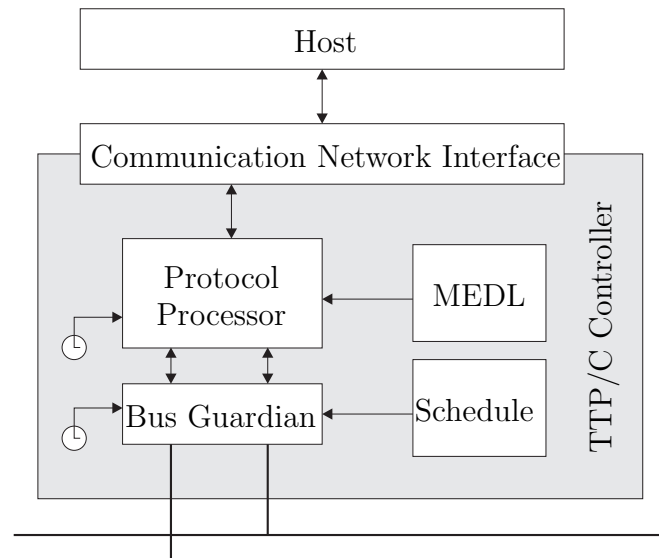


Abbildung 2.20: Vereinfachter Aufbau eines TTP-Controllers, mit Änderungen aus [PK99]

satz im Kraftfahrzeug eignet, werden die wichtigsten Eigenschaften an dieser Stelle erwähnt. Die Informationen dieses Abschnitts sind hauptsächlich [Rus03], [TTT03] sowie [AHM04] entnommen.

TTP entstand ursprünglich im Rahmen des MARS Forschungsprojekts der Universität Wien. Fokus waren von Anfang an sicherheitsrelevante Anwendungen im Automobil- und Luftfahrtbereich, weshalb die Wahl auf TDMA als Buszugriffsmechanismus fiel, da es nach Ansicht der TTP Entwickler die zeitliche Fehlerisolierung vereinfacht [Kop98]. Die einzelnen Knoten verfügen wie bei FlexRay über lokales Wissen über den Buszugriff, das heißt in der message descriptor list (MEDL) eines Knotens sind lediglich Informationen über die Sende- beziehungsweise Empfangszeiten des eigenen Knotens gespeichert. Die Struktur eines TTP-Controllers ist der eines FlexRay-Controllers ähnlich und in Abbildung 2.20 dargestellt. Im Unterschied zu FlexRay dürfen Bus Guardian und Protocol Processor nicht auf einem Chip integriert werden.

Das Communication Network Interface (CNI) stellt die Verbindung zwischen dem TTP-Controller und der eigentlichen Anwendung, die auf dem Host läuft, dar. Die Anwendung schreibt dazu Daten in zuvor fest definierte Speicherbereiche, die in der MEDL gespeichert sind. Die Länge dieser Speicherbereiche ist wie die Reihenfolge der Sendeslots a priori festgelegt, was eine effiziente Nutzung des vorhandenen Speicherplatzes ermöglicht. Durch ihren Ort im CNI-Speicher sind Daten also eindeutig identifizierbar, weshalb auf Identifier verzichtet werden kann. Dies schließt also eine Ursache des Masquerading-Fehlers aus und trägt so zur Zuverlässigkeit des Gesamt-

2 Kommunikation verteilter Automobilanwendungen

systems bei.

Der Protocol Processor ist das Herzstück des TTP-Controllers und ist unter anderem für die Ausführung der in den folgenden Abschnitten beschriebenen höheren Protokolldienste zuständig. Hierzu benötigt er eine eigenständige Zeitbasis, Zugriff auf die MEDL sowie, durch den Bus Guardian, Zugriff auf das physikalische Medium.

Der Bus Guardian überwacht das Sendeverhalten des Protocol Processors und ist in TTP im Gegensatz zu FlexRay nicht optional¹⁰. Um das unerlaubte Senden eines Knotens zu verhindern, benötigt der lokale Bus Guardian Kenntnis über die Sendeslots des eigenen Knotens sowie eine eigene externe Uhr, welche aus einer von der MEDL unabhängigen Quelle, der Schedule in Abbildung 2.20, bezogen wird. Die Uhr kann, falls der Bus Guardian sie nicht selbst synchronisieren kann, in regelmäßigen Abständen mit dem Protocol Processor abgeglichen werden um die Drift der einzelnen Uhren auszugleichen. Diese Korrektur kann jedoch nur innerhalb festgelegter Grenzen erfolgen da somit eine Fehlerabgrenzung zwischen Protocol Processor und Bus Guardian gewährleistet werden kann. Bus Guardians sind in TTP eine separate Einheit, was zu einer höheren Fehlerabdeckung führt, als dies bei einer gemeinsamen Implementierung von Protocol Processor und Bus Guardian der Fall wäre [Kop01].

Der zentrale Bus Guardian im aktiven Sternpunkt benötigt im Gegensatz zum lokalen Bus Guardian den gesamten Sendeplan des Systems. Es ist in diesem Fall auch darauf zu achten, dass die beiden Kanäle von verschiedenen Bus Guardians abgesichert werden, um einen SPOF auszuschließen, siehe auch Abbildung 2.21.

TTP unterstützt wie FlexRay sowohl Bus- und Sternanordnung der ECUs als auch Mischstrukturen aus Bus und Stern. Im Falle der Sternstruktur kommunizieren die ECUs über einen aktiven Stern (Sternkoppler). Passive Sternkoppler wie in FlexRay sind in TTP nicht vorgesehen. Die Sternstruktur bietet gegenüber der Busstruktur den Vorteil, dass das System robuster gegenüber SOS-Fehlern wird und sollte daher für Systeme verwendet werden, die die fail-operational Anforderung erfüllen müssen, siehe auch [DHSZ03]. Auffällig ist dabei, dass TTP sich mit einem Bus Guardian im Sternpunkt begnügt, während FlexRay jeweils einen Bus Guardian pro Knoten im Sternpunkt benötigt [Rus03]. Dadurch ist es möglich, den Bus Guardian im Sternpunkt bei gleichem Preis aufwändiger zu gestalten und dadurch zum Beispiel genauere Uhren zu verwenden.

¹⁰An dieser Stelle ist die Spezifikation [TTT03] nicht eindeutig: Einerseits ist die Rede davon, dass ein Knoten zwingend einen Bus Guardian besitzen muss, andererseits ist in der Darstellung der Sternstruktur der Bus Guardian lediglich im aktiven Stern eingezeichnet.

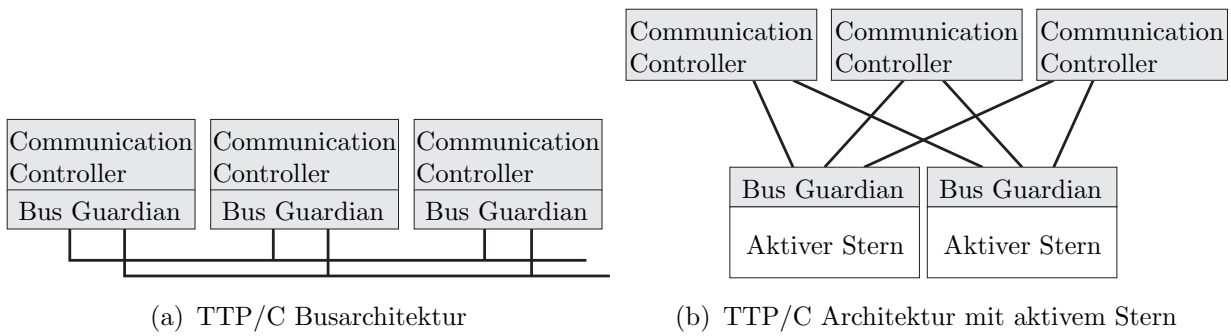


Abbildung 2.21: Vergleich der wichtigsten Strukturen für TTP/C-Netzwerke

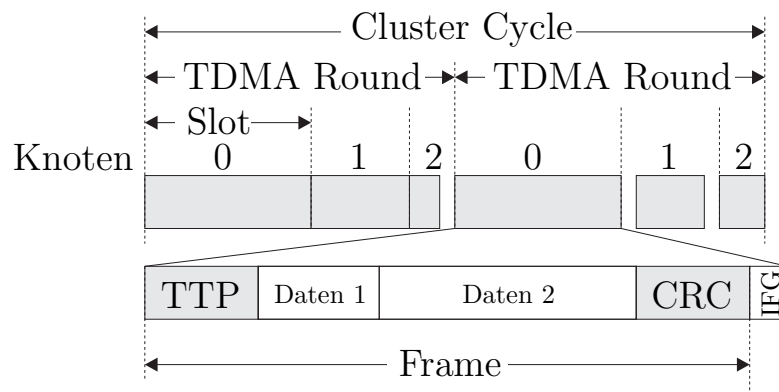


Abbildung 2.22: TTP Frame

Nachrichtenformat Da die Nachrichten in TTP bereits durch ihren Zeitschlitz eindeutig gekennzeichnet sind, ist die Verwendung von Identifiern wie bei CAN nicht notwendig. Identifier können innerhalb der Nachricht verwendet werden, um der Anwendung die Zuordnung der Daten zu ermöglichen oder um den Zeitschlitz eines Gerätes effizienter zu nutzen. Das Fehlen eines Identifiers erhöht die Effizienz der Nachrichtenübertragung und erlaubt das in Abbildung 2.22 gezeigte Format.

Die in der MEDL abgelegte Zeitspanne umfasst dabei einen Cluster Cycle, der in einzelne TDMA Rounds unterteilt ist. Innerhalb einer TDMA Round besitzen Knoten einen Slot, in dem sie senden dürfen. Am Ende des Cluster Cycles beginnt der Ablauf von Neuem.

Die Sendezeit innerhalb eines Slots ist aufgeteilt in die *inter frame gap* (IFG), einer Pause zwischen den Frames, und den eigentlichen Frame. Für die minimale Länge der IFG wird in [Kop01] 5 μ s–20 μ s angegeben, was bei einer Busgeschwindigkeit von 10 MBit/s etwa 50 bis 200 Bitzeiten entspricht. Innerhalb eines Frames können mehrere Nachrichten untergebracht werden die von den Empfängern getrennt werden können. Ihre gesamte Länge darf

2 Kommunikation verteilter Automobilanwendungen

allerdings 236 Byte nicht überschreiten¹¹ [Kop01]. Das CRC-Feld am Ende des Frames ist 3 Byte lang, wobei das verwendete Polynom von der Nachrichtenlänge abhängt und laut Spezifikation eine Hammingdistanz von $d_H = 6$ aufweisen muss. Rushby [Rus03] bezeichnet diesen CRC als äquivalent zu digitaler Signatur der Nachrichten, da ein Empfänger davon ausgehen kann, dass eine Nachricht mit korrektem CRC nicht verfälscht wurde. Bei dem CRC handelt es sich normalerweise um einen so genannten erweiterten CRC, bei dem der Zustand des Geräts bei der Berechnung mit berücksichtigt wird. Hierdurch kann ohne weiteren Overhead überprüft werden, ob Sender und Empfänger sich im gleichen Zustand befinden. Wird der Zustand explizit mit übertragen, so erhöht sich der Overhead natürlich.

Bei einer Gesamtlänge eines Frames von 240 Byte steht noch 1 Byte für Informationen über die Art des Frames und andere Informationen im TTP-Feld zur Verfügung.

Uhrensynchronisation Die Synchronisation der Uhren ist wie bei allen auf TDMA basierenden Kommunikationsmedien auch bei TTP wichtig für den zuverlässigen Betrieb. Ein Knoten, dessen Uhr nicht mit denen der anderen Knoten synchronisiert ist, könnte ohne weitere Maßnahmen wie zum Beispiel Bus Guardians die Kommunikation im gesamten Netzwerk stören. Aber auch im fehlerfreien Fall hängt die Effizienz der Übertragung von der Synchronisation der einzelnen Uhren ab, da bei besserer Synchronisation weniger Puffer (IFG) zwischen den Frames liegen muss, um den Drift der lokalen Uhren zu kompensieren.

Die Synchronisation der Uhren in TTP erfolgt mit Hilfe des Welch-Lynch Algorithmus. Hierbei wird ein fehlertoleranter Mittelwert (fault-tolerant midpoint, [Rus03]) gebildet, auf den alle fehlerfreien Steuergeräte synchronisiert werden. Hierzu werden die Werte geordnet und für n Knoten im Netzwerk, das t Fehler tolerieren soll, der Mittelwert des t -ten und $(n - t)$ -ten Wertes gebildet. Damit ist sichergestellt, dass die weit vom Mittelwert der korrekten Uhren abweichenden Werte der inkorrekten Uhren den Mittelwert nicht beeinflussen. Da TTP von der Ein-Fehler-Hypothese ausgeht, ist hier $t = 1$ gewählt. Für $n \geq 4$ ist daher sichergestellt, dass der Algorithmus eine korrekte Synchronisation der Uhren ermöglicht. Aus Gründen der Skalierbarkeit ist der implementierte Algorithmus unabhängig von n . Es ist für die Berechnung des fehlertoleranten Mittelwertes ausreichend, die zwei kleinsten und die zwei größten Werte, also insgesamt $2t$ Werte, zu kennen. TTP verwendet nur die

¹¹In anderen Dokumenten (zum Beispiel [Kop98] und [PK99]) ist von einer maximalen Datenlänge von 128 Bit mit 21 Bit Overhead die Rede. Da die Spezifikation in der Version 1.1 [TTT03] keine explizite Aussage zum genauen Frame-Aufbau macht, ist davon auszugehen, dass hier auf eine ältere Implementierung Bezug genommen wird.

vier *aktuellsten* Werte für die Uhrensynchronisation, was ebenfalls ausreicht, um einen Einfachfehler zu tolerieren, da hierdurch ebenfalls sichergestellt werden kann, dass höchstens einer dieser Werte fehlerhaft ist.

Jeder Knoten berechnet lokal anhand der Abweichung des Empfangszeitpunktes einer Nachricht von dem von ihm berechneten Soll-Empfangszeitpunkt einen Korrekturterm für jeden Knoten im Netz. Dies ist möglich, da jeder Nachricht eindeutig der sendende Knoten zugeordnet werden kann. Aus Kostengründen verfügt nicht jeder Knoten über eine genaue Uhr. Knoten mit einer solchen Uhr sind in der MEDL mit dem Synchronisation Flag (SYF) gekennzeichnet¹² und nur sie werden zur Synchronisation herangezogen. Die Synchronisation wird durch einen Frame angestoßen, bei dem das ClkSyn-Flag in der MEDL gesetzt ist. Die Synchronisation benötigt also keine Zeit auf dem Bus und erhöht damit den Overhead nicht.

Abschließende Betrachtung TTP ist im Vergleich zu FlexRay stärker an den Bedürfnissen sicherheitsrelevanter Systeme orientiert. Es steht außer Frage, dass TTP prinzipiell die Anforderungen für sicherheitsrelevante Systeme im Kraftfahrzeug erfüllen kann. Durch die Verwendung von TDMA zur Regelung des Buszugriffs und die a priori festgelegte Sendereihenfolge der Geräte kann im fehlerfreien Fall eine worst-case Aussage über die Latenzzeiten der einzelnen Nachrichten gemacht werden. Fehlerhafte Geräte werden durch die weitgehend unabhängigen Bus Guardians daran gehindert, die Kommunikation der fehlerfreien Geräte zu stören. Durch den Group Membership Algorithmus, der in Unterabschnitt 5.3.3 näher erläutert wird, wird zudem sichergestellt, dass vereinzelte Empfangsfehler nicht zu einem inkonsistenten Zustand des Systems führen. Durch die Verwendung einer Sternstruktur werden diese Eigenschaften noch verbessert. Diese Sicherheit hat jedoch auch ihren Preis: TTP-Controller sind mit ca. € 40.-¹³ um ein Vielfaches teurer als CAN-Controller. Außerdem müssen, wie bereits bei FlexRay diskutiert, die Anforderungen des ereignisdiskreten Systems Kraftfahrzeugelektronik und des Zeitplans des Kommunikationsmediums angeglichen werden.

Kritisch ist auch die Fehlerhypothese zu betrachten. Wie in Abschnitt 2.3.2 angeführt wurde, sind EMI Störungen, also kippende Bit, im Kraftfahrzeug vor allem auf Einspritzung und Zündung zurückzuführen. Aufgrund der relativ kurzen Zykluszeit von circa 1 ms erscheint es bei realistischen Nachrichtenlängen und Teilnehmerzahlen zumindest theoretisch sehr wahrscheinlich, dass mehr als der in der Fehlerhypothese angegebene eine Fehler

¹²Die Uhren der anderen Knoten werden als *slave clocks* bezeichnet.

¹³Über austriamicrosystems bei Abnahme von 500 Stück laut Internetseite. Einzelpreis: € 140.- Stand: Juli 2006

2 Kommunikation verteilter Automobilanwendungen

pro zwei Rounds auftritt. Diese Annahme müsste jedoch mit der entsprechenden Bitübertragungsschicht unter realistischen Umweltbedingungen verifiziert werden.

TTP erscheint aber insgesamt als aus technischer Sicht für den Einsatz in sicherheitsrelevanter Kraftfahrzeugelektronik gut geeignet, vor allem, wenn die Sternarchitektur verwendet wird.

SAFEbus

Bei SAFEbusTM handelt es sich um die erweiterte Honeywell-Implementierung des ARINC 659 Standards, die in der Boeing 777 als *backplane bus* des *airplane information management systems* (AIMS) zum Einsatz kommt. Seit der Indienststellung der Boeing 777 im Jahre 1995 kam es dabei zu keinem katastrophalen Ausfall, was SAFEbus zu einer Referenz für sicherheitsrelevante Anwendungen im Luftfahrtbereich macht¹⁴. Wie bereits zuvor erwähnt, werden Systeme in der zivilen Luftfahrtindustrie über mehrfache Redundanz (im Falle der Steuergeräte der Boeing 777 bis zu triple-triple) abgesichert. Daher muss auch der Bus zur Verbindung der redundanten Steuergeräte fehlertolerant sein (hier fail operational/fail silent), um keinen SPOF darzustellen. Dieser Abschnitt soll lediglich die wichtigsten Unterschiede zwischen SAFEbus und den im Kraftfahrzeug verwendeten Ansätzen verdeutlichen. Eine genauere Betrachtung findet sich zum Beispiel in [HD93] oder [Rus03], aus denen auch die Informationen dieses Abschnitts entnommen sind.

Die zentralen Bestandteile eines SAFEbus *line replaceable moduls* (LRM) sind das *bus interface unit* (BIU) ASIC, das grob dem Controller bei CAN oder FlexRay entspricht, das *table memory* EEPROM, sowie die *backplane transceiver logic* (BTL) und das *intermodule memory*, siehe Abbildung 2.23. Aus Sicht der Anwendung (des Hosts in Abbildung 2.23) stellt sich SAFEbus als statisch aufgeteiltes RAM dar, um mit Hilfe von Hardware *memory management units* (MMU) die räumliche Trennung der Daten sicherzustellen. Jede LRM verfügt über zwei BIUs, die ihrerseits wieder über je zwei BTL an je einen Bus der *self-checking busses* (SCB) angeschlossen sind. Diese Konfiguration wurde gewählt, um sofortige Fehleraufdeckung und Fehlerkapselung zu ermöglichen.

Der Buszugriff in SAFEbus wird über in den Table Memories fest abgelegte Befehle gesteuert. SAFEbus verzichtet damit vollständig auf die Möglichkeit zur Arbitrierung während der Laufzeit. Anhand dieser Befehle entscheidet der Host, ob er senden, empfangen oder sich passiv verhalten soll. Der Verlust

¹⁴Neben SAFEbus existieren weitere Lösungen, wie zum Beispiel der ARINC 629.

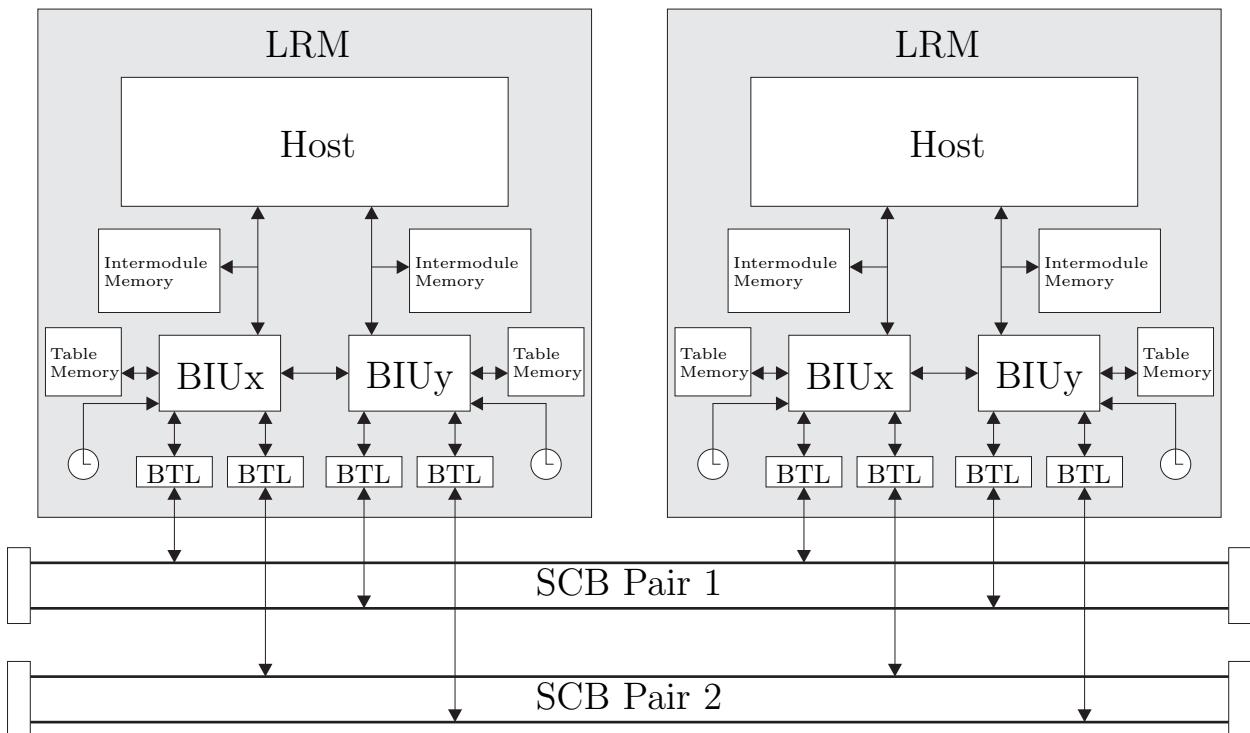


Abbildung 2.23: Struktureller Aufbau eines SAFEbus-LRMs. Die Verbindungen zwischen BIU und BTL sind stark vereinfacht wiedergegeben [HD93].

an Flexibilität ist in dem Anwendungsbereich des SAFEbus weniger relevant als der Vorteil, dass es mit diesen festgelegten Buszugriffszeiten möglich ist, dem Gesamtsystem ein deterministisches Verhalten zu geben. Die geringe Anforderung an Variantenbildung in der Luftfahrtindustrie, die in der aufwändigen Zertifizierung begründet ist, begünstigte diese Entscheidung, da ein deterministisches System Vorteile bei der Zertifizierung mit sich bringt [Rus03]. Unter bestimmten Voraussetzungen ist es aber möglich, zwischen verschiedenen Schemata für den Buszugriff zu wechseln, die jedoch alle zur Entwurfszeit des Systems festgelegt und in den table memories gespeichert werden.

Jeder der zwei Busse der zwei SCB besteht aus drei Leitungen (Clock, Data0, Data1), wodurch zwei Bit parallel übertragen werden können. Dabei ist die Codierung der Bits auf den Bussen so gewählt, dass Bus Ax normale Parität hat, während Bx invertiert ist. Auf Bus Ay wird jedes zweite Bit beginnend mit dem zweiten vertauscht und die Bits auf By sind zusätzlich invertiert. Durch diese Codierung sind unabhängig von Inhalt der Nachricht immer vier Leitungen low und vier high und bei jedem Taktschritt ändert sich der Pegel auf vier der insgesamt acht Leitungen. Ein Vorteil dieser Codierung ist, dass die Leistungsaufnahme konstant ist und nicht auf worst-case Fälle

2 Kommunikation verteilter Automobilanwendungen

ausgelegt werden muss. Durch Abweichen der Buspegel von diesem spezifizierten Schema ist es möglich, fehlerhafte Übertragungen sofort zu erkennen. Als physical layer verwendet SAFEbus die BTL nach IEEE 1194 [HD93]¹⁵.

Die Zeit für eine vollständige Übertragung aller Nachrichten (Frame) ist in Windows aufgeteilt, die ihrerseits aus der eigentlichen Nachricht (Message) und einem Abstand zwischen den Nachrichten (Gap, vergleichbar dem Inter Frame Space in CAN, siehe Abschnitt 2.4.1) besteht. Eine Nachricht kann dabei zwischen 32 und 8192 Bit (1 MByte) enthalten und wird daher in 16 bis 4096 Taktschritten übertragen. Die Länge der Gap kann zur Entwurfszeit zwischen 2 und 9 Bit festgelegt werden. Da SAFEbus weder Arbitrierung noch CRC/FEC einsetzt, wird für den typischen Anwendungsfall [HD93] von 32 Bit Nachrichten mit 4 Bit Gap eine Effizienz von $\eta = 0.89$ erreicht. Bei der maximal möglichen Geschwindigkeit von 30 MBit/s auf jeder der zwei Datenleitungen ergibt sich eine Nettodatenrate von ca. 54 MBit/s. Um trotz des geforderten Determinismus ein gewisses Maß an Flexibilität zu wahren, lässt SAFEbus die Verwendung von sogenannten Master/Shadow Paaren zu, denen das gleiche Window zugeteilt ist. Im Regelfall sendet der Master und beginnt dabei direkt am Anfang des Windows. Ist der Master defekt oder aus einem anderen Grund nicht in der Lage zu senden, beginnt der erste Shadow nach Δ Taktschritten zu senden, wobei Δ im Normalfall 1 Bit länger als die gewählte Gap ist. Sollte auch der erste Shadow nicht senden, kann nach 2Δ Zeitschritten der zweite Shadow senden, usw. Um die Nachricht dennoch innerhalb des Windows ausliefern zu können, werden an die eigentliche Nachricht leere Bits angehängt, um das Window zu füllen, im Fall, dass der Master die Nachricht versendet also $(\text{Anzahl der Shadows}) \cdot \Delta$ Bit. Durch das künstliche Verlängern des Windows im fehlerfreien Fall ist es also möglich einen spätesten Zeitpunkt für die Auslieferung der Nachricht anzugeben.

Ein weiteres Zugeständnis an die Flexibilität ist die Möglichkeit, zwischen verschiedenen Frames zu wechseln. Framewechsel finden dabei zu Synchronisationszeitpunkten statt, wodurch durch eine unterschiedliche Folge verschiedener Synchronisationssequenzen Geräte im falschen Frame vom Bus getrennt werden. Der Framewechsel ist jedoch auf ein sehr grobes Raster beschränkt. So sind beispielsweise Frames für Hardware- und Softwareinitialisierung sowie für den Selbsttest und den eigentlichen Flug denkbar. Dabei enthält der Frame für den Flug keine weiteren Framewechselkommandos, da diese sonst in deterministischen Abständen den Betriebszustand des SAFEbus ändern würden.

Abschließend ist zu sagen, dass es sich bei SAFEbus um eine sehr kompromisslose Umsetzung der TDMA Strategie handelt, wodurch sowohl die

¹⁵Die Norm IEEE 1194 wurde zwischenzeitlich durch die Norm 1194.1 ersetzt.

Effektivität des Systems steigt als auch die Zertifizierung des Systems vereinfacht wird. So ist SAFEbus in der Lage, einen byzantinischen Fehler zu tolerieren [DHSZ03]. Bezahlt wird dies durch die beschränkte Flexibilität und nicht zuletzt durch den hohen Preis von mehr als \$ 100 für die Anbindung eines LRMs. SAFEbus ist daher für den Einsatz im Kraftfahrzeug ungeeignet und lediglich als Referenz zu betrachten.

2.4.4 Zusammenfassende Betrachtung

Wie bereits in den einzelnen Abschnitten erwähnt, ist jedes der beschriebenen Bussysteme aus dem einen oder anderen Grund nur eingeschränkt für die Verwendung in sicherheitsrelevanten Automobilanwendungen geeignet. Zwar erfüllt zum Beispiel SAFEbus die technischen Aspekte in Hinblick auf die Sicherheitsanforderungen und übertrifft diese zum Großteil auch, ist aber aus Kostengründen nicht für den Einsatz im Kraftfahrzeug geeignet. Lösungen, die restriktiven Kostenanforderungen erfüllen, wie zum Beispiel CAN oder FlexRay, weisen andererseits konzeptionelle Mängel auf, die einen Einsatz in sicherheitsrelevanten Teilen der Kraftfahrzeugelektronik ohne Rückfallebene erschweren. Aus diesem Grund wird im folgenden Kapitel ein netzwerkbasierter Ansatz für ein Kommunikationssystem für sicherheitsrelevante Automobilelektronik vorgestellt.

2 Kommunikation verteilter Automobilanwendungen

3 Netzwerkbasieretes Kommunikationsmedium (**SafeNet**)

3.1 Zielsetzung

Den in Abschnitt 2.4 vorgestellten Kommunikationsmedien ist gemeinsam, dass sie auf einer zentralen Komponente beruhen. In allen Fällen teilen sich alle Knoten das physikalische Medium, in aktiven Sternstrukturen ist dies der aktive Stern. Da dieser gemeinsame Zugriff auf das physikalische Medium koordiniert werden muss, entweder dezentral durch Arbitrierung oder zentral durch einen aktiven Stern, muss auch sichergestellt werden, dass alle an dem Medium angeschlossenen Geräte ihre Entscheidung zu einem gegebenen Zeitpunkt zu senden oder nicht zu senden anhand von konsistenten Datensätzen treffen können. Bei CAN wird hierzu die Bitdauer so gewählt, dass trotz der durch die begrenzte Ausbreitungsgeschwindigkeit entstehenden Signallaufzeiten alle Geräte in einem Großteil der Bitdauer den gleichen Buspegel sehen, siehe Abbildung 3.1(a).

Die erwähnten TDMA-Systeme FlexRay und TTP/C gehen noch einen Schritt weiter, da sich hier alle Geräte, die an dem Kommunikationsmedium angeschlossen sind, auch eine logische Komponente, nämlich die Systemzeit, teilen. Basierend auf dieser Zeit werden die Zeitschlitze, in denen ein Gerät senden darf, verteilt. Auch hier spielt die Ausbreitungsgeschwindigkeit und die durch sie verursachte Verzögerung τ_p eine Rolle, da zwar innerhalb eines Zeitschlitzes die Bit im Rahmen der spezifizierten Geschwindigkeiten beliebig schnell hintereinander gesendet werden dürfen, im Anschluss an einen Zeit-

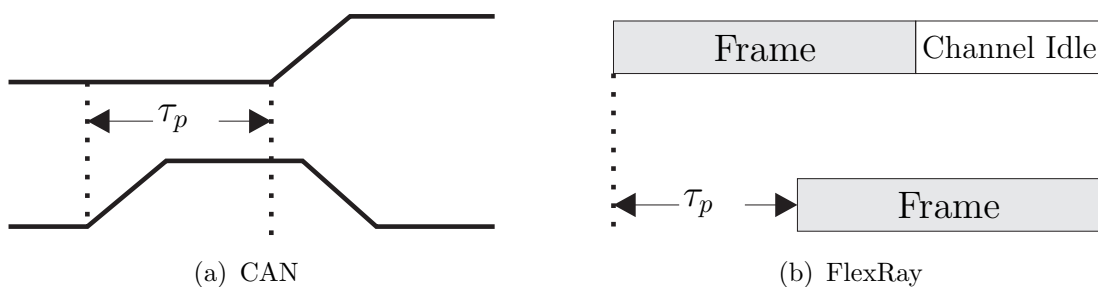


Abbildung 3.1: Einfluss der durch die maximale Ausbreitungsgeschwindigkeit verursachten Verzögerung τ_p in CAN and FlexRay

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

schlitz aber abgewartet werden muss, bis definitiv alle Geräte am Kommunikationsmedium die Daten, die innerhalb des Zeitschlitzes versendet wurden, vollständig empfangen haben. Andernfalls könnte sich eine Verletzung der TDMA-Eigenschaften ergeben, siehe Abbildung 3.1(b), was zur Cliquenbildung führen könnte, siehe auch Abbildung 2.2.

Der gemeinsame Zugriff auf das Medium verlangsamt also durch die physikalischen Eigenschaften der Leitungen die Übertragung, da zu einem Bit beziehungsweise einem Frame jeweils ein Pufferintervall addiert werden muss, um die Effekte der Laufzeit auszugleichen. Erschwerend kommt bei TDMA Systemen hinzu, dass der Abgleich der lokalen Uhren eine komplexe Aufgabe ist, die den Implementierungsaufwand des Kommunikationssystems steigert.

Ein weiterer Nachteil eines zentralen physikalischen Mediums ist, dass ohne weitere Vorkehrungen, ein defektes Gerät das Medium stören und so die Kommunikation der anderen, korrekt funktionierenden Teilnehmer untereinander verhindern kann. Die aktive Sternstruktur stellt hier eine Ausnahme dar, da der aktive Stern defekte Teilnehmer vom physikalischen Medium trennen kann. Erkauft wird dies durch eine erhöhte Ausfallwahrscheinlichkeit des Gesamtsystems, da der aktive Stern ja wie jeder normale Teilnehmer aktiv ausfallen kann und somit einen SPOF darstellt. Andere Maßnahmen, um aktiv ausgefallene Geräte von Kommunikationsmedium zu trennen, erhöhen ebenfalls die Komplexität des Gesamtsystems. Beispielhaft sind hier der Bus Guardian von FlexRay oder der in [Roo05] vorgestellte BUSPWR-Block für CAN zu nennen. Durch eine spezielle Anordnung seiner Komponenten, die so genannte switching signal path structure, ist der BUSPWR in der Lage, auch stuck-at Ausfälle aufzudecken und weist daher im Vergleich zu dem Bus Guardian eine höhere Fehlerabdeckung auf. Hierfür müssen jedoch mehrere Komponenten, die in ihrer Komplexität dem Bus Guardian vergleichbar sind, verschaltet werden, um ein Gerät an den CAN Bus anzuschließen.

Es zeigt sich also, dass der gemeinsame Zugriff auf ein physikalisches Medium einerseits die Kommunikation der Geräte an einem Kommunikationsmedium verlangsamt und andererseits den Implementierungsaufwand für die einzelnen Geräte beziehungsweise ihre Anbindung an das Kommunikationssystem erhöht. Durch den Verzicht auf ein gemeinsames physikalisches Medium sollte es demnach möglich sein, ein effizientes, schnelles und einfach zu implementierendes Kommunikationsmedium zu entwerfen. Dieses Kommunikationsmedium muss jedoch einen großen Vorteil des gemeinsamen physikalischen Mediums wettmachen, nämlich die *broadcast*-Eigenschaft, die impliziert, dass alle an das gemeinsame physikalische Medium angeschlossenen Geräte eine Nachricht gleichzeitig empfangen, und zwar unabhängig von der Anzahl der an das Kommunikationsmedium angeschlossenen Geräte.

Ein solches Kommunikationssystem, *SafeNet*, wird in diesem Kapitel vor-

gestellt. Nach einer kurzen Übersicht über das Funktionsprinzip, bestehend aus der Netzwerkstruktur und verteilten Algorithmen wird dieses Prinzip mit Hilfe von Markov-Ketten analysiert und anschließend um eine zentrale Eigenschaft im Sendalgorithmus erweitert. Besonderes Augenmerk wird dabei auf die Fehleranalyse gelegt. Im Anschluss wird die Spezifikation von SafeNet in Kapitel 4 vorgestellt.

3.2 Funktionsprinzip

Die Grundidee hinter SafeNet ist der in dem vorhergehenden Abschnitt angedeutete Verzicht auf das gemeinsame physikalische Medium. Hieraus ergibt sich eine Struktur, die aus Eigenschaften für gültige Topologien für die Verbindungen der Knoten untereinander einerseits und verteilten Algorithmen andererseits besteht.

Hierbei sollen die Anforderungen an die Topologie möglichst wenig einschränkend sein, um die Anpassung des Systems an geometrische und strukturelle Gegebenheiten zu erlauben. Für die Algorithmen, die in den einzelnen Geräten implementiert werden müssen, ist es von Vorteil, die Komplexität gering zu halten. Werden diese Aspekte kombiniert, ergibt sich ein Kommunikationssystem, welches die Anforderungen an echtzeitfähige Automobilelektronik, die in Abschnitt 2.1 und Abschnitt 2.2 formuliert wurden, erfüllt.

3.3 Netzwerktopologie

Um die Anforderung, nach einem beliebigen Fehler¹ noch die Kommunikation zwischen den fehlerfrei funktionierenden Geräten gewährleisten zu können, erfüllen zu können, ist es nötig, dass bei dem Ausfall einer Verbindung zwischen zwei beliebigen Knoten eine funktionierende Verbindung existiert.

Darüber hinaus muss bei dem aktiven Ausfall eines Geräts gewährleistet sein, dass dieses Gerät sicher vom funktionierenden Rest des Netzwerkes separiert werden kann. In diesem Zusammenhang besitzt *sicher* auch einen zeitlichen Aspekt: Es muss möglich sein, ein aktiv ausgefallenes Gerät innerhalb einer definierten Zeit nach dem Ausfall zu identifizieren und zu isolieren. Es ist anzumerken, dass für diese Zeit nicht das Auftreten des Fehlers sondern der Ausfall des Geräts ausschlaggebend ist, da ein Fehler, wie in Definition 2.2 bemerkt, von der Umgebung des Systems erst identifiziert werden kann, wenn er die Grenze des Systems erreicht hat, also einen Ausfall verursacht

¹Wie bei den zuvor betrachteten existierenden Systemen werden Common-Mode Ausfälle der Teile, die für die Anbindung der Geräte an das Kommunikationssystem verantwortlich sind, nicht berücksichtigt.

3 Netzwerkbasierendes Kommunikationsmedium (*SafeNet*)

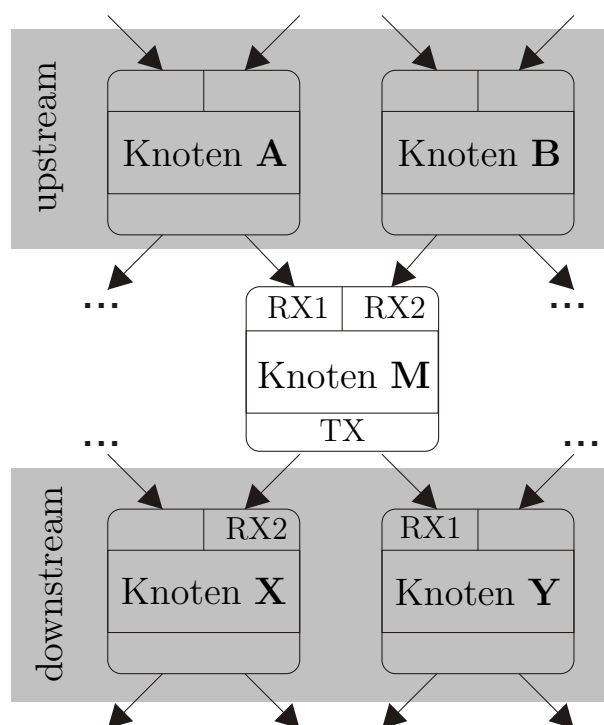


Abbildung 3.2: Netzwerkprimittiv für SafeNet

hat. Diese Identifizierung ist jedoch durch Netzwerktopologie nicht zu bewerkstelligen und wird im folgenden Unterabschnitt genauer beleuchtet.

Die Anforderungen „sichere Trennung“ und „redundante Wege“ können durch eine Struktur, die aus Bausteinen wie in Abbildung 3.2 dargestellt, aufgebaut werden. Dargestellt ist ein Knoten M aus der Menge aller Knoten \mathcal{N} , also $M \in \mathcal{N}$, sowie seine direkte Umgebung, die Knoten $A, B, X, Y \in \mathcal{N}$.

Dieser Knoten besteht aus zwei Empfangsmodulen (RX1 und RX2) sowie einem Sendemodul (TX). Der Knoten ist durch unidirektionale Verbindungen mit anderen Knoten des Netzwerks verbunden.

Auf Grund der Tatsache, dass sämtliche Verbindungen unidirektional sind, breiten sich Nachrichten von einem Knoten ausgehend nur in einer Richtung aus. Diese wird im Folgenden als Ausbreitungsrichtung von Nachrichten bezeichnet.

Die allgemeinste Möglichkeit zur Beschreibung der relativen Position von Knoten in einer SafeNet-Topologie sind dementsprechend die Richtungsbeschreibungen „*upstream*“ und „*downstream*“.

Ein Knoten liegt *upstream* relativ zu einem Referenz-Knoten, in Abbildung 3.2 ist dies der Knoten M , wenn man sich von diesem Referenz-Knoten aus *entgegen* der Ausbreitungsrichtung von Nachrichten entlang der unidirektionalen Verbindungen bewegt. In Abbildung 3.2 liegen die Knoten A und B unmittelbar *upstream* von M und senden damit Nachrichten direkt an M .

Formal sind A und B in der Menge der direkten upstream Knoten oder der direkten Vorgänger von M enthalten: $A, B \in M^-$.

Hingegen liegt ein Knoten ausgehend von einem Referenz-Knoten *downstream*, wenn man sich *mit* der Ausbreitungsrichtung von Nachrichten entlang unidirektionaler Verbindungen bewegt. In Abbildung 3.2 liegen die Knoten X und Y unmittelbar downstream von M , empfangen also Nachrichten direkt von M . Hieraus folgt $X, Y \in M^+$.

An dieser Stelle ist anzumerken, dass die Bezeichnungen upstream und downstream den Informationsfluss abbilden und im Allgemeinen abhängig von der Nachricht beziehungsweise dem ursprünglichen Absender der Nachricht sind. Da im Folgenden von einer eindeutigen Zuordnung der Nachrichten zu ihrem ursprünglichen Absender ausgegangen wird, werden Knoten, die bezüglich der Nachricht μ upstream von M liegen in einer Menge M_μ^- , die die downstream von M bezüglich der Nachricht μ liegen als Element der Menge M_μ^+ bezeichnet. Diese Eigenschaft muss bei der Betrachtung der Zyklizität des Netzes im Rahmen der Nachrichtenausbreitung berücksichtigt werden.

Wie in Abbildung 3.2 bereits angedeutet, besitzt jeder Knoten mindestens zwei Eingänge und einen Ausgang, der mit zwei Knoten verbunden ist. Diese Formulierung wurde gewählt, um die Anzahl der möglichen Fehlerursachen zu reduzieren. An dieser Stelle soll jedoch nicht näher auf diese Eigenschaft eingegangen und lediglich auf die folgende Diskussion der Fehlerbehandlung in Abschnitt 3.7 verwiesen werden. Gleichwertig kann auch die Formulierung, dass ein Knoten zwei Ausgänge besitzt, auf denen zu jeder Zeit die gleichen Nachrichten versendet werden, verwendet werden. Wichtig ist nun, dass weder beide Eingänge noch beide Ausgänge mit dem gleichen Knoten verbunden sind. Wäre dies der Fall könnte ein Knoten durch den Ausfall eines einzigen andern Knotens von der Kommunikation des Netzwerkes abgeschnitten werden. Es müssen also für die Knoten $A, B, M, X, Y \in \mathcal{N}$

$$A \in M^- \wedge B \in M^- \Rightarrow A \neq B \quad (3.1)$$

und

$$X \in M^+ \wedge Y \in M^+ \Rightarrow X \neq Y \quad (3.2)$$

gelten um einen SPOF zu vermeiden.

Darüber hinaus muss sichergestellt werden, dass der Ausgang eines Knoten nicht direkt auf einen Eingang desselben Knotens zurückgeführt wird. Hierdurch würden der andere Eingang beziehungsweise der andere Ausgang zu SPOFs, da nach dem Ausfall einer dieser Verbindungen der Knoten entweder empfangs- oder sendeseitig isoliert ist. Hierfür wird für einen Knoten $M \in \mathcal{N}$

$$M \notin M^+ \wedge M \notin M^- \quad (3.3)$$

3 Netzwerkbasierendes Kommunikationsmedium (*SafeNet*)

gefordert.

Um nun trotz der unidirektionalen Verbindungen Nachrichten zwischen allen Knoten versenden zu können, ist es nötig, die losen Enden, die in Abbildung 3.2 dargestellt sind, so zu verbinden, dass Pfade zwischen allen beliebigen Knoten existieren. Hierfür muss zusätzlich zu den oben formulierten Bedingungen gelten, dass das Netz auch nach dem Ausfall eines beliebigen Knotens zusammenhängend bleibt. Für Nachrichten μ , die dem Knoten M zugeordnet sind, wird also gefordert, dass

$$X \in M_{\mu}^{+} | I \text{ defekt } \forall I, M, X \in \mathcal{N} | I \neq X, M \quad (3.4)$$

gilt.

Der Ausfall einer Verbindung muss hierbei nicht gesondert berücksichtigt werden, da jeder Ausfall einer Verbindung in dem Ausfall mindestens eines Knotens enthalten ist. Eine solche Topologie wird als *gültig* bezeichnet. Im Folgenden wird, falls nicht anders erwähnt, davon ausgegangen, dass es sich bei dem Netzwerk um ein gültiges *SafeNet* handelt.

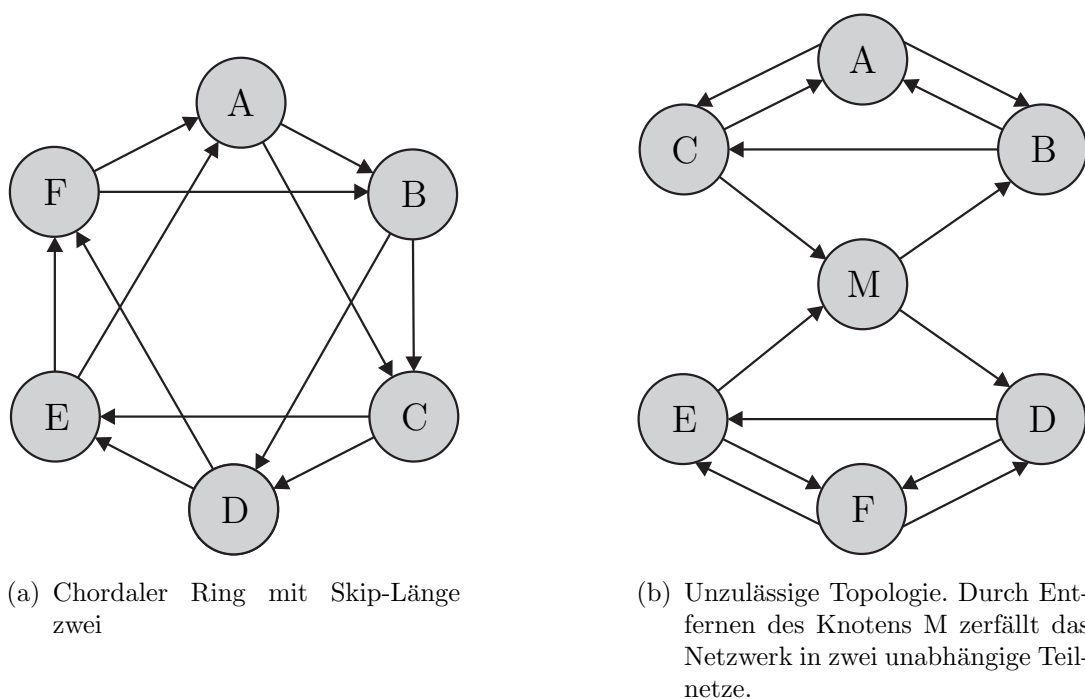


Abbildung 3.3: Verschiedene Topologien in *SafeNet*

In Abbildung 3.3 sind zwei Topologien für *SafeNet* dargestellt. Bei dem chordalen Ring der Skip-Länge zwei handelt es sich um eine universelle Topologie in dem Sinne, dass sie für eine beliebige Anzahl von Knoten $N \geq 3$ in dem Netzwerk angegeben werden kann und eine gültige Topologie darstellt. Ein chordaler Ring der Skip-Länge zwei, im Folgenden kurz „chordaler Ring“

genannt, wird gebildet, indem Knoten A an die Knoten B und C sendet, B an C und D, und so weiter.

Der chordale Ring wird im Folgenden oft als Beispielarchitektur verwendet. Zum einen liegt dies natürlich an der Tatsache, dass dieses Beispiel dann für beliebige $N \geq 3$ verwendet werden kann; zum anderen besitzt der chordale Ring mehrere Eigenschaften, die ihn unter den mögliche Topologien für Safe-Net besonders interessant machen. Der chordale Ring ist in in seiner Struktur dem idealen Aufbau von Kraftfahrzeugkommunikationssystemen relativ nahe. Im Gegensatz zur Sternarchitektur wächst die benötigte Leitungslänge nicht mit der Anzahl der Teilnehmer sondern konvergiert bei geeigneter Normierung gegen einen festen Wert. Auf diese Eigenschaften wird in Unterabschnitt 5.3.1 näher eingegangen.

Der chordale Ring ist auch eine der wenigen und wohl die einzige nicht-triviale Topologie, für die die maximale Anzahl der Nachrichten in der Sendewarteschlange einfach angegeben werden kann. Diese Eigenschaft spielt bei der Sicherstellung von deterministischem Verhalten eine wichtige Rolle, siehe auch Abschnitt 3.6.

Im Gegensatz zu dem chordalen Ring ist die in Abbildung 3.3(b) gezeigte Topologie ungültig, da der Knoten M einen nicht zulässigen SPOF darstellt. Fällt dieser Knoten aus, so zerfällt das Netzwerk in zwei Teile, die nicht miteinander kommunizieren können, wodurch die Gesamtfunktionalität des Systems nicht mehr gewährleistet ist.

Eine weitere zulässige Topologie, die allerdings nur für gerade N gültig ist, ist in Abbildung 3.4 gezeigt. Auch diese Struktur, im Folgenden als „Leitertopologie“ bezeichnet, besitzt günstige Eigenschaften, was beispielhaft in Abbildung 3.4 gezeigt wird. Der chordale Ring und die Leitertopologie sind einander ähnlich. Zeichnet man beide als planare Graphen, so unterscheiden sie sich lediglich im Umlaufsinn des mittleren Ringes sowie in der Reihenfolge der äußeren Knoten, wie in Abbildung 3.5 dargestellt ist.

Die Leitertopologie kann mit Hilfe der in Abbildung 3.6 dargestellten Struktur auf beliebige, also auch ungerade Knotenzahlen erweitert werden. Hierzu wird ein Knoten durch ein Knotenpaar ersetzt. Jeder Knoten dieses Paares übernimmt einen Ein- sowie einen Ausgang des ursprünglichen Knotens. Der andere Eingang beziehungsweise Ausgang wird mit dem anderen Knoten des Paares verbunden. Hierdurch bleibt die Gültigkeit des Netzes erhalten, da der Ausfall eines Knotens des Paares weniger Verbindungen unterbricht als der Ausfall des ursprünglichen Knotens.

Es muss jedoch beachtet werden, dass besonders in Verbindung mit dem erweiterten Sendalgorithmus aus Abschnitt 3.6 einige der Eigenschaften der ursprünglichen Topologie verloren gehen. So kann sich zum Beispiel nach der Ersetzung eines Knotens durch ein Knotenpaar die maximale im Netzwerk

3 Netzwerkbasierendes Kommunikationsmedium (*SafeNet*)

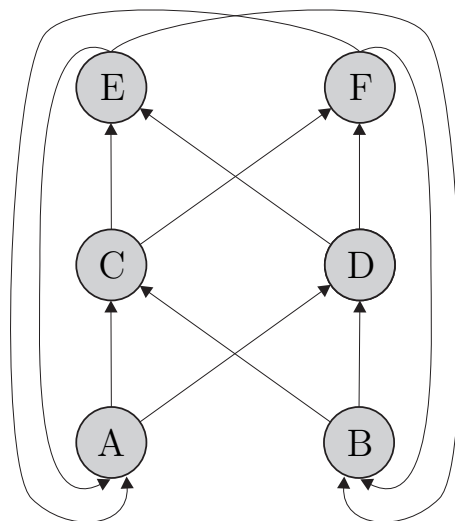
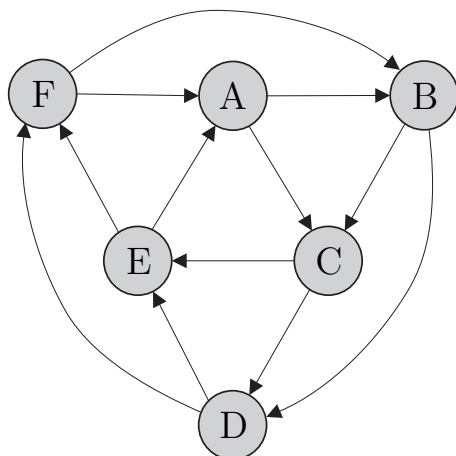
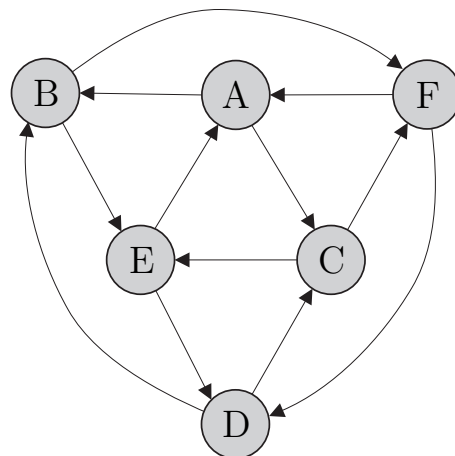


Abbildung 3.4: Die Leiter-Topologie für gerade Knotenzahlen in *SafeNet*



(a) Chordaler Ring mit Skip-Länge 2, planar dargestellt



(b) Leiter-Topologie, planar dargestellt

Abbildung 3.5: Planare Darstellung verschiedener Topologien in *SafeNet*

vorkommende Sendewarteschlangenlänge ändern, womit sich das deterministische Verhalten des Systems ändern kann. Im Falle hoher Auslastung des Netzwerkes ist unter Verwendung der in Abschnitt 3.4 und Abschnitt 3.6 vorgestellten Algorithmen auch nicht gewährleistet, dass alle Knoten nach der Ersetzung den gleichen Anteil an der Auslastung erhalten.

Neben den hier vorgestellten Topologien existieren weitere, zum Beispiel dreidimensionale Architekturen. Für die grundsätzliche Analyse von *SafeNet* sind die gegebenen Topologien jedoch ausreichend, weshalb an dieser Stelle auf die Vorstellung weiterer Topologien verzichtet werden soll.

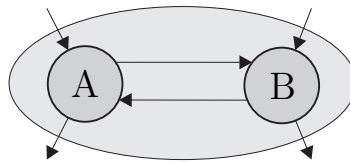


Abbildung 3.6: Mögliche Ersetzung eines Knotens durch ein Knotenpaar

3.4 Verteilte Algorithmen

Neben der Topologie, also den Regeln für die Verbindung der Knoten untereinander, ist SafeNet auch auf einfache, verteilte Algorithmen aufgebaut. Die Algorithmen werden als verteilt bezeichnet, da erst durch ihre Ausführung in mehreren, im Extremfall allen, Knoten das beabsichtigte Resultat entsteht. Die einzelnen Knoten haben dabei unterschiedliche „Meinungen“ über den Zustand des gesamten Netzwerkes, da nicht sichergestellt werden kann, dass alle Knoten zur gleichen Zeit über die gleichen Informationen verfügen. Dies ist zur Fehlerkapselung oft auch nicht erwünscht.

3.4.1 Sendealgorithmus

Der zentrale Algorithmus in SafeNet ist der in Abbildung 3.7 dargestellte Sendealgorithmus, mit dessen Hilfe alle Knoten alle Nachrichten im fehlerfreien Fall exakt ein Mal versenden.

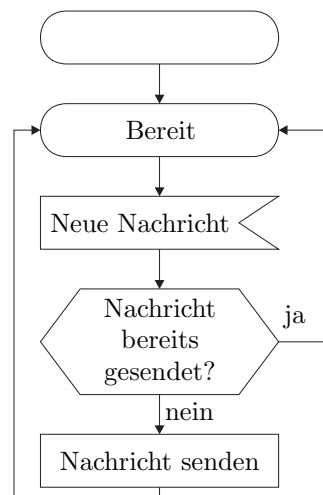


Abbildung 3.7: Der SafeNet Sendealgorithmus

Dieser Algorithmus lässt sich auf die einfache Abfrage reduzieren, ob ein Knoten die erhaltene, syntaktisch korrekte Nachricht schon an seine beiden

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

Nachfolger gesendet hat oder nicht. Ist dies der Fall, sind beide Nachfolger bereits über den Inhalt der Nachricht informiert und es ist keine weitere Aktion nötig. Wurde die Nachricht noch nicht versendet, so wird sie weitergeleitet. Als gesendet gelten in diesem Zusammenhang auch Nachrichten, die in einer Warteschlange auf ihr Versenden warten. Es kann daher auch abgefragt werden, ob der Knoten die Nachricht bereits empfangen hat, da er diese nach dem ersten Empfangen direkt weiterleitet.

Eigene Nachrichten darf der Knoten nach dieser Formulierung direkt bei Empfang vom Host ohne Beschränkung an das Ende der Sendewarteschlange einreihen. Ist eine Nachricht in der Sendewarteschlange, so wird sie danach nicht mehr entfernt oder durch das Einfügen anderer Nachrichten verzögert. Die Reihenfolge in der Warteschlange wird auch nicht durch Umsortieren geändert. Mit anderen Worten, es existieren in der Sendewarteschlange keine Nachrichten, die anderen gegenüber priorisiert wären. Dies ist akzeptabel, wenn die Länge der Warteschlange auf einen kleinen Maximalwert beschränkt ist.

Wie eingangs erwähnt, stellt dieser Algorithmus sicher, dass jeder Knoten jede Nachricht exakt ein Mal versendet.

Satz 3.1 *Im fehlerfreien Fall versendet in SafeNet jeder Knoten jede Nachricht exakt ein Mal.*

Beweis 3.1 *Der Beweis erfolgt in zwei Schritten. Zuerst wird gezeigt, dass jeder Knoten seine eigenen Nachrichten nur ein Mal versendet, später dass jeder Knoten empfangene Nachrichten, die von anderen Knoten generiert wurden, nur ein Mal versendet, womit alle Knoten abgedeckt sind. Dies geschieht unter der Annahme, dass Nachrichten weder in den Knoten noch bei der Übertragung verfälscht werden. Betrachtet wird der Knoten $M \in \mathcal{N}$ sowie $A, B \in M^- \in \mathcal{N}$ und $X, Y \in M^+ \in \mathcal{N}$.*

- 1. Empfängt M seine eigene Nachricht μ auf einem seiner Eingänge, so wird diese Nachricht nicht an die Knoten in M^+ gesendet, da μ bereits von M verschickt wurde.*
- 2. Empfangen X und Y als direkte Nachfolger von M die Nachricht μ direkt von M , so senden sie diese Nachricht an ihre Nachfolger. Erhalten sie μ ein zweites Mal, wird keine Nachricht versendet.*

Es ist also gezeigt, dass jeder Knoten jede Nachricht im fehlerfreien Fall genau ein Mal versendet.

Ebenso gelten die folgenden Sätze:

Satz 3.2 *Auf jeder Verbindung in SafeNet wird im fehlerfreien Fall jede Nachricht exakt ein Mal versendet.*

Beweis 3.2 *Da jede Verbindung unidirektional von einem Knoten zu seinen downstream Knoten führt und nach Satz 3.1 jeder Knoten jede Nachricht exakt ein Mal versendet, wird also jede Nachricht auf jeder Verbindung genau ein Mal versendet.*

Satz 3.3 *Jeder Knoten erhält im fehlerfreien Fall jede Nachricht von zwei verschiedenen Knoten.*

Beweis 3.3 *Da jeder Knoten jede Nachricht nach Satz 3.1 genau ein Mal sendet und nach Gleichung 3.2 jeder Knoten mit zwei verschiedenen Knoten direkt upstream verbunden ist, erhält jeder Knoten jede Nachricht von zwei verschiedenen Knoten.*

Mit Satz 3.2 ist klar, dass die mittlere Buslast auf allen Verbindungen in einem SafeNet gleich ist. Der Begriff „Buslast“ wird hier auch für die unidirektionalen Verbindungen verwendet, obwohl diese streng genommen keinen Bus darstellen.

Durch den Sendalgorithmus ist in Verbindung mit einer gültigen Topologie also die Funktion des Systems im fehlerfreien Fall sichergestellt. Da zwischen zwei Knoten im Netz immer mindestens zwei Wege existieren müssen, die sich in mindestens einem Knoten und einer Verbindung unterscheiden, existiert auch immer mindestens ein Weg, auf dem sich eine fehlerhafte Nachricht ausbreiten kann, sofern sie nicht früh gekapselt wird. Diese Eigenschaft kann in den Fehlererkennungsalgorithmen ausgenutzt werden.

3.4.2 Fehlererkennung

Neben dem Sendalgorithmus sind die Algorithmen zur Fehlererkennung die zweite Klasse von Algorithmen in SafeNet. Während der Sendalgorithmus eine sehr einfache Netzwerkschicht im OSI-Schichtenmodell darstellt, gehen die Fehlererkennungsalgorithmen in SafeNet über die Kanalcodierung hinaus und sind daher in der Transportschicht anzusiedeln, vergleiche auch Abbildung 2.8. Da die Transportschicht bereits zu den anwendungsorientierten Schichten gezählt wird, ist sie in vielen Kommunikationssystemen, wie zum Beispiel FlexRay nicht oder nur rudimentär implementiert, vergleiche auch Unterabschnitt 2.4.2. Das Hauptargument für diese Verlagerung ist, dass die Fehlerbehandlung in den höheren Schichten besser an die Anwendung angepasst werden kann. In SafeNet ist eine Fehlererkennung und -behandlung

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

jedoch auf Protokollebene vorgesehen, da sich hierdurch die Belastung des Hosts durch die Fehlerbehandlung deutlich verringern lässt. Zudem wird ein „drakonisches“ Vorgehen, wie zum Beispiel bei dem Membership Algorithmus von TTP/C, der auch korrekt funktionierende Geräte ausschließen kann, vermieden, siehe auch Unterabschnitt 5.3.3. Es soll also sichergestellt werden, dass nur das fehlerhafte Gerät von der Kommunikation ausgeschlossen wird.

Bei der Fehlererkennung werden in *SafeNet*, wie in Echtzeitkommunikationssystemen allgemein, **drei Fehlerdimensionen** für Nachrichten unterschieden. Zum einen kann es sich bei einem Fehler um einen **kontextuellen Fehler** handeln. Die Nachricht wird also den Konventionen gemäß ausgeliefert, der Inhalt ist aber verfälscht. Kontextuelle Fehler werden meist auf den höheren Ebenen behandelt, da für ihre Entdeckung Plausibilitätstest, Voting-Systeme, Kalman-Filter oder Ähnliches eingesetzt werden müssen, also ein Wissen über das System oder zumindest über den Verlauf der in der Nachricht enthaltenen Information. In *SafeNet* ist die Erkennung von kontextuellen Fehlern darauf beschränkt, dass mindestens ein Knoten in einem *SafeNet* sowohl eine richtige als auch eine falsche Version der Nachricht erhält, beziehungsweise dass die fehlerhafte Nachricht zu dem ursprünglichen Sender weitergereicht wird, der die Abweichung von der von ihm versendeten Nachricht feststellen kann.

Eine weitere Fehlerdimension ist die **Überschreitung einer maximalen Zeit**. Daten werden in Echtzeitsystemen zu festgelegten Zeitpunkten benötigt. Kommt eine Nachricht nach diesem Zeitpunkt an, ist sie als fehlerhaft zu werten. Es ist daher beim Entwurf von Echtzeitkommunikationssystemen wichtig, dass eine obere Grenze für die Zeit, die eine Nachricht von ihrer Generierung bis zu ihrer Auslieferung an alle Knoten benötigt, angegeben werden kann. Die Einhaltung der von den darüber liegenden Schichten definierten Auslieferungszeitpunkte kann *SafeNet* nur insofern garantieren, als diese sich an der maximalen Auslieferungszeit von *SafeNet* orientieren. Die obere Grenze, die für die Nachrichtenauslieferung angegeben wird, ist normalerweise auf den fehlerfreien Fall bezogen. Wird beispielsweise eine kontextuell falsche Nachricht über einen der beiden FlexRay-Kanäle versendet, so können die angeschlossenen Geräte nicht unterscheiden, welche der Nachrichten korrekt ist und müssen die nächste Übertragung abwarten. In diesem Fall beträgt die maximale Auslieferungszeit nicht eine sondern zwei Runden.

Bei einem Fehler kann es sich auch um einen **syntaktischen Fehler** handeln. Die Nachricht kommt also beim Empfänger an, verstößt aber gegen syntaktische Konventionen. Im Sinne der Codierungstheorie liegt also ein ungültiges Codewort vor, siehe auch Unterabschnitt 2.3.2. Dieses Codewort wird nach dem Sendalgorithmus nicht weiter versendet, wodurch sich der Fehler nicht auf weitere Geräte ausbreiten kann, also gekapselt wird. Dieses

Vorgehen ist genau dann nicht problematisch, wenn in dem System zu jedem Zeitpunkt nur ein Fehler vorliegt, der auch nur einen Knoten betrifft. In diesem Fall würde sich die korrekte Nachricht nämlich weiter im Netz ausbreiten. Für SafeNet muss also die Ein-Fehler-Hypothese gelten.

Satz 3.4 (Ein-Fehler-Hypothese) *Tritt in einem SafeNet ein Fehler auf, so wird das Auftreten eines weiteren, unabhängigen Fehlers während der Fehlerentdeckung und Fehlerbehandlung ausgeschlossen.*

Die Ein-Fehler-Hypothese ist an dieser Stelle bewusst unpräzise formuliert. Vor allem die Fehleraufdeckungszeit sowie die Fehlerbehandlungszeit sind im Folgenden näher zu untersuchen. Dies ist allerdings erst mit der Spezifikation von SafeNet möglich, und wird daher an gegebener Stelle nachgeholt.

Eine wichtige Konsequenz der Ein-Fehler-Hypothese ist aber, dass durch einen Fehler dessen Aufdeckung und Behandlung nicht gestört wird. Konkret bedeutet dies, dass Fehlernachrichten auch von einem eventuell aktiv ausgefallenen Knoten nicht verfälscht sondern entweder richtig oder nicht versendet werden. Des Weiteren stört ein Knoten die Fehleranalyse nicht durch das Versenden falscher Fehlernachrichten.

Im Zusammenhang mit der Fehlerentdeckung und -behandlung ist es wichtig, sich der Tatsache bewusst zu sein, dass die Knoten in einem SafeNet unterschiedliche Wahrnehmungen des restlichen Netzes haben. Es existieren Fehler, die lediglich von einem Knoten entdeckt werden können. Beispielhaft ist hier der Ausfall einer einzelnen Verbindung, der nur von dem empfangenden Knoten an dieser Verbindung diagnostiziert werden kann, zu nennen. Es kann in diesem Fall nicht unterschieden werden, ob der diagnostizierende Knoten fehlerhaft ist und einen korrekt funktionierenden Knoten beschuldigt, oder ob die Diagnose korrekt ist.

Abstrakt kann ein Fehler also auf die Verbindung zwischen zwei Knoten projiziert werden. Daher ist die einzige nötige Konsequenz zur Fehlerbehandlung in SafeNet die Abschaltung von Verbindungen. Bei einem aktiven Ausfall wird der Knoten vom Netzwerk getrennt, indem beide downstream Knoten keine Nachrichten des ausgefallenen Knotens weitersenden. Im Falle einer Störung der Übertragung wird lediglich die betroffene Verbindung abgeschaltet. Auch hier ist anzumerken, dass sich der Ausfall eines Knotens wie der Ausfall zweier Verbindungen darstellt, die jeweils von dem Folgeknoten diagnostiziert werden. Aufgrund der Struktur ist kein Knoten in der Lage, einen anderen zu isolieren, da nach Voraussetzung mindestens ein Weg zwischen zwei Knoten besteht, der einen beliebigen dritten Knoten nicht enthält. Ein Knoten kann nur isoliert werden, wenn beide downstream Knoten in der Diagnose, dass der betreffende Knoten ausgefallen ist, übereinstimmen.

Eine genauere Betrachtung der Fehlerfälle erfolgt in Abschnitt 3.7.

3.5 Modellierung mit Warteschlangen

Wie bereits erwahnt, ist eine zentrale Anforderung an Echtzeitkommunikationssysteme die Moglichkeit, eine maximale Nachrichtenlaufzeit anzugeben. Die Nachrichtenlaufzeit soll nun auch fur SafeNet analysiert werden. Hierzu werden die einzelnen Knoten auf ihre Sendewarteschlangen reduziert. Die Argumentation in diesem Unterabschnitt folgt bis zur Berechnung der stationaren Zustandsverteilung [Kie06] und ist dort detaillierter ausgefuhrt. An dieser Stelle sei darauf hingewiesen, dass die in den folgenden Abschnitten vorgestellte Modellierung nur ungenaue Ergebnisse liefert, weshalb Modellierung und Sendalgorithmus in der Folge modifiziert wird.

Der stochastische Prozess, mit dem Elektronikgerate im Automobil typischerweise Ereignisse und damit Nachrichten erzeugen, kann als Poisson-Prozess modelliert werden [Sch98]. Da keine Priorisierung und kein TDMA-Schema verwendet werden, kann dieser Poisson-Prozess auch direkt als Eingangsprozess der Warteschlange verwendet werden. Dieser Poisson-Ankunftsprozess ist in den Verlassenszeitpunkten gedachnislos, weshalb der Ansatz einer eingebetteten Markov-Kette verwendet werden kann. Wird ein Knoten mit einer $M/D/1$ Warteschlange mit einer first-come-first-serve Bedienstrategie mit Hilfe einer $M/G/1$ Warteschlange modelliert, so gilt fur die Wahrscheinlichkeitsdichte der Bedienzeit $f_s(\tau) = \delta(\tau - \tau_s)$.

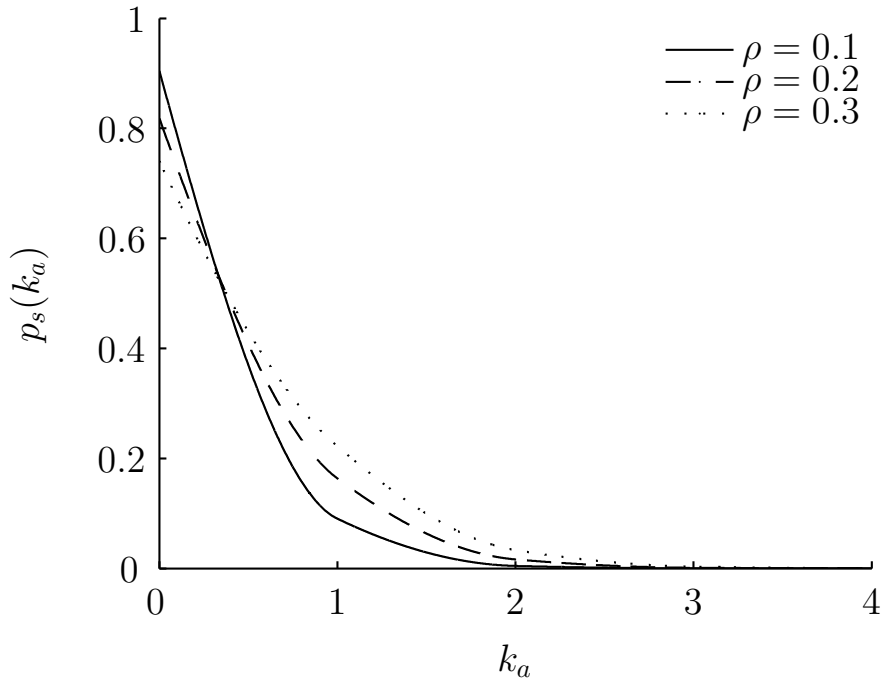
Es sei nun λ die Ankunftsrate des Poisson-Eingangsprozesses, $f_s(\tau)$ die Verteilung der Bedienzeit und τ_s die deterministische Bedienzeit. Dann gilt fur die Marginalwahrscheinlichkeit $p_s(k_a)$, die eine Aussage uber die Wahrscheinlichkeit, dass in einem Serviceintervall k_a neue Nachrichten ankommen, erlaubt

$$p_s(k_a) = \int_0^{\infty} \frac{(\lambda\tau)^{k_a}}{k_a!} e^{-\lambda\tau} f_s(\tau) d\tau \quad (3.5)$$

$$= \int_0^{\infty} \frac{(\lambda\tau)^{k_a}}{k_a!} e^{-\lambda\tau} \delta(\tau - \tau_s) d\tau \quad (3.6)$$

$$= \frac{(\lambda\tau_s)^{k_a}}{k_a!} e^{-\lambda\tau_s}. \quad (3.7)$$

In Abbildung 3.8 ist die Marginalwahrscheinlichkeit fur verschiedene Auslastungen $\rho = \lambda\tau_s$ gezeigt. Die Auslastung bewegt sich dabei in einem Rahmen, in dem beim CAN-Bus quasi-deterministisches Verhalten erwartet werden kann ($0.2 < \rho < 0.3$).


 Abbildung 3.8: Interpolierte Marginalwahrscheinlichkeit $p_s(k_a)$

Die Matrix der bedingten Übergangswahrscheinlichkeiten \underline{P} der zeitdiskreten Markov-Kette ergibt sich zu

$$\underline{P} = \begin{bmatrix} p_s(0) & p_s(1) & p_s(2) & p_s(3) & \cdots \\ p_s(0) & p_s(1) & p_s(2) & p_s(3) & \cdots \\ 0 & p_s(0) & p_s(1) & p_s(2) & \cdots \\ 0 & 0 & p_s(0) & p_s(1) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \quad (3.8)$$

Damit kann nun die Zustandswahrscheinlichkeit $\underline{\Pi}(t_{n+1})$ zum Zeitpunkt t_{n+1} berechnet werden:

$$\underline{\Pi}^T(t_{n+1}) = \underline{\Pi}^T(t_n) \cdot \underline{P}. \quad (3.9)$$

Die stationäre Zustandsverteilung kann als

$$\underline{\Pi}^T = \underline{\Pi}^T \cdot \underline{P} \quad (3.10)$$

berechnet werden.

Hierbei gilt für die Elemente des Zustandsvektors

$$\pi_j = \pi_0 \cdot p_s(j) + \sum_{i=1}^{j+1} \pi_i \cdot p_s(j-i+1) \quad \forall j \in [1, \infty) \quad (3.11)$$

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

und $\pi_0 = 1 - \rho$ mit $\rho = \lambda\tau_s$. Daher gilt

$$\pi_{j+1} = \frac{\pi_j - \pi_0 p_s(j) - \sum_{i=1}^j \pi_i p_s(j-i+1)}{p_s(0)} \quad (3.12)$$

$$= \frac{\pi_j - \pi_0 \frac{(\lambda\tau_s)^j}{j!} e^{-\lambda\tau_s} - \sum_{i=1}^j \frac{(\lambda\tau_s)^{j-i+1}}{(j-i+1)!} e^{-\lambda\tau_s}}{e^{-\lambda\tau_s}} \quad (3.13)$$

$$= \pi_j e^{\lambda\tau_s} - \pi_0 \frac{(\lambda\tau_s)^j}{j!} - \sum_{i=1}^j \frac{(\lambda\tau_s)^{j-i+1}}{(j-i+1)!}. \quad (3.14)$$

Hierbei gibt π_{j+1} in Gleichung 3.14 die Wahrscheinlichkeit an, dass zum Verlassenszeitpunkt $j+1$ Nachrichten in der Warteschlange befinden. In Abbildung 3.9 ist π_j für verschiedene ρ aufgezeichnet.

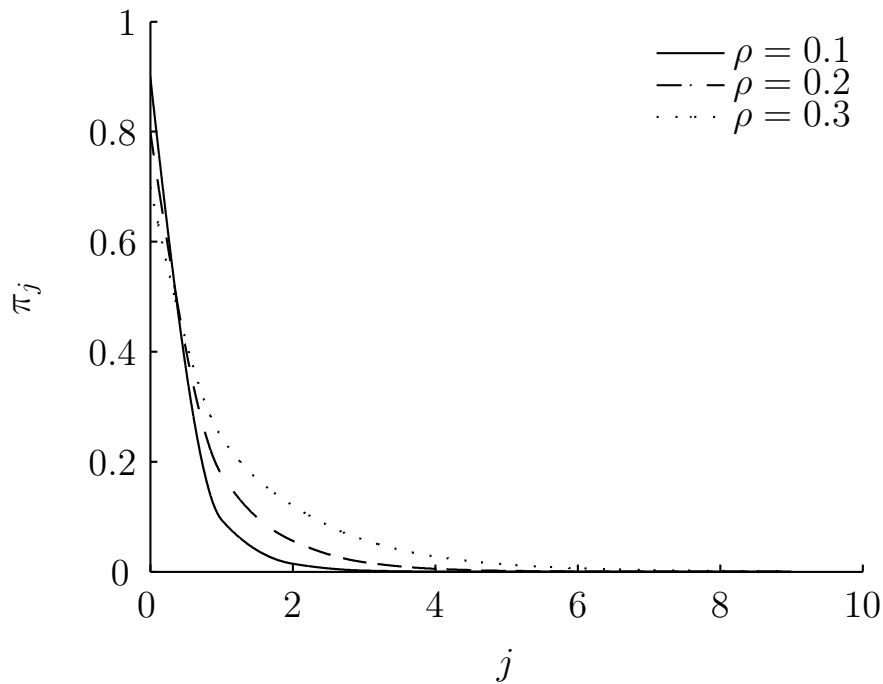


Abbildung 3.9: Interpolierte stationäre Zustandsverteilung π_j

Ist die Länge der Warteschlange nicht begrenzt, so ist die stationäre Zustandsverteilung in den Verlassenszeitpunkten gleich der stationären Zustandsverteilung zu zufälligen Zeitpunkten [Tak93]. Daher kann Π verwendet werden, um die Anzahl der Nachrichten abzuschätzen, die ein Knoten speichern können muss, um den Verlust von Nachrichten aufgrund einer vollen Warteschlange zu verhindern. Mit anderen Worten, Π ist ein Maß für die Länge der Sendewarteschlange.

Werden mehrere Systeme in Reihe geschaltet, so ergibt sich die Gesamtzustandsverteilung aus der Faltung der Einzelzustandsverteilungen. Im Falle von SafeNet müssen also die Zustandsverteilungen entlang des längsten Weges gefaltet werden. Dieser ist theoretisch gleich der Anzahl der Knoten im Netzwerk minus eins, $N - 1$, realistischer sind aber Annahmen in der Größenordnung von $N/2$. Hieraus kann dann die Anzahl der Nachrichten, die vor einer Nachricht auf ihrem Weg durch das Netz ermittelt werden.

$$\underline{\Pi}_{lp} = \underline{\Pi} * \underline{\Pi} * \dots * \underline{\Pi} \quad (3.15)$$

In Gleichung 3.15 wird vorausgesetzt, dass alle Knoten die gleiche Servicezeit τ_s besitzen und dass der Ausgangsprozess einer mit einem Poisson-Prozess der Ankunftsrate λ angeregten $M/D/1$ Warteschlange wieder ein Poisson-Prozess der Ankunftsrate λ ist. Diese Annahme über den Ausgangsprozess gilt streng genommen genau dann, wenn sowohl Ankunftsintervalle als auch Servicezeit exponentialverteilt sind, also bei $M/M/1$ Warteschlangen. Für eine konstante Servicezeit τ_s und unter der Annahme, dass $\rho \ll 1$ gilt, liefert die Modellierung des Ausgangsprozesses als Poisson-Prozess jedoch zufrieden stellende Ergebnisse. Bei den im Folgenden berechneten Ergebnissen sollte diese Einschränkung jedoch berücksichtigt werden.

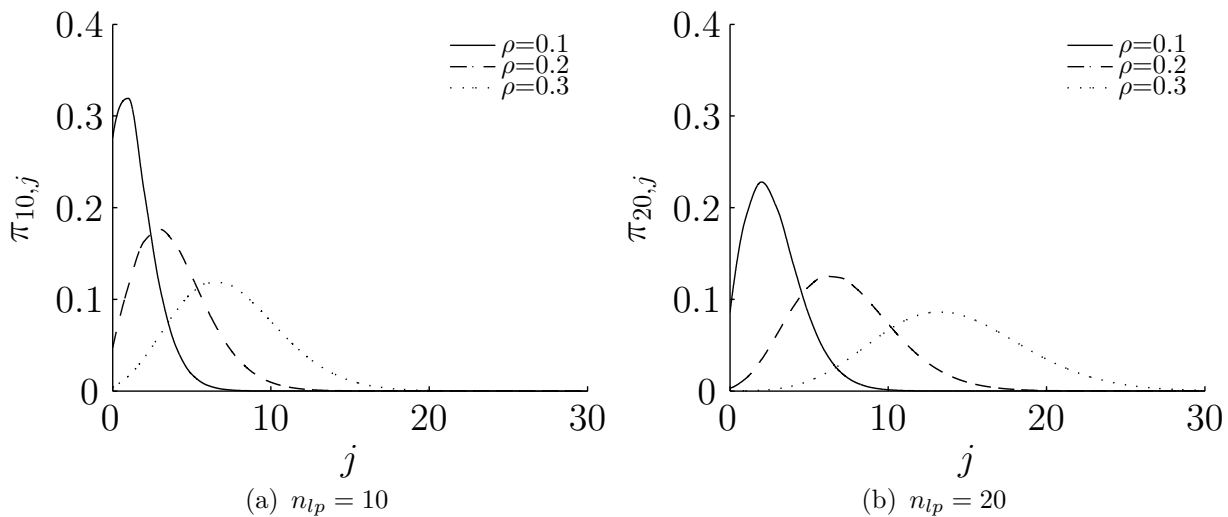


Abbildung 3.10: Wahrscheinlichkeit, dass sich j Nachrichten in den Warteschlangen vor einer Nachricht befinden

Betrachtet man nun die Ergebnisse in Abbildung 3.10, so werden zwei Eigenschaften schnell deutlich. Zum einen fällt die Wahrscheinlichkeit, dass eine Nachricht auf ihrem Weg durch das Netz hinter j Nachrichten warten muss, für große j nicht auf Null ab, sondern wird sehr klein. Der Grund hierfür liegt

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

in den Poisson-Ankunftsprozessen der Warteschlangen, die auch die Ankunft sehr vieler Nachrichten zulassen, wenn auch mit sehr geringer Wahrscheinlichkeit. Dieser Umstand ist unproblematisch wenn man die Wahrscheinlichkeiten betrachtet, die für realistische Werte von j bei $\pi_{10,j} < 10^{-9}$ liegen.

Problematischer ist in diesem Zusammenhang die stationäre Zustandsverteilung π_j , aus der auf die Länge der Warteschlange geschlossen werden kann. Wird die maximale Länge der Warteschlange überschritten, können ankommende Nachrichten nicht gespeichert werden und werden verworfen. Diese Nachrichten sind also verloren. Um den Verlust von Nachrichten zu vermeiden, muss die Warteschlange daher so lang dimensioniert werden, dass der Fall, dass eine Nachrichten bei voller Warteschlange ankommt, ausgeschlossen werden kann. Dieser Wert für die Länge der Warteschlange kann mit dieser Analyse nur in Verbindung mit einer Wahrscheinlichkeit für den Verlust von Nachrichten angegeben werden. Hier ist nicht der eigentliche Wert problematisch, wie bei $\pi_{10,j}$ auch ist er sehr klein, als vielmehr die Frage, ob das Modell in dieser Größenordnung noch gültig ist. Dieser Umstand würde einen Einsatz von *SafeNet* in sicherheitsrelevanten Systemen in der bisher vorgestellten Form nicht zulassen, da keine obere Schranke für die Wartezeit der Nachrichten τ_w existiert.

Wichtig ist auch, dass sowohl π_j als auch $\pi_{10,j}$ von der Auslastung des Netzwerkes, also der Buslast auf den einzelnen Verbindungen, abhängen. Ändert sich die Buslast, so ändert sich das Verhalten des Systems. Fällt nun ein Knoten aktiv aus (*babbling idiot*), so ist, abgesehen von der Tatsache, dass basierend auf dem bisher vorgestellten Algorithmus kein Kriterium für diesen Ausfall definiert werden kann, nicht gewährleistet, dass die Länge der Sendewarteschlange ausreicht und dass die Randbedingungen, die an den Versand von Nachrichten gestellt werden, eingehalten werden können. Um *SafeNet* also einsetzen zu können, muss die Auslastung des Netzwerkes auf einen Wert begrenzt werden, der eine quasi-deterministische Auslieferung der Nachrichten ermöglicht. Erfahrungswerte mit dem CAN-Bus zeigen, dass bei einer Auslastung von bis zu circa 20 % von einem annähernd deterministischen Verhalten ausgegangen werden kann [EPZ02], [NRK04a], siehe auch Kapitel 5.

3.6 Erweiterung des lokalen Sendalgorithmus

SafeNet ist also mit dem bisher vorgestellten Vorgehen beim Versenden von Nachrichten, welches als $M/D/1$ -Warteschlange modelliert werden kann, nicht in der Lage, eine deterministische Auslieferung von Nachrichten zu gewährleisten. Ein Grund hierfür ist, dass ein Knoten mit einem aktiven

3.6 Erweiterung des lokalen Sendalgorithmus

Ausfall das gesamte Netzwerk wenn nicht komplett lahmlegen, so doch über Gebühr belasten kann.

Eine Begrenzung der Sendefrequenz eines Knotens ist aber, aufgrund des verteilten Aufbaus von SafeNet, nur lokal möglich. Hierzu soll zuerst die Grenze für ein stabiles Netz untersucht werden. Anschließend wird eine detailliertere Modellierung mit Hilfe von Warteschlangen vorgestellt und das Sendeverhalten weiter erläutert.

Allgemein muss gelten, dass im Mittel die Empfangsraten $\lambda_{RX1,K}, \lambda_{RX2,K}$ auf den Eingängen eines Knotens K und die Rate λ_K , mit der der Knoten selbst Nachrichten erzeugt kleiner der Senderate $\lambda_{TX,K}$ sein müssen.

$$\lambda_{RX1,K} + \lambda_{RX2,K} + \lambda_K < \lambda_{TX,K} \quad K \in \mathcal{N} \quad (3.16)$$

Aufgrund der Tatsache, dass ein SafeNet stark zusammenhängend ist, ein Knoten also auch an sich selbst sendet, ergibt sich aus Gleichung 3.16, dass

$$\lambda_{TX,K} > \lambda_{TX,K} \quad \forall K \in \mathcal{N}$$

gelten muss, was auf ein instabiles Netzwerk führen würde.

Betrachtet man ein Netzwerk mit sechs Knoten $A - F \in \mathcal{N}$ in dem die Knoten so verbunden sind, dass A an B und C, B an C und D, ... sendet, also in einem chordalen Ring angeordnet sind, so ergibt sich für den Sender der Nachrichten, die an A ankommen, der in Abbildung 3.11 gezeigte aufspannende Binärbaum.

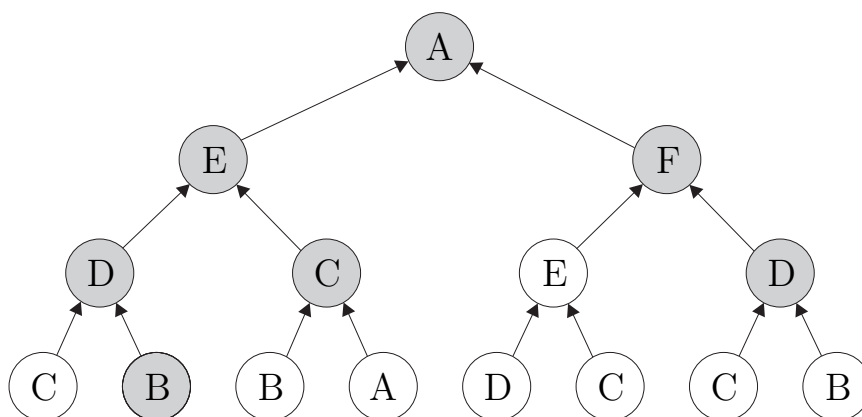


Abbildung 3.11: Aufspannender Binärbaum der ursprünglichen Sender der Nachrichten, die an Knoten A ankommen

Dabei fallen lediglich Nachrichten von den grau hinterlegten Knoten ins Gewicht, da die Nachrichten der anderen Knoten redundant sind und daher

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

nicht zur Sendewarteschlange beitragen. Aus Abbildung 3.11 ist ersichtlich, dass jeder Knoten in der Lage sein muss, die Nachrichten aller N Knoten zu versenden. Es muss also

$$\sum_{K \in \mathcal{N}} \lambda_K < \lambda_{TX,I} \quad \forall K, I \in \mathcal{N} \quad (3.17)$$

gelten.

Wird nun angenommen, dass kein Knoten im Netzwerk Priorität gegenüber einem anderen hat und alle Knoten über die gleiche Sendegeschwindigkeit verfügen, so muss die Senderate der Knoten bei

$$\lambda_K < \frac{1}{N} \lambda_{TX,K} \quad \forall K \in \mathcal{N} \quad (3.18)$$

liegen um ein stabiles Verhalten des Systems zu gewährleisten. Jeder Knoten darf also im Mittel lediglich weniger als ein N -tel der Netzwerkkapazität belegen.

Trotz der deterministischen Servicezeit ist das hier gezeigte Modell von *SafeNet* für $\rho = 1$ nicht stabil, weshalb in Gleichung 3.18 kleiner und nicht kleiner gleich verwendet werden muss. Nach dem Gesetz von Little [Kie06] gilt für die mittlere Kundenzahl \bar{k} in einer Warteschlange

$$\bar{k} = \frac{\mathbb{E}\{\tau_d\}}{\mathbb{E}\{\tau_a\}} = \frac{\bar{\tau}_s + \bar{\tau}_w}{\bar{\tau}_a} \quad (3.19)$$

wobei τ_s die Service-, τ_w die Warte-, τ_d die Durchlauf- und τ_a die Ankunftszeit darstellt. Mit

$$\bar{k} \cdot \bar{\tau}_a = \sum_{k=1}^{\infty} k \pi_k \cdot \int_0^{\infty} \tau f_a(\tau) d\tau \quad (3.20)$$

$$= \sum_{k=1}^{\infty} k \pi_k \cdot \int_0^{\infty} \tau \lambda e^{-\lambda \tau} d\tau \quad (3.21)$$

$$= \frac{1}{\lambda} \sum_{k=1}^{\infty} k \pi_k \quad (3.22)$$

und $\bar{\tau}_s = \tau_s$ ergibt sich für die mittlere Wartezeit also

$$\bar{\tau}_w = \frac{1}{\lambda} \sum_{k=1}^{\infty} k \pi_k - \tau_s. \quad (3.23)$$

3.6 Erweiterung des lokalen Sendealgorithmus

Die Annahme, dass π_k nach einem globalen Maximum streng monoton fallend ist wird an dieser Stelle nicht bewiesen, da sonst mit dem Minorantenkriterium und dem Vergleich mit

$$\lim_{k \rightarrow \infty} \sum_{k=1}^{\infty} k = \infty$$

die Divergenz der Reihe in Gleichung 3.23 und damit die Unbeschränktheit von τ_w gezeigt werden kann.

Da also π_k ab einem π_{max} streng monoton fallend ist, gilt

$$\bar{\tau}_w = \frac{1}{\lambda} \sum_{k=1}^{k_{max}-1} k\pi_k + \frac{1}{\lambda} \sum_{k=k_{max}}^{\infty} k\pi_k - \tau_s \quad (3.24)$$

$$> \lim_{k^* \rightarrow \infty} \left(\frac{1}{\lambda} \sum_{k=1}^{k_{max}-1} k\pi_k + \frac{1}{\lambda} \sum_{k=k_{max}}^{k^*} \pi_k k \right) - \tau_s. \quad (3.25)$$

Da aber

$$\sum_{k=k_{max}}^{k^*} k = \frac{k^*(k^* + 1)}{2} - \sum_{k=1}^{k_{max}-1} k \quad (3.26)$$

ist und damit für $k^* \rightarrow \infty$ nicht gegen Null konvergiert, gilt mit $\pi_k > 0 \forall k$ dass

$$\sum_{k=k_{max}}^{k^*} \pi_k k < \sum_{k=k_{max}}^{k^{*'}} \pi_k k \Leftrightarrow k^* < k^{*'} \forall k^*, k^{*'} > k_{max}.$$

Hierdurch ist das notwendige Kriterium [BSMM97] für die Konvergenz von $\bar{\tau}_w$ verletzt und $\bar{\tau}_w$ divergiert.

Der Grund für die Divergenz ist die Tatsache, dass durch die Modellierung des Ankunftsprozesses als Poisson-Prozess die Ankunft vieler Nachrichten während der Servicezeit einer Nachricht unwahrscheinlich aber nicht unmöglich ist, $p_s(k_a)$ also für große k_a nicht Null wird.

Betrachtet man aber die an einen Knoten während einer Servicezeit ankommenden Nachrichten, siehe Abbildung 3.12, so wird deutlich, dass die Anzahl der maximal ankommenden Nachrichten genau dann endlich ist, wenn sowohl die kürzeste als auch die längste Servicezeit, die in dem System vorkommen, endlich sind.

Genauer gesagt können bei gegebener minimaler und maximaler Servicezeit $\tau_{s,min}$, beziehungsweise $\tau_{s,max}$ genau $\frac{\tau_{s,max}}{\tau_{s,min}}$ Nachrichten auf jedem Eingang

3 Netzwerkbasierendes Kommunikationsmedium (*SafeNet*)

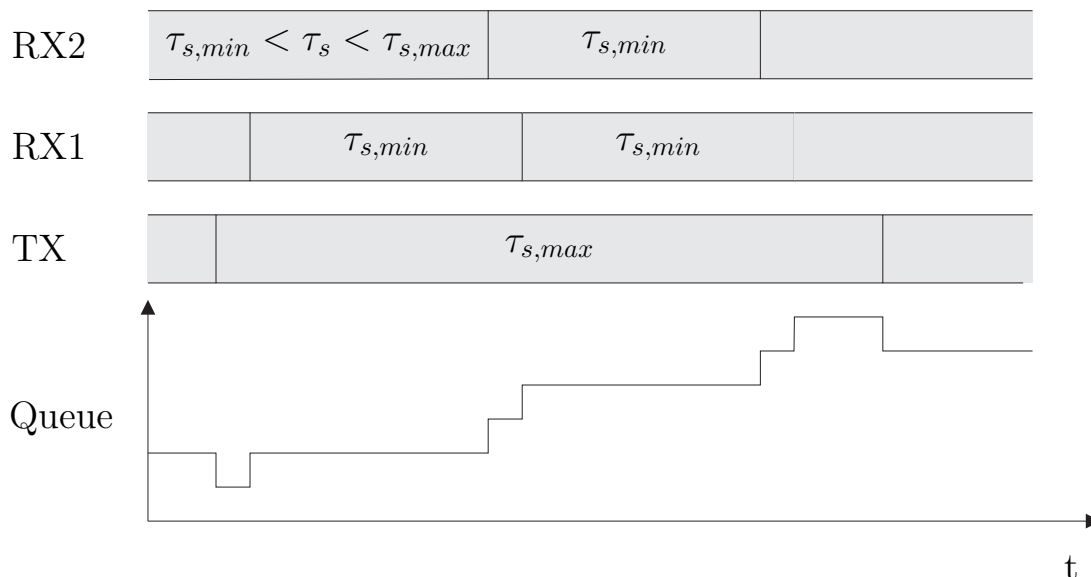


Abbildung 3.12: An den Eingängen eines SafeNet-Knotens ankommende Nachrichten

ankommen, da diese ja von den upstream Knoten gesendet werden müssen, also um die entsprechende Servicezeit verzögert werden. Hinzu kommen noch die Nachrichten, die ein Knoten selbst generiert.

Auf dieser Grundlage wird nun der Sendalgorithmus erweitert. Ein Knoten darf, wie in Abbildung 3.13 gezeigt, nur dann eine eigene Nachricht in die Sendewarteschlange stellen, wenn seine letzte eigene Nachricht bereits auf mindestens einem der Eingänge empfangen wurde.

Im Folgenden wird mit dem Begriff „SafeNet“ ein SafeNet, das mit dem erweiterten Sendalgorithmus arbeitet bezeichnet, falls nicht explizit anders angegeben.

Durch den so erweiterten Sendalgorithmus wird die in Gleichung 3.18 formulierte Bedingung für die Stabilität des Netzwerkes nicht zwingend eingehalten. Die Einschränkung erlaubt es aber, aus einem lokal beobachtbaren Parameter, der Zeit, die eine eigene Nachricht braucht, um wieder zum Ursprungsknoten zurückzukehren, auf die globale Auslastung des Netzes zu schließen, da mit steigendem ρ die Wahrscheinlichkeit, dass die eigene Nachricht auf ihrem Weg durch das Netz warten muss, ebenfalls steigt.

Betrachtet man sich den Sendeprozess genauer, so fällt auf, dass ein wichtiger Aspekt in den bisherigen Modellen nicht repräsentiert ist. Nachrichten, die bereits auf dem anderen Eingang eines Knotens empfangen wurden, werden nicht in die Sendewarteschlange gestellt. Deutlich wird dies an dem in Abbildung 3.14 gezeigten Ausschnitt aus einem chordalen Ring.

Für die folgende Betrachtung gelte, dass die Nachrichten die gleiche Länge,

3.6 Erweiterung des lokalen Sendalgorithmus

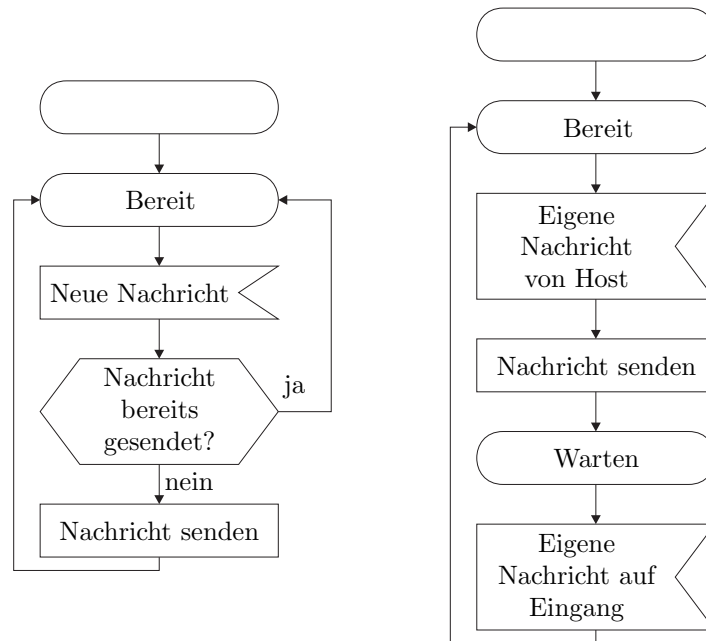


Abbildung 3.13: Der erweiterte Sendalgorithmus von SafeNet

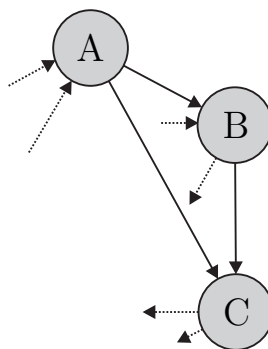


Abbildung 3.14: Ausschnitt aus einem chordalen Ring

also die gleiche Servicezeit τ_s besitzen. Die Zeit, die zur Verarbeitung in den Geräten benötigt wird, wird im Folgenden vernachlässigt. Mit Änderungen können die angestellten Überlegungen auf den allgemeinen Fall variabler Nachrichtenlänge erweitert werden. Bei Verwendung von gleich langen Nachrichten kommen an einem Knoten unter Umständen zwei Nachrichten gleichzeitig an. In diesem Beispiel sei daher der äußere Ring, also A-B-C-..., priorisiert.

Geht man von einem Netzwerk in Ruhe aus, in dem also keine Nachrichten auf den Verbindungen versendet werden und alle Sendewarteschlangen leer sind, so können sowohl Knoten A als auch Knoten B im ersten Schritt eine Nachricht generieren, α und β . Nach Ablauf der Servicezeit, also nach dem Versenden der Nachrichten, liegen in C also zwei neue Nachrichten in der

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

Sendewarteschlange, von denen eine direkt versendet werden kann. Generiert C nun noch eine eigene Nachricht γ , so beträgt die Länge der Sendewarteschlange zwei. Aufgrund der Priorisierung wird entweder die Nachricht β oder die Nachricht γ versendet, abhängig davon, ob γ generiert wurde bevor β in die Sendewarteschlange gestellt wurde oder nicht. Während C diese Nachricht sendet, wird auf den Eingängen eine Nachricht, die A von seinen upstream Knoten empfangen hat sowie α , dieses Mal von B gesendet, empfangen. Diese Nachricht wurde von C bereits empfangen und verlängert die Sendewarteschlange daher nicht. Die Länge der Sendewarteschlange bleibt also auch in den folgenden Schritten konstant bei zwei. Erst wenn γ auf einem der beiden Eingänge oder β von A empfangen wird, sinkt die Länge der Warteschlange wieder.

Es gilt der folgende Satz:

Satz 3.5 (Maximale Queuelänge im chordalen Ring) *Die maximale Länge der Sendewarteschlange $q_{max,I} \forall I \in \mathcal{N}$ in einem chordalen Ring der Skiplänge zwei, bei dem der äußere Ring priorisiert ist, beträgt bei konstanter Nachrichtenlänge unabhängig von der Anzahl der im Netzwerk vorhandenen Knoten $q_{max,I} = 2$.*

Durch die Erweiterung des Sendalgorithmus ist es also möglich, die maximale Länge der Sendewarteschlange der einzelnen Knoten zu bestimmen. Wichtig ist es aber zu berücksichtigen, dass die maximale Länge der Sendewarteschlange topologieabhängig ist. Schon das Vertauschen der Eingänge eines Knotens im chordalen Ring würde dazu führen, dass die Beschränkung bezüglich der Warteschlangenlänge auf q_{max} und der sich daraus ergebenden maximalen Laufzeit hinfällig würde.

Der chordale Ring stellt in dieser Beziehung² ein Optimum dar. Die maximale Länge der Sendewarteschlange wird durch die Tatsache begrenzt, dass die Hälfte der Nachrichten, die empfangen werden, nicht in die Sendewarteschlange gestellt werden, da sie bereits gesendet wurden. Eigene Nachrichten können so interpretiert werden, dass beim ersten Empfangen einer eigenen Nachricht eine neue eigene Nachricht generiert werden kann und somit eine, wenn auch nicht die empfangene Nachricht in die Warteschlange gestellt wird. Im chordalen Ring ist die Zeit zwischen ersten und zweiten Empfangen einer Nachricht kurz und aufgrund der Struktur von der Anzahl der Knoten im Netzwerk unabhängig.

Auch in anderen Topologien ist die maximale Länge der Sendewarteschlange beschränkt, wenn auch nicht auf $q_{max} = 2$. Wird in einem chordalen Ring

²Da die Beschränkung der Sendewarteschlange eine wichtige Eigenschaft ist, aus der vor allem die maximale Laufzeit abgeleitet werden kann, ist der chordale Ring im Allgemeinen eine vorteilhafte Struktur.

3.6 Erweiterung des lokalen Sendealgorithmus

ein Knoten durch eine Struktur, wie sie in Abbildung 3.6 gezeigt ist, ersetzt, so erhöht sich q_{max} . Grund hierfür ist der Unterschied zwischen dem kürzesten Weg und dem kürzesten Weg über den anderen upstream Knoten. Hierdurch dürfen die Knoten des Paares bereits wieder eine Nachricht generieren, unabhängig davon, wie weit die erste Nachricht bereits durch das Netzwerk propagiert ist. Daher ist die maximale Länge der Sendewarteschlange in dieser Topologie abhängig von dem kürzesten Weg über den zweiten upstream Knoten und damit von der Anzahl der im Netzwerk vorhandenen Knoten.

Die beiden Knoten des Paares dürfen im Vergleich zu den anderen Knoten im Netzwerk auch mehr Nachrichten versenden, da ihre eigene Nachricht sie ja schneller wieder erreicht. Es ist also in SafeNet möglich, in gewissen Grenzen ein *topologisches load-balancing* oder eine *topologische Priorisierung* von Knoten zu ermöglichen, indem Knoten, die viele Nachrichten versenden müssen, in einem Paar angeordnet werden. Dieser Effekt tritt aber erst bei sehr hohen Netzwerkauslastungen von $\rho \approx 1$ auf. In Abschnitt 5.1 wird auf diese Eigenschaft detaillierter eingegangen.

Neben der Beschränkung der Länge der Sendewarteschlange besitzt der erweiterte Sendealgorithmus auch die Eigenschaft, die Auslastung des Netzwerkes zu beschränken. Diese Beschränkung erfolgt aber nicht wie in der Analyse mit Hilfe der Warteschlangen zu Beginn dieses Kapitels angenommen zu circa $\rho \leq 0.2$ sondern zu $\rho \leq 1$.

Satz 3.6 (Auslastung eines SafeNets) *In einem SafeNet, das mit dem erweiterten Sendealgorithmus arbeitet, ist die Auslastung $\rho \leq 1$.*

Beweis 3.4 *Ein Knoten benötigt für das Senden und den Empfang von Nachrichten die gleiche Zeit τ_s . Eine Nachricht eines anderen Knotens wird genau ein Mal auf jedem Eingang empfangen, siehe Satz 3.2, beziehungsweise Satz 3.3, aber nur ein Mal versendet. Jede eigene Nachricht wird nach ihrem Versenden ein Mal auf jedem der Eingänge empfangen, ohne weiterversendet werden zu müssen. Die Auslastung des Systems ist also halb so groß wie die Auslastung beider Eingänge zusammen. Da an den Eingängen aber höchstens immer eine Nachricht anliegen kann, die Auslastung also $\rho = 1$ sein kann, ist die Auslastung des Systems $\rho \leq 1$.*

Vor den weiteren Ausführungen zu SafeNet wird an dieser Stelle eine kurze Betrachtung zur Netzwerkauslastung gegeben. Im Fall von $M/M/1$ Warteschlangen bedeutet eine Auslastung $\rho = 1$, dass die Parameter λ im Ankunfts- und μ im Serviceprozess gleich sind, mithin also

$$f_a(\tau) = f_s(\tau) \Leftrightarrow \lambda e^{-\lambda\tau} = \mu e^{-\mu\tau}, \quad \tau \geq 0$$

3 Netzwerkbasierendes Kommunikationsmedium (*SafeNet*)

gilt. Die mittlere Kundenzahl \bar{k} ist nach [Kie06] in einer $M/M/1$ Warteschlange als

$$\bar{k} = \frac{\rho}{1 - \rho}$$

gegeben. Geht nun $\rho \rightarrow 1$, so folgt also $\bar{k} \rightarrow \infty$ und damit auch für die mittlere Wartezeit $\bar{\tau}_w = \bar{k} \cdot \bar{\tau}_s \rightarrow \infty$. Es ist also in einer $M/M/1$ Warteschlange mit der Auslastung $\rho = 1$ nicht möglich, die Kunden in endlicher Zeit zu bedienen.

Anders verhält sich ein *SafeNet*. Wie bereits zuvor ausgeführt, ist die Dichtefunktion des Ankunftsprozesses beschränkt. Da es nach den bisher gegebenen Ausführungen auch möglich ist, dass zwei oder drei Nachrichten während der Servicezeit einer Nachricht ankommen, kann es also dazu kommen, dass bei der Auslastung $\rho = 1$ mehr Nachrichten ankommen als versendet werden können. Da die Wahrscheinlichkeit, dass keine oder eine Nachricht während der Servicezeit einer Nachricht ankommt, die Länge der Sendewarteschlange also sinkt oder zumindest konstant bleibt, mit

$$P_s(1) = \sum_{i=0}^1 p_s(k_a) = \frac{2}{e}$$

kleiner ist als die Wahrscheinlichkeit, dass zwei oder drei Nachrichten ankommen, die Länge der Warteschlange also steigt, ist auch in dieser Modellierung nicht mit einer endlichen mittleren Länge der Sendewarteschlange und somit mit Stabilität zu rechnen. In der Logistik spricht man in diesem Zusammenhang von einem *backlog*, also einem Rückstand, der nicht abgearbeitet werden kann. Mit jedem der wahrscheinlichen Ereignisse, dass zwei oder drei Nachrichten innerhalb eines Serviceintervalls ankommen, verlängert sich die Wartezeit der nachfolgenden Nachrichten.

Eine Auslastung von $\rho > 1$ würde also informell beschrieben bedeuten, dass wesentlich mehr Nachrichten am Eingang des Systems ankommen, als es am Ausgang versenden kann. Dies ist zum Beispiel möglich, wenn ein System zwei Eingänge, aber nur einen Ausgang besitzt, was bei *SafeNet* der Fall ist, oder aber die Zeit zum Abarbeiten des Kunden länger ist als die minimale Zwischenankunftszeit der Kunden am Eingang, man denke an eine Schicht im OSI-Modell, die aufgrund der Protokolle längere Nachrichten an die unten liegenden Schichten senden muss als sie selbst von den höher gelegenen Schichten empfängt (Anfügen eines CRC, etc.).

In *SafeNet* ist jedoch die Hälfte der Nachrichten redundant in dem Sinne, dass sie nicht zur Verlängerung der Sendewarteschlange beitragen. Außerdem ist sichergestellt, dass nach einem Intervall, indem in jedem Zeitschritt

3.6 Erweiterung des lokalen Sendealgorithmus

mehr als eine neue Nachricht empfangen wurde, wieder redundante Nachrichten ankommen. Dies bedeutet explizit, dass die Markov-Eigenschaft der Gedächtnislosigkeit des Ankunftsprozesses, wie sie in [Kie06] definiert ist, verletzt wird.

Die Markov-Eigenschaft ist aber der Grund dafür, dass eine $M/M/1$ Warteschlange für $\rho = 1$ instabil wird. Da SafeNet diese Eigenschaft verletzt, ist es sinnvoll, zu überprüfen, ob SafeNet auch für den Fall $\rho = 1$ stabil ist.

Satz 3.7 (Stabilitätsgrenze) *SafeNet ist für eine Auslastung $\rho \leq 1$ stabil.*

Beweis 3.5 *Durch die beschränkte Länge der Sendewarteschlange q_{max} lässt sich Gleichung 3.19 als*

$$\bar{k} = \frac{\mathbb{E}\{\tau_d\}}{\mathbb{E}\{\tau_a\}} = \frac{\bar{\tau}_s + \bar{\tau}_w}{\bar{\tau}_a}$$

schreiben. Aufgrund der der FCFS Bedienstrategie der Warteschlange gilt für τ_w die Abschätzung $\tau_w \leq (q_{max} - 1)\tau_s$ und damit

$$\bar{k} \leq \frac{\tau_s + (q_{max} - 1)\tau_s}{\bar{\tau}_a} = \frac{q_{max}\tau_s}{\bar{\tau}_a}.$$

Da für $\rho \leq 1$ $\bar{\tau}_a \geq \tau_s > 0$ gilt, ist \bar{k} beschränkt und SafeNet somit auch für $\rho = 1$ stabil.

Für ein SafeNet mit dem erweiterten Sendealgorithmus gilt also der folgende Satz.

Satz 3.8 (Stabilität von SafeNet) *Ein SafeNet mit dem erweiterten Sendealgorithmus ist unabhängig von seiner Topologie und der Anzahl der Knoten im Netzwerk stabil.*

Beweis 3.6 *Aus Satz 3.6 und Satz 3.7 folgt, dass ein SafeNet mit dem erweiterten Sendealgorithmus immer stabil ist.*

Durch die Beschränkung der Länge der Sendewarteschlange ist also gewährleistet, dass die mittlere Kundenzahl in einem SafeNet beschränkt ist. Eine Konsequenz hieraus ist, dass die mittlere Wartezeit, die eine Nachricht in einem Knoten verbringt, ebenfalls immer beschränkt ist. Für den Einsatz in sicherheitsrelevanten Systemen ist aber ein anderes Datum relevant, nämlich die exakte Zeit, die eine Nachricht für den Durchlauf durch ein SafeNet benötigt, beziehungsweise für den Fall, dass diese Zeit nicht exakt angegeben werden kann, eine worst-case Abschätzung.

3 Netzwerkbasiertes Kommunikationsmedium (SafeNet)

Für diese Abschätzung wird zuerst wieder die Ausbreitung von Nachrichten allgemein betrachtet. In Abbildung 3.15(a) ist die Sendereihenfolge eines Knotens dargestellt, wobei der linke Eingang priorisiert ist. Die Nachricht γ' , die von dem rechten rechten upstream Knoten von M stammt, hat zu dem Zeitpunkt ihres Auftretens zwei Nachrichten vor sich in der Sendewarteschlange.

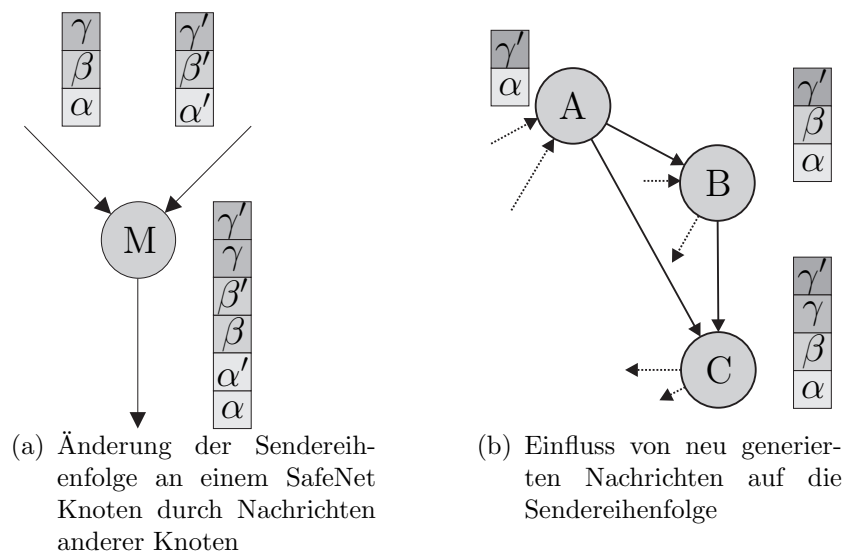


Abbildung 3.15: Sendereihenfolge empfangener Nachrichten in SafeNet

Im ersten Schritt empfängt M auf seinen beiden Eingängen α und α' . Von diesen wird im zweiten Schritt, in dem M β und β' empfängt, α wieder aus der Sendewarteschlange entfernt und versendet. Im dritten Schritt empfängt M γ und γ' und sendet α' . Die Nachricht γ' trifft also in M auf drei Nachrichten in der Sendewarteschlange, da sie auf dem nicht priorisierten Eingang empfangen wurde. Hierbei wird angenommen, dass es bei den Nachrichten keine Überschneidungen gibt, also jede Nachricht, die von M empfangen wurde, auch wirklich zur Länge der Sendewarteschlange beiträgt.

Allgemein lässt sich formulieren, dass eine Nachricht α , die in einem Knoten A q Zeitschritte auf ihr Versenden warten muss, also q Nachrichten vor sich in der Warteschlange sieht, in dem Knoten $M \in A_\alpha^+$ durch diesen Mechanismus höchstens $q + 1$ Nachrichten vor sich sieht.

In Abbildung 3.15(b) ist ein weiterer Aspekt der Nachrichtenpropagation zu sehen. Bei dem Weg einer Nachricht durch das Netz darf im worst-case jeder Knoten auf dem Weg eine eigene Nachricht vor der betrachteten Nachricht einfügen, wodurch sich die Laufzeit der Nachricht verlängert.

Betrachtet man die beiden oben genannten Faktoren gemeinsam, so lässt sich für die worst-case Laufzeit einer Nachricht von dem Knoten A zu dem

3.7 Analyse der Fehlerursachen und -auswirkungen

Knoten B die Abschätzung

$$\tau_{max} = (q + 2) \cdot l_{A,B} \cdot \tau_s \quad (3.27)$$

angeben, wobei q die Anzahl der Nachrichten, die Nachricht bei ihrer Erzeugung in A vor sich hat, darstellt und $l_{A,B}$ der geometrisch kürzeste Weg von A nach B ist [Tur96]. Die Verzögerungen stammen dabei von den auf dem anderen Eingang empfangenen Nachrichten, während die Nachrichten, die im gleichen Knoten in der Warteschlange stehen, die Laufzeit der Nachricht nicht mehr beeinflussen, also vor der Nachricht her geschoben werden.

Nach Gleichung 3.27 steigt die maximale Zeit, die eine Nachricht auf ihrem Weg durch das Netz benötigt linear mit der Anzahl der Knoten, die dabei passiert werden. Nun ist aber, wie in den vorherigen Abschnitten dargestellt, die maximale Länge der Sendewarteschlange begrenzt. Setzt man nun q_{max} für die maximale Länge der Sendewarteschlange ein, so ergibt sich für die maximale Laufzeit

$$\tau_{max} = (q_{max} + 1) \cdot l_{A,B} \cdot \tau_s \quad (3.28)$$

Aufgrund von unterschiedlichen momentanen Auslastungen der Knoten im Netzwerk kann eine Nachricht auch einen geometrisch längeren Weg zwischen zwei Knoten nehmen. Dies ist der Fall, wenn die Summe der Längen der Sendewarteschlangen der Knoten auf dem geometrisch kürzesten Weg länger ist als die Summe der Längen der Sendewarteschlangen auf einem geometrisch längeren Weg. In diesem Fall gilt die Abschätzung in Gleichung 3.28 jedoch weiterhin.

Mit Gleichung 3.28 ist es also möglich, die Forderung nach einer oberen Schranke für die Nachrichtenauslieferung, die an Echtzeitkommunikationssysteme gestellt wird, zu erfüllen. Es ist also trotz eines zufälligen Nachrichtenaufkommens möglich, eine deterministische maximale Laufzeit für *alle* Nachrichten zu garantieren. Für die anderen in Abschnitt 2.2 angeführten Anforderungen, nämlich die Fehlertoleranz und die sichere Trennung von aktiv ausgefallenen Knoten wurden Lösungsansätze bisher nur grob skizziert.

3.7 Analyse der Fehlerursachen und -auswirkungen

Für die genaue Analyse wurde bisher stets der fehlerfreie Fall betrachtet. In diesem Unterabschnitt wird der Einfluss verschiedener Ausfallarten auf SafeNet untersucht und hierauf aufbauend, Sicherungsmechanismen aufgezeigt. Besonderes Augenmerk wird dabei auf das Format der zu versendenden Nachrichten gelegt.

3.7.1 Ausfall einer Verbindung

Ein häufiger Fehler in einem Kommunikationssystem ist der Ausfall einer Verbindung, verursacht zum Beispiel durch eine defekte Steckverbindung. In einem klassischen Bussystem kann es durch den Wegfall einer Terminierung zu einem schlechteren Übertragungsverhalten des Kanals kommen. Da die Knoten in *SafeNet* über Punkt-zu-Punkt Verbindungen kommunizieren, ist es nicht möglich, dass der Ausfall einer Leitung die Signalqualität auf einer anderen beeinflusst.

Die Ursachen für den Ausfall einer Verbindung können vielfältig sein. Beispiele sind mechanische Belastungen wie Schwingbeschleunigungen oder Mikrobewegungen oder auch in die Steckverbindungen eindringende Feuchtigkeit und Schwallwasser [Bos98].

Es kann allgemein angenommen werden, dass Verbindungen, wie sie bei *SafeNet* verwendet werden, im Gegensatz zu aktiven Sternen immer passiv ausfallen, also nur existierende Nachrichten stören oder ihren Versand ganz verhindern aber keine neuen Nachrichten generieren. Hier ist mit dem Begriff *Ausfall einer Verbindung* gemeint, dass auf dieser Verbindung keine Daten mehr versendet werden. Andere Arten passiver Ausfälle werden im Folgenden unter syntaktischen und byzantinischen Fehlern behandelt.

Diagnostiziert wird der Ausfall einer Verbindung, indem ein Knoten M eine Nachricht μ von dem Knoten $A \in M^-$ zum Zeitpunkt t_0 zum ersten Mal erhält. Eine Nachricht muss nun vor dem Zeitpunkt $t_0 + \tau_{max}$ auf dem anderen Eingang empfangen werden, ansonsten ist die Verbindung ausgefallen.

Nachdem ein Knoten den Ausfall einer Verbindung diagnostiziert hat, muss dieser Ausfall allen anderen Teilnehmern im Netz mitgeteilt werden. Wird von zwei Knoten M und N , für die $M, N \in A^+$ gilt, der Ausfall einer Verbindung gemeldet, so ist nicht die Verbindung, sondern der Knoten passiv ausgefallen, was im nächsten Unterabschnitt behandelt wird.

Eine Reintegration von ausgefallenen Verbindungen im laufenden Betrieb ist nicht sinnvoll, da die oben aufgeführten Fehlerursachen tendenziell permanente Ausfälle verursachen. Es ist außerdem damit zu rechnen, dass sich die Kanaleigenschaften aufgrund der Fehlerursachen verschlechtern. Auch wenn Signale auf dieser Verbindung empfangen werden, ist damit zu rechnen, dass weitere Fehler entstehen.

Am Beispiel der ausgefallenen Verbindung zeigt sich ein anderes grundlegendes Problem der Fehlerbehandlung in *SafeNet*: Anhand eines Symptoms kann die Ursache nicht genau diagnostiziert werden. Grund hierfür ist, dass aufgrund der Struktur in *SafeNet* oft nur ein Knoten einen Fehler feststellen kann, es also nur einen Beobachter gibt. Es ist bei der Diagnose der ausgefallenen Verbindung nicht klar, ob der TX-Teil des Senders, der RX-Teil des

Empfängers oder die physikalische Verbindung defekt ist. Im Gegensatz zu Kommunikationssystemen mit gemeinsamem physikalischen Medium ist eine genauere Bestimmung der Fehlerursache nicht möglich, siehe auch den Group Membership Algorithmus von TTP/C in Unterabschnitt 5.3.3.

Was hier negativ formuliert ist, stellt sich aber bei genauerer Betrachtung als vorteilhaft heraus. Es ist im Gegensatz zu herkömmlichen Systemen in SafeNet nicht nötig, dass jeder Knoten in regelmäßigen Abständen oder im Extremfall mit jeder Nachricht dem übrigen System seine Sicht der Welt mitteilt. Wird keine Fehlermeldung versendet, so haben alle Knoten die Nachricht korrekt empfangen und sind daher im Besitz von konsistenten Datensätzen. Ein spezieller Mechanismus zur Sicherstellung der Datenintegrität ist in SafeNet nicht nötig, da die Fehlerdiagnosealgorithmen dieses Problem abdecken.

3.7.2 Passiver Ausfall eines Knotens

Ruft man sich den Aufbau eines SafeNets wie in Abbildung 3.2 dargestellt in Erinnerung so wird deutlich, dass durch den passiven Ausfall eines Knotens Verbindungen unterbrochen werden, da dieser Knoten keine Nachrichten mehr weiterleitet. Ist das Netzwerk gültig, so kann aber jede Nachricht trotz dieses Ausfalls alle funktionierenden Knoten des Netzwerkes erreichen. Die Sicherstellung der Funktionalität der Anwendung muss auf einer höheren Ebene sichergestellt werden und ist daher nicht Teil der hier angestellten Betrachtungen. Auf diesen Aspekt wird in Anhang A näher eingegangen.

Ursachen für den Ausfall eines Knotens können mechanischer Natur sein, also zum Beispiel durch einen Unfall verursacht werden. Unfälle im Kraftfahrzeug, die zur Zerstörung eines Steuergerätes führen, beschädigen aber mit hoher Wahrscheinlichkeit mehrere Steuergeräte beziehungsweise die Verbindungen zwischen ihnen und verletzen so die Ein-Fehler-Hypothese aus Satz 3.4. Ein Argument dafür, dass diese Hypothese trotzdem sinnvoll ist, lässt sich in der räumlichen Kompaktheit von Kraftfahrzeugen finden. Ein Fahrzeug, bei dem eine Beschädigung in einer Größenordnung auftritt, dass mehrere Steuergeräte mechanisch zerstört werden, kann als fahruntauglich gelten. Im Gegensatz zu einem Flugzeug ist es bei einem Kraftfahrzeug kaum möglich, bei einer Zerstörung eines Teils des Fahrzeuges den Betrieb des Gesamtsystems aufrecht zu erhalten. Ist dies in einem Flugzeug nötig, so befindet sich ein Kraftfahrzeug nach einem solchen Unfall in einem „sicheren“, also energielosen Zustand.

Muss die mechanische Zerstörung eines einzelnen Steuergerätes nicht berücksichtigt werden, so besteht noch die Möglichkeit, dass ein Steuergerät auf Grund von zum Beispiel Alterung ausfällt. Die Symptome von Alte-

3 Netzbasiertes Kommunikationsmedium (*SafeNet*)

rungeffekten sind vielfältig, werden aber oft bei dem Entwurf von BIST berücksichtigt, weshalb der Ausfall eines Knotens als wahrscheinliches Ereignis berücksichtigt werden kann. Die in einem Gerät vorhandenen BIST können auch andere Fehlerursachen auf das Symptom Ausfall eines Knotens abbilden.

Der Ausfall eines Knotens ist daher nicht zwingend als permanent anzusehen. Wird ein Fehler von dem BIST eines Knotens erkannt und dieser Knoten in einen sicheren Zustand überführt, beispielsweise durch ein Reset, so kann der Knoten während dieser Zeit als passiv ausgefallen diagnostiziert werden. Es ist jedoch sinnvoll einen Knoten, der wieder in einen sicheren Zustand versetzt wurde, in das Netzwerk aufzunehmen.

Diagnostiziert wird der Ausfall eines Knotens analog zum Ausfall einer Verbindung. Erhält ein Knoten M eine Nachricht μ von dem Knoten $A \in M^-$ zum Zeitpunkt t_0 zum ersten Mal, so muss eine Nachricht vor dem Zeitpunkt $t_0 + \tau_{max}$ auf dem anderen Eingang empfangen werden, ansonsten ist die Verbindung ausgefallen. Wird nun von beiden Knoten $N, M \in A^+$ der Ausfall der Verbindung zu A gemeldet, so ist A ausgefallen. Da der Ausfall passiv ist, müssen nach der Diagnose keine weiteren Maßnahmen getroffen werden. Die anderen Knoten des Netzwerkes sind bereits über den Ausfall informiert, da von zwei Knoten der Ausfall einer Verbindung gemeldet wurde. Die von dem ausgefallenen Knoten aus gesehen downstream gelegenen Knoten können auf ein Signal zu Wiedereingliederung des ausgefallenen Knotens hören, sofern dies spezifiziert ist. Dies kann sinnvoll sein, da durch fail-silent Verhalten viele Fehler auf den passiven Ausfall eines Knotens abgebildet werden.

Die Diagnose des passiven Ausfalls eines Knotens ist wieder ein Beispiel für einen verteilten Algorithmus. Obwohl jeder Knoten für sich genommen nur den Ausfall einer Verbindung diagnostizieren kann, ergibt sich im Gesamtsystem durch die Kommunikation zwischen den Knoten ein neues Bild. Durch die Kommunikation der Knoten und die Kombination ihrer Sichtweisen entsteht so konsistentes, umfangreicheres Wissen über den Zustand des Gesamtsystems als es die Teilsysteme mit ihren eingeschränkten Informationen einzeln herstellen können. Um den Ausfall eines Knotens durch einen zweiten so zu diagnostizieren, dass er von dem Ausfall einer Verbindung zu unterscheiden ist, müsste ein ungültiges *SafeNet* aufgebaut werden, in dem beide Ausgänge eines Knotens mit den Eingängen des anderen verbunden sind.

Ein weiterer Aspekt, der sich bereits hier zeigt ist, dass ein Knoten alleine nicht die völlige Isolation eines anderen verursachen kann.

3.7.3 Byzantinischer Fehler

Empfangen die beiden downstream Knoten eines gegebenen Knotens unterschiedliche, syntaktisch korrekte Nachrichten, so liegt ein byzantinischer Fehler vor³. Ein byzantinischer Fehler wird von mindestens einem Knoten im Netzwerk diagnostiziert, da sich die Ausbreitungswege der korrekten und der fehlerhaften Nachricht im Netzwerk kreuzen. Wird in dem Knoten, in dem sich die Wege kreuzen, die zweite Version der Nachricht empfangen, so kann der Fehler diagnostiziert werden.

Die Fehlerursachen, die zu byzantinischen Fehlern führen, sind in Abschnitt 2.1 skizziert. Die Frage, ob byzantinische Fehler als permanent zu betrachten sind, hängt auch davon ab, ob und wenn ja wie sie auf andere Fehlersymptome abgebildet werden können. Daher kann eine Antwort erst unter Berücksichtigung der verwendeten Fehleraufdeckungs- und Fehlerkorrekturmechanismen der Kanalcodierung gegeben werden.

Im Gegensatz zu den bisher behandelten Fehlern ist im Fall des byzantinischen Fehlers aber eine relativ aufwändige Diagnose nötig. Liegt eine Struktur vor, wie sie in Abbildung 3.16 gezeigt ist, so ist der aufdeckende Knoten nicht zwingend ein direkter downstream Knoten des fehlerhaften Knotens.

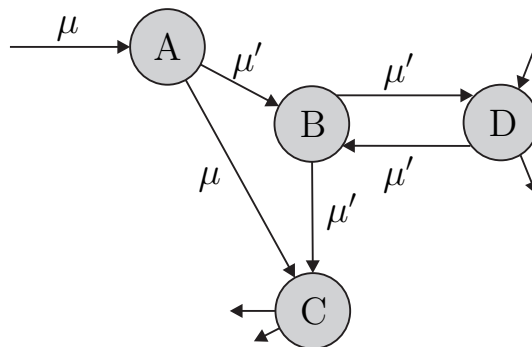


Abbildung 3.16: Propagation eines byzantinischen Fehlers bis zu seiner Aufdeckung

Ist aber die Topologie des Netzwerkes bekannt, so kann mit Hilfe der Adjazenzmatrix auf den Ursprung der fehlerhaften Version geschlossen werden. Voraussetzung hierfür ist, dass jeder Knoten im Netzwerk über die Versionen, die jeder andere Knoten empfangen hat, informiert ist, da die Fehlerfindung

³In Definition 4.5 wird der byzantinische Fehler abweichend definiert. Es wird dort von einem byzantinischen Fehler gesprochen, wenn ein Knoten zwei unterschiedliche Versionen einer Nachricht auf seinen Eingängen erhält. Grund für diese Verwendung des Begriffes ist, dass die Fehleraufdeckung verteilt in jedem Knoten stattfindet und sich die Knoten vor der Fehlerentdeckung nicht über die empfangenen Daten austauschen. Die in Definition 4.5 verwendete Sprachregelung bezieht sich im engeren Sinne auf den byzantinischen *Ausfall* eines Knotens, also auf Symptome des byzantinischen Fehlers. Da eine Unterscheidung in SafeNet aber im regulären Betrieb nicht möglich ist, wird dennoch der Begriff *byzantinischer Fehler* verwendet.

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

wieder in jedem Knoten lokal ausgeführt wird. Der Verursacher des byzantinischen Fehlers ist der upstream des Knotens, der am weitesten upstream liegt und die falsche Nachricht empfangen hat. Die betroffene Verbindung wird abgeschaltet.

Die Fehleraufdeckung kann vereinfacht werden, wenn der Ursprungsknoten der Nachricht beim Empfang einer Fehlermeldung über einen byzantinischen Fehler den Inhalt der Nachricht erneut versendet. In diesem Fall müssen nur Knoten, die die fehlerhafte Version der Nachricht erhalten haben, eine Fehlermeldung versenden. Problematisch ist hierbei, dass die erneut gesendete Version der Nachricht wieder eine geringe Hammingdistanz zu der fehlerhaften Version aufweist. Es ist also nützlich, den Inhalt der Nachricht durch eine möglichst eindeutige Repräsentation zu ersetzen, die zwei nahe beieinander liegende Datenworte auf zwei weit auseinander liegende Codeworte abbildet. Solche Funktionen werden *Hash* genannt und sind in für verschiedene Anwendungen optimierten Versionen Stand der Technik. Ein einfacher Hash ist beispielsweise die byteweise XOR-Verknüpfung. Hashfunktionen haben typischerweise eine feste Länge, die deutlich geringer ist als die Länge der Daten, die mit Hilfe des Hashes repräsentiert werden, beispielsweise *MD5*. Es kommt also zu Überschneidungen, so genannten Kollisionen, bei denen zwei unterschiedliche Datenworte auf den gleichen Hashwert abgebildet werden. Diese zwei Datenworte liegen aber bei einem guten Hash weit auseinander und sind daher für die Anwendung in *SafeNet* irrelevant. Das Versenden der korrekten Nachricht ist dennoch nötig, da alle Knoten sicher die korrekte Version erhalten müssen.

Byzantinische Fehler erfordern ein relativ hohes Maß an Kommunikation zwischen den Knoten und verbunden damit eine relativ lange Zeit zwischen Fehleraufdeckung und Fehlerlokalisierung. Ziel des Entwurfes der Randbedingungen eines Kommunikationssystems sollte es also sein, byzantinische Fehler wenn möglich zu vermeiden oder auf andere, einfacher zu lokalisierende Fehler abzubilden. Hier bietet es sich an, die Hammingdistanz zwischen den Codeworten groß zu wählen, da so die Wahrscheinlichkeit, dass eine Störung ein syntaktisch korrektes Codewort erzeugt, sinkt.

Byzantinische Fehler sind auch der Grund für die Parallelschaltung der Ausgänge eines *SafeNet* Knotens. Besitzt ein Knoten zwei unabhängige Ausgänge, wie zum Beispiel in [NMBK06] und [Nen06] angenommen, so existiert per Definition kein byzantinischer Fehler, da für jede Verbindung nur ein Beobachter existiert, wie dort argumentiert. Problematisch ist hierbei, dass durch die unabhängige Berechnung der Code- aus den Datenworten für jeden Ausgang innerhalb des Knotens eine weitere Fehlerquelle entsteht, die sich als „lügender“ Knoten, der dauerhaft unterschiedliche Nachrichten an seine downstream Knoten versenden kann, äußert. Für die Diagnose byzan-

tinischer Fehler ist diese Annahme nicht hilfreich, da dieselben, oben vorgestellten Algorithmen ausgeführt werden müssen.

Zudem wird durch einen zweiten, unabhängigen Ausgang die Fehlerkapselung erschwert, da zusätzlich zu den bisher beschriebenen Fehlerfällen auch berücksichtigt werden muss, dass ein Knoten dauerhaft kontextuell falsche Nachrichten an lediglich einen downstream Knoten sendet.

Es wird im Folgenden daher davon ausgegangen, dass ein Knoten auf beiden Ausgängen das selbe Signal anlegt, wie dies bereits in der grundlegenden Beschreibung der Netzwerktopologie angedeutet wurde, siehe Abschnitt 3.3.

3.7.4 Fehlerhafte Nachricht

Analog zu der Beziehung zwischen einer ausgefallenen Verbindung und einem passiv ausgefallenen Knoten existiert auch einen Fehler, der in beiden downstream Knoten als byzantinischer Fehler diagnostiziert wird. In diesem Fall wird also eine syntaktisch korrekte aber inhaltlich fehlerhafte Nachricht an beide downstream Knoten versendet.

Da dieser Fehler eng mit dem byzantinischen Fehler verwandt ist, liegt auch die Vermutung nahe, dass die Fehlerursachen ähnlich sind. Im Unterschied zu dem byzantinischen Fehler liegt die Fehlerursache hier aber vor der Aufspaltung der Signalfade, also in dem SafeNet Knoten selbst.

Die Fehleraufdeckung erfolgt analog zum byzantinischen Fehler, allerdings entdecken beide downstream Knoten des betroffenen Knotens den Fehler und trennen die Verbindungen zu dem betroffenen Knoten.

3.7.5 Babbling Idiot

Im Gegensatz zu den bisher aufgeführten passiven Fehlern handelt es sich bei dem *Babbling Idiot* Fehler um einen aktiven Ausfall. Verletzt ein Knoten die Sendebeschränkung, sendet er also bevor er seine eigene Nachricht auf mindestens einem Kanal wieder erhalten hat, so wird er als Babbling Idiot bezeichnet. Der Babbling Idiot ist in der Lage, die Funktionalität des Gesamtsystems zu gefährden, wenn er nicht entdeckt und behandelt wird.

Ein Babbling Idiot Fehler entsteht innerhalb eines SafeNet Knotens. Der Host, also das an das SafeNet angeschlossene Steuergerät selbst, unterliegt keiner Beschränkung, was die minimale Zwischenankunftszeit anbelangt. Im normalen, fehlerfreien Betrieb wird durch den erweiterten Sendalgorithmus verhindert, dass ein Knoten das Netz über Gebühr beansprucht, die Sendefrequenz des Hosts wird also begrenzt. Fällt diese Begrenzung weg, so kann der Host ungehindert zu jedem Zeitpunkt senden. Der Wegfall der Beschränkung

3 Netzwerkbasierendes Kommunikationsmedium (*SafeNet*)

ist also ein inaktiver Fehler, der erst durch eine unter anderen Umständen zulässige hohe Senderate des Hosts aktiviert wird.

Die Verletzung der Sendebeschränkung wird durch die beiden downstream Knoten des Babbling Idiots detektiert. Diese senden keine Nachrichten des betroffenen Knotens weiter, trennen die Verbindungen und informieren die restlichen Knoten des Netzes.

Erfahrungen mit einem am IIT entwickelten Hardware-in-the-Loop Prüfstand legen nahe, dass ein Babbling Idiot Fehler oft durch das Überführen des Knotens in einen sicheren Zustand, beispielsweise durch ein Reset, behoben werden kann. Daher ist der Babbling Idiot als transient einzustufen und die downstream Knoten des betroffenen Knotens können auf Signale zur Reintegration hören, sofern dies spezifiziert ist.

3.7.6 Masquerading

Da die Fehlerlokalisierung in *SafeNet* nur mit Hilfe der in Nachrichten versendeten Informationen erfolgen kann, ist es wichtig, den Ursprung einer Nachricht feststellen zu können. In Abschnitt 2.4 wurden dafür bereits mehrere Verfahren vorgestellt, nämlich das inhaltsbasierte Verfahren, welches bei CAN zum Einsatz kommt einerseits und das zeitbasierte der TDMA Systeme andererseits.

Wird die Zuordnung zwischen Inhalt der Nachricht und Herkunft hergestellt, so kann auf ein spezielles Feld innerhalb der Nachricht verzichtet werden, wenn ein Nachrichtentyp einem Knoten zugeordnet ist. Andererseits muss sichergestellt werden, dass keine zwei Knoten in einem Netzwerk den gleichen Inhalt versenden, wodurch unter Umständen mehr Nachrichtentypen benötigt werden.

Bei einer zeitbasierten Zuordnung zwischen Nachricht und Absender wird eine Verletzung der Zuordnung auf eine Verletzung des Sendalgorithmus und somit auf einen Babbling Idiot Fehler abgebildet. Ist das System ausreichend gegen einen Babbling Idiot Fehler geschützt, so ist eine eindeutige Zuordnung einer Nachricht zu ihrem Absender möglich.

Verletzt ein Knoten die Zuordnungsregel, die einer Nachricht ihren Absender zuordnet, so liegt ein Masquerading Fehler vor. Da in *SafeNet* ohne ein gemeinsames physikalisches Medium auch keine einheitliche Zeitbasis vorliegt, ist eine inhaltsbasierte Zuordnung hier vorteilhaft. Wird dabei sichergestellt, dass jeder Nachrichtentyp nur einem Knoten zugeordnet ist, so kann auf das Versenden einer Knotenkennung, wie sie in [NMBK06] vorgeschlagen wurde, verzichtet werden.

Die Aufdeckung von Masquerading Fehlern wird in *SafeNet* durch die Tatsache erschwert, dass für das korrekte Funktionieren des Netzwerkes jeder

Knoten jede Nachricht versenden muss. Ein Knoten kann daher nicht unterscheiden, ob eine Nachricht von seinem upstream Knoten korrekt weiter versendet wurde, oder ob sie von diesem Knoten fälschlicherweise generiert wurde.

Allgemein gilt aber, dass der Knoten, dem die Nachricht eigentlich zugeordnet ist, einen Masquerading-Fehler aufdecken kann. Für den detaillierteren Algorithmus zur Fehlerlokalisierung ist ein genaueres Verständnis des Aufbaus eines SafeNet Knotens nötig, weswegen dieser an gegebener Stelle (Abschnitt 3.9) vorgestellt wird.

3.7.7 Unterschlagen einer Nachricht

Wird eine Nachricht μ_i von einem Knoten A unterschlagen, also nicht weiter versendet, so ist es für die Knoten in A_μ^+ unter Umständen nicht möglich festzustellen, ob es sich bei der nächsten Nachricht um eine fehlerhafte Version von μ handelt oder bereits um μ_{i+1} .

Grund hierfür ist das Fehlen einer globalen Zeitbasis in SafeNet, die in Kommunikationssystemen mit gemeinsamem physikalischem Medium durch die Synchronität von Sende- und Empfangsvorgängen in den verschiedenen Geräten bedingt ist. Da diese Vorgänge in SafeNet entkoppelt voneinander geschehen und kein festes zeitliches Sendeschema existiert, kann unter Umständen eine eindeutige Bestimmung, ob es sich bereits um eine neue Nachricht oder um eine verfälschte Version einer bereits empfangenen Nachricht handelt, unmöglich sein, wenn nur Nachrichtentyp und Daten übertragen werden.

3.7.8 Ungültige Header-Daten

Die bisher vorgestellten Verfahren zur Fehlererkennung und Fehlerlokalisierung basieren alle auf einer korrekten Information über den Nachrichtentyp. Wird diese Information bei der Übertragung verfälscht, so versagen diese Verfahren und unter Umständen wird ein korrekt funktionierender Knoten aus dem Netzwerk ausgeschlossen.

Es ist daher notwendig, Informationen wie den Nachrichtentyp, die so genannten Header-Daten, so abzusichern, dass sie entweder zuverlässig korrekt sind oder als inkorrekt identifiziert werden können. Hierfür bieten sich die in Unterabschnitt 2.3.2 vorgestellten Verfahren der Kanalcodierung an. Wie bei FlexRay können die sensiblen Header-Daten separat abgesichert werden.

3.7.9 Ungültige Nachricht

Für den Rest der Nachricht, die Nutzdaten, gelten andere Regeln. Hier ist es weniger kritisch, wenn bei einer Übertragung aus einer Nachricht durch äußere Veränderung wieder eine syntaktisch korrekte Nachricht wird, da in diesem Fall ja noch immer die zweite, korrekte Version der Nachricht verschickt wird und die Fehlersicherungsmechanismen von *SafeNet* greifen.

Ein anderer, bereits in Unterabschnitt 2.3.2 diskutierter Aspekt kommt hingegen zum Tragen. Tritt bei der Übertragung zwischen zwei Knoten ein Fehler auf, so dauert es aufgrund der unidirektionalen Verbindungen lange, bis die Fehlermeldung den upstream Knoten erreicht hat und eine Übertragung erneut erfolgen kann. Mit den in Unterabschnitt 2.3.2 eingeführten Begriffen ausgedrückt bedeutet dies, dass ein sehr langsamer Rückkanal vorhanden ist.

Während der wiederholten Übertragung ist die Nachricht aber wieder den Störungen auf den einzelnen Übertragungstrecken ausgesetzt. Es besteht also die Wahrscheinlichkeit, dass die Nachricht erst nach mehreren Übertragungsversuchen alle Knoten im Netz korrekt erreicht.

Es ist daher sinnvoll, die Nutzdaten mit Hilfe der Kanalcodierung so zu schützen, dass kleine Fehler von dem jeweiligen Empfänger selbst zuverlässig korrigiert werden können. Hierzu bietet die Kanalcodierung die FEC-Verfahren an.

3.7.10 Nachrichtenformat

Aus den vorhergehenden Unterabschnitten lassen sich einige Forderungen an das zu verwendende Nachrichtenformat ableiten:

- eindeutige Identifizierbarkeit des Nachrichtentyps
- eindeutige Reihenfolge der Nachrichten gleichen Typs
- Absicherung dieser Header-Daten so, dass Übertragungsfehler entweder erkannt oder mit großer Wahrscheinlichkeit ausgeschlossen werden können
- Absicherung der Nutzdaten so, dass eine wiederholte Übermittlung durch häufige Fehler ein seltenes Ereignis wird.

Eine Struktur, die diese Anforderungen erfüllt, ist in Abbildung 3.17 zu sehen.

3.7 Analyse der Fehlerursachen und -auswirkungen

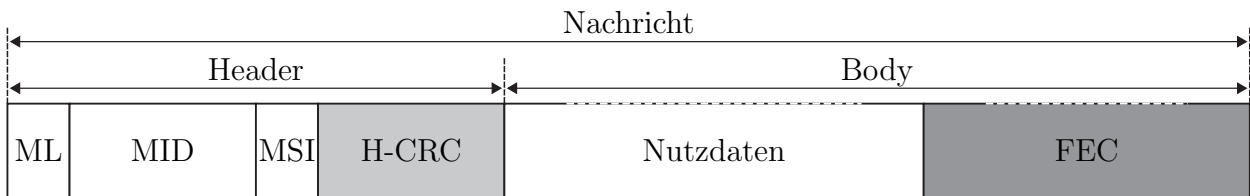


Abbildung 3.17: Grundstruktur einer Nachricht in SafeNet

Eine Nachricht besteht aus Header, also den Informationen, die für die Verwertung der Nachricht benötigt werden, einerseits und den Nutzdaten selbst, dem Body.

Der Header wiederum besteht aus vier Teilen:

Message Length (ML) Um das Versenden der Nachricht möglichst effizient zu gestalten, ist es vorteilhaft, die Länge der Nachricht an die Länge der tatsächlich zu versendenden Daten anzupassen. Die Länge des Datenfeldes im Body, in Abbildung 3.17 mit Nutzdaten beschriftet, kann über das Message Length Feld variiert werden.

Es ist sinnvoll, die maximale Länge des Feldes nicht zu hoch zu wählen, da insbesondere die Fehlerentdeckungs- und Fehlerkorrekturalgorithmen auf die höchste Nachrichtenlänge angepasst werden müssen. Werden kürzere Nachrichten versendet, so muss dieser overhead mit der kürzeren Nachricht mitversendet werden, was die Effizienz senkt.

Ein in Echtzeitanwendungen zudem wichtiger Punkt ist, dass für die worst-case Analyse der Laufzeiten bei variabler Nachrichtenlänge jeweils die längste Nachrichtenlänge anzunehmen ist, was sich negativ auf die Systemleistung auswirken kann, sollte die maximale Nachrichtenlänge übermäßig groß gewählt sein.

Sind nur gelegentlich lange Datensätze zu versenden so ist es sinnvoll, diese durch höhere Schichten in mehrere Nachrichten zu zerlegen, anstatt die maximale Nachrichtenlänge auf diese anzupassen. Hierdurch wird die worst-case Zeit, die zum Versenden von kürzeren Nachrichten benötigt wird, nicht negativ beeinflusst.

Message Identifier (MID) Zur korrekten Zuordnung des Inhaltes einer Nachricht, also zur eindeutigen Zuordnung einer Nachricht zu einem Nachrichtentyp dient das Message Identifier Feld. Wichtig ist hierbei, dass eine MID eindeutig einem Knoten zugeordnet werden kann. Hierdurch können durch die MID nicht nur Rückschlüsse auf den Inhalt einer Nachricht sondern auch auf den Absender gezogen werden.

3 Netzwerkbasierendes Kommunikationsmedium (*SafeNet*)

Die Länge des MID Feldes sollte sich an der Anzahl der in einem System nötigen Nachrichten orientieren. An diesem Punkt besteht ein Konflikt zwischen der Forderung nach einfacher Erweiterbarkeit, was für ein möglichst langes MID Feld sprechen würde, und einem möglichst effizienten Versand von Nachrichten, was ein kurzes MID Feld nahe legt.

Bei dieser Entscheidung ist eine Berücksichtigung des Einsatzbereiches des Kommunikationssystems nötig. Geht man davon aus, dass *SafeNet* aufgrund des gegenüber einer einfachen Busleitung erhöhten Aufwandes in der Verkabelung durch die topologieinherente Redundanz nur für sicherheitsrelevante Anwendungen eingesetzt wird, so ist es wahrscheinlich, dass ein *SafeNet* aus einer relativ geringen Zahl von Knoten besteht, die über wenige Nachrichten miteinander kommunizieren, da die Kommunikation zwischen Geräten im Vergleich zu der Kommunikation von Prozessen innerhalb eines Gerätes ineffizient ist, siehe auch [NRK04a]. Es ist daher davon auszugehen, dass sich die Anzahl der benötigten Nachrichten innerhalb eines Netzwerkes in einer ähnlichen Größenordnung bewegen wird, wie dies bei einem Standard-CAN der Fall ist.

Message Sequence Identifier (MSI) Um Nachrichten eines Typs unterscheiden zu können, also um eine nicht gesendete Nachricht beim Empfang der folgenden gleichen Typs von einer fehlerhaften Übertragung unterscheiden zu können, wird hier der Ansatz gewählt, alle Nachrichten eines Typs fortlaufend zu nummerieren. Da bereits eine fehlende Nachricht als Fehler interpretiert werden muss, genügt eine kurzes Feld.

Dieses Feld wird vom ursprünglichen Absender der Nachricht, dem die MID der Nachricht zugeordnet ist, bei jedem Versenden einer Nachricht erhöht.

Header Cyclic Redundancy Check (H-CRC) Die im Folgenden vorgestellten Algorithmen zur Fehlererkennung, Fehlerlokalisierung und Fehlerkapselung basieren alle auf der Annahme, dass die Headerdaten, also ML, MID und MSI korrekt übertragen werden. Daher ist eine starke Absicherung des Headers nötig.

Von den in Unterabschnitt 2.3.2 vorgestellten Verfahren, Fehlererkennung und Fehlerkorrektur, bietet sich hier die Fehlererkennung an. Grund hierfür ist, dass die Berücksichtigung falscher Headerdaten die Komplexität der Fehlererkennungs- und Fehlerlokalisierungsalgorithmen stark ansteigen lässt. Daher ist es vorteilhaft, Codeworte mit möglichst großer Hammingdistanz zu verwenden, um einer Verfälschung

3.7 Analyse der Fehlerursachen und -auswirkungen

bei der Übertragung, aus der wieder ein gültiges Codewort entsteht, vorzubeugen.

Fehlerkorrekturalgorithmen weiten nun im Signalraum aber die Codeworte aus und verringern so unbeabsichtigt auch die minimale Hammingdistanz von Codeworten, die auf unterschiedliche gültige Codeworte abgebildet werden. Es ist daher von Vorteil, auf Fehlerkorrektur zu verzichten und lediglich Fehlererkennung zu betreiben. Zudem ist der für eine effektive Fehlerkorrektur benötigte overhead bei kurzen Datensätzen relativ zur Nachrichtenlänge besonders hoch.

Der Datenteil (*Body*) einer SafeNet Nachricht besteht lediglich aus den Daten selbst und den mit FEC bezeichneten Daten für die Fehlerkorrektur. Die Abwägungen für die Länge des Datenfeldes wurden bereits im Zusammenhang mit dem ML Feld diskutiert.

Es stellt sich jedoch die Frage, wie die Daten und eventuell auch der Header zusätzlich durch fehlerkorrigierende Algorithmen abgesichert werden können, wie also das FEC Feld definiert werden kann.

3.7.11 Kanalcodierung in SafeNet

Bei der Auswahl eines Kanalcodierungsverfahrens für SafeNet ist darauf zu achten, dass die besonderen Rahmenbedingungen, die durch die Verteiltheit des Systems und die unidirektionalen Verbindungen gegeben sind, berücksichtigt werden. In SafeNet liegt, wie bereits erwähnt, durch die Verwendung von unidirektionalen Verbindungen ein langsamer Rückkanal vor. Daher ist die Verwendung eines stop-and-wait ARQ-Verfahrens, wie es in Unterabschnitt 2.3.2 vorgestellt wurde, nicht sinnvoll, zumal dieses Verfahren durch explizite Bestätigung die Netzwerkauslastung erhöht oder durch implizite Bestätigung, also dem Ausbleiben von Fehlermeldungen, die maximal mögliche Auslastung unterhalb von $\rho = 1$ limitiert.

Zudem muss sichergestellt werden, dass alle Nachrichten in der korrekten Version von allen Knoten empfangen wurden. Die Verwendung eines reinen FEC-Verfahrens, in dem nicht vorgesehen ist, dass eine Rückmeldung über den Empfang einer Nachricht gesendet wird, ist daher nicht möglich, da hier der Fall eintreten kann, dass ein Knoten zwar die richtige Version erhält, sie aber nicht von der falschen unterscheiden kann.

Es liegt daher nahe, ein hybrides ARQ-Verfahren zu verwenden, in dem sowohl eine Fehlerkorrektur zur Korrektur kleiner Bitfehler, als auch ein ARQ-Mechanismus eingesetzt wird. Der ARQ-Mechanismus kommt dabei nur zum Tragen, falls durch die Fehlerkorrektur nicht alle Fehler behoben werden

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

können. Konkret äußert sich dies beispielsweise durch zwei syntaktisch korrekte Versionen einer Nachricht. Durch den Einsatz eines FEC Algorithmus werden kleiner Fehler, die bei der Übertragung zwischen den Knoten auftreten, entkoppelt bevor sie zu einem nicht korrigierbaren oder sogar einem nicht entdeckbaren Fehler führen.

Wie zuvor schon diskutiert, sind syntaktische Fehler in *SafeNet* einfach und effektiv zu behandeln, während byzantinische Fehler mit einem deutlich höheren Aufwand in der Fehlerlokalisierung und Fehlerbehandlung verbunden sind. Wird die Korrekturfähigkeit eines Codes voll ausgeschöpft, so wird ein Teil der zuvor syntaktischen Fehler durch die Korrektur auf ein gültiges Codewort auf byzantinische Fehler abgebildet. Da byzantinische Fehler aber ein seltenes Ereignis darstellen sollten, ist es hilfreich, nicht alle Fehler zu korrigieren, sondern einen Teil der Codeworte in einer Rückweisungsklasse zusammenzufassen. Wird ein solches Codewort empfangen, so ist ein syntaktischer Fehler aufgetreten und der ARQ Mechanismus wird aktiv. Die Rückweisungsklasse trennt im Signalraum die Bereiche der gültigen Codeworte. Da Bereiche, in denen Codeworte als syntaktisch falsch angesehen werden zumindest näherungsweise symmetrisch zu beiden Seiten der Grenze zwischen Codeworten entsteht, sinkt durch die Einführung einer Rückweisungsklasse die Wahrscheinlichkeit, dass eine Nachricht bei der Korrektur auf ein falsches Codewort abgebildet wird.

Bei der Entscheidung, welches Verfahren bei der Fehlererkennung in *SafeNet* zum Einsatz kommt, musste auch berücksichtigt werden, dass über den Kanal relativ wenig bekannt ist, da die Erstellung eines genauen Kanalmodells über den Rahmen dieser Arbeit hinausgeht. Die Möglichkeit mit Hilfe der Blockcodierung sowohl Fehlerkorrektur als auch reine Fehlererkennung effizient durchführen zu können, gab daher den Ausschlag zugunsten von Blockcodes gegenüber Faltungscodes. Sollte im Laufe der Verifikation festgestellt werden, dass eine Fehlerentdeckung benötigt wird, so kann dies durch einen neuen Algorithmus im Empfänger, also ohne Änderungen im Nachrichtenformat oder im Sender bewerkstelligt werden.

Gegen die Verwendung von RS-Codes spricht die kurze Nachrichtenlänge und die Tatsache, dass in die Übertragung in *SafeNet* unmoduliert stattfindet. Diese Rahmenbedingungen sprechen für den Einsatz von BCH-Codes.

Ein weiterer Freiheitsgrad ist nun die Dimensionierung der verwendeten Codes. Die gesonderte Absicherung des Headers wird, wie in Unterabschnitt 2.4.2 bereits erwähnt, auch bei *FlexRay* eingesetzt. Da die Länge der zu sichernden Daten gleich ist, wird für *SafeNet* ebenfalls der in *FlexRay* eingesetzte (31,20) BCH-Code verwendet. Das Generatorpolynom findet sich in der *SafeNet* Spezifikation, Kapitel 4. Dieser Code besitzt eine Hammingdistanz von $d_H = 6$ und kann damit bis zu fünf fehlerhafte Bit in dem Header

erkennen [Fle05b].

Für den FEC Code wird ein $(127,92)$ BCH-Code eingesetzt, siehe auch [LC83]. Dieser ist aufgrund der Hammingdistanz $d_H = 11$ in der Lage, bis zu fünf Fehler in einem Codewort zu korrigieren. Wie oben diskutiert ist aber der Einsatz einer Rückweisungsklasse sinnvoll, wobei die Korrekturfähigkeit durch Satz 2.3 eingeschränkt ist.

3.8 Fehlerbehandlungsstrategien

Tritt ein Fehler in einem Netzwerk auf, so muss es oberstes Ziel der Fehlerbehandlung sein, die Kommunikation zwischen den fehlerfrei funktionierenden Teilen des Netzwerkes, also den von dem Fehler nicht betroffenen Knoten, weiter zu ermöglichen. Hierzu muss der Fehler möglichst schnell aufgedeckt, lokalisiert und gekapselt werden.

Prinzipiell gilt in SafeNet, dass ein Fehler besser gekapselt werden kann, also eine fehlerhafte Nachricht an weniger Knoten versendet wird, je schneller der Fehler entdeckt wird. Im Idealfall sind bereits die downstream Knoten des Verursachers in der Lage, den Fehler zu entdecken und somit eine Ausbreitung zu verhindern. Dies ist zum Beispiel der Fall, wenn eine Nachricht syntaktisch inkorrekt ist und von den downstream Knoten nicht weiterversendet wird.

Problematischer ist der byzantinische Fehler, da hier nicht mit Sicherheit bestimmt werden kann, wie weit sich die fehlerhafte Nachricht ausbreitet. Auch hier kann im Idealfall bereits der downstream Knoten den Fehler identifizieren, nämlich dann, wenn die korrekte Version der Nachricht bereits an dem downstream Knoten eingegangen ist. Es kann aber auch der Fall eintreten, dass fast alle Knoten zuerst die fehlerhafte Version der Nachricht erhalten und sich diese somit durch einen Großteil des Netzes, mit Ausnahme des ursprünglichen Senders, ausbreiten kann.

Den worst-case in dieser Beziehung stellt der Masquerading Fehler dar. Da er prinzipiell nur an zwei Stellen entdeckt werden kann, nämlich an dem fehlerhaften Knoten selbst, wie dies bei dem Bus Guardian beziehungsweise der Sternarchitektur der Fall ist, oder an dem Knoten, für den sich der fehlerhafte Knoten ausgibt. Alle anderen Knoten haben unabhängig von dem Kommunikationsmedium keine Möglichkeit die fehlerhafte von der fehlerfreien Nachricht zu unterscheiden.

In SafeNet gestaltet sich die Entdeckung des Masquerading Fehlers besonders schwer, da eine Entdeckung an dem fehlerhaften Knoten selbst nicht möglich ist. Wie bereits bei der Fehleranalyse bemerkt, muss für das Funktionieren des Protokolls jeder Knoten jeden Nachrichtentyp versenden. Die downstream Knoten können dabei nicht unterscheiden, ob es sich um ein

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

legitimes Weiterversenden handelt oder ein von dem Masquerading Fehler vorliegt.

Wird der Masquerading Fehler von dem Knoten, dem die MID eigentlich zugeordnet ist, entdeckt, so verbreitet sich die fehlerhafte Nachricht trotzdem durch das gesamte Netz, da aufgrund der Topologie immer mindestens ein Weg von dem Masquerading Fehler betroffenen Knoten zu allen anderen besteht. Es ist also davon auszugehen, dass sich ein Masquerading Fehler durch das gesamte Netz verbreitet.

Ziel der Fehlerentdeckung muss es also sein Fehler, insbesondere aktive, möglichst schnell zu entdecken und zu kapseln. Dies geschieht prinzipiell dadurch, dass Knoten fehlerhafte Daten nicht weitersenden. Liegt ein Fehler vor, kann aber nicht entschieden werden, welche der empfangenen Nachrichten korrekt ist, so muss eine Fehlermeldung statt der zweiten Nachricht versendet werden.

Es stellt sich nun die Frage, wie diese Fehlernachrichten im Detail auszu-sehen haben. Besonders zu berücksichtigen sind hier zwei Punkte.

Lokales Wissen Aufgrund der verteilten Struktur von *SafeNet* besitzt jeder Knoten nur lokales Wissen. Insbesondere ist es bei byzantinischen Fehlern nicht möglich, zwischen fehlerhafter und korrekter Version der Nachricht zu unterscheiden. Es kann auch, wie bereits zuvor erwähnt, der Fall eintreten, dass ein Knoten die fehlerhafte Version einer Nachricht auf beiden Eingängen erhält.

Falsche Fehlernachricht Prinzipiell ist es möglich, dass ein Knoten so ausfällt, dass er mit einer Fehlernachricht einen anderen „beschuldigt“, dort also fälschlicherweise einen Fehler lokalisiert. Auch dieser Fehler darf nicht zum Abschalten des korrekt funktionierenden Knotens oder gar zu einem Ausfall des gesamten Netzwerkes führen.

Die Lösung des Problems der falschen Fehlernachricht ist eng mit der bereits in Abschnitt 3.4 angesprochenen Problematik verwandt, dass die genaue Lokalisierung eines Fehlers oft nicht möglich ist. Die Fehlerursache, die zu einer syntaktisch inkorrekten Nachricht führt, kann beispielsweise in dem Bustreiber des Senders, in der elektromagnetischen Einkopplung auf der Verbindung oder im Bustreiber des Empfängers liegen. In Abschnitt 3.4 wurde bereits argumentiert, dass eine genauere Lokalisierung des Fehlers nicht nötig ist, da die Auswirkungen der Fehlerbehandlung in allen Fällen die gleiche wäre. Dieser Ansatz wird nun auch für das Problem der falschen Fehlernachricht verwendet.

Satz 3.9 (Fehlerlokalisierung) *Eine Fehlernachricht in SafeNet bezieht sich immer abstrakt auf die Verbindung zwischen zwei Knoten.*

Der Begriff *abstrakte Verbindung* wurde gewählt, da angenommen wird, dass Verbindungen, also die physikalische Verbindung über ein Medium nur passiv ausfallen können. Die abstrakten Verbindungen können jedoch auch aktiv ausfallen und zum Beispiel Babbling Idiot Verhalten an den Tag legen.

Eine konkrete Folgerung aus Satz 3.9 ist, dass ein Knoten, der einen Fehler in seinem upstream Knoten lokalisiert, auch immer die Möglichkeit offen lässt, dass der Fehler bei ihm selbst liegt. Diese Unterscheidung ist nicht erforderlich, da in beiden Fällen die gleiche Konsequenz gezogen wird.

Einfache Fehlermeldungen

Die Fehlermeldungen für die Fehler, die direkt in den downstream Knoten entdeckt werden können, sind ähnlich den Fehlermeldungen anderer Kommunikationssysteme aufgebaut. Es genügen hier wenige Daten:

1. verursachender Knoten
2. identifizierender Knoten
3. Fehlerart
4. gegebenenfalls Informationen über die Nachricht, bei der der Fehler aufgetreten ist

Mit diesen vier Informationen ist es für alle Knoten möglich den Fehler zu lokalisieren, wenn die Topologie des Netzwerkes zum Beispiel über die Adjazenzmatrix in jedem Knoten bekannt ist.

Über diesen Fehlermeldungstyp kann bereits ein Großteil der Fehlerfälle abgedeckt werden. Allen diesen Fehlern ist gemeinsam, dass sie nach der Identifikation keine weitere Behandlung benötigen, also bereits durch ihre Entdeckung lokalisiert und gekapselt sind.

Fällt nun ein Knoten so aus, dass er seine Vorgänger als fehlerhaft identifiziert, so sendet dieser Knoten zwei Fehlermeldungen, je eine für jede abstrakte Verbindung an seinen Eingängen. Hierdurch werden beide Eingänge des Knotens für alle anderen Knoten im Netz als fehlerhaft gekennzeichnet und der Knoten ist isoliert. Durch eine fehlerhafte Fehlernachricht kann sich ein Knoten also schlimmstenfalls selbst isolieren und somit aus dem Netzwerk nehmen.

Fehlermeldung für byzantinische Fehler

Bisher spielte die Verteiltheit der Information keine Rolle bei der Formulierung der Fehlernachrichten. Anders bei byzantinischen Fehlern. Da Strukturen wie in Abbildung 3.16 zulässig sind, ist es möglich, dass der Knoten, der

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

den Fehler identifiziert, nicht direkt downstream von dem Verursacher liegt. Mehr noch, es ist dem identifizierenden Knoten basierend auf seinem lokalen Wissen im Moment der Identifizierung nicht möglich zu entscheiden, ob sein upstream Knoten der Verursacher des byzantinischen Fehlers ist.

Bevor also der Fehler einer abstrakten Verbindung zugeordnet werden kann, ist es nötig, dass alle Knoten, welche die fehlerhafte Nachricht erhalten haben, eine Fehlermeldung senden. Da aber Knoten, welche die fehlerhafte Nachricht auf beiden Eingängen erhalten haben, der Meinung sind, sie haben die korrekte Nachricht erhalten, muss der ursprüngliche Sender, der als einziger Knoten im Netz sicher den korrekten Inhalt der Nachricht kennt, eine Korrekturnachricht senden.

Da byzantinische Fehler oft durch SOS-Fehlerursachen bedingt sind, ist es empfehlenswert, eine Nachricht mit möglichst hoher Hammingdistanz zu der ursprünglichen Nachricht zu verwenden, um eine zufällige Verfälschung auf die gleiche fehlerhafte Nachricht während der Übertragung zu vermeiden. Es ist zudem zur Fehlerlokalisierung nicht nötig, den gesamten Inhalt der Nachricht erneut zu senden. Es muss lediglich möglich sein, die beiden Versionen der Nachricht voneinander zu unterscheiden. Anhand dieser Korrekturnachricht können nun auch Knoten, die auf beiden Eingängen die fehlerhafte Nachricht empfangen haben, eine Fehlermeldung versenden.

Betrachtet man die Ausbreitung einer fehlerhaften Nachricht im Netz so fällt auf, dass sich immer die von einem Knoten zuerst empfangene Nachricht ausbreitet. Die fehlerhafte Version der Nachricht wird also lediglich versendet, wenn sie vor der korrekten Version empfangen wird. In den Fehlermeldungen zum byzantinischen Fehler muss also die Reihenfolge, in der die Nachrichten auf den Eingängen empfangen werden, vermerkt sein. Ist diese bekannt, so lässt sich von einem beliebigen Knoten, der eine falsche Version der Nachricht empfangen hat, aus der Verursacher des Fehlers finden. Als Verursacher wird der upstream Knoten des letzten Knotens, der bewusst eine fehlerhafte Nachricht versendet hat, identifiziert.

Auch diese Methode zur Fehlerlokalisierung ist wieder robust gegen falsche Fehlernachrichten. Behauptet der Verursacher des byzantinischen Knotens, eine fehlerhafte Nachricht erhalten zu haben, so wird die Verbindung zu seinem upstream Knoten getrennt. Tritt erneut ein byzantinischer Fehler auf und verweist der fehlerhafte Knoten erneut auf seinen upstream Knoten, so ist er isoliert und der Fehler damit behoben.

Masquerading Fehler

Wie schon zuvor erwähnt, ist die Aufdeckung des Masquerading Fehlers in *SafeNet* vergleichsweise schwer zu bewerkstelligen, da sich die fehlerhafte Nach-

richt immer durch das gesamte Netzwerk ausbreitet. Eine Lokalisierung ist direkt durch den Fehler mit den bekannten Algorithmen nicht möglich, weshalb hier ein anderer Ansatz verwendet werden soll. Es wird statt dem Fehler, also einem Zustand innerhalb des Systems, nach der Fehlerursache gesucht. Hierfür ist jedoch der innere Aufbau eines SafeNet Knotens wichtig, der im folgenden Unterabschnitt vorgestellt werden soll.

3.9 Aufbau eines Knotens

Bereits aus Abbildung 3.2 wird die Grundstruktur eines SafeNet Knotens deutlich. Die zwei Eingänge müssen in einer internen Instanz so serialisiert werden, dass trotz der halben Bandbreite am Ausgang keine Nachrichten verloren gehen. Hierzu werden drei Warteschlangen eingeführt, je eine zwischen jedem der Eingänge und besagter Instanz und eine vor dem Ausgang. Die koordinierende Instanz wird als *Operation Control* (OC) bezeichnet.

Auf die topologieabhängige Länge der Sendewarteschlange wurde in Abschnitt 3.5 bereits ausführlich eingegangen. Die Länge der Empfangswarteschlangen hingegen ist lediglich von der Verarbeitungsgeschwindigkeit der OC relativ zur minimalen Nachrichtenlänge abhängig. Kann gewährleistet werden, dass die Bearbeitung zweier gleichzeitig auf den Eingängen eingegangenen Nachrichten abgeschlossen ist, bevor erneut eine Nachricht vollständig empfangen wird, so kann die Länge der Empfangswarteschlangen auf eins gesetzt werden.

Zwischen die eigentlichen, physikalischen Eingänge des Knotens und die Empfangswarteschlangen wird je Eingang noch ein Prozess (*DECODE_A*, *DECODE_B*) eingefügt, der die am Eingang ankommenden Bit zu einer Nachricht zusammenfügt. Dies erlaubt bei der Implementierung eine unterschiedliche Taktrate für die einfachen Aufgaben der DECODE Prozesse und der deutlich komplexeren OC.

Analog zu den DECODE Prozessen an den Eingängen wird ein ENCODE Prozess zwischen OC und den physikalischen Ausgang eingefügt. Dessen Aufgabe ist es, Nachrichten aus der Sendewarteschlange auf das physikalische Medium zu versenden.

Um eine Kommunikation mit dem Host zu ermöglichen wird ein unabhängiger Teil eingeführt, das *Controller Host Interface* (CHI). Über das CHI werden Nachrichten, die auf den Eingängen des Knotens ankommen, an den Host weitergeleitet. Durch Zugriff auf die Warteschlange, beziehungsweise durch Kommunikation mit der OC kann das CHI auch das Versenden von Nachrichten veranlassen.

Der Aufbau eines SafeNet Knotens ist in Abbildung 3.18 dargestellt.

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

Zusätzlich zu den bereits beschriebenen Teilen sind auch die Blöcke \underline{A} , in dem Wissen über die Topologie des Netzwerkes, repräsentiert durch die Adjazenzmatrix \underline{A} des Netzwerkes, sowie Wissen über MIDs und MSIs gespeichert ist und der Block RAM in dem Wissen über bereits gesendete und empfangene Nachrichten enthalten ist, eingezeichnet.

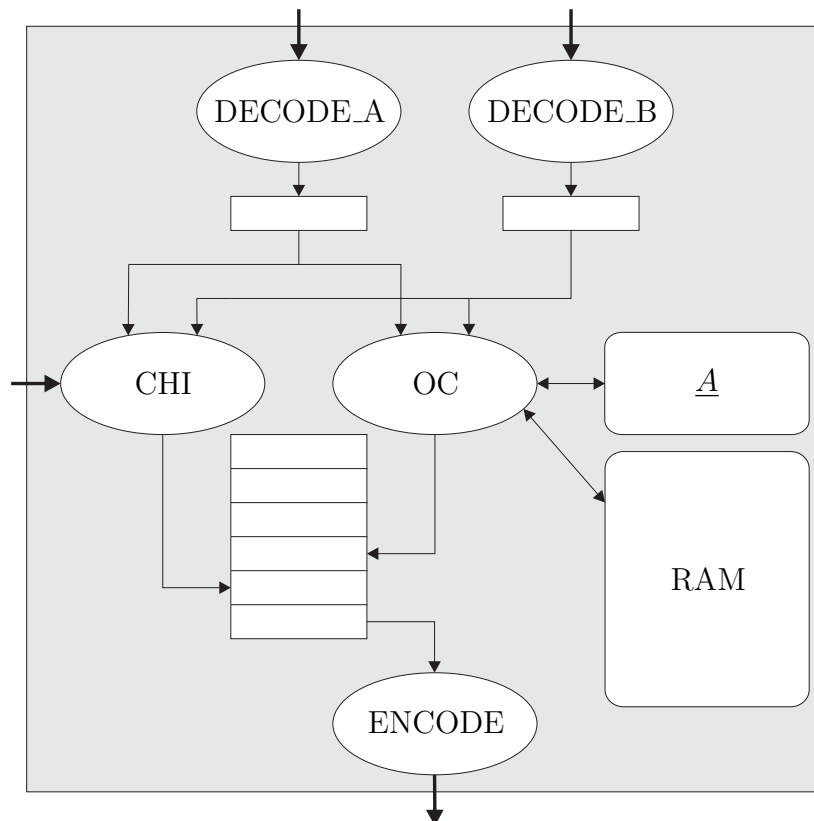


Abbildung 3.18: Interner Aufbau eines SafeNet Knotens

Mit dem Wissen über den Aufbau eines Knotens ist es nun möglich, die Fehlerursachen des Masquerading Fehlers zu analysieren. Es müssen hierbei zwei Fehlerursachen unterschieden werden, die sich beide in einem Masquerading Fehler äußern.

bewusster Masquerading Fehler Eine mögliche Fehlerursache für einen Masquerading Fehler ist, dass die betroffene MID fälschlicherweise in der Liste der von diesem Knoten zu versendenden MIDs steht. Es handelt sich hierbei um eine passive Fehlerursache, die erst durch einen Fehler im Host, nämlich den Versuch, eine Nachricht mit eben dieser MID zu versenden, aktiviert wird. Der Knoten versendet die fehlerhafte Nachricht also wie eine eigene Nachricht und somit „bewusst“ und stellt sie auch in die Liste der von ihm versendeten Nachrichten.

Entdeckt der Knoten, dem die MID eigentlich zugeordnet ist nun die fehlerhafte Nachricht, so sendet er eine Fehlermeldung. Empfängt ein Knoten eine Masquerading Fehlernachricht über eine seiner eigenen MIDs, so sendet er zusätzlich zu dieser Nachricht auch eine eigene Fehlernachricht. Da nun beide Knoten Anspruch auf die betroffene MID erheben, ist es den restlichen Knoten und speziell den downstream Knoten möglich, den Knoten, der für den Masquerading Fehler verantwortlich ist, zu identifizieren und zu isolieren.

unbewusster Masquerading Fehler Liegt die Fehlerursache in dem Prozess der Nachrichtengenerierung, so steht die Nachricht, mit der der Masquerading Fehler ausgelöst wurde, nicht in der Liste der bereits versendeten Nachrichten. Erhält der fehlerhafte Knoten die Nachricht von seinen upstream Knoten, so versendet er sie erneut an seine downstream Knoten, die diesen Fehler bemerken und auch lokalisieren können, da dieses Verhalten Satz 3.2 verletzt.

Durch die Unterscheidung der Fehlerursachen ist es nun möglich, einen Masquerading Fehler zu lokalisieren. An dieser Stelle sei angemerkt, dass es zu einer Überschneidung zwischen Masquerading Fehler und byzantinischen Fehler kommen kann. Versendet der Knoten, dem die MID zugeordnet ist, eine Nachricht und erhält er die Nachricht, die aus dem Masquerading Fehler hervorgegangen ist, vor seiner eigenen, so wird er dies als byzantinischen Fehler deuten und eine Fehlerbehandlung anstoßen. Der unbewusste Masquerading Fehler wird wie oben beschrieben aufgedeckt. Um den bewussten Masquerading Fehler aufdecken zu können ist es jedoch notwendig, dass ein Knoten auf eine Korrekturnachricht für einen byzantinischen Fehler, die sich auf eine der ihm zugeordneten MIDs bezieht, mit einer Masquerading Fehlernachricht antwortet um die Lokalisierung des Fehlers zu ermöglichen.

Mit der Möglichkeit, auch den Masquerading Fehler zu lokalisieren, ist der erste Teil der Fehlerbehandlung, nämlich die Entdeckung und Lokalisierung von Fehlern abgeschlossen. Es stellt sich nun aber die Frage, wie mit Knoten, die als fehlerhaft identifiziert wurden, umgegangen werden soll. Es wurde bereits bei der Diskussion über die verschiedenen Fehlerarten mehrfach angedeutet, dass ein nicht zu vernachlässigender Teil von Fehlern von permanenten Fehlerursachen ausgelöst wird, die im laufenden Betrieb nicht zu beheben sind. Es ist also zu erwarten, dass aufgrund dieser nicht behobenen Fehlerursachen die Auftrittswahrscheinlichkeit von Fehlern gesteigert wird.

Die Entscheidung, welche Fehler als permanent und welche als temporär zu beurteilen sind, ist stark von der Anwendung abhängig. Ist beispielsweise

3 Netzwerkbasiertes Kommunikationsmedium (*SafeNet*)

eine Überführung von Geräten in einen bekannt sicheren Zustand nach dem Auftreten eines Babbling Idiot Fehlers durch ein Reset spezifiziert, so ist der Babbling Idiot Fehler als temporär einzustufen. Ist dies nicht der Fall, so ist eine dauerhafte Trennung des betroffenen Knotens nötig. Aus diesem Grund wird im Folgenden der Ansatz gewählt, dass nach der Lokalisierung eines Fehlers der betroffene Knoten von dem Netzwerk getrennt wird, indem seine beiden downstream Knoten ihre entsprechenden Eingänge abschalten. Eine Reintegration von zuvor abgeschalteten Knoten ist momentan nicht vorgesehen, kann aber zum Beispiel mit der Erweiterung des Start-up Vorgangs um die Möglichkeit eines dynamischen Start-ups eingeführt werden.

Im folgenden Kapitel wird nun basierend auf den bisher angestellten Überlegungen eine Spezifikation vorgestellt, die als Grundlage für die Modellierung und Implementierung von *SafeNet* dient.

4 Spezifikation

4.1 Topologie

SafeNet besteht aus Knoten, die untereinander durch unidirektionale Verbindungen kommunizieren. Jeder Knoten besitzt dabei zwei Ein- sowie zwei Ausgänge. Die Ausgänge sind dabei so verbunden, dass auf beiden das gleiche Signal anliegt.

Ein Knoten $M \in \mathcal{N}$ wird so mit anderen Knoten $A, B \in \mathcal{N}$ verbunden, dass für die Mengen seiner direkten Vorgänger M^- und die Menge seiner direkten Nachfolger M^+ folgende Bedingungen erfüllt sind:

1. $A \in M^- \wedge B \in M^-, \Rightarrow A \neq B$
2. $A \in M^+ \wedge B \in M^+, \Rightarrow A \neq B$
3. $M \notin M^+ \wedge M \notin M^-$
4. das Netz muss auch ohne einen beliebigen Knoten zusammenhängend sein

Jedem Knoten der Menge \mathcal{N} ist eineindeutig eine UID (Unique Identifier, 8 Bit) zugeordnet. Jedem Nachrichtentyp ist eineindeutig eine MID (Message Identifier, 10 Bit). Jede MID ist eindeutig einem Knoten, also einer UID, zugeordnet und darf nur von diesem generiert werden, wobei jedem Knoten mehrere MIDs zugeordnet sein dürfen. Jeder Knoten darf Nachrichten mit MIDs, die anderen UIDs zugeordnet sind, dem Sendalgorithmus folgend weitersenden.

4.2 Nachrichtenformat

Eine Nachricht in SafeNet hat die in Abbildung 4.1 dargestellte Struktur. Die in Abbildung 4.1 verwendeten Bezeichnungen sind in Tabelle 4.1 erklärt.

Die Prüfsumme HCRC(μ) wird dabei mit Hilfe des Generatorpolynoms $g_{HCRC}(x)$ aus Gleichung 4.1 berechnet.

$$g_{HCRC}(x) = x^{11} + x^9 + x^8 + x^7 + x^2 + 1 \quad (4.1)$$

4 Spezifikation

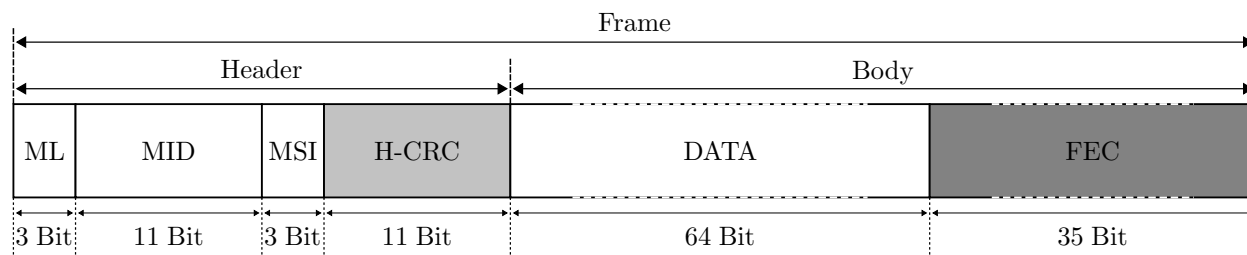


Abbildung 4.1: Grundstruktur einer Nachricht in SafeNet

Tabelle 4.1: In einer SafeNet Nachricht enthaltene Datenfelder

Bezeichnung		Länge	Erklärung	Formal
ML	Message Length	3 Bit	Länge des Application Data Felds	$ML(\mu)$
MID	Message Identifier	11 Bit	Eindeutige Identifikationsnummer eines Nachrichtentyps. Davon: 1 Bit Error Frame Flag (nicht gesetzt) 10 Bit eigentlicher Identifier	MID
MSI	Message Sequence Identifier	3 Bit	Reihenfolge der Nachrichten eines Typs	$MSI(\mu)$
H-CRC	Header CRC	11 Bit	Prüfsumme über ML, MID und MSI	$HCRC(\mu)$
DATA		64 Bit	Nutzdaten der Nachricht (feste Länge)	$DATA(\mu)$
FEC		35 Bit	Kanalcodierungsdaten	$FEC(\mu)$

Das Generatorpolynom $g_{FEC}(x)$ für die Fehlerkorrektur lautet:

$$g_{FEC}(x) = x^{35} + x^{34} + x^{31} + x^{29} + x^{26} + x^{25} + x^{24} + x^{22} + x^{21} + x^{13} + x^{10} + x^7 + x^6 + x^4 + x^2 + x + 1 \quad (4.2)$$

Ein Frame beginnt immer mit einem HIGH Bit und endet mit drei LOW Bit, die in Abbildung 4.1 nicht abgebildet sind.

Um dem downstream Knoten eine Synchronisation zu ermöglichen, wird nach acht Bit ein SYNC-Bit eingeschoben, so dass es zu einem Flankenwechsel kommt. Hierzu besitzt das SYNC-Bit immer den gegenteiligen Wert des vorhergegangenen Bits. Dieses Verfahren wird über die gesamte Nachricht angewandt. Diese Bit sind in den Abbildungen 4.1 bis 4.5 nicht eingezeichnet. Die Länge der gesamten Nachricht beträgt also 146 Bit.

Für die Behandlung von Fehlern ist das in Abbildung 4.2 dargestellte Format vorgesehen. Die einzelnen Felder der Nachricht sind in Tabelle 4.2 näher erläutert. Weitere Typen von Fehlermeldungen, die für spezielle Fehlertypen

benötigt werden, werden bei der Betrachtung dieser Fehler in den folgenden Unterabschnitten vorgestellt. Sie entsprechen aber von ihrem Grundformat immer der in Abbildung 4.1 gezeigten Nachricht. Es wird lediglich das Error Frame Flag gesetzt und der Inhalt der Nachricht den Bedürfnissen entsprechend angepasst.

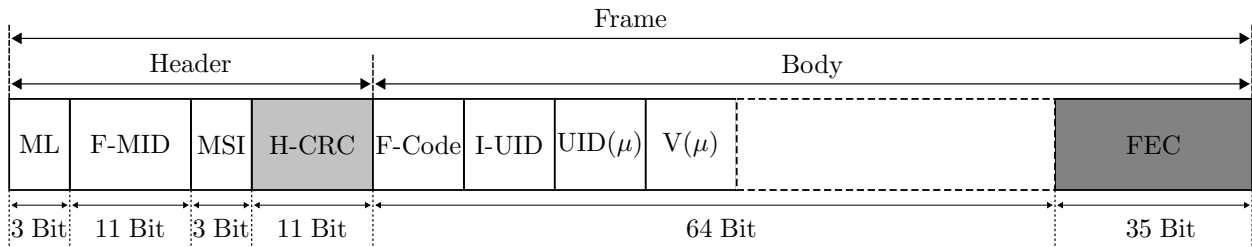


Abbildung 4.2: Nachrichtenformat einer Fehler-Nachricht

Tabelle 4.2: In einer SafeNet Fehlernachricht enthaltene Datenfelder

Bezeichnung	Länge	Erklärung	Formal
ML	3 Bit	Länge des Application Data Felds	$ML(\mu)$
MID	11 Bit	Eindeutige Identifikationsnummer eines Nachrichtentyps. Davon:	MID
	1 Bit	Error Frame Flag (gesetzt)	
	10 Bit	Identifizier der fehlerhaften Nachricht	
MSI	3 Bit	MSI der fehlerhaften Nachricht	$MSI(\mu)$
H-CRC	11 Bit	Prüfsumme über ML, MID und MSI	$HCRC(\mu)$
F-Code	8 Bit	Fehlercode nach Tabelle 4.6	
I-UID	8 Bit	UID des identifizierenden Knotens	
UID(μ)	8 Bit	UID des Knotens, der den Fehler verursacht hat, falls bekannt, sonst 0.	
V(μ)	8 Bit	Hash-Code der Nachrichten	
FEC	35 Bit	Kanalcodierungsdaten	$FEC(\mu)$

4.3 Fehlerbehandlung

SafeNet muss auch nach dem Auftreten von Fehlern innerhalb des Netzwerkes korrekt funktionieren. In diesem Unterabschnitt sind daher alle Fehler auf-

4 Spezifikation

geführt, die bei der Spezifikation berücksichtigt wurden. Neben einer Definition werden auch die lokale Behandlung des Fehlers in den einzelnen Knoten sowie das globale Verhalten des Netzwerkes auf das Auftreten eines Fehlers aufgezeigt. Es wird zudem die worst-case Zeit zwischen Auftreten und Abschluss der Behandlung eines Fehlers gegeben.

Unabhängig von der Art des Fehlers wird davon ausgegangen, dass innerhalb der Zeit, die das Netzwerk für die Bearbeitung des Fehlers benötigt, kein weiterer, unabhängiger Fehler auftritt (Ein-Fehler-Hypothese). Insbesondere kann also davon ausgegangen werden, dass die Fehlerentdeckungs- und -lokalisierungsalgorithmen immer fehlerfrei abgearbeitet werden.

Durch den in Gleichung 4.1 definierten BCH-Code wird der Header der Nachricht stark abgesichert. Es können in den 28 Bit des Headers bis zu zehn fehlerhafte Bit erkannt werden. Es kann daher davon ausgegangen werden, dass ein Header entweder richtig übertragen wird oder so verfälscht wird, dass dieser Fehler entdeckt werden kann. Bei der Übertragung entstehen also keine Nachrichten mit syntaktisch korrektem aber kontextuell falschem Header.

4.3.1 Syntaktisch inkorrekte Nachricht

Definition 4.1 (Syntaktisch inkorrekte Nachricht) *Eine Nachricht, deren Format nicht den in Abschnitt 4.2 vorgestellten Formaten entspricht, deren HCRC nicht ihrem Header entspricht oder bei der der FEC Algorithmus nicht terminiert oder ein Element der Rückweisungsklasse liefert, wird als syntaktisch inkorrekt bezeichnet.*

Eine syntaktisch inkorrekte Nachricht wird im downstream Knoten des Verursachers entdeckt. Diese Nachricht wird nicht weiterversendet, wodurch der Fehler gekapselt wird. Der Eingang, auf dem die syntaktisch inkorrekte Nachricht empfangen wurde, wird abgeschaltet. Statt der Nachricht wird eine Fehlermeldung versendet, die alle anderen Knoten über den Empfang einer syntaktisch inkorrekten Nachricht informiert.

Der empfangende Knoten stellt schon beim Decodieren der Nachricht fest, dass diese syntaktisch inkorrekt ist. Der Fehler ist daher nach maximal τ_s aufgedeckt und behoben. Die maximale Zeit, nach der alle Knoten über das Abschalten der Verbindung informiert sind, ist die maximale Laufzeit einer Nachricht im Netzwerk. Ab dem Auftreten des Fehlers vergehen also maximal $\tau_s + \tau_{max}$, bis in allen Knoten eine konsistente Sicht des Fehlers vorliegt.

Für die Funktionalität des Netzwerkes hat das Abschalten einer Verbindung keine Folgen. Alle Knoten erhalten weiterhin jede Nachricht mindestens ein Mal. Die Fehleraufdeckungsfähigkeit des Netzwerkes ist aber einge-

schränkt, da byzantinische Fehler in einem Knoten nicht mehr diagnostiziert werden können.

Versendet ein Knoten syntaktisch inkorrekte Nachrichten an beide downstream Knoten, so wird der Fehler in zwei Knoten diagnostiziert. Der verursachende Knoten wird also vollständig isoliert. In diesem Fall muss auf einer höheren Schicht sichergestellt werden, dass die Funktionalität des isolierten Knotens anderweitig zur Verfügung gestellt wird.

4.3.2 Ausfall einer Verbindung

Definition 4.2 (Ausfall einer Verbindung) *Eine Verbindung ist ausgefallen, wenn nach dem erstmaligen Empfang einer Nachricht auf dem anderen Kanal des selben Knotens innerhalb von τ_{max} keine Nachricht ankommt.*

Der Ausfall einer Verbindung ist ein passiver Ausfall, weshalb keine falsche Nachricht entstanden ist. Ist der Ausfall einer Verbindung diagnostiziert, so informiert der empfangende Knoten den Rest des Netzwerkes mit einer Fehlermeldung, wie in Abbildung 4.2 dargestellt.

Mit den in Abschnitt 2.1 gegebenen Definitionen ist der Ausfall einer Verbindung streng genommen aus der Perspektive von SafeNet betrachtet eine Fehlerursache, die sich für den Rest des Systems durch den Fehler „ausbleibende Nachrichten“ äußert. Zudem handelt es sich um eine passive Fehlerursache, die erst durch das Versenden von Nachrichten aktiviert wird. Es kann daher keine maximale Zeit zwischen Ausfall einer Verbindung und der Diagnose angegeben werden, zumal der Zustand, dass keine Nachricht in dem Netzwerk versendet wird, zulässig ist.

Die Zeit zwischen Aktivierung der Fehlerursache, nämlich dem Empfang einer Nachricht auf dem anderen Eingang und der Diagnose ist aber mit τ_{max} bekannt. Spätestens nach einer weiteren Zeitspanne τ_{max} sind alle Knoten in dem Netzwerk über den Ausfall der Verbindung informiert.

Global schränkt der Ausfall einer Verbindung lediglich die Fehleraufdeckungsfähigkeit des Netzwerkes ein, während die Funktionalität erhalten bleibt.

4.3.3 Passiver Ausfall eines Knotens

Definition 4.3 (Passiver Ausfall eines Knotens) *Ein Knoten ist passiv ausgefallen, wenn beide downstream Knoten ihre Verbindung zu diesem Knoten als ausgefallen melden.*

Die Diagnose eines passiv ausgefallenen Knotens verläuft in den downstream Knoten analog zu der Diagnose einer ausgefallenen Verbindung. Durch

4 Spezifikation

die Fehlermeldungen der beiden downstream Knoten wird es aber möglich, den Ausfall des betroffenen Knotens in jedem anderen Knoten lokal zu identifizieren.

Wie schon bei der syntaktisch inkorrekten Nachricht angemerkt, ist es bei einem passiven Ausfall eines Knotens Aufgabe der höheren Schichten, für die Bereitstellung der Funktionalität des ausgefallenen Knotens zu sorgen.

4.3.4 Babbling Idiot

Definition 4.4 (Babbling Idiot) *Ein Babbling Idiot Fehler tritt auf, wenn ein Knoten eine eigene Nachricht versendet, ohne zuvor seine zuletzt gesendete eigene Nachricht auf mindestens einem Kanal wieder empfangen zu haben.*

Aufgrund der Tatsache, dass es sich bei SafeNet um ein verteiltes System handelt, ist es dem downstream Knoten nicht möglich zu überprüfen, ob der betroffene Knoten seine eigene Nachricht tatsächlich schon wieder erhalten hat. Daher wird der kürzeste Weg durch das Netz als untere Schranke angenommen. Versendet ein Knoten Nachrichten schneller, versenden seine beiden downstream Knoten die Nachricht nicht weiter, melden stattdessen einen Babbling Idiot Fehler und schalten ihre betroffenen Eingänge ab.

Da der Babbling Idiot direkt in den downstream Knoten erkannt werden kann, beträgt seine Fehleraufdeckungszeit τ_s . Nach dem Versenden einer Fehlermeldung, also τ_{max} nach der Aufdeckung, sind alle Knoten in dem Netzwerk über den Ausfall eines Knotens informiert.

Da auch ein Babbling Idiot von dem Netzwerk getrennt wird, ist es auch hier Aufgabe der höheren Schichten, die Funktionalität des Knotens bereitzustellen.

4.3.5 Byzantinischer Fehler

Definition 4.5 (Byzantinischer Fehler) *Empfängt ein SafeNet Knoten zwei syntaktisch korrekte aber unterschiedliche Versionen einer Nachricht, so ist ein byzantinischer Fehler aufgetreten.*

Der byzantinische Fehler wird hier so definiert, dass eine Abweichung zwischen den zwei Beobachtungen eines Ereignisses, nämlich des Sendens einer Nachricht, existiert, wobei von dem Ursprung der Nachricht, also den upstream Knoten, abstrahiert wird.

Der byzantinische Fehler unterscheidet sich von den bisher behandelten insofern, als seine Behandlung und insbesondere seine Lokalisierung nicht

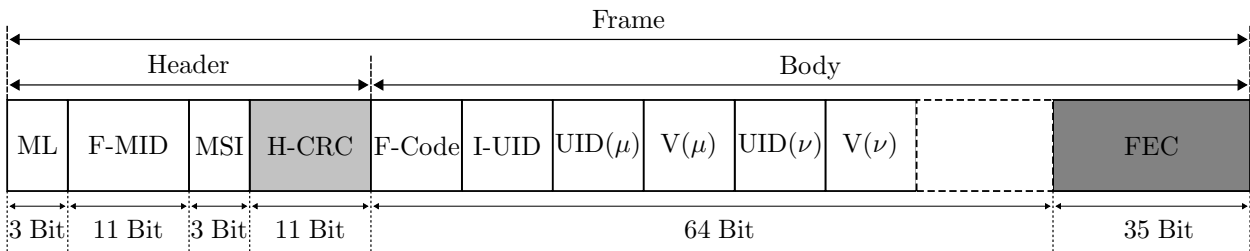


Abbildung 4.3: Fehlernachricht, die zwei abweichende Versionen einer Nachricht in einem Knoten signalisiert (byzantinischer Fehler)

lokal stattfinden kann, sondern erst durch eine globale Analyse möglich wird. Hierzu sendet jeder Knoten eine Fehlermeldung nach Abbildung 4.3.

Empfängt der ursprüngliche Absender der Nachricht eine solche Fehlermeldung, versendet er eine Nachricht, die den Hash-Code der korrekten Nachricht enthält, wie in Abbildung 4.4 dargestellt.

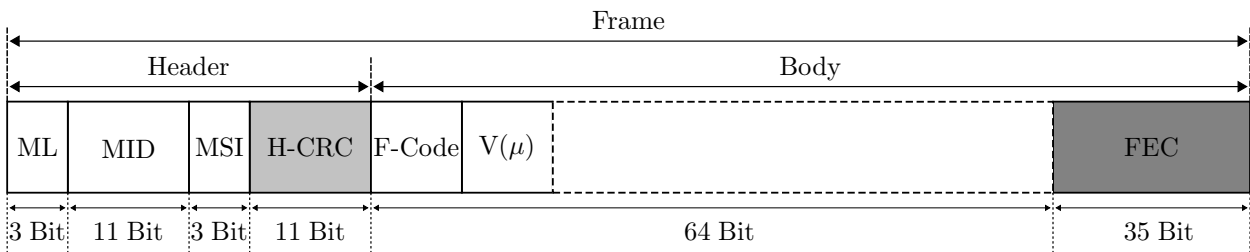


Abbildung 4.4: Korrekturnachricht für byzantinische Fehler

Somit können alle Knoten, die die verfälschte Version der Nachricht auf beiden Eingängen empfangen haben, ebenfalls eine Fehlermeldung, wie in Abbildung 4.5 gezeigt, senden.

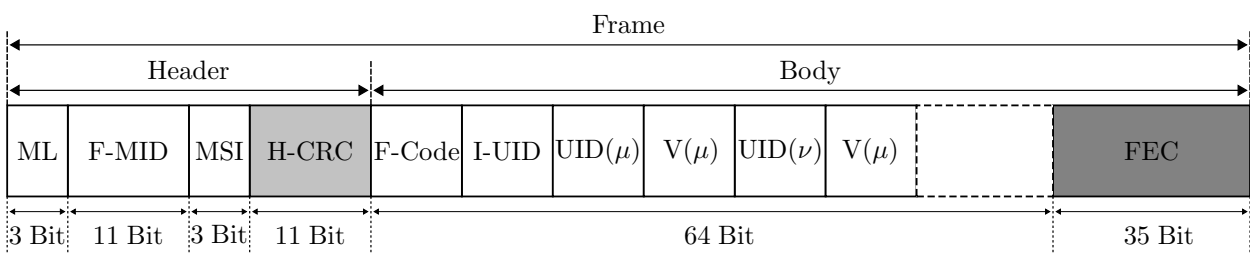


Abbildung 4.5: Fehler-Nachricht, die durch fehlerhafte Nachrichten auf beiden Eingängen hervorgerufen wird

Mit diesen Fehlermeldungen ist die Lokalisierung des Fehlers möglich.

Eine Abweichung kann erst mit dem Empfang der zweiten Nachricht, also spätestens τ_{max} nach dem Eintreffen der Ersten diagnostiziert werden. Ist die zweite Nachricht verfälscht, so wird der Fehler sofort aufgedeckt. Spätestens

4 Spezifikation

Tabelle 4.3: In einer SafeNet Fehlernachricht für byzantinische Fehler enthaltene Datenfelder

Bezeichnung		Länge	Erklärung	Formal
ML	Message Length	3 Bit	Länge des Application Data Felds	$ML(\mu)$
MID	Message Identifier	11 Bit	Eindeutige Identifikationsnummer eines Nachrichtentyps. Davon:	MID
		1 Bit	Error Frame Flag (gesetzt)	
		10 Bit	Identifier der fehlerhaften Nachricht	
MSI	Message Sequence Identifier	3 Bit	MSI der fehlerhaften Nachricht	$MSI(\mu)$
H-CRC	Header CRC	11 Bit	Prüfsumme über ML, MID und MSI	$HCRC(\mu)$
F-Code	Fehlercode	8 Bit	Fehlercode nach Tabelle 4.6, hier 8	
I-UID	Identifying UID	8 Bit	UID des identifizierenden Knotens	
$UID(\mu)$	UID	8 Bit	UID des Knotens, von dem die Nachricht zuerst empfangen wurde	
$V(\mu)$	Hash-Code	8 Bit	Hash-Code der zuerst empfangenen Nachricht	
$UID(\nu)$	UID	8 Bit	UID des Knotens, von dem die Nachricht zuletzt empfangen wurde	
$V(\nu)$	Hash-Code	8 Bit	Hash-Code der zuletzt empfangenen Nachricht	
FEC		35 Bit	Kanalcodierungsdaten	$FEC(\mu)$

τ_{max} nach der Entdeckung erreicht die Fehlermeldung den ursprünglichen Knoten, dessen Korrekturnachricht nach weiteren τ_{max} alle Knoten erreicht hat. Wiederum nach τ_{max} liegen alle Fehlermeldungen in allen Knoten vor und die Lokalisierung des Fehlers kann beginnen. Nach spätestens $4 \cdot \tau_{max}$ ist der byzantinische Fehler also lokalisiert und kann behandelt werden.

Durch geeignete Rückweisungsklassen in dem Fehlerkorrekturalgorithmus kann verhindert werden, dass ein Knoten dauerhaft byzantinische Fehler erzeugt. Es kann daher davon ausgegangen werden, dass, sollte ein Knoten nach einem byzantinischen Fehler und vor Ende der Diagnose erneut eine fehlerhafte Nachricht versenden, diese Nachricht syntaktisch inkorrekt ist und von den downstream Knoten direkt entdeckt wird. Der Fall, dass zu einem Zeitpunkt mehrere kontextuell inkorrekte Nachrichten in einem Netzwerk im Umlauf sind, wird daher nicht berücksichtigt.

Tabelle 4.4: In einer SafeNet Korrekturnachricht für byzantinische Fehler enthaltene Datenfelder

Bezeichnung		Länge	Erklärung	Formal
ML	Message Length	3 Bit	Länge des Application Data Felds	$ML(\mu)$
MID	Message Identifier	11 Bit	Eindeutige Identifikationsnummer eines Nachrichtentyps. Davon: 1 Bit Error Frame Flag (gesetzt)	MID
MSI	Message Sequence Identifier	10 Bit	Identifizier der fehlerhaften Nachricht	$MSI(\mu)$
H-CRC	Header CRC	3 Bit	MSI der fehlerhaften Nachricht	$MSI(\mu)$
F-Code	Fehlercode	11 Bit	Prüfsumme über ML, MID und MSI	$HCRC(\mu)$
$V(\mu)$	Hash-Code	8 Bit	Fehlercode nach Tabelle 4.6, hier 8	
FEC	Hash-Code	8 Bit	Hash-Code der korrekten Nachricht	
		35 Bit	Kanalcodierungsdaten	$FEC(\mu)$

4.3.6 Versenden einer kontextuell falschen Nachricht

Definition 4.6 (Versenden einer kontextuell falschen Nachricht) *Diagnostizieren beide downstream Knoten eines Knotens byzantinische Fehler auf ihren entsprechenden Eingängen, so ist der betrachtete Knoten der Versender einer kontextuell falschen Nachricht.*

Analog zu den bisher betrachteten Fehlerfällen ist auch ein byzantinischer Fehler lokal zunächst nur für einen Ausgang eines upstream Knotens diagnostizierbar. Entdecken beide downstream Knoten eines Knotens byzantinische Fehler, so wurde die Nachricht zwar syntaktisch korrekt, aber verfälscht an beide downstream Knoten versendet. In diesem Fall wird der Knoten vom restlichen Netzwerk isoliert.

4.3.7 Masquerading Fehler

Definition 4.7 (Masquerading Fehler) *Generiert ein Knoten eine Nachricht mit einer MID, die einem anderen Knoten im Netzwerk zugeordnet ist, liegt ein Masquerading Fehler vor.*

Da für die korrekte Funktion des Sendalgorithmus' alle Knoten Nachrichten mit allen im Netz vorhandenen MIDs weiterversenden müssen, kann der Masquerading Fehler nur von dem Knoten sicher aufgedeckt werden, dem die MID zugeordnet ist. Dieser versendet eine Fehlernachricht wie in

4 Spezifikation

Tabelle 4.5: In einer SafeNet Fehlernachricht nach Abbildung 4.5 enthaltene Datenfelder

Bezeichnung		Länge	Erklärung	Formal
ML	Message Length	3 Bit	Länge des Application Data Felds	$ML(\mu)$
MID	Message Identifier	11 Bit	Eindeutige Identifikationsnummer eines Nachrichtentyps. Davon:	MID
		1 Bit	Error Frame Flag (gesetzt)	
		10 Bit	Identifier der fehlerhaften Nachricht	
MSI	Message Sequence Identifier	3 Bit	MSI der fehlerhaften Nachricht	$MSI(\mu)$
H-CRC	Header CRC	11 Bit	Prüfsumme über ML, MID und MSI	$HCRC(\mu)$
F-Code	Fehlercode	8 Bit	Fehlercode nach Tabelle 4.6, hier 12	
I-UID	Identifying UID	8 Bit	UID des identifizierenden Knotens	
$UID(\mu)$	UID	8 Bit	UID des Knotens, von dem die Nachricht zuerst empfangen wurde	
$V(\mu)$	Hash-Code	8 Bit	Hash-Code der Nachrichten	
$UID(\nu)$	UID	8 Bit	UID des Knotens, von dem die Nachricht zuletzt empfangen wurde	
$V(\mu)$	Hash-Code	8 Bit	Hash-Code der Nachrichten	
FEC		35 Bit	Kanalcodierungsdaten	$FEC(\mu)$

Abbildung 4.2 dargestellt. Die weitere Fehlerbehandlung erfolgt wie in Abschnitt 3.9 dargestellt mit der Unterscheidung zwischen bewusstem und unbewusstem Masquerading Fehler.

Für den bewussten Masquerading Fehler ist es durch dieses Vorgehen möglich, den Masquerading Fehler innerhalb von τ_{max} nach seinem Auftreten zu entdecken. Innerhalb von weiteren τ_{max} erreicht die erste Fehlermeldung, nämlich die des Knotens, dem die MID eigentlich zugeordnet ist, alle Knoten, insbesondere den Verursacher. Dieser antwortet mit einer Fehlernachricht, die nach weiteren τ_{max} alle Knoten erreicht hat. Nach insgesamt $3 \cdot \tau_{max}$ sind also alle Knoten über den Verursacher des Masquerading Fehlers informiert.

Im Falle eines unbewussten Masquerading Fehlers erreicht die fehlerhafte Nachricht den Verursacher nach spätestens τ_{max} wieder. Da dieser sie wieder versendet, entdecken die downstream Knoten nach insgesamt $\tau_{max} + \tau_s$ den Fehler und senden eine Fehlernachricht, die nach insgesamt $2 \cdot \tau_{max} + \tau_s$ bei allen Knoten im Netz angekommen ist.

4.3.8 Tabelle der Fehlercodes

Um die Behandlung der Fehler zu vereinfachen, werden die in Tabelle 4.6 definierten Fehlercodes verwendet.

Tabelle 4.6: In SafeNet verwendete Fehlercodes

Fehlercode	Beschreibung	Fehlernachrichtentyp (Abbildung)
0	Kein Fehler	
1	FEC-Algorithmus terminiert nicht	4.2
2	FEC-Algorithmus liefert Element der Rückweisungsklasse	4.2
3	H-CRC-Fehler (Prüfsumme fehlerhaft)	4.2
4	Inhalt des ML-Feldes logisch fehlerhaft	4.2
5	Inhalt des MID-Feldes logisch fehlerhaft	4.2
6	MSI-Feld logisch fehlerhaft	4.2
8	Byzantinischer Fehler	4.3, 4.4, 4.5
9	Inhalte sowohl des ML- als auch des MID-Feldes sind fehlerhaft (Fehler 4 und Fehler 5 treten gleichzeitig auf)	4.2
10	Inhalte sowohl des ML- als auch des MSI-Feldes sind fehlerhaft (Fehler 4 und Fehler 6 treten gleichzeitig auf)	4.2
11	Inhalte sowohl des MID- als auch des MSI-Feldes sind fehlerhaft (Fehler 5 und Fehler 6 treten gleichzeitig auf)	4.2
12	Fehlerhafte Nachricht auf beiden Eingängen erhalten	4.5
13	Ausfall einer in dem sendenden Knoten endenden Verbindung	4.2
14	Upstream Knoten des versendenden Knotens weist Babbling Idiot Verhalten auf	4.2
15	Inhalte der Felder ML, MID und MSI sind logisch fehlerhaft (Fehler 4, 5 und 6 treten gleichzeitig auf)	4.2

4.4 Bitübertragungsschicht

Es besteht in SafeNet nicht die Notwendigkeit, eine Bitübertragungsschicht für das gesamte Netzwerk zu verwenden. Es ist durchaus zulässig, für einen Teil der Verbindungen beispielsweise eine optische Übertragung zu wählen, während andere Knoten über Powerline kommunizieren. Wichtig ist nur, dass die Verbindung zwischen zwei Knoten einheitlich ist. Um hierfür einen Anhaltspunkt zu geben, wird im Folgenden die Übertragungsschicht der proof-of-concept Implementierung vorgestellt, siehe auch Abschnitt 5.2.

Um den Einfluss von Störungen zu minimieren, wurde eine differentielle Übertragung auf twisted-pair Leitungen mit einem Wellenwiderstand von

4 Spezifikation

75 Ω gewählt. Die Verwendung von STP-Kabel ist dabei optional. Wichtig ist, dass die Kabel auf der Sende- und Empfangsseite angepasst sind, um Reflektionen zu verhindern.

In Abbildung 4.6 sind die Toleranzen für die differentielle Spannung angegeben. Es ist zu erkennen, dass ein NRZ-Code verwendet wird. Die Spannungsdifferenz muss für mindestens $5/8$ einer Bitzeit in dem für den jeweiligen Wert spezifizierten Bereich liegen, um eine korrekte Erkennung mit dem Bit-strobing Mechanismus zu erlauben. Bei Verwendung einer 100 MBit/s Bitübertragungsschicht beträgt die Länge eines Bits 10 ns.

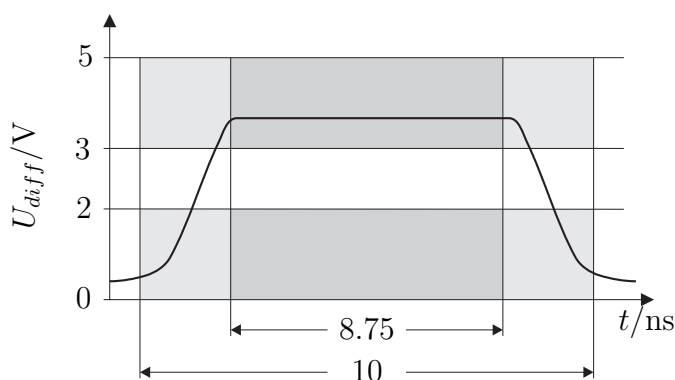


Abbildung 4.6: Bitdiagramm

4.5 Aufbau eines SafeNet-Knotens

Ein SafeNet Knoten besteht auf der obersten Ebene aus fünf unabhängigen Prozessen, von denen in Abbildung 4.7 vier Prozesse dargestellt sind.

Diese Prozesse sind:

- CHI** Das Channel-Host Interface stellt die Verbindung zwischen Host und dem SafeNet dar. Es wird in Abschnitt 4.6 erklärt.
- OC** Die Operation Control ist für die Ausführung der verteilten Fehlerfindungsalgorithmen verantwortlich, auf die in Abschnitt 4.9 eingegangen wird.
- ENCODE** Der ENCODE Prozess ist für das korrekte Versenden der Nachrichten verantwortlich, siehe auch Abschnitt 4.8. Zusammen mit dem DECODE Prozess stellt er die Sicherungsschicht dar.
- DECODE** Im DECODE Prozess werden empfangene Nachrichten für die Weiterverarbeitung in der OC vorbereitet, siehe Abschnitt 4.7. In einem

SafeNet Knoten existiert ein unabhängiger DECODE Prozess für jeden Eingang. Die Signale, mit denen diese Prozesse angesprochen werden müssen eindeutig unterscheidbar sein.

Diese Prozesse kommunizieren untereinander durch Signale, in Abbildung 4.7 durch Beschriftung der Pfeile dargestellt, und gemeinsamen Variablen, die unter den jeweiligen Prozessen aufgeführt sind.

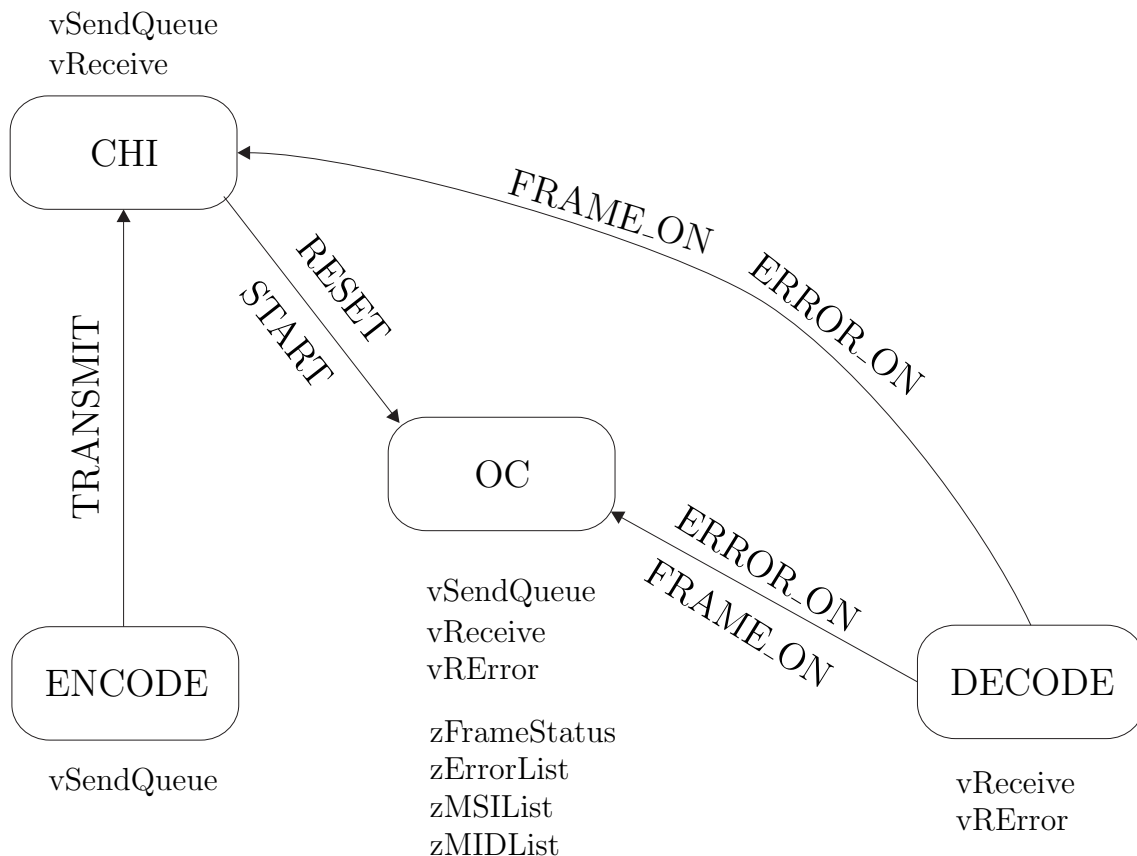


Abbildung 4.7: Aufbau eines SafeNet Knotens aus unabhängigen Prozessen

Die Namen von Variablen, die von mehr als einem Prozess auf dieser Ebene verwendet werden, beginnen mit 'v', die von solchen, die nur von einem Prozess verwendet werden mit 'z'. Die genauen Anforderungen an die Variablen sind im Folgenden zusammengefasst.

4.5.1 Globale Variablen

In vSendQueue werden die zu sendenden Nachrichten gespeichert. Hierzu müssen alle wichtigen Informationen über die Nachricht, also MSI, MID und der Inhalt der Nachricht gespeichert werden. Die Variable vSendQueue dient

4 Spezifikation

als Schnittstelle zwischen CHI und OC, die Nachrichten in die Sendewarteschlange stellen einerseits und dem ENCODE Prozess, der die Nachrichten versendet und wieder aus der Warteschlange entfernt, andererseits. Die maximale Länge von `vSendQueue` ist topologieabhängig.

Für den umgekehrten Weg ist `vReceive` zuständig. Mit Hilfe dieser Variable, die wie ein Feld von `vSendQueue` aufgebaut ist, werden Nachrichten, die empfangen wurden, von DECODE an OC und CHI ausgeliefert. Da OC und CHI in der Lage sein müssen, empfangene Nachrichten schneller zu verarbeiten, als sie eingeht können, ist hier keine Warteschlange nötig.

Die Variable `vRError` dient zur Signalisierung der Art des aufgetretenen Fehlers und muss momentan lediglich ganze Zahlen im Intervall $[0, 15]$ beinhalten können. Da das Fehlerfeld der Nachrichten jedoch auf 8 Bit ausgelegt ist, ist es sinnvoll, diese Bandbreite auch in `vRError` zu ermöglichen.

4.5.2 Lokale Variablen in der Operation Control

Die zentrale lokale Variable in der Operation Control ist `zFrameStatus`. In ihr sind alle relevanten Daten über die bereits empfangenen Nachrichten, also MID, MSI und der Hash-Code des Inhaltes gespeichert. Zusätzlich muss gespeichert werden, auf welchen der Eingänge die Nachricht bereits empfangen wurde. Für die Fehlerlokalisierung des byzantinischen Fehlers ist zudem die Reihenfolge der Eingänge auf denen die Nachricht empfangen wurde, nötig. Wie auch die Länge von `vSendQueue` ist die Länge von `zFrameStatus` topologieabhängig.

Die Lokalisierung von byzantinischen Fehlern erfolgt in einem Knoten mit Hilfe von `zErrorList`. Daher muss aus dieser Liste ersichtlich sein, in welcher Reihenfolge (Eingänge) der Knoten welche Version der Nachricht erhalten hat. Die Länge von `zErrList` ist abhängig von der Anzahl der im Netzwerk vorhandenen Knoten, maximal $2^8 = 256$.

In der Liste `zMIDList` sind die Zuordnungen von MIDs zu UIDs gespeichert. Diese Liste darf nur während der Start-up Phase verändert werden. In der Liste `zMSIList` sind die momentanen Zählerstände, der einzelnen MIDs gespeichert. Sowohl die Länge von `zMIDList` als auch die Länge von `zMSIList` ist abhängig von den im Netzwerk vorhandenen MIDs, maximal jedoch $2^{10} = 1024$.

Zusätzlich zu den hier eingezeichneten Signalen muss es dem CHI und der OC möglich sein, die ENCODE und DECODE Prozesse zu beenden und durch ein Reset in einen bekannten sicheren Zustand zu überführen. Zusätzlich kann es möglich sein, die ENCODE und DECODE Prozesse zur Laufzeit zu starten. Ist dies nicht der Fall, werden diese Prozesse im Start-up gestartet und nach dem Beenden nicht wieder gestartet.

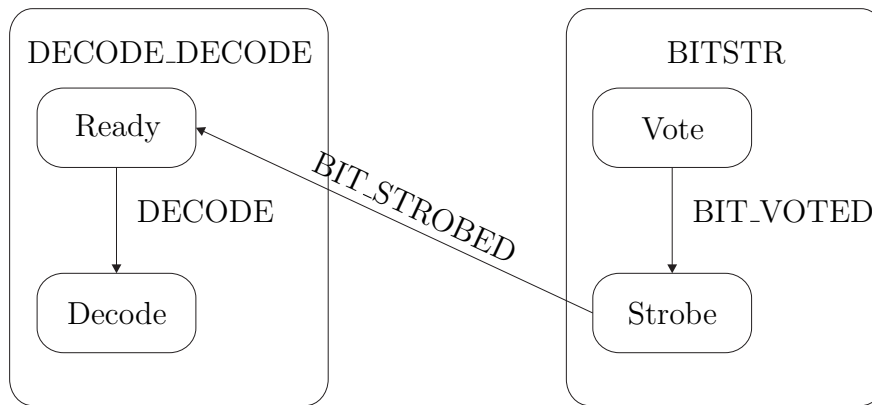


Abbildung 4.8: Aufbau des DECODE Prozesses

Zu Beginn des Betriebs müssen die Informationen über die im Netzwerk vorhandenen UIDs und MIDs in jedem Knoten vorliegen. Diese können entweder fest in jedem Knoten des Netzwerks gespeichert sein oder aber dynamisch zu Beginn in einer Start-up Phase festgelegt werden, siehe Abschnitt 4.10. Nach Ende der Start-up Phase dürfen diese Daten nicht verändert werden.

4.6 Controller-Host Interface (CHI)

Das Controller-Host Interface bildet die Schnittstelle zwischen dem SafeNet-Controller und dem Host, der über SafeNet Daten versenden will. Das Channel-Host Interface überwacht die Einhaltung des Sendalgorithmus und stellt, wenn dies zulässig ist, Nachrichten, die von dem Host verschickt werden sollen, direkt in vSendQueue. Des Weiteren leitet das CHI alle empfangenen Nachrichten anderer Knoten an den Host weiter.

Der genaue Aufbau des CHI ist nicht spezifiziert.

4.7 DECODE

Der DECODE Prozess selbst besteht aus zwei Unterprozessen, DECODE_DECODE und BITSTR, wie in Abbildung 4.8 dargestellt.

Diese Unterprozesse bestehen selbst wieder aus Prozessen. DECODE_DECODE besteht aus Ready und Decode.

Die Aufgabe von Ready ist es, auf eine HIGH Bit, welches den Anfang einer Nachricht markiert, zu warten, die Nachricht bitweise einzulesen und nach dem Empfang der drei Schlussbits das DECODE Signal zu setzen.

Für das Einlesen eines Frames wird die Unterprozedur DECODE_DECODE Frame_Start verwendet. Sie ordnet jedes von dem BITSTR

4 Spezifikation

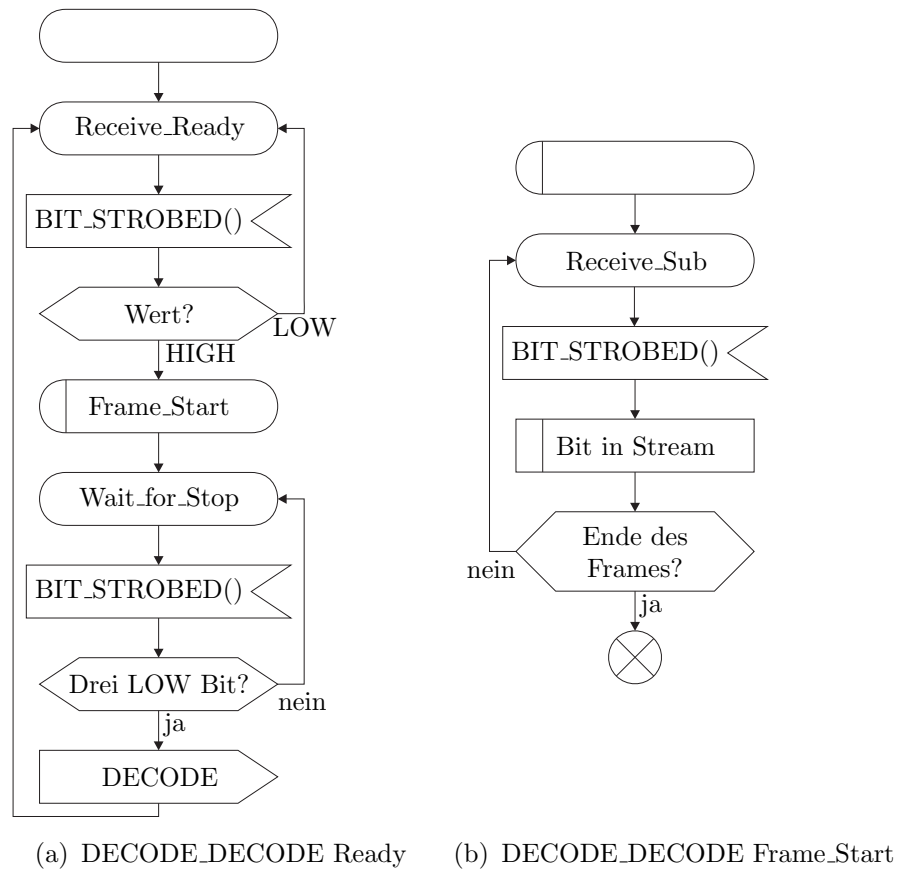


Abbildung 4.9: Der DECODE_DECODE Prozess Ready und die Unterprozedur Frame_Start

Prozess erkannte Bit in einem Stream an. Die nötigen SYNC-Bits werden in der Funktion Bit in Stream erkannt und nicht in den Stream eingeordnet. Am Ende jeden Frames springt die Unterprozedur zurück und der Prozess DECODE Ready wird weiter abgearbeitet.

Die Decodierung von HCRC und FEC findet in DECODE_DECODE Decode statt, welcher durch das DECODE Signal aktiviert wird. Dabei wird zuerst der FEC angewandt um eventuell vorkommende Bitfehler zu korrigieren. Anschließend wird der HCRC auf Korrektheit geprüft. Terminiert der FEC Algorithmus und ist der HCRC korrekt, wird das Signal FRAME_ON gesetzt. Wird hierbei die Nachricht geändert, wird im Folgenden mit der korrigierten Version der Nachricht gearbeitet. Treten bei der Decodierung Fehler auf, wird das Signal ERROR_ON gesetzt.

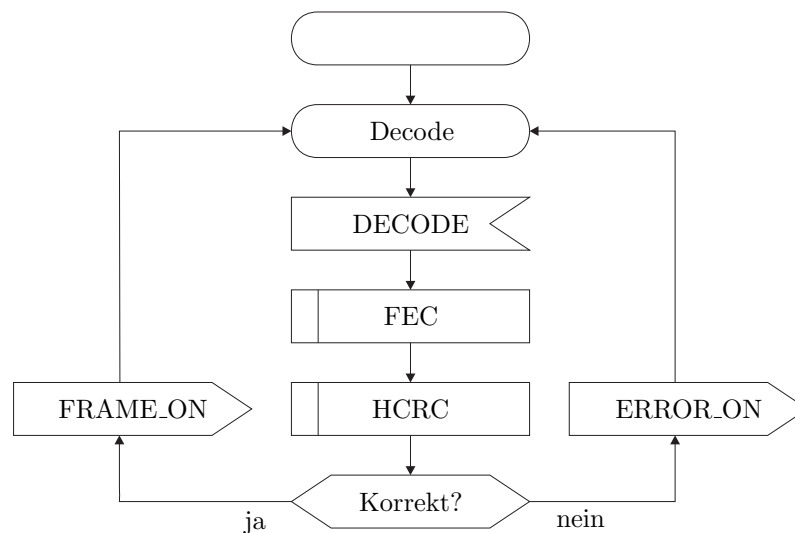


Abbildung 4.10: DECODE_DECODE Decode

Die Entscheidung, ob ein Bit HIGH oder LOW empfangen wurde, fällt in den in den Abbildungen 4.11(b) und 4.11(a) gezeigten BITSTR Prozessen. Es wird hierzu ein Voting Window der Länge 8 Samples verwendet. Innerhalb dieses Voting Wondows wird der Wert des Bits durch einen Mehrheitsentscheid bestimmt. Werden also 5 Samples des gleichen Pegels empfangen so steht der Wert des Bits fest.

4.8 ENCODE

Die zwei Unterprozesse, aus denen der ENCODE Prozess besteht, sind in Abbildung 4.12(a) und Abbildung 4.12(b) dargestellt. Gemeinsam sind diese

4 Spezifikation

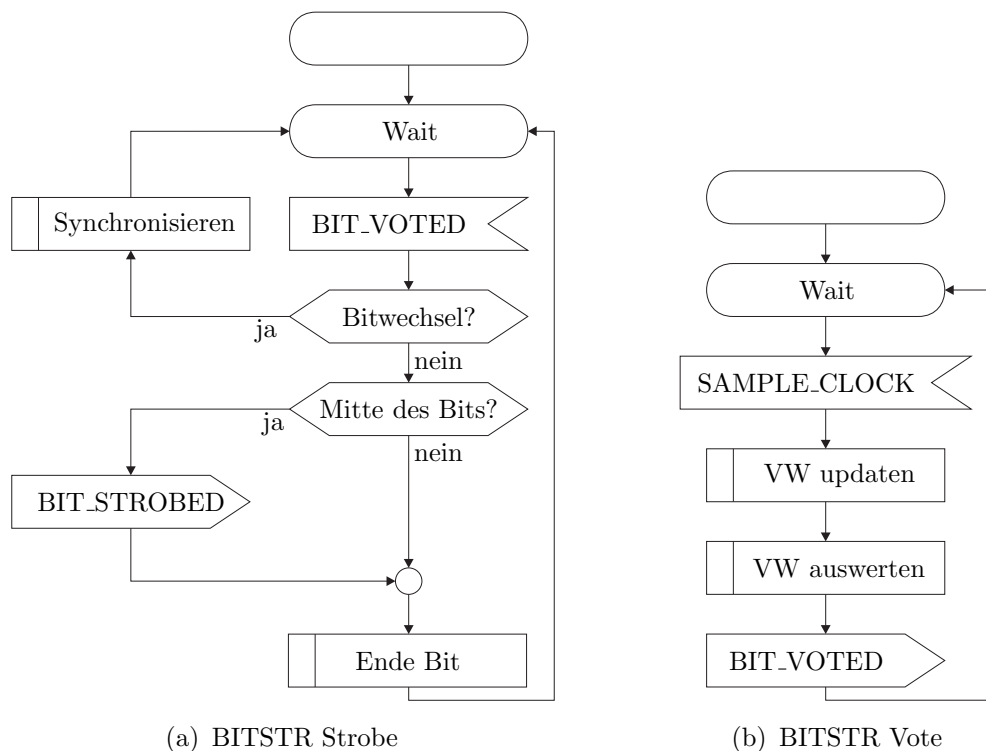


Abbildung 4.11: Die BITSTR Prozesse

Prozesse dafür zuständig, eine Nachricht, die von der OC in vSendQueue gestellt wurde, an die downstream Knoten zu versenden.

Wird eine Nachricht, die von den upstream Knoten empfangen wurde, weiterversendet, so ist das Receive Flag gesetzt. Da diese Nachricht bereits mit FEC und HCRC empfangen wurde und bei der Decodierung bereits mit Hilfe des FEC Algorithmus korrigiert wurde, muss die Nachricht nur versendet werden. Eigene Nachrichten werden in dem Block Generiere Nachricht mit HCRC und FEC versehen.

Bei dem bitweisen Versenden werden die nötigen SYNC-Bits eingefügt.

4.9 Operation Control (OC)

Das Herzstück des SafeNet Knotens ist die Operation Control. Die Operation Control ist aus sieben unabhängigen Prozessen aufgebaut, von denen im Folgenden sechs näher erklärt werden. Der interne Aufbau der OC ist in Abbildung 4.13 dargestellt.

Teil der OC ist der Sample_Clock Prozess, der aus dem von außen angelegten Takt den Takt für alle Prozesse generiert. Da alle Prozesse dieses Signal empfangen, ist es in den Abbildungen nicht enthalten.

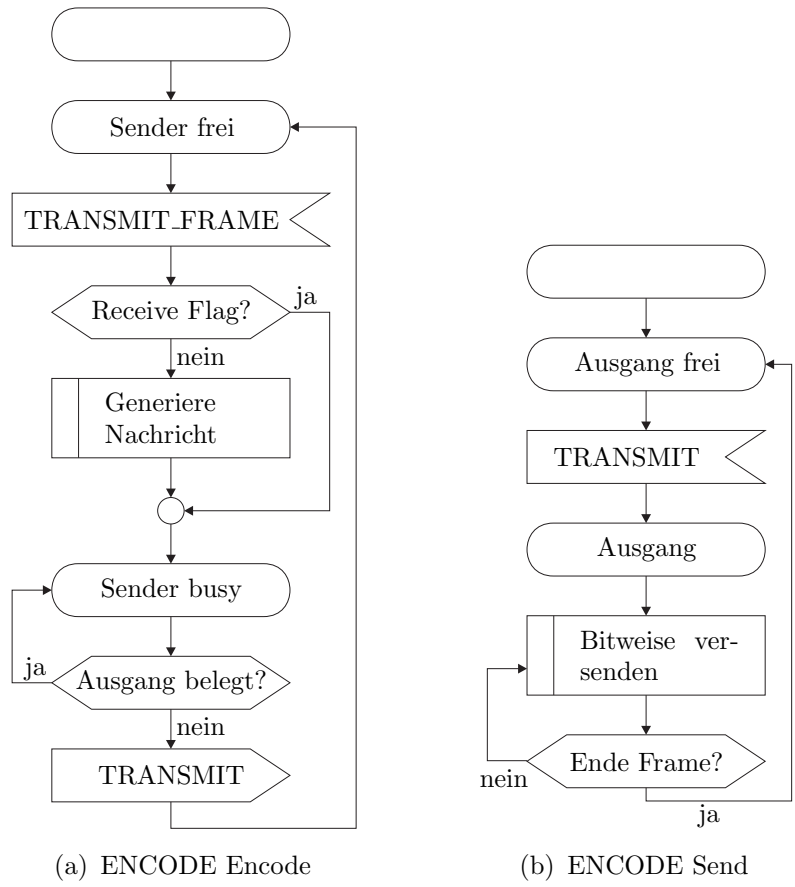


Abbildung 4.12: Der ENCODE Prozess

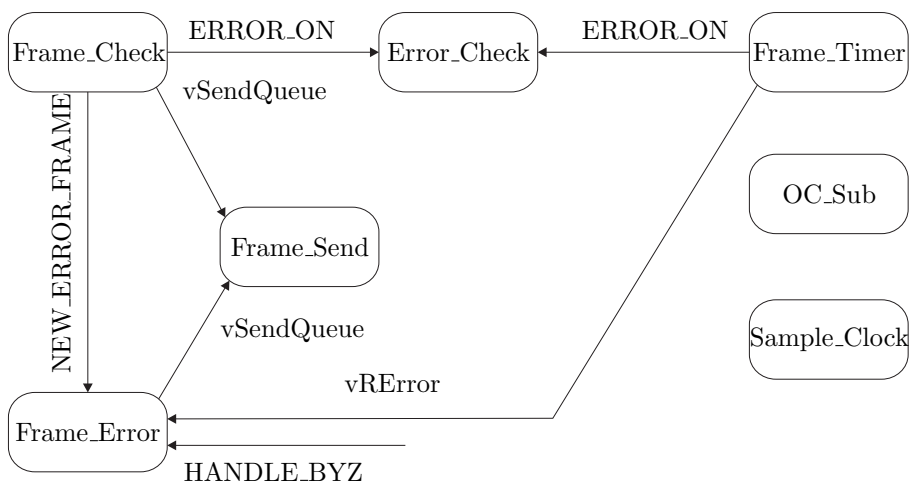


Abbildung 4.13: Aufbau des OC Prozesses

4 Spezifikation

Ist die Decodierung eines Frames abgeschlossen, so wird von DECODE_DECODE Decode das FRAME_ON Signal gesetzt. Dieses wird in der OC von Frame_Check abgefangen, woraufhin die Verarbeitung der Nachricht in der OC beginnt. Aufgabe von Frame_Check ist es, den Zugriff der Eingänge auf die Sendewarteschlange, etc. zu serialisieren. Daher muss für jeden der beiden Eingänge ein Frame_Check Prozess existieren.

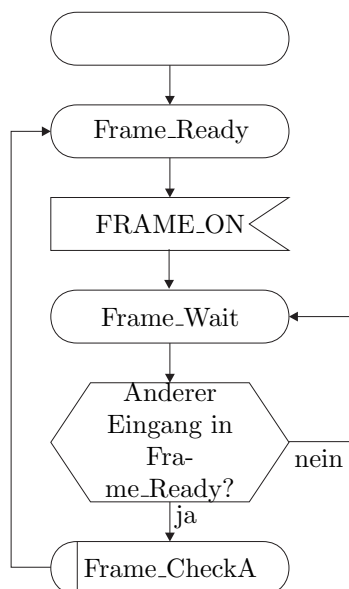


Abbildung 4.14: OC Frame_Check

Ist sichergestellt, dass der andere Eingang nicht auf die Warteschlange oder andere gemeinsam genutzte Ressourcen zugreift, so wird die Nachricht in der Unterprozedur Frame_Check_A weiterverarbeitet. Hierzu wird zuerst der Header auf kontextuelle Fehler überprüft. Stimmt der in dem ML Feld angegebene Wert nicht mit der tatsächlichen Länge der in der Nachricht enthaltenen Daten überein, so wird `vRError=4` gesetzt und andere Prozesse mit dem ERROR_ON Signal informiert. An dieser Stelle ist zu bemerken, dass eine falsche Datenlänge bei variabler Nachrichtenlänge als syntaktischer Fehler zu werten ist. Da jedoch hier der Syntax mit fester Datenlänge definiert ist, resultiert eine Abweichung von dieser Länge in einem Decodierfehler.

Ist die empfangene MID keiner UID im Netz zugeordnet, so wird `vRError=5` gesetzt und andere Prozesse mit dem ERROR_ON Signal informiert.

Weicht der MSI von dem erwarteten ab, so wird `vRError=6` gesetzt. Die MSI wird für jeden Nachrichtentyp, also jede MID, gezählt. Wird nur ein Zähler pro Knoten verwendet, so kann der Fall eintreten, dass durch Überlauf des MSI Zählers zwei aufeinanderfolgende Nachrichten der gleichen MID auch die gleiche MSI besitzen, was die Unterscheidung zwischen byzantinischem Fehler und falscher MSI erschwert.

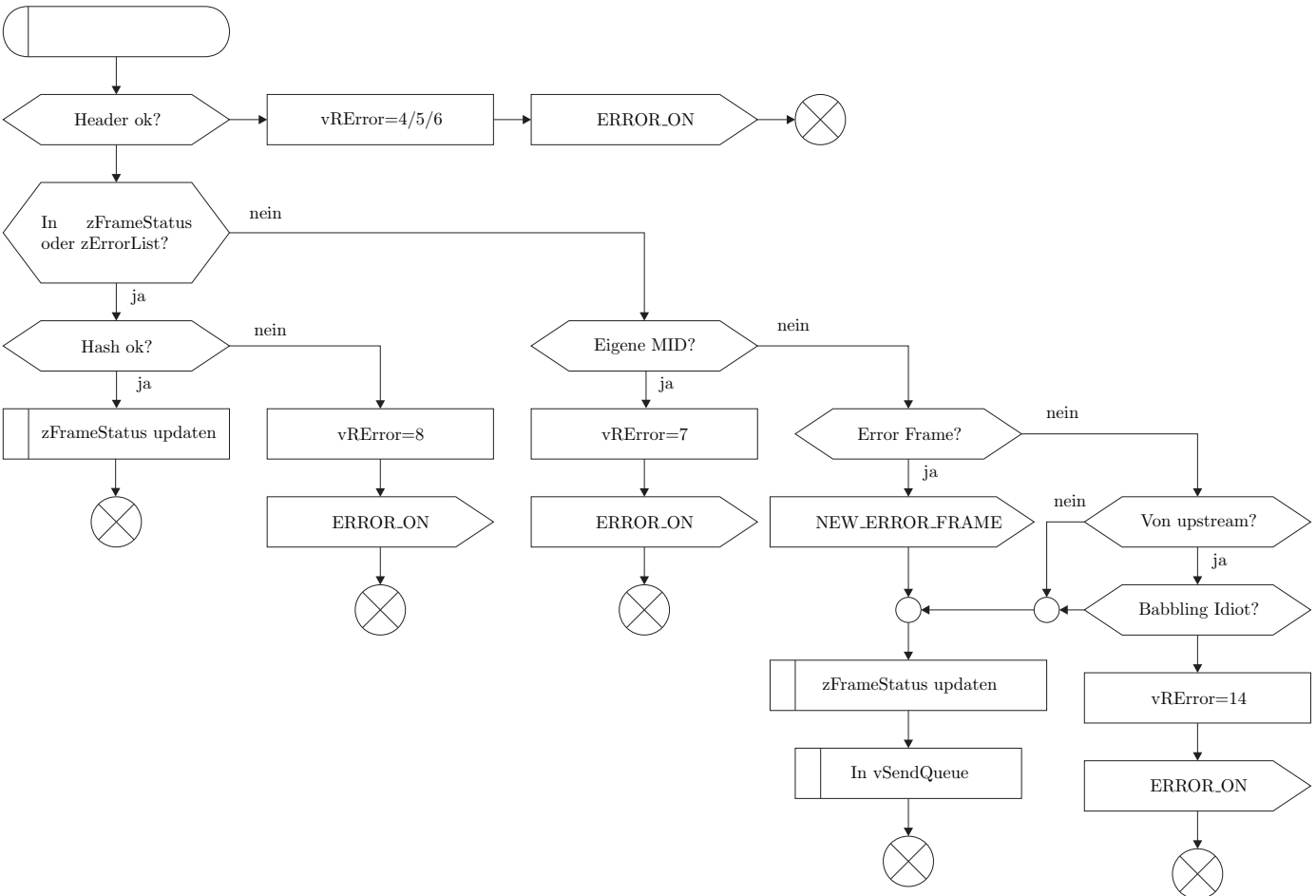


Abbildung 4.15: OC Frame-CheckA

4 Spezifikation

Ist der Header kontextuell korrekt, so wird überprüft, ob diese Nachricht weiterversendet werden muss. Hierzu wird `zFrameStatus` sowie die Liste eigener versendeter Nachrichten überprüft. Wurde die Nachricht bereits auf dem anderen Eingang empfangen und stimmen auch die Daten, beziehungsweise der Hash überein, so wird lediglich in `zFrameStatus` vermerkt, dass die Nachricht auf beiden Kanälen empfangen wurde. Stimmt der Hash nicht mit dem der bereits empfangenen Nachricht überein, so liegt ein byzantinischer Fehler vor¹. Daher wird `vRError=8` gesetzt und `ERROR_ON` signalisiert.

Wird bei der Überprüfung festgestellt, dass eine Nachricht mit einer MID empfangen wurde, die dem überprüfenden Knoten zugeordnet ist, von diesem aber nicht versendet wurde, so liegt ein Masquerading Fehler vor (`vRError=7`) und `ERROR_ON` wird gesetzt.

Beim erstmaligen Empfang einer Fehlernachricht wird die Nachricht direkt in `zFrameStatus` und `vSendQueue` gestellt und damit weiterversendet, wenn gewährleistet ist, dass der Header korrekt ist und kein Masquerading Fehler aufgetreten ist.

Bei regulären Nachrichten wird zusätzlich überprüft, ob der sendende upstream Knoten ein Babbling Idiot (`vRError=14`) ist. Ist dies der Fall, wird `ERROR_ON` gesetzt. Anderenfalls wird die Nachricht wie für Fehlerframes beschrieben, versendet.

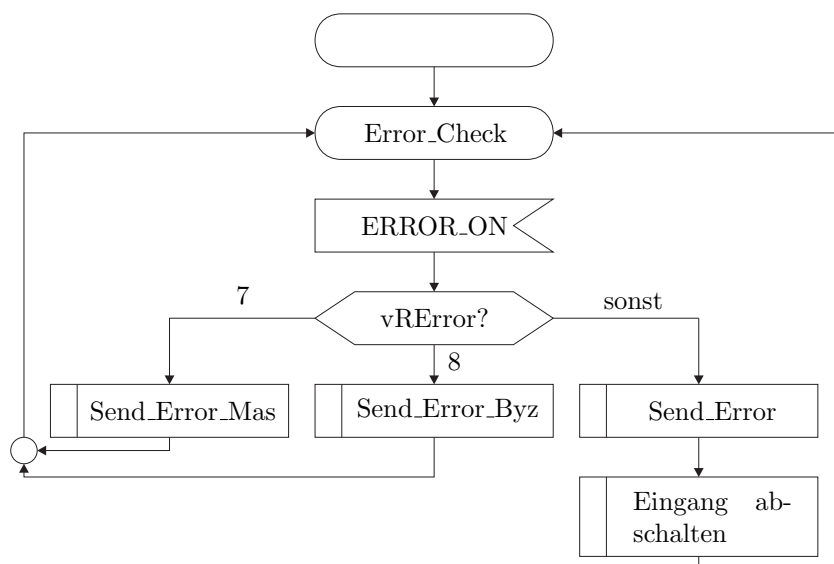


Abbildung 4.16: OC Error_Check

¹Streng genommen ist dies nicht zwingend der Fall, da sich ja nur die beiden Versionen in einem Knoten unterscheiden, die aber von unterschiedlichen Sendeereignissen stammen. Ob der andere downstream Knoten des Senders ebenfalls eine falsche Nachricht erhalten hat oder nicht ist für den betrachteten Knoten nicht ersichtlich. Daher wird von dem schwerer zu behandelnden Fall, eben einem byzantinischen Fehler, ausgegangen.

Die in `Frame_Check_A` und `Frame_Timer` gemeldeten Fehler werden in `Error_Check` behandelt. Zu unterscheiden sind drei Fälle:

Masquerading Fehler (vRError=7) Versenden einer Fehlernachricht wie in Abbildung 4.2 gezeigt

byzantinischer Fehler (vRError=8) Versenden einer Fehlernachricht wie in Abbildung 4.3 gezeigt

andere Fehler Versenden einer Fehlernachricht wie in Abbildung 4.2 gezeigt und Abschalten des Eingangs

Bei Masquerading und byzantinischen Fehlern wird der betroffene Eingang nicht direkt abgeschaltet, da nicht sicher davon ausgegangen werden kann, dass der upstream Knoten den Fehler verursacht hat. Daher wird der betroffene Knoten erst nach der Abarbeitung des Fehlerfindungsalgorithmus' abgeschaltet.

Neue Fehlernachrichten, die `Frame_Check_A` mit `NEW_ERROR_FRAME` meldet, werden in `Frame_Error` behandelt. Unterschieden werden die Aktionen anhand von `vRError`. Alle neuen Fehlernachrichten werden in `vSendQueue` und `vErrList` aufgenommen. Zusätzlich werden der Masquerading Fehler und der byzantinische Fehler noch behandelt, was für byzantinische Fehler in Abbildung 4.17 ausführlich dargestellt ist.

Wird eine Fehlernachricht für einen byzantinischen Fehler empfangen, die sich auf eine eigene Nachricht bezieht, so muss zusätzlich zu der Fehlernachricht eine Korrekturnachricht wie in Abbildung 4.4 dargestellt gesendet werden. Hierdurch können alle Knoten im Netzwerk entscheiden, ob sie die richtige oder die verfälschte Version der Nachricht erhalten haben.

Der Prozess geht nun in den Zustand `Error_Byz` über und neu ankommende Fehlernachrichten werden auch in die `zErrList` aufgenommen. Nach Ablauf einer Wartezeit oder nach dem Empfang aller Fehlernachrichten, für den Prozess der Empfang des Signals `HANDLE_BYZ`, wird der eigentliche Fehlerfindungsalgorithmus gestartet.

Liegt eine Nachricht in der Sendewarteschlange `vSendQueue`, so sendet der `Frame_Send` Prozess innerhalb der OC bei freiem Sender, also falls der `ENCODE` Send Prozess im Zustand Ausgang frei ist, das `TRANSMIT` Signal. Hiermit wird das Versenden einer Nachricht im `ENCODE` Prozess angestoßen.

Der `Frame_Timer` Prozess ist die Schnittstelle zwischen dem nicht spezifizierten Prozess, der die Aktivitäten der beiden Eingänge überwacht, und der OC. Erhält er das Signal `TIMER`, so setzt er `vRError=13` und das Signal `ERROR_ON`.

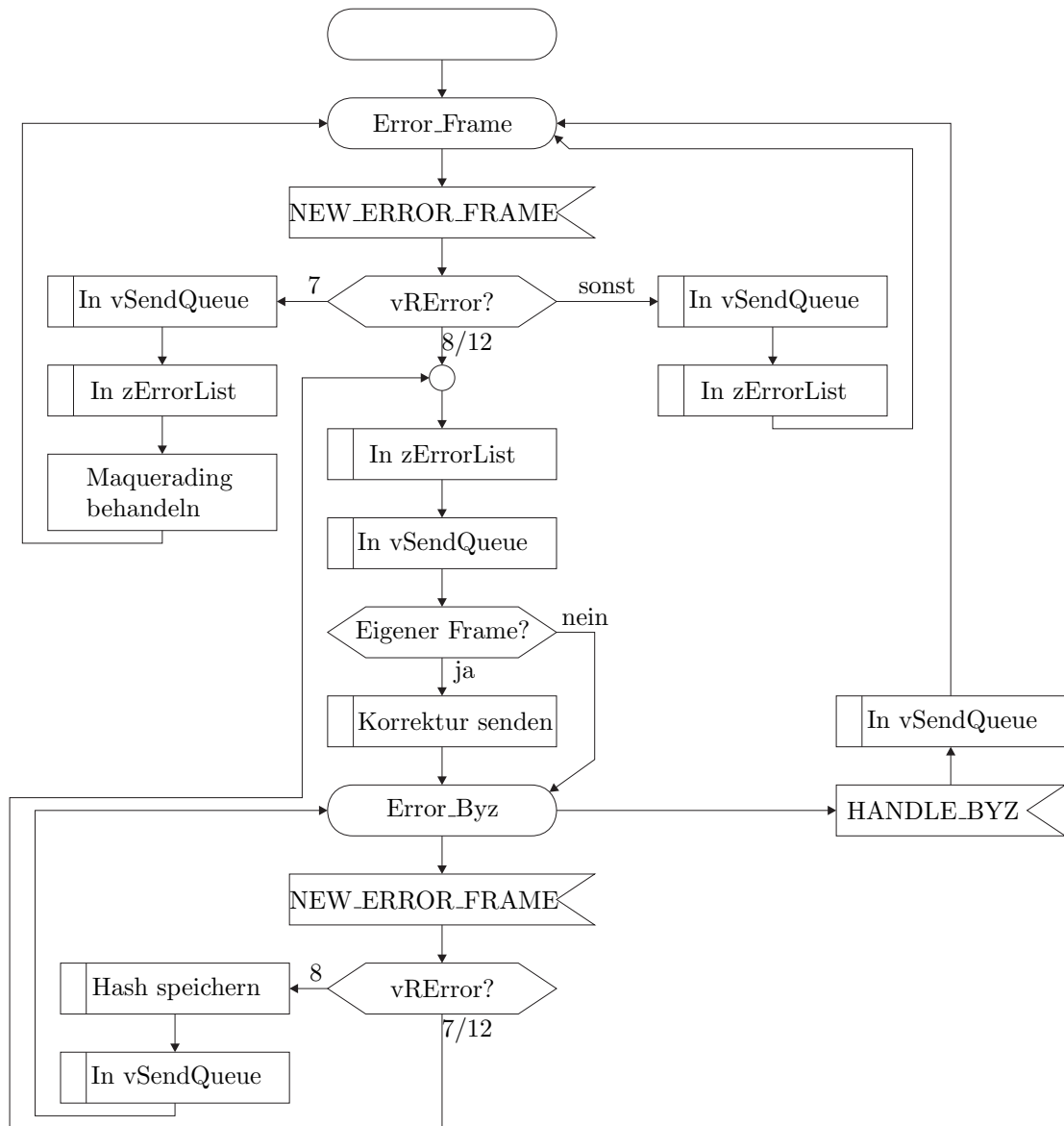


Abbildung 4.17: OC Frame_Error

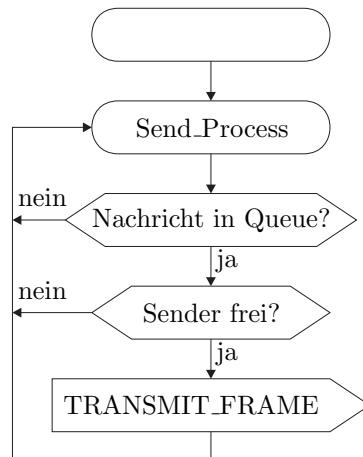


Abbildung 4.18: OC Frame_Send

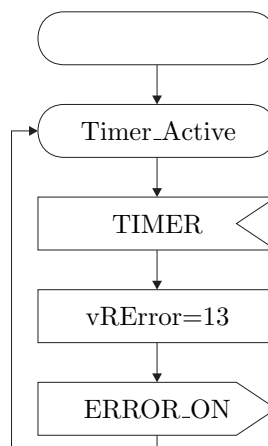


Abbildung 4.19: OC Frame_Timer

4.10 Dynamischer Start-up

Bisher wurde der Fall betrachtet, dass alle für den Betrieb eines SafeNets nötigen Daten, also Wissen über die MIDs und die Netzwerkstruktur in Form der Adjazenzmatrix zu Beginn des Betriebs vorliegen. Dies kann beispielsweise durch Ablegen der Daten in einem ROM geschehen. Es wird davon ausgegangen, dass diese Daten in allen Knoten vorliegen und dabei konsistent sind. Diese Annahme ist für den Einsatz in sicherheitsrelevanten Systemen nicht gültig. Daher müssen Strategien entwickelt werden, um vor dem eigentlichen Betrieb konsistente Datensätze zu gewährleisten.

Bemerkung 4.1 Die in diesem Unterabschnitt vorgestellten Vorgehensweisen sind weder für das Modell (Unterabschnitt 5.1.1) noch für die Implementierung (Abschnitt 5.2) relevant.

Im Folgenden werden Strategien entwickelt, um den Einschaltvorgang auch ohne die Beschränkung zu Beginn konsistenter Datensätze zu ermöglichen. Es wird hierzu zuerst der Fall betrachtet, dass die Daten zwar verteilt in den Geräten vorliegen, ihre Konsistenz aber nicht gewährleistet ist. Danach wird diese Strategie auf den Fall erweitert, dass zu Beginn nur lokale Daten, also Wissen über den Knoten selbst, in jedem Knoten vorliegen. In beiden Fällen wird angenommen, dass der Start-up klar von dem normalen Betrieb getrennt ist.

4.10.1 Verifizieren der vorliegenden Daten

Liegen nach dem Einschalten die notwendigen globalen Daten in den Geräten vor, so müssen diese Daten vor dem regulären Betrieb lediglich auf Konsistenz überprüft werden. Hierzu reichen zwei Klassen von Nachrichten aus. Zuerst wird von jedem Knoten eine Nachricht versendet, die, entweder direkt über die MID oder über den Inhalt der Nachricht, Rückschlüsse über den Absender zulässt. Diese Nachricht muss auch die relevanten Daten in geeigneter Form zusammenfassen, zum Beispiel in einem Hash. Hierdurch ist es den anderen Knoten des Netzwerkes möglich, Abweichungen von ihren eigenen Daten festzustellen. Tritt eine solche Abweichung in einem Gerät auf, muss dies signalisiert werden und darf das Netzwerk nicht in den normalen Betrieb übergehen. Die zweite Nachrichtenart, die hier benötigt wird, benachrichtigt alle anderen Geräte über die Tatsache, dass ein Gerät die Nachrichten aller anderen Knoten im Netz empfangen hat und nun bereit ist, in den regulären Betrieb überzugehen.

Bei dem Versenden der ersten Nachrichtenart, also der, mit deren Hilfe ein Knoten eine Prüfsumme seiner eigenen Daten versendet, kann noch

nicht davon ausgegangen werden, dass alle Knoten in dem Netzwerk empfangsbereit sind. Daher muss diese Nachricht periodisch versendet werden, bis eine entsprechende Nachricht von allen anderen Knoten vorliegt. Liegen diese Nachrichten vollständig vor, so kann nach nochmaligem Senden der eigenen Nachricht, und natürlich dem Ausbleiben von Fehlermeldungen bezüglich dieser Nachricht, davon ausgegangen werden, dass alle Knoten diese eigene Nachricht korrekt empfangen haben. Um den Knoten noch die Möglichkeit zu geben, interne Konfigurationen, zum Beispiel Kommunikation mit dem Host, durchzuführen, wird über die zweite Nachricht die Bereitschaft zum Übergang in den regulären Betrieb signalisiert.

4.10.2 Vollständig dynamischer Start-up

Wird der Start-up auf ein Verifizieren der globalen Daten reduziert, so ist eine Änderung der Konfiguration des Netzwerkes nicht möglich. Dies kann in sicherheitsrelevanten Systemen unter Umständen gewünscht sein, um Veränderungen des Systems durch Unbefugte aus Haftungsgründen auszuschließen. Andererseits wird durch dieses Vorgehen die in der Automobilindustrie ausgeprägte Variantenbildung erschwert, da für jede Variante eine Version der globalen Daten vorliegen muss und diese konsistent in den Geräten des Systems gespeichert werden muss. Wird besonderer Wert auf die Konfigurierbarkeit des Systems gelegt, so ist ein vollständig dynamischer Start-up, zu dessen Beginn keine globalen Daten, also kein Wissen über das übrige Netzwerk in einem Knoten, vorliegen, zu bevorzugen.

Um die globalen Daten in den Geräten korrekt zu generieren, wird der Start-up weiter unterteilt. In der ersten Phase wird von jedem Gerät lediglich eine Nachricht gesendet, die dieses Gerät eindeutig identifiziert. Empfängt ein Knoten während dieser Phase eine Nachricht von seinen upstream Knoten, so fügt er diese Information seiner eigenen Nachricht hinzu. Weitergeleitet wird dieser Nachrichtentyp nicht, womit der eigentliche Sendalgorithmus von SafeNet verletzt wird. Hat ein Knoten diese Nachrichten von seinen beiden upstream Knoten erhalten, so sendet er zusätzlich einen Nachrichtentyp, die dem Knoten zugeordnet werden kann und die Information über beide upstream Knoten enthält. Diese Nachricht wird wie übliche Nachrichten weitergeleitet. Hierdurch wird die Topologie des Netzwerkes allen Knoten mitgeteilt.

Diese Phase endet, wenn die topologische Information aller Knoten in einem Knoten vorliegt, also für jeden Knoten beide Vorgänger und beide Nachfolger bekannt sind. Ist dies der Fall, wird der Sendalgorithmus auf alle Nachrichten angewandt. Die Knoten versenden nun je eine Nachricht für jede ihnen zugeordnete MID. Diese Phase ist τ_{max} nach dem Empfang der

4 Spezifikation

letzten Nachricht beendet. Die Knoten können nun wie zuvor beschrieben die vorliegenden Informationen verifizieren.

Der vollständig dynamische Start-up erfordert lediglich eine Änderung der OC, da DECODE und ENCODE Prozesse wie gewohnt funktionieren. Das CHI wird zur Kommunikation mit dem Host noch nicht benötigt. Abhängig davon, wo der Sendealgorithmus implementiert ist, wird ein rudimentäres CHI benötigt. Innerhalb der OC können die Fehlerentdeckungsalgorithmen wie gewohnt verwendet werden, da auch in der Start-up Phase Fehler auftreten können.

4.10.3 Bewertung des dynamischen Start-ups

Ob ein dynamischer Start-up benötigt oder gewünscht ist, hängt stark von dem Gesamtsystem ab, in dem SafeNet eingesetzt wird. Die Algorithmen des dynamischen Start-ups können aber auch zur Wiedereingliederung von Knoten nach dem Auftreten von transienten Fehlern dienen und sind daher auch sinnvoll, wenn während dem Start-up lediglich die vorliegenden globalen Daten verifiziert werden.

5 Simulation und Implementierung

Nach der theoretischen Analyse in Kapitel 3 soll SafeNet nun in diesem Kapitel mit Hilfe einer umfangreichen Simulation sowie einer proof-of-concept Implementierung verifiziert werden. Hierfür wird die in Kapitel 4 spezifizierte Version zu Grunde gelegt.

5.1 Simulation

Trotz der einfachen Algorithmen, die bei SafeNet zum Einsatz kommen, ist die genaue Nachrichtenausbreitung komplex, da sie neben der Topologie auch von dem zukünftigen Sendeverhalten der übrigen Knoten abhängt. Sie ist daher nur mit Hilfe einer akausalen Systemfunktion zu beschreiben. Auf eine detaillierte Beschreibung dieser Funktion wird an dieser Stelle verzichtet, da aus ihr für den allgemeinen Fall lediglich τ_{max} hergeleitet werden kann, was schon auf anschaulichere Weise erfolgt ist. Zur Analyse des Zeitverhaltens von SafeNet wird auf ein Modell zurückgegriffen, welches in Unterabschnitt 5.1.1 näher vorgestellt wird, bevor in Unterabschnitt 5.1.2 die Ergebnisse der Simulation vorgestellt werden.

5.1.1 Modell

Das zur Analyse verwendete Modell wurde in Matlab/Simulink erstellt. Diese Plattform wurde gewählt, da sich so leicht Erweiterungen, wie beispielsweise ein Kanalmodell, einfügen lassen. Auf der obersten Ebene ist jeder Knoten eines SafeNets ein Simulink Block, der über zwei Dateneingänge sowie zwei Datenausgänge verfügt. Zusätzlich besitzt jeder Knoten noch einen Eingang für einen Taktgeber. Dies ermöglicht die Analyse des Einflusses von Abweichungen der lokalen Uhren auf die Kommunikation.

Die Algorithmen sind wo möglich der einfachen Lesbarkeit wegen in Stateflow implementiert. Zur Beschleunigung der Ausführung wurden zeitkritische Passagen in C beziehungsweise in Matlab-Code geschrieben. Die Konfiguration des Modells sowie die Ausgabe der Ereignisse erfolgt in Dateien, da diese einerseits aus C einfach einzulesen sind und die Daten so für eine offline-Analyse zur Verfügung stehen.

5.1.2 Simulationsergebnisse

In den bisherigen Ausführungen wurde die Nachrichtenausbreitung stets in Teilen betrachtet, so zum Beispiel lediglich an einem Knoten. Die charakteristischen Größen eines SafeNets wie die Länge der Sendewarteschlange wurden dabei meist an dem einfachen Beispiel des chordalen Rings und auch dann nur unter Berücksichtigung des worst-case aufgezeigt. Mit Hilfe des in Unterabschnitt 5.1.1 vorgestellten Modells ist es nun möglich, die Ausbreitung von Nachrichten zu analysieren.

Im Folgenden werden exemplarisch die Ergebnisse für einige Topologien vorgestellt und diskutiert. Simulationsergebnisse für weitere Topologien finden sich in Anhang B.

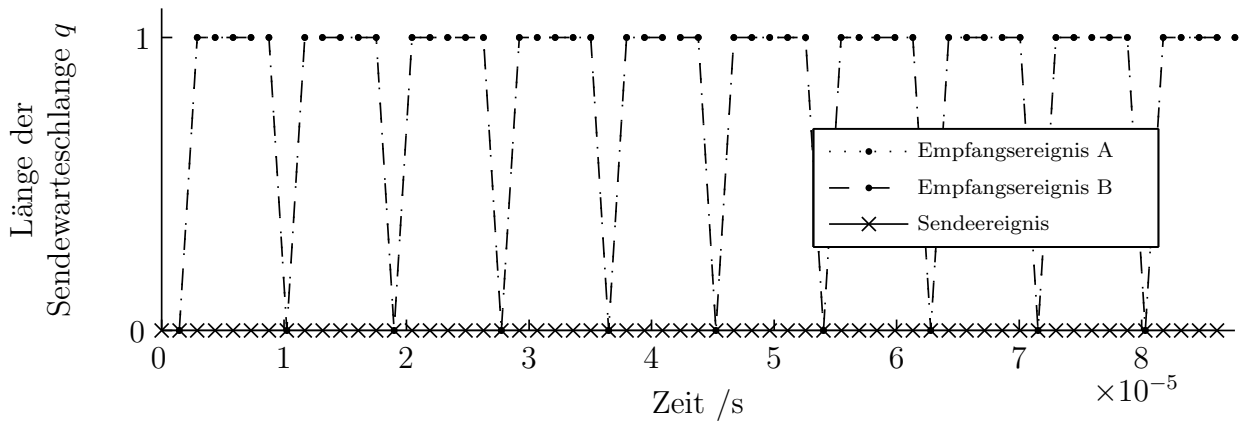
Für die Simulationen wurde das Modell mit einem Takt von 800 MHz betrieben, was einer achtfachen Überabtastung bei einer Netzwerkgeschwindigkeit von 100 MBit/s entspricht. Um eine statistisch relevante Datenmenge zu erhalten, wurde eine Zeitspanne von 5 ms simuliert. Die Netzwerklast ρ wurde zu $\rho = 1$ gewählt, was in der Implementierung bedeutet, dass jedes Gerät immer eine neue eigene Nachricht sendet, wenn es die letzte eigene auf einem der Eingänge erhält. Hierdurch ist gewährleistet, dass auf den einzelnen Verbindungen immer Nachrichten versendet werden. Da das Verhalten des Systems nach einer kurzen Einschwingphase größtenteils über den gesamten Simulationsverlauf in den charakteristischen Größen ähnlich bleibt, wird die Darstellung der Zeitverläufe aus Gründen der Übersichtlichkeit beschränkt.

Chordaler Ring mit sechs Knoten

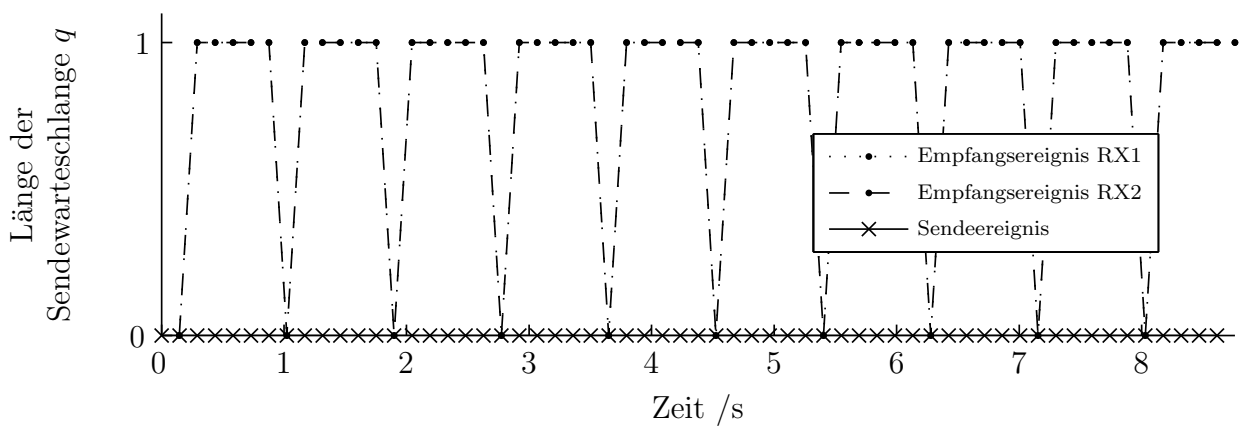
Zuerst werden die Simulationsergebnisse für den chordalen Ring, wie er in Abbildung 3.3(a) dargestellt ist, vorgestellt. In den vorhergehenden Kapiteln wurden bereits einige Eigenschaften des chordalen Rings, nämlich die Beschränktheit der maximalen Länge der Sendewarteschlange und der von ihr abhängigen maximalen Laufzeit einer Nachricht, theoretisch hergeleitet.

Durch die Ergebnisse der Simulation werden die Ergebnisse der theoretischen Überlegungen bestätigt. In Abbildung 5.1 ist der Verlauf der Länge der Sendewarteschlange über der Zeit dargestellt. Die Länge ist ereignisdiskret zu Ankunfts- und Verlassenszeitpunkten von Nachrichten für zwei der sechs Knoten aufgetragen. Es wird bei den Ankunftsereignissen unterschieden, ob die Nachricht auf Eingang RX1 oder RX2 empfangen wird.

Es ist zu erkennen, dass die Länge der Sendewarteschlange nie über $q = 1$ steigt. Grund hierfür ist, dass im ersten Schritt jeder Knoten eine eigene Nachricht sendet und somit das Generieren der eigenen Nachricht nicht mit dem Empfang zweier verschiedener Nachrichten auf den Eingängen zusam-



(a) Knoten A



(b) Knoten B

Abbildung 5.1: Länge der Sendewarteschlange q über der Zeit

menfallen kann. Da eine Nachricht direkt nach dem Empfang versendet wird, ohne in die Sendewarteschlange gestellt zu werden, falls der Ausgang zu dem Empfangszeitpunkt frei ist, steigt die Länge der Sendewarteschlange nicht über $q = 1$.

Interessant sind auch die periodischen Einbrüche der Länge der Sendewarteschlange auf $q = 0$. Da ein Knoten bei $\rho = 1$ immer zwei Nachrichten pro Zeitschritt erhält, muss es sich hier um zwei dem Knoten bereits bekannte Nachrichten handeln. Dies ist beispielsweise für Knoten C der Fall, wenn von B eine Nachricht, die auch schon von A empfangen wurde, gesendet wird, und von A die Nachricht β , die ursprünglich von B stammt, also C erst nach einem kompletten Durchlauf durch das Netz wieder erreicht, gesendet wird. In dieser Situation kann C die noch in der Sendewarteschlange stehende Nachricht versenden, ohne dass eine weitere Nachricht in die Sendewarteschlange ge-

5 Simulation und Implementierung

stellt würde. Die Länge der Sendewarteschlange sinkt also in jedem $1/N$ -ten Schritt.

Da, wie erwähnt, jeder Knoten immer eine Nachricht sendet, wenn dies durch den Algorithmus möglich ist, ist die ereignisdiskrete Funktion der Länge der Sendewarteschlange periodisch mit der Anzahl der Knoten im Netz.

In Abbildung 5.2 ist eine andere Perspektive, nämlich die relative Häufigkeit der Längen der Sendewarteschlangen zu den Empfangs- und Sendezeitpunkten zu sehen. Da nach den Empfangszeitpunkten höchstens eine Nachricht in der Warteschlange steht, ist klar, dass nach den Sendezeitpunkten die Sendewarteschlange immer leer sein muss.

Die relativen Häufigkeiten, also der Anteil einer Warteschlangenlänge an den gesamten Beobachtungen, in den Empfangszeitpunkten sind durch die Periodizität der Funktion der Länge der Sendewarteschlange zu erklären, siehe auch Abbildung 5.1. Die Wahrscheinlichkeit, dass in einem Empfangszeitpunkt keine Nachricht in der Warteschlange steht, ist, wie oben diskutiert, $1/N$.

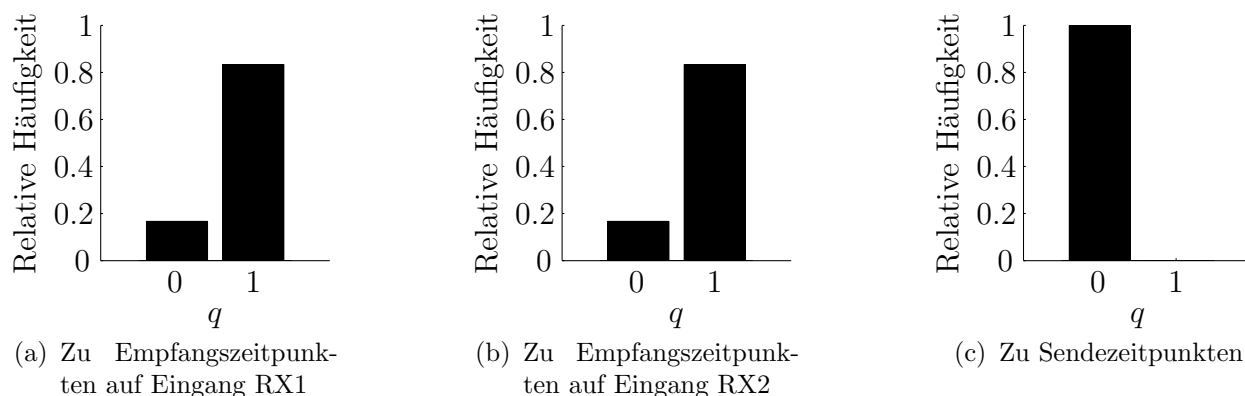


Abbildung 5.2: Relative Häufigkeit der Warteschlangen-Längen q in Knoten A

Die Verteilung der MIDs der von Knoten A empfangenen beziehungsweise gesendeten Nachrichten ist in Abbildung 5.3 zu sehen. Wie schon zuvor angedeutet, ist hier die Verteilung der MIDs gleich, was auf die Symmetrie der zugrunde liegenden Topologie zurückzuführen ist, siehe auch Abschnitt 3.3. Geringe Abweichungen sind darauf zurückzuführen, dass zum Ende der Simulationszeit noch immer Nachrichten im Netzwerk versendet wurden.

Ein wichtiges Kriterium zur Beurteilung der Echtzeitfähigkeit von SafeNet ist die Zeit, die eine Nachricht benötigt, um durch das Netz wieder zu ihrem ursprünglichen Absender zurückzukehren. Im chordalen Ring entspricht sie der Zeit, die maximal benötigt wird, um eine Nachricht an alle Knoten zu senden, also τ_{max} . Diese Zeit ist, unterschieden nach den beiden Eingängen, in

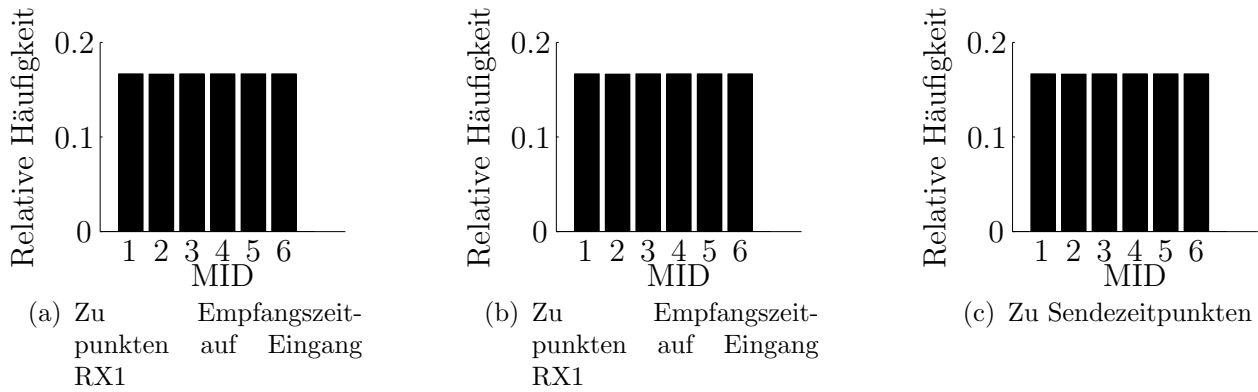


Abbildung 5.3: Relative Häufigkeit des Auftretens aller MIDs in Knoten A

Abbildung 5.4 zu sehen. Dabei ist zu berücksichtigen, dass von dem Modell des Sendeereignisses das *Ende* als Zeitpunkt des Versendens der Nachricht verwendet wird. Daher muss für die tatsächlich benötigte Zeit noch $\tau_s = 1.46 \mu\text{s}$ zu dem in Abbildung 5.4 dargestellten Wert τ_{RX1} beziehungsweise τ_{RX2} addiert werden. Die Differenz zwischen dem ersten und dem zweiten Empfangsereignis ist mit Δ_τ bezeichnet.

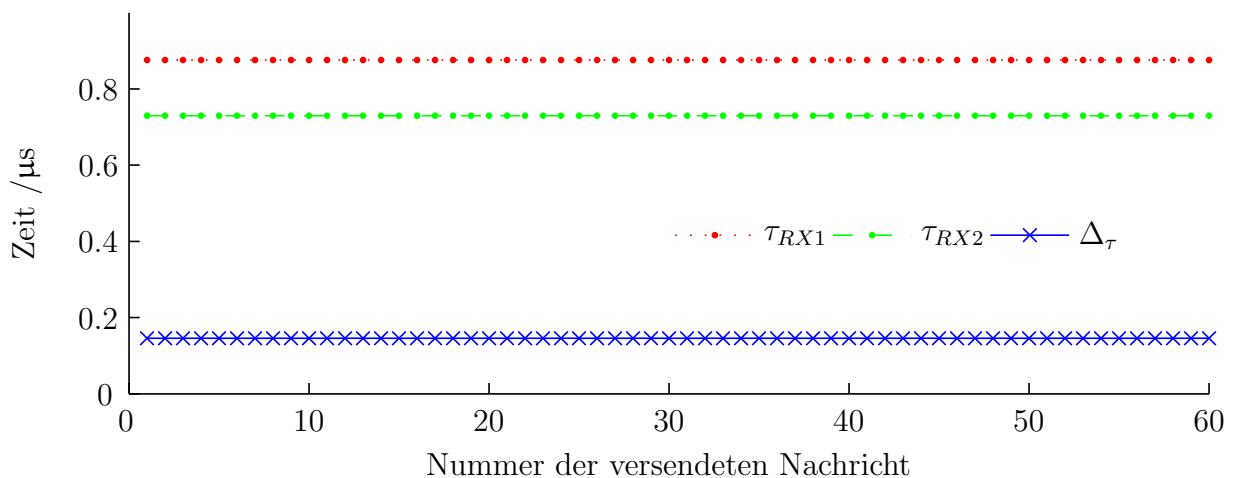


Abbildung 5.4: Laufzeiten aller von Knoten A versendeten Nachrichten bis zu ihrem Empfang auf Eingang RX1 und Eingang RX2

Es wird deutlich, dass die Zeitdauer über nahezu den gesamten Verlauf der Simulation konstant bleibt, von einem transienten Vorgang zu Beginn abgesehen. Es wird auch deutlich, dass der tatsächliche Wert von τ_{max} mit $\tau_{max} = 8.76 \mu\text{s}$ unter dem durch Gleichung 3.28 bestimmten Wert von $\tau_{max} = 13.14 \mu\text{s}$ für den chordalen Ring mit $N = 6$ bleibt. Grund hierfür ist, dass

5 Simulation und Implementierung

die Länge der Warteschlange durch das synchrone Senden der Knoten nicht größer als $q = 1$ wird und nicht, wie in Satz 3.5 auf $q = 2$ steigen kann.

Modifizierter chordaler Ring mit sieben Knoten

In den vorangegangenen Kapiteln wurde oft auf die positiven Eigenschaften des chordalen Rings eingegangen, ohne ein Gegenbeispiel zu geben. Dies soll nun an dieser Stelle nachgeholt werden. Hierzu wird in dem eben analysierten chordalen Ring ein Knoten durch ein Knotenpaar, wie es in Abbildung 3.6 dargestellt ist, ersetzt. Das resultierende Netz ist in Abbildung 5.5 gezeigt.

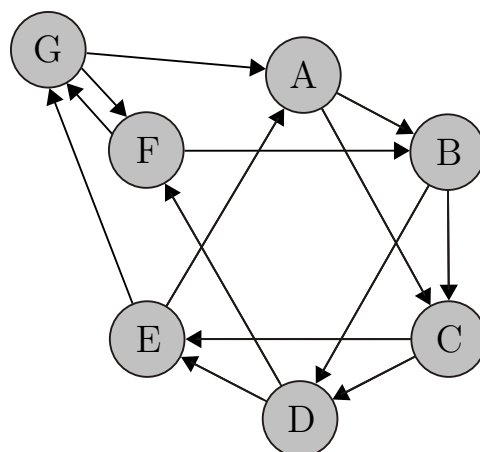
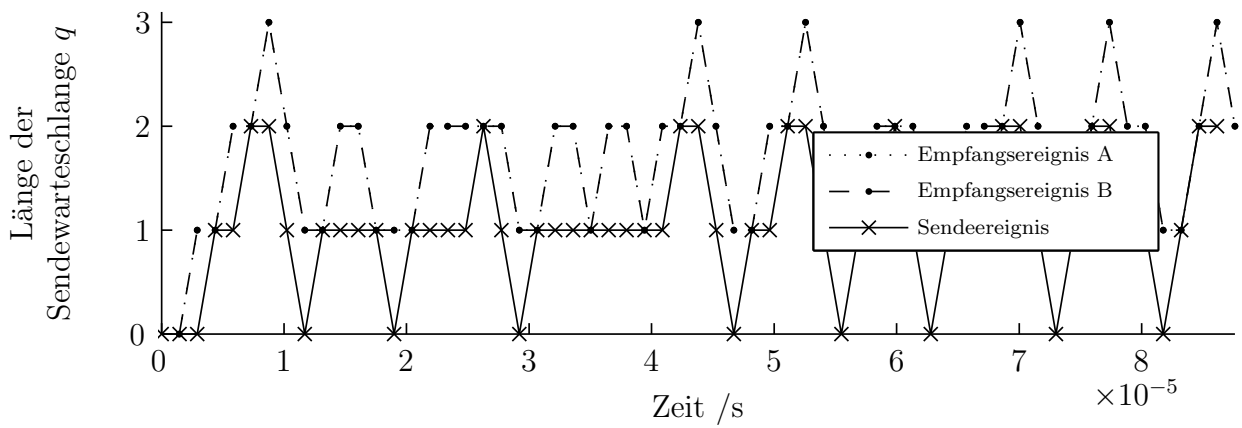


Abbildung 5.5: Chordaler Ring mit einem Knoten-Paar ($N = 7$)

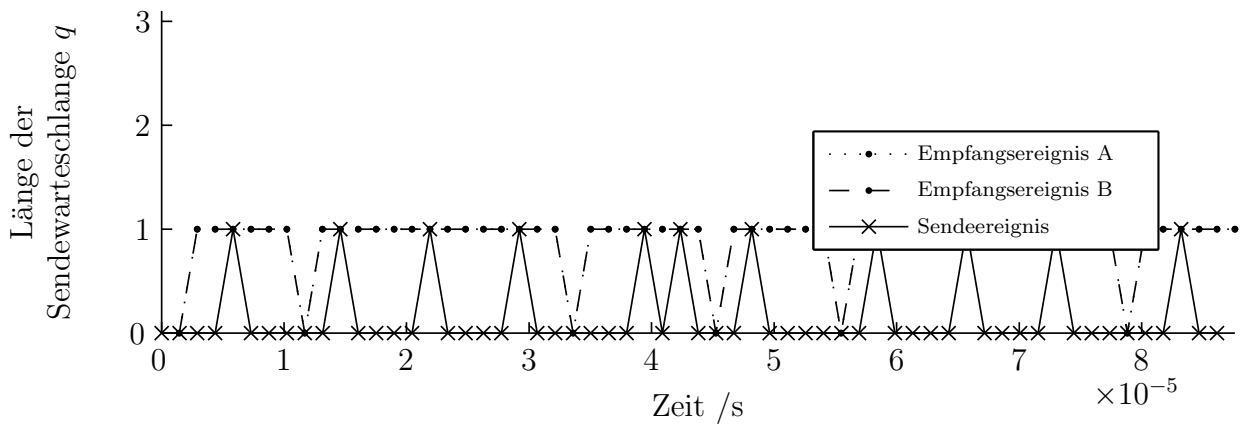
Durch das Ersetzen eines Knotens wird eine Ungleichmäßigkeit in dem Netzwerk geschaffen, die sich topologisch dadurch zeigt, dass zwei Knoten einen sehr kurzen Weg von ihren Ausgängen zurück zu einem Eingang besitzen. Hierdurch ist es diesen Knoten möglich, relativ schnell wieder eigene Nachrichten zu generieren. Daher sind die Voraussetzungen für Satz 3.5 verletzt und die Länge der Sendewarteschlange kann $q = 2$ überschreiten. Eine weitere Folge der Unsymmetrie ist, dass, im Gegensatz zu den in Abbildung 5.1 dargestellten Längen der Sendewarteschlangen in verschiedenen Knoten des chordalen Ringes, sich die gezeigten Längen der Sendewarteschlangen in dem modifizierten chordalen Ring zwischen den Knoten unterscheiden. In Abbildung 5.6 ist eine Auswahl der Längen der Sendewarteschlangen dargestellt.

Die Längen der Sendewarteschlangen in den Knoten B und C sind der in Knoten A ähnlich, während die in Knoten E der in Knoten D und der Verlauf in Knoten F dem in G entspricht.

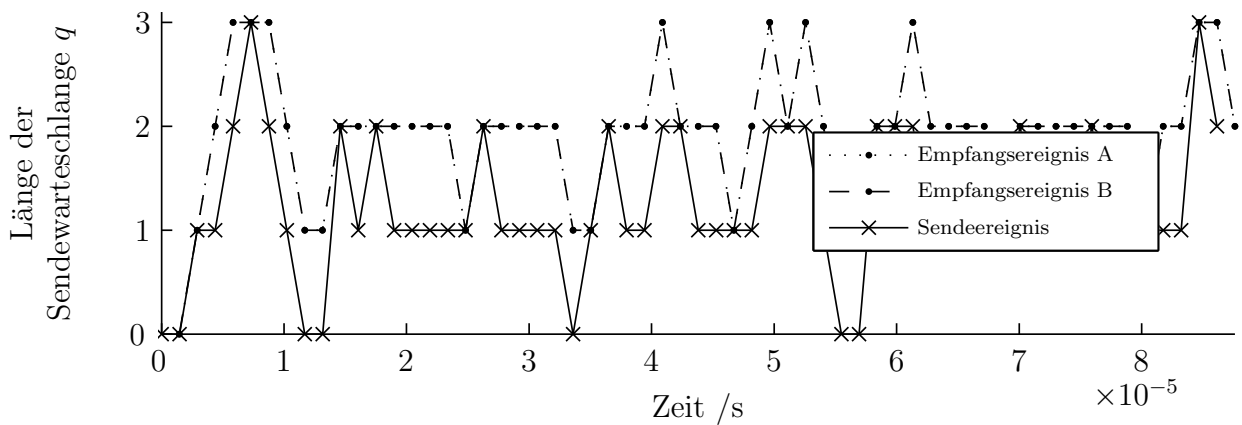
In der Länge der Sendewarteschlange ist insbesondere für Knoten G keine Periodizität zu erkennen. Deutlich sind jedoch die Spitzen der Länge der



(a) Knoten A



(b) Knoten D



(c) Knoten G

Abbildung 5.6: Länge der Sendewarteschlange in verschiedenen Knoten im modifizierten chordalen Ring

5 Simulation und Implementierung

Sendewarteschlange in einigen Knoten. Diese Spitzen schlagen sich auch in der relativen Häufigkeit der Sendewarteschlange zu den Ereigniszeitpunkten nieder wie in den Abbildungen 5.7 bis 5.9 dargestellt ist.

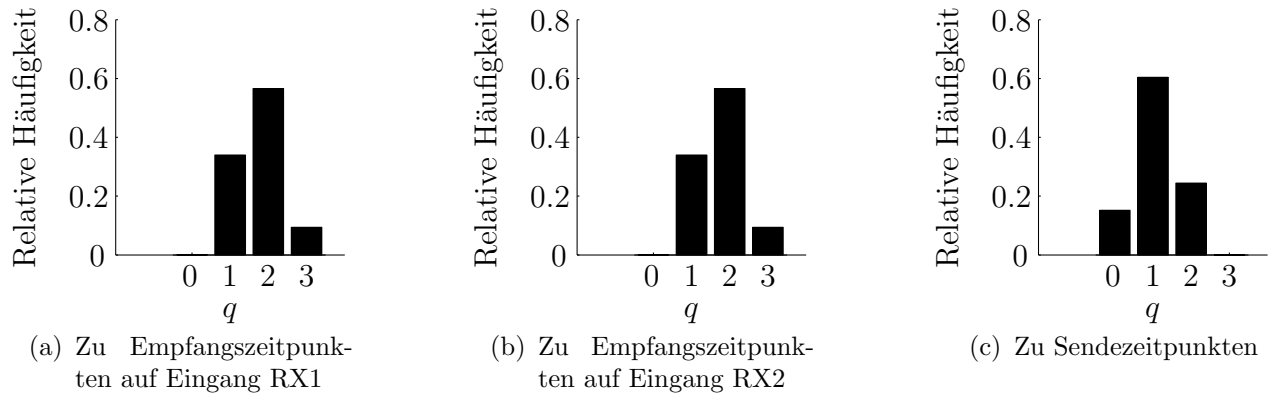


Abbildung 5.7: Relative Häufigkeit der Warteschlangen-Längen q in Knoten A

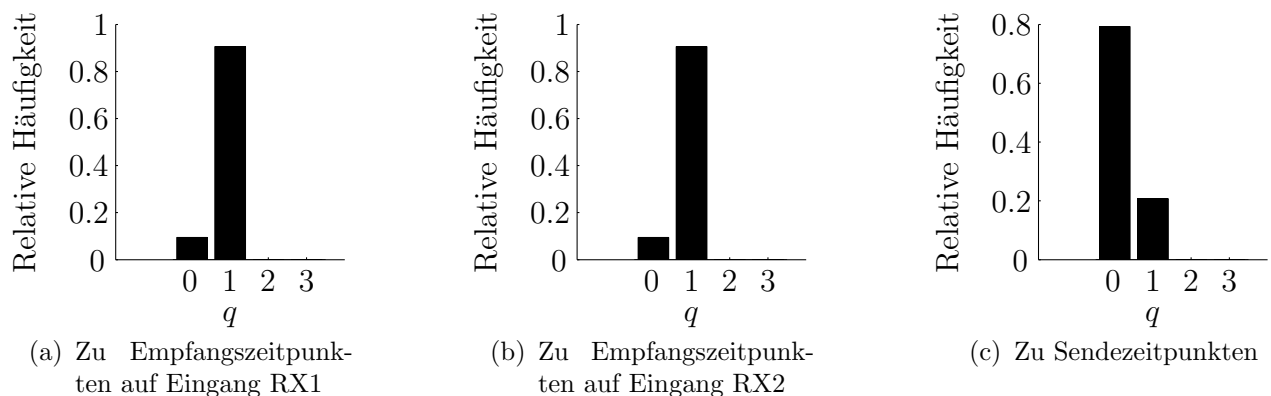


Abbildung 5.8: Relative Häufigkeit der Warteschlangen-Längen q in Knoten D

Auffallend ist hierbei, dass sich die Histogramme der Längen der Sendewarteschlangen in dem Knotenpaar am deutlichsten von denen des chordalen Rings unterscheiden, während in Knoten D zwar eine Abweichung zu erkennen ist, auch in den Sendezeitpunkten tritt beispielsweise der Fall ein, dass sich noch eine Nachricht in der Sendewarteschlange befindet, diese aber in Knoten D deutlich geringer ist. Die durch die Unsymmetrie verursachte Störung klingt also gewissermaßen auf dem Weg der Nachrichten durch das Netz ab.

Die Auswertung der von den Knoten versendeten MIDs ist in Abbildung 5.10 beispielhaft für Knoten A zu sehen. Da alle Knoten ja bekanntlich

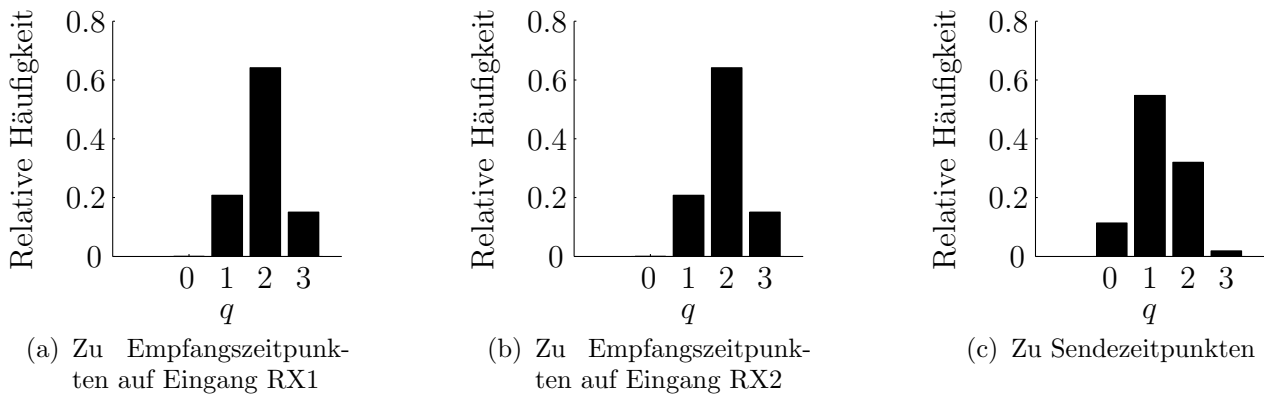


Abbildung 5.9: Relative Häufigkeit der Warteschlangen-Längen q in Knoten G

alle Nachrichten exakt ein Mal versenden müssen, ist die Anzahl der Nachrichten bestimmter MIDs für alle Knoten gleich.

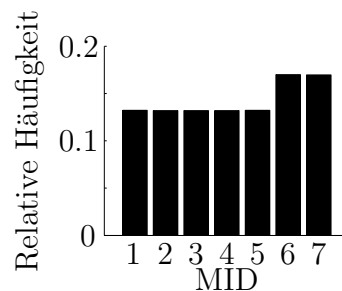


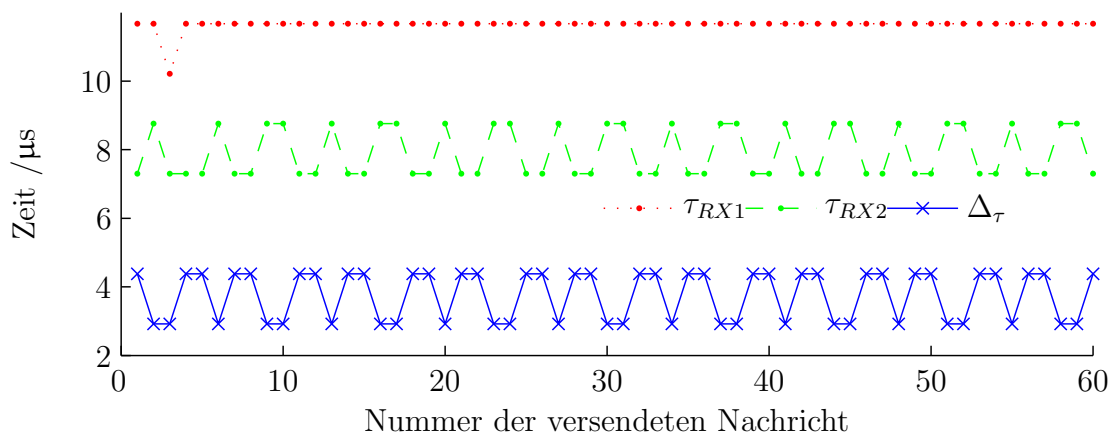
Abbildung 5.10: Relative Häufigkeit des Auftretens aller MIDs in Knoten A

Hier wird der bereits in Abschnitt 3.6 angesprochene Effekt deutlich, dass das Knotenpaar in der Lage ist, deutlich mehr eigene Nachrichten zu versenden als die übrigen Knoten. Hierdurch kann ein topologisches load-balancing erfolgen.

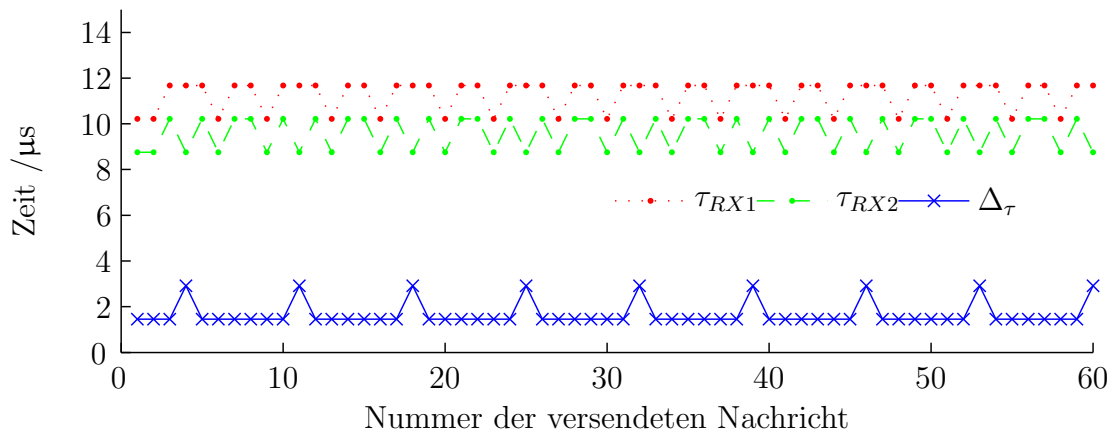
Hervorgerufen wird dieser Effekt durch den relativ kurzen Weg von den Ausgängen des Knotenpaars zu je einem der Eingänge der Knoten. Hierdurch ist es diesen Knoten möglich, relativ schnell wieder eigene Nachrichten zu senden, wie in Abbildung 5.11 dargestellt.

Teilweise kompensiert wird dieser Effekt bei hoher Buslast jedoch durch höhere Längen der Sendewarteschlangen in diesen Knoten, welcher die Sendefrequenz dieser beiden Knoten vergleichsweise schnell absenkt.

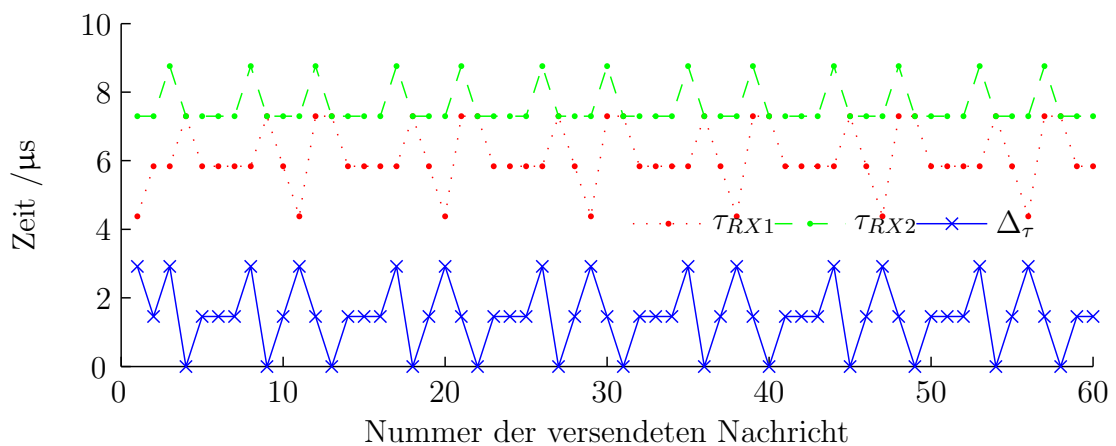
5 Simulation und Implementierung



(a) Knoten A



(b) Knoten D



(c) Knoten G

Abbildung 5.11: Laufzeiten eigener Nachrichten im modifizierten chordalen Ring

5.2 Implementierung

Mit Hilfe des in Unterabschnitt 5.1.1 beschriebenen Modells können reproduzierbare Untersuchungen unter kontrollierten Bedingungen durchgeführt werden. Wie in Unterabschnitt 5.1.2 dargestellt, wird die Ausbreitung von Nachrichten in SafeNet durch das Modell korrekt wiedergegeben, weshalb die Analyse von SafeNet anhand des Modells erfolgen kann.

Bei der Parametrierung des Modells wurden jedoch zwei Annahmen getroffen, die durch eine proof-of-concept Implementierung verifiziert werden müssen:

- die Übertragung mit 100 MBit/s ist möglich
- die Verarbeitungszeit in einem Knoten ist gegenüber der Sendezeit einer Nachricht vernachlässigbar.

Um diesen Nachweis zu erbringen, wurde ein System, wie in Abbildung 3.18 dargestellt, in VHDL implementiert. Als Hardware stand ein Board, dessen Kern ein Cyclone FPGA (*Field Programmable Gate Array*) der Firma Altera bildet, zur Verfügung, siehe auch [Alt07]. Es handelt sich hierbei um ein vergleichsweise günstiges FPGA am unteren Ende des Leistungsspektrums. Die Wahl fiel dabei auf dieses Board, da es am IIT auch für weitere Projekte genutzt wird und so in ausreichender Anzahl zur Verfügung stand.

Bei der Implementierung wurden die nebenläufigen Prozesse in zwei Gruppen unterschieden. Die DECODE und der ENCODE Prozess müssen dabei mit der vollen Geschwindigkeit betrieben werden, um eine Kommunikation zwischen den Knoten zu ermöglichen. Diese Prozesse bestimmen also die maximale Übertragungsgeschwindigkeit.

Der komplexere OC Prozess hingegen ist hiervon entkoppelt und kann mit einer niedrigeren Taktrate betrieben werden. Er begrenzt die Übertragungsgeschwindigkeit nur, wenn die Zeit, die für die Verarbeitung in der OC benötigt wird, in der gleichen Größenordnung wie die Übertragungszeit einer Nachricht liegt. Die Verarbeitungsgeschwindigkeit in der OC ist aber maßgeblich für die in einem Knoten entstehende Verzögerung verantwortlich.

Innerhalb des DECODE Prozesses ist für die hohen Geschwindigkeitsanforderungen besonders der BITSTR Prozess bestimmend, da dieser während der Dauer eines Bits mehrfach aktiviert werden muss. Die Anzahl der Aktivierungen pro Bit ist dabei nicht spezifiziert. Für eine Übertragungsgeschwindigkeit von 100 MBit/s und einer Überabtastung von acht muss der BITSTR Prozess also $8 \cdot 10^8$ Mal pro Sekunde aktiviert werden. Hieraus ergibt sich eine maximale Dauer 1.25 ns, was mit dem Cyclone FPGA nicht zu realisieren

5 Simulation und Implementierung

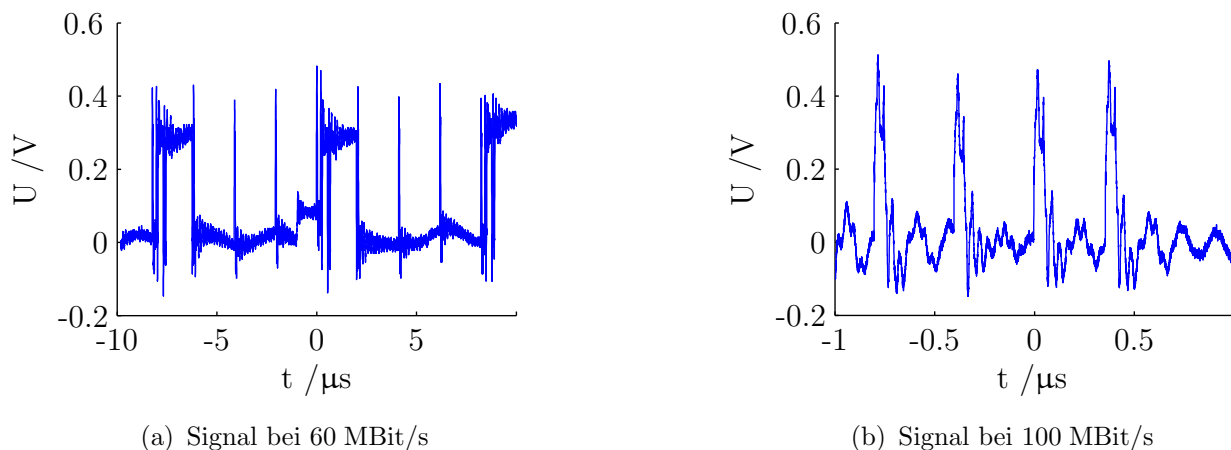


Abbildung 5.12: Ausschnitt aus einer SafeNet Nachricht in dem proof-of-concept System

ist. Eine Möglichkeit zur Erhöhung der Abtastung besteht darin, den BIT-STR Prozess mehrfach parallel zu implementieren und zeitversetzt aufzurufen. Hierbei ist jedoch auf ausreichende Genauigkeit der Aufrufe zu achten.

Der DECODE Prozesses ist auch für die Auswertung des H-CRC sowie des FEC Feldes verantwortlich. Insbesondere die Fehlerkorrektur ist komplex und daher langsamer als die Fehlererkennung, weshalb auf die Implementierung der FEC verzichtet wurde. Die Fehlerkorrektur muss nicht mit der Biterkennung synchron laufen, weshalb durch eine Ausgliederung von Fehlerkorrektur und Fehlererkennung aus dem DECODE Prozess eine weitere Beschleunigung erreicht werden kann. Erfahrungen mit Powerline-Modems bestätigen, dass die Fehlerkorrektur auch für Datenraten > 100 MBit/s in Echtzeit möglich ist [Bab07].

Die Komplexität des OC Prozesses stammt größtenteils aus der Überprüfung, ob eine Nachricht bereits erhalten wurde. Kann dies schnell realisiert werden, so kann im fehlerfreien Fall die Verarbeitungszeit in den Knoten gegenüber der Nachrichtenlaufzeit vernachlässigt werden.

Das proof-of-concept System erreicht auf einem Cyclone FPGA eine maximale Bruttodatenrate von 95 MBit/s, woraus sich der in Abbildung 5.12(b) dargestellte Signalverlauf am FPGA ergibt. Eine Steigerung dieser Datenrate ist durch die Verwendung eines schnelleren FPGAs sowie eine sorgfältigere Implementierung möglich.

Hierbei erreichen sowohl der DECODE Prozess als auch die OC mit einer auf einfache Fehler beschränkten Fehlerbehandlung eine Taktrate von 95 MHz. Hierdurch kann eine Nachricht während einer Bitzeit durch die OC verarbeitet werden. Durch die Vernachlässigung der Verarbeitungszeit in den Knoten bei der Berechnung der worst-case Laufzeit entsteht also ein Feh-

ler von $6.85 \cdot 10^{-3}$. Für eine exakte deterministische Betrachtung ist diese Verzögerung auch zu spezifizieren. An dieser Stelle soll jedoch die Annahme, dass durch die Verarbeitung in den Knoten keine Verzögerung entsteht, als gerechtfertigt angesehen werden.

Deutlich erkennbar ist jedoch die schlechte Signalqualität bei einer Datenrate von 95 MBit/s verglichen mit einer Datenrate von 60 MBit/s. Grund hierfür ist die Tatsache, dass der Cyclone FPGA an seiner Leistungsgrenze betrieben wird. Hier kann die Wahl eines leistungsfähigeren FPGAs Abhilfe schaffen. Es muss zusätzlich durch externe Beschaltung des FPGA sichergestellt werden, dass das Signal auf der Leitung der jeweiligen Spezifikation der Bitübertragungsschicht entspricht.

5.3 Vergleichende Betrachtung

In dem folgenden Abschnitt wird SafeNet mit den in Abschnitt 2.4 vorgestellten Kommunikationsmedien für den Einsatz im Kraftfahrzeug verglichen. Der Fokus des Vergleichs liegt dabei auf CAN und FlexRay sowie auf TTP/C. SAFEbus ist, wie zuvor schon diskutiert, durch seine hohen Kosten für den Einsatz im Kraftfahrzeug nur bedingt geeignet.

Der Vergleich ist auf sicherheitsrelevante Systeme im Kraftfahrzeug ausgelegt, weshalb Systeme mit wenigen Knoten zum Einsatz kommen. Die Kommunikationssysteme werden dabei möglichst vorteilhaft ausgelegt, was bedeutet, dass wo möglich, jedes System so parametrisiert wird, dass es unter Einhaltung der Randbedingungen bestmögliche Ergebnisse liefert.

SafeNet wird in der Version verglichen, die in der Simulation verwendet wird, siehe auch Abschnitt 5.1. Dies bedeutet, dass die Nachrichtenlänge fest 8 Byte ist und das ML Feld lediglich den Teil der Nachricht angibt, dessen Inhalt relevant ist. Des Weiteren wird kein Interleaver eingesetzt.

Verglichen werden drei Teilbereiche, die drei Aspekte des Einsatzes von Kommunikationsmedien abdecken. Zuerst wird dabei auf die Netzwerktopologie eingegangen. Da der Kabelbaum im Kraftfahrzeug nach dem Antriebsstrang die zweitschwerste Komponente ist, ist die Länge des Kabelbaums ein Kostenfaktor sowohl in der Produktion als auch im Betrieb. Der zweite Aspekt in dem Vergleich ist die Datenrate, und mit ihr die Effizienz der einzelnen Systeme und maximale Laufzeit von Nachrichten. Hier wird also auf die Echtzeitfähigkeit der Systeme abgehoben. Der dritte und letzte Aspekt ist die Fehlerbehandlung der einzelnen Kommunikationssysteme.

5.3.1 Netzwerktopologie

Für den Vergleich der Kabellängen werden zwei normierte Topologien eingeführt, zum einen der normierte Kreis und zum anderen die normierte Linientopologie. In der Linientopologie sind die Knoten mit gleichmäßigem Abstand auf einer Linie angeordnet, wobei die beiden äußersten Knoten den Abstand $l_L = 1$ haben. Für den aktive und passive Stern wird angenommen, dass sie sich bei $l = 0.5$, also in der Mitte der Linie befinden. Bei der normierten Kreistopologie ist der Umfang konstant $l_K = 1$ und die Knoten werden, wie bei der Linientopologie auch in gleichmäßigen Abständen auf dem Kreis angeordnet, während Sterne im Zentrum des Kreises liegen. Für alle Strukturen wird die Anzahl der in dem Netzwerk vorhandenen Knoten N variiert.

Mit den normierten Topologien werden drei Strukturen verglichen. Zum einen die Busstruktur, wie sie bei CAN und FlexRay Verwendung findet. Da der Vergleich, wie eingangs erwähnt, auf sicherheitsrelevante Systeme abzielt, wird der Bus redundant ausgelegt. Ebenfalls redundant ausgelegt wird die Sternstruktur, die für FlexRay und TTP/C spezifiziert ist. Für SafeNet wird der chordale Ring betrachtet.

Linientopologie

Bus Die Länge des Busses in der Linientopologie ist trivial

$$l_{L,Bus}(N) = 2 \quad \forall N$$

und damit konstant.

Stern Mit der Annahme, dass die Sternpunkte in der Mitte der Linie liegen, ergibt sich die Länge der Sternstruktur zu

$$\begin{aligned} l_{L,Stern}(N) &= 2 \cdot \sum_{i=0}^{N-1} \left| i \cdot \frac{1}{N-1} - \frac{1}{2} \right| \\ &\leq 2 \cdot N \frac{1}{2} = N. \end{aligned} \tag{5.1}$$

Hierbei ist zu beachten, dass $l_{L,Stern}(N)$ im Unterschied zu $l_{L,Bus}$ eine Funktion von N ist. Die Abschätzung nach oben legt den Schluss nahe, dass $l_{L,Stern}(N)$ für große N nicht gegen einen festen Wert konvergiert.

Beweis 5.1 (Divergenz von $l_{L,Stern}(N)$) Für den Beweis der Divergenz werden in Gleichung 5.1 lediglich die ersten $\lfloor \frac{N}{2} - 1 \rfloor$ Knoten betrachtet. Es wird gezeigt, dass $l_{L,Stern}(N)$ streng monoton steigend ist, also

$$l_{L,Stern}(N_1) > l_{L,Stern}(N) \Leftrightarrow N_1 > N_2 \Leftrightarrow N_1 - N_2 > 0$$

gilt.

$$\begin{aligned} l_{L,Stern}(N) &= 4 \cdot \sum_{i=0}^{\lfloor \frac{N}{2}-1 \rfloor} \left| \frac{i}{N-1} - \frac{1}{2} \right| \\ &= 4 \cdot \sum_{i=0}^{\lfloor \frac{N}{2}-1 \rfloor} \left(\frac{1}{2} - \frac{i}{N-1} \right) \\ &= \frac{4}{N-1} \sum_{i=0}^{\lfloor \frac{N}{2}-1 \rfloor} \left(\frac{N-1}{2} - i \right) \\ &= 2 \left\lfloor \frac{N}{2} - 1 \right\rfloor - \frac{4}{N-1} \sum_{i=0}^{\lfloor \frac{N}{2}-1 \rfloor} i \end{aligned} \quad (5.2)$$

Weiter wird nun gezeigt, dass Gleichung 5.2 streng monoton steigend ist. Zuerst wird hierfür der Fall betrachtet, dass N gerade, $N+1$ also ungerade ist. Es ergeben sich nun

$$l_{L,Stern}(N) = 2 \left(\frac{N}{2} - 1 \right) - \frac{4}{N-1} \sum_{i=0}^{\frac{N}{2}-1} i$$

und

$$l_{L,Stern}(N+1) = 2 \left(\frac{N}{2} - 1 \right) - \frac{4}{N} \sum_{i=0}^{\frac{N}{2}-1} i.$$

Berechnet man nun die Differenz, so ergibt sich

$$l_{L,Stern}(N+1) - l_{L,Stern}(N) = \frac{4}{N-1} \sum_{i=0}^{\frac{N}{2}-1} i - \frac{4}{N} \sum_{i=0}^{\frac{N}{2}-1} i > 0 \quad \forall N, \text{ gerade.}$$

Da diese Differenz für alle N aus dem Definitionsbereich größer null ist, steigt $l_{L,Stern}(N)$ also.

5 Simulation und Implementierung

Betrachtet man nun den Fall, dass N ungerade, $N + 1$ also gerade ist, so ergeben sich

$$l_{L,Stern}(N) = 2 \left(\frac{N-1}{2} - 1 \right) - \frac{4}{N-1} \sum_{i=0}^{\frac{N-1}{2}-1} i$$

und

$$l_{L,Stern}(N+1) = 2 \left(\frac{N+1}{2} - 1 \right) - \frac{4}{N} \sum_{i=0}^{\frac{N+1}{2}-1} i.$$

Betrachtet man nun wieder die Differenz, so erhält man

$$\begin{aligned} l_{L,Stern}(N+1) - l_{L,Stern}(N) &= 2 - \frac{4}{N} \sum_{i=0}^{\frac{N+1}{2}-1} i + \frac{4}{N-1} \sum_{i=0}^{\frac{N-1}{2}-1} i \\ &= 2 + 4 \left(- \sum_{i=0}^{\frac{N-1}{2}-1} \frac{i}{N} - \frac{\frac{N+1}{2} - 1}{N} + \sum_{i=0}^{\frac{N-1}{2}-1} \frac{i}{N-1} \right) \\ &= 2 \underbrace{\left(1 - \frac{N-1}{N} \right)}_{>0} + 4 \sum_{i=0}^{\frac{N-1}{2}-1} \underbrace{\left(\frac{i}{N-1} - \frac{i}{N} \right)}_{>0}. \end{aligned} \quad (5.3)$$

Da alle Summanden in Gleichung 5.3 für $N \geq 2$ größer null sind, ist $l_{L,Stern}(N)$ also streng monoton steigend.

Chordaler Ring Im chordalen Ring wird angenommen, dass die Knoten auf der Linie aufsteigend nummeriert mit $i \in [0, N-1]$ sind und der Knoten i an die Knoten $(i+1 \bmod N)$ und $(i+2 \bmod N)$ sendet, also in deren upstream Menge enthalten ist. Unter dieser Annahme ergibt sich

$$\begin{aligned} l_{L,CR}(N) &= \sum_{i=0}^{N-3} \left(\frac{1}{N-1} + \frac{2}{N-1} \right) + \frac{1}{N-1} + 1 \\ &\quad - \frac{1}{N-1} + 1 + 1 - \frac{1}{N-1} \\ &= \frac{6N-10}{N-1} = 6 - \frac{4}{N-1}. \end{aligned}$$

Für große N ergibt sich $\lim_{N \rightarrow \infty} l_{L,CR}(N) = 6$. Die Länge des chordalen Ringes in der Linientopologie konvergiert also gegen die dreifache Länge, die für eine redundante Busleitung benötigt wird.

Die Längen der einzelnen Strukturen in einer Linientopologie sind in Abbildung 5.13 aufgetragen.

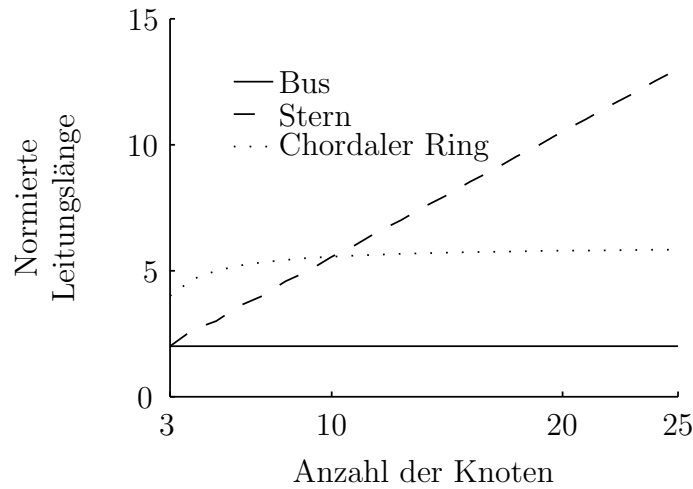


Abbildung 5.13: Länge der verschiedenen Strukturen in der normierten Linientopologie

Kreistopologie

Bus Werden die Knoten auf einem Kreis angeordnet, so kann der Bus auf dem Kreis verlegt werden. Dabei kann immer das letzte Teilstück frei bleiben, da alle Knoten bereits verbunden sind. Die Länge der benötigten Leitung ergibt sich also zu $l_{K,Bus}(N) = 2 \cdot \left(1 - \frac{1}{N}\right)$. Für große N konvergiert dies $\lim_{N \rightarrow \infty} l_{K,Bus}(N) = 2$.

Stern Mit dem Sternpunkt in der Mitte des Kreises lässt sich $l_{K,Stern}$ leicht zu $l_{K,Stern} = \frac{N}{\pi}$ berechnen. Es ist ersichtlich, dass dies für große N divergiert.

Chordaler Ring Die Länge der Leitungen im chordalen Ring kann als

$$l_{K,CR}(N) = \frac{N}{\pi} \cdot \left(\sin\left(\frac{\pi}{N}\right) + \sin\left(\frac{2\pi}{N}\right) \right)$$

berechnet werden. Die Taylor-Reihe dritter Ordnung ergibt sich zu

$$\begin{aligned} l_{K,CR}(N) &\approx \frac{N}{\pi} \left(\frac{\pi}{N} - \frac{1}{3!} \frac{\pi^3}{N^3} + \frac{2\pi}{N} - \frac{1}{3!} \frac{8\pi^3}{N^3} \right) \\ &= 3 - \frac{9\pi^2}{3! \cdot N^2}. \end{aligned}$$

5 Simulation und Implementierung

Daher gilt

$$\lim_{N \rightarrow \infty} l_{K,CR}(N) = \lim_{N \rightarrow \infty} 3 - \frac{9\pi^2}{3! \cdot N^2} = 3.$$

Abbildung 5.14 zeigt die verschiedenen Strukturen im Vergleich.

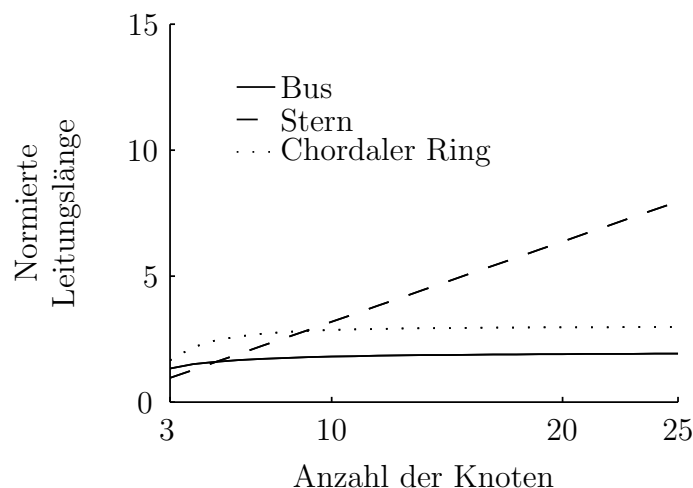


Abbildung 5.14: Länge der verschiedenen Strukturen in der normierten Kreistopologie

Vergleich der Topologien

Durch die Analyse der Leitungslänge zeigt sich, dass die Verwendung eines Busses zur Verbindung der einzelnen Knoten in beiden Fällen, sowohl bei der Kreis- als auch der Linientopologie, für eine realistische Anzahl von Knoten unter dem Kostenaspekt die optimale Lösung bietet. Gegen die Verwendung der Bustopologie spricht, dass ein fehlerhafter Knoten das gesamte Netzwerk inklusive des redundanten Kanals blockieren kann, weswegen für TTP/C beispielsweise die Sterntopologie zum Einsatz in sicherheitsrelevanten Anwendungen empfohlen wird, siehe auch Unterabschnitt 2.4.3.

Stattdessen wird in der Spezifikation [TTT03] die Sternarchitektur empfohlen. Diese löst zwar das Problem des möglichen SPOF, bringt aber, wie hier gezeigt wurde, deutlich mehr Aufwand bei der Verkabelung mit sich, die Kosten für zwei Sternpunkte noch nicht eingerechnet. Besonders schwerwiegend ist, dass die nötige Leitungslänge weder in der Kreis- noch in der Linientopologie konvergiert. Dass dies keine rein theoretische Betrachtung ist, zeigt sich daran, dass bereits für realistische Größenordnungen $N < 10$ die Sternarchitektur beiden anderen betrachteten Strukturen unterlegen ist.

Der chordale Ring verursacht hingegen in beiden hier betrachteten Fällen im Vergleich zu der Busstruktur lediglich um einen Faktor 1.5 – 3 höhere Kosten bei der Verkabelung, löst dabei aber das Problem des bei Verwendung eines Busses auftretenden SPOF. Kritisch zu betrachten ist jedoch die bei dem chordalen Ring, wie bei dem Einsatz von SafeNet allgemein, die höhere Anzahl von benötigten Steckverbindungen. Benötigen sowohl Bus als auch Stern jeweils zwei Steckverbindungen pro Knoten, so erhöht die Verwendung von unidirektionalen Verbindungen in SafeNet diese Zahl auf vier pro Knoten. Die Anzahl der Steckverbindungen ist nicht nur aus Kostensicht relevant, Steckverbindungen sind auch häufige Fehlerursachen. Daher ist es wichtig, die Steckverbindungen für SafeNet so zu entwerfen, dass die erhöhte Anzahl von Steckverbindungen nicht in einer höheren Ausfallrate resultiert.

5.3.2 Datenrate

Wie bereits zu Beginn dieses Abschnittes angedeutet, ist ein wichtiger Faktor in der Bewertung von Kommunikationssystemen im Kraftfahrzeug ihre Datenrate sowie ihre Echtzeitfähigkeit. Mit SafeNet verglichen werden hier zum einen die TDMA-basierten Systeme TTP und FlexRay sowie das CSMA basierte CAN. In allen Fällen wird die Effizienz η des Kommunikationssystems bewertet.

Die Echtzeitfähigkeit der Systeme kann für die TDMA Systeme als gegeben angenommen werden und wurde für SafeNet bereits in Abschnitt 3.6 gezeigt. Sie ist daher nur noch für CAN zu untersuchen.

CAN

Betrachtet man den Arbitrierungsmechanismus von CAN, so fällt auf, dass hier ein Kompromiss zwischen schneller Auslieferung einiger weniger sicherheitsrelevanter Nachrichten und der guten Ausnutzung der knappen Ressource Buskapazität geschlossen wurde.

Besonders für geringe Buslast ist dies auch gut möglich. In [EPZ02] wird $\rho \leq 0.2$ als Richtwert angeführt. Für diese geringe Buslast ist das Verhalten des CAN Busses quasi-deterministisch. Dies bedeutet, dass auch Nachrichten geringer Priorität kaum auf ihre Auslieferung warten müssen. Die maximale Wartezeit von Nachrichten verschiedener Priorität ist in Abbildung 5.15 dargestellt.

Die Daten für die gezeigten Diagramme stammen aus einem im Rahmen dieser Arbeit erstellten Modells, welches den CAN Bus bitgenau simuliert. Sie stellen also lediglich Messwerte und im Allgemeinen keine worst-case Werte

5 Simulation und Implementierung

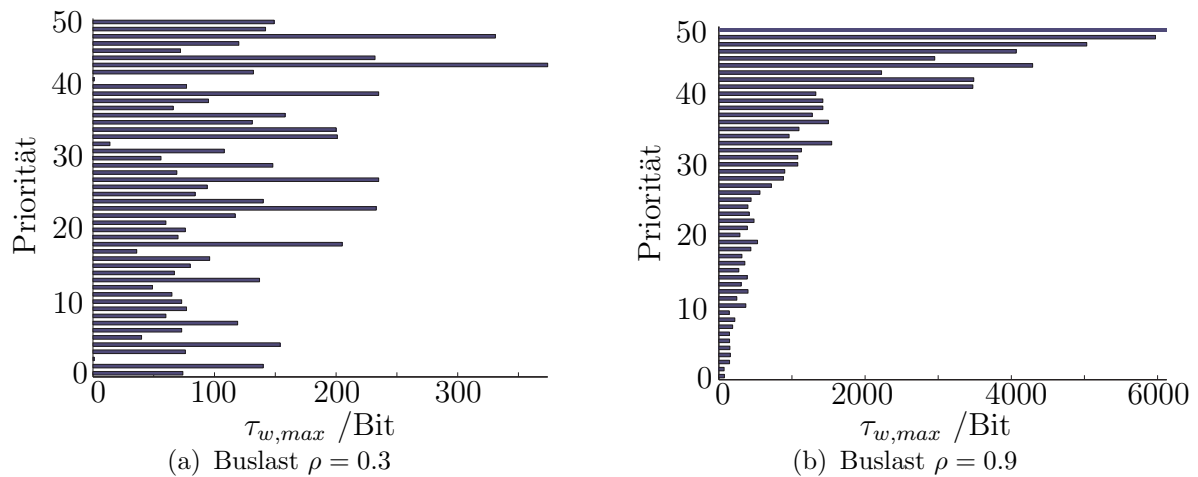


Abbildung 5.15: Maximale Wartezeiten $\tau_{w,max}$ von Nachrichten verschiedener Prioritäten in CAN bei niedriger und hoher Buslast

dar. Für die Simulation wurden 50 Nachrichten verschiedener Priorität verwendet, die jeweils 79 Bit lang sind. Dies erklärt die maximale Wartezeit der Nachricht mit der höchsten Priorität, welche kleiner oder gleich 79 Bit ist.

Es lässt sich deutlich erkennen, dass für den Fall geringer Buslast, $\rho = 0.3$, dargestellt in Abbildung 5.15(a), die maximale Wartezeit der Nachrichten für alle Prioritäten ähnlich ist. Dies gilt besonders, wenn man berücksichtigt, dass eine Nachricht in CAN im fehlerfreien Fall nicht unterbrochen werden kann, also schon ohne Priorisierung maximal 79 Bit warten muss. Es ist zwar ein leichter Trend zu erkennen, dass niederpriorie Nachrichten längere Zeit in der Warteschlange verbringen, die maximale Wartezeit liegt aber deutlich unter den 3950 Bit, die für eine Runde in einem TDMA System benötigt würde¹. Hier ist also die Aussage, der CAN Bus würde sich quasi-deterministisch verhalten, gerechtfertigt.

Anders stellt sich der Fall für eine hohe Buslast $\rho = 0.9$, wie in Abbildung 5.15(b) gezeigt, dar. Hier liegt die maximale Wartezeit für einen nicht zu vernachlässigenden Teil der Nachrichten in der Größenordnung der TDMA Runde oder auch deutlich darüber. Problematisch ist hier nicht der absolute Wert der maximalen Wartezeit, sondern die Tatsache, dass aufgrund der CAN zugrunde liegenden Arbitrierung keine worst-case Wartezeit angegeben werden kann. An dieser Stelle sei auch auf die Analyse der Warteschlangen in Abschnitt 3.5 verwiesen. In CAN existiert auf Netzwerkebene keine Begrenzung der Sendefrequenz der einzelnen Knoten, wodurch die Wartezeit der Nachrichten nicht begrenzt ist. Die mittlere Wartezeit geht für $\rho \rightarrow 1$ gegen

¹Details über den Frameaufbau in FlexRay werden an dieser Stelle zugunsten einer einfachen Abschätzung vernachlässigt.

unendlich, wodurch das System instabil wird.

Ein weiterer Effekt der fehlenden Begrenzung der Sendefrequenz ist, dass die Buslast in einem CAN Bus nicht begrenzt werden kann. Der babbling-idiot Fehler ist in CAN streng genommen also kein Fehler sondern eine eingeplante Eigenschaft, die es Nachrichten mit hoher Priorität erlaubt, unwichtige Nachrichten mit niedriger Priorität vom Bus zu verdrängen. Hierdurch wird die maximale Wartezeit der wichtigen Nachrichten auf Kosten der Nachrichten mit niedriger Priorität verkürzt. Es kann also nicht sichergestellt werden, dass sich ein Bus im Zustand $\rho \leq 0.3$ befindet. Hier zeigt sich deutlich die Auslegung von CAN, die sicherheitsrelevante und nicht sicherheitsrelevante Nachrichten auf einem Bus zulässt. Werden jedoch lediglich sicherheitsrelevante Daten über einen Bus versendet, so ist die Bevorzugung einiger Nachrichten auf Kosten anderer kritisch zu betrachten.

Es bleibt also festzustellen, dass CAN zwar für den Fall geringer Buslast quasi-deterministisch arbeitet, diese Eigenschaft aber durch den CAN Bus alleine nicht gesichert werden kann. CAN ist daher ohne die Verwendung weiterer Mechanismen für sicherheitsrelevante Systeme nur bedingt geeignet. Dennoch soll im Folgenden die Effizienz von CAN untersucht werden.

Die maximale Länge einer CAN Nachricht in Bit bei einem Datenfeld von k Byte Länge lässt sich durch

$$n = 8k + 47 + \left\lfloor \frac{33 + 8k}{4} \right\rfloor$$

angeben, siehe auch [Ram06]. Müssen Daten länger als 8 Byte versendet werden, so müssen sie auf mehrere Nachrichten verteilt werden. Da die Länge der Nachricht nur in Byte-Schritten verändert werden kann, ergibt sich die Effizienz in Abhängigkeit der zu versendenden k Byte zu

$$\eta = \frac{8k}{8k + 47 + \left\lfloor \frac{33+8k}{4} \right\rfloor}. \quad (5.4)$$

Da die Effizienz von CAN im fehlerfreien Fall von keinen anderen Faktoren abhängt, ist η in Abbildung 5.16 lediglich über der Länge der Nachricht aufgetragen. Die Nettodatenrate ergibt sich durch die Multiplikation mit der Busgeschwindigkeit.

FlexRay

Im Gegensatz zu CAN sind bei FlexRay die sicherheitsrelevanten Daten im statischen Teil sauber von den nicht sicherheitsrelevanten im dynamischen Teil getrennt. Aus diesem Grund lässt sich für FlexRay im fehlerfreien Fall

5 Simulation und Implementierung

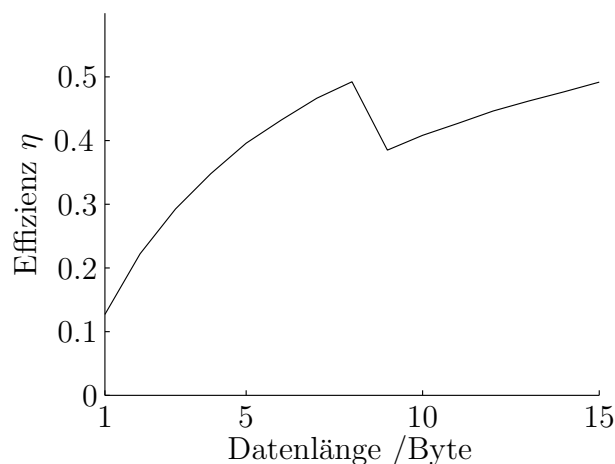


Abbildung 5.16: Effizienz von CAN

auch eine worst-case Laufzeit angeben, welche die Länge einer Runde beträgt. Da das dynamische Segment aber mit einer festen Länge in die Overheadberechnung eingeht, beeinflusst die Länge des dynamischen Teils auch die worst-case Laufzeit. Für die folgenden Betrachtungen wird auf einen dynamischen Teil verzichtet und lediglich der statische Teil der sicherheitsrelevanten Nachrichten betrachtet.

FlexRay lässt im Vergleich zu CAN sehr viel Spielraum zum Beispiel bei der Parametrierung der Intervalle zwischen den Frames. Daher ist die Berechnung der Effizienz von sehr vielen Faktoren abhängig, siehe auch [Fle05b]. Wie schon zuvor erwähnt, werden für die Berechnung der Effizienz realistische, also zulässige, aber vorteilhafte Parameter angenommen. Für die genaue Berechnung der Länge eines Frames und damit der Effizienz sei hier auf [Fle05b] verwiesen. Der Overhead entsteht dabei durch den Header sowie den mit 24 Bit relativ langen CRC.

Durch die Tatsache, dass innerhalb des statischen Teils alle Frames die gleiche Länge besitzen, ist anzunehmen, dass trotz sorgfältiger Planung im Voraus nicht die gesamte zur Verfügung stehende Bandbreite genutzt werden kann. Dieser Tatsache wird durch den Auslastungsfaktor A Rechnung getragen. Die Effizienz ist für zwei Auslastungsfaktoren in Abbildung 5.17(a) dargestellt. Die auffällige Oszillation entsteht durch die Tatsache, dass die Länge eines FlexRay Frames nur in zwei-Byte Schritten variiert werden kann.

Zur Berechnung der Nettodatenrate müssen zusätzlich andere Parameter mit einbezogen werden, die auch die Zeiten zwischen der einzelnen Frames sowie Zeiten zur Uhrensynchronisation regeln. Werden diese mit den jeweils optimalen Werten einbezogen, so ergibt sich die in Abbildung 5.17(b) dargestellte Nettodatenrate. Es ist anzumerken, dass in realen Systemen mit einer geringeren Nettodatenrate zu rechnen ist, besonders falls die Parametrierung

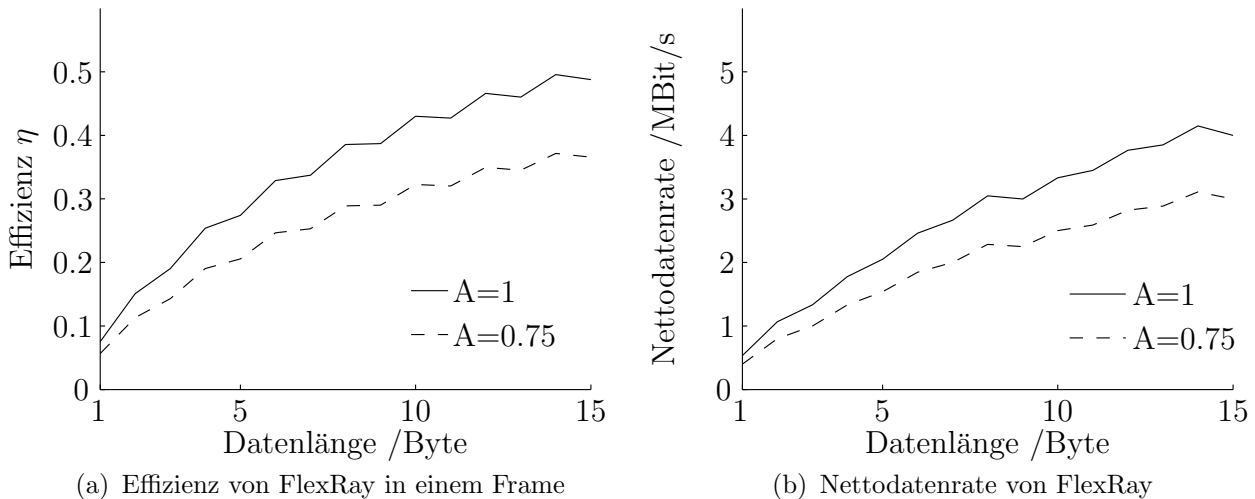


Abbildung 5.17: Effizienz und Nettodatenrate von FlexRay

auf sicherheitsrelevante Systeme ausgelegt wurde.

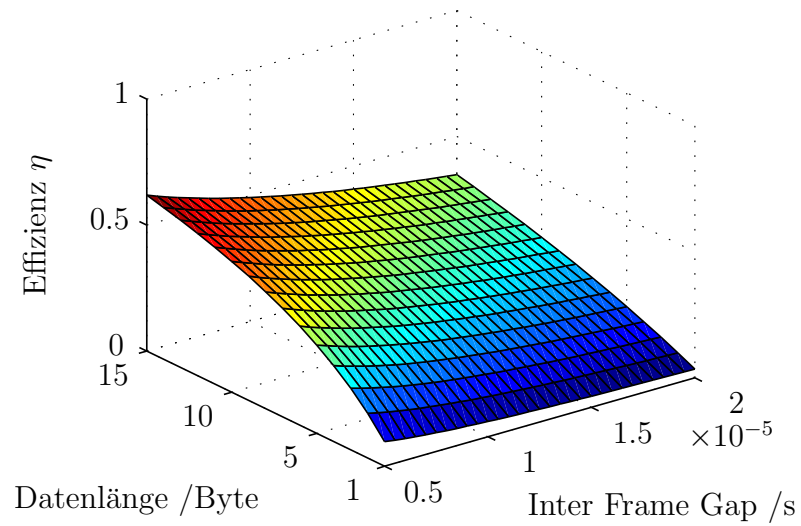
TTP/C

Da die frei erhältliche Spezifikation von TTP [TTT03] wie bereits in Abschnitt 2.4 erwähnt, den Frameaufbau nicht detailliert beschreibt, wird hier lediglich der beschriebene Overhead zusammen mit der IFG der Analyse zugrunde gelegt. Da die Länge der Slots in TTP für unterschiedliche Knoten verschieden gewählt werden kann, wurde hier auf einen Auslastungsfaktor, wie er bei FlexRay verwendet wurde, verzichtet.

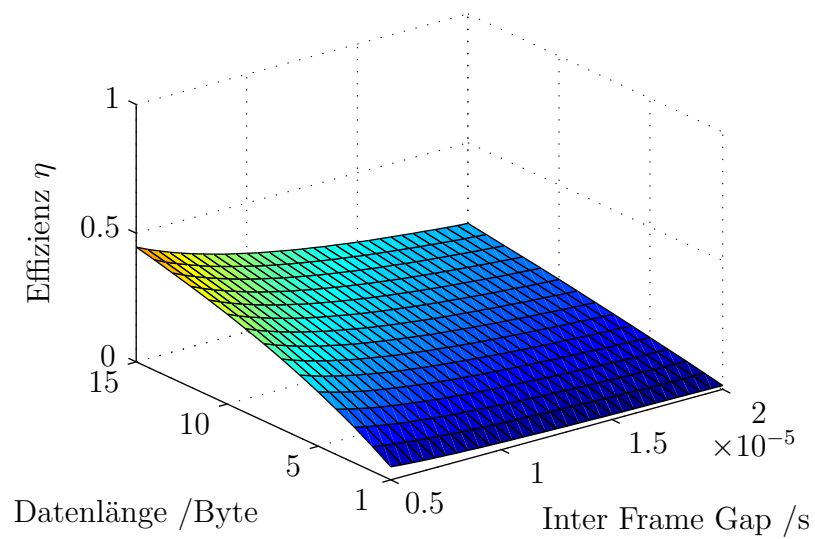
Die Effizienz von TTP ist in Abbildung 5.18 in Abhängigkeit von der IFG und der Datenlänge für in realistischen Szenarien häufig zu erwartende Nachrichtenlängen dargestellt. Dabei ist zu erkennen, dass sich sowohl die IFG als auch der durch CRC und das Protokoll bedingte Overhead negativ auf die Effizienz auswirken. Der in [Kop01] angeführte Maximalwert von $\eta = 0.958$ ist ein eher theoretischer Grenzwert. Hinzu kommt, dass mindestens ein Mal in zwei Rounds ein Frame mit explizitem Zustand übertragen werden muss, was in Abbildung 5.18 nicht berücksichtigt ist.

In Abbildung 5.18(a) ist die Effizienz für eine Geschwindigkeit von 10 MBit/s aufgetragen, um einen Vergleich mit FlexRay zu ermöglichen. Wird die maximale Geschwindigkeit von 25 MBit/s, wie sie laut [Kop01] in Planung ist, verwendet, steigt der Einfluss der IFG und die Effizienz nimmt weiter ab. Dies wird bei einem Vergleich mit Abbildung 5.18(b) deutlich. Es zeigt sich hier, dass die IFG den Einfluss der Erhöhung der Geschwindigkeit auf die Nettodatenrate deutlich senkt. Betrachtet man beispielsweise Nach-

5 Simulation und Implementierung



(a) Effizienz von TTP/C bei 10 MBit/s



(b) Effizienz von TTP/C bei 25 MBit/s

Abbildung 5.18: Effizienz von TTP/C bei verschiedenen Geschwindigkeiten

richten mit 8 Byte Nutzdatenlänge und einer IFG von 5 μs , so beträgt die Nettodatenrate bei einer Bruttodatenrate von 10 MBit/s circa 4.64 MBit/s. Erhöht man nun die Bruttodatenrate auf 25 MBit/s, so steigt die Nettodatenrate auf 6 MBit/s, was statt der zu erwartenden Verdopplung lediglich einer Verbesserung vom ungefähr 50 % entspricht. Mit steigender IFG wird dieser Effekt deutlicher.

SafeNet

Die Effizienz von SafeNet kann, dank der festen Länge für Nachrichten, deren Daten kürzer als $k \leq 8$ Byte sind, einfach als

$$n = \frac{8k}{146}$$

berechnet werden. Überschreitet die Länge der zu versendenden Daten 8 Byte, so müssen die Daten auf weitere Nachrichten verteilt werden.

Da wie bei CAN keine weiteren Faktoren die Effizienz beeinflussen, ist diese in Abbildung 5.19 über der Länge der zu versendenden Daten aufgetragen.

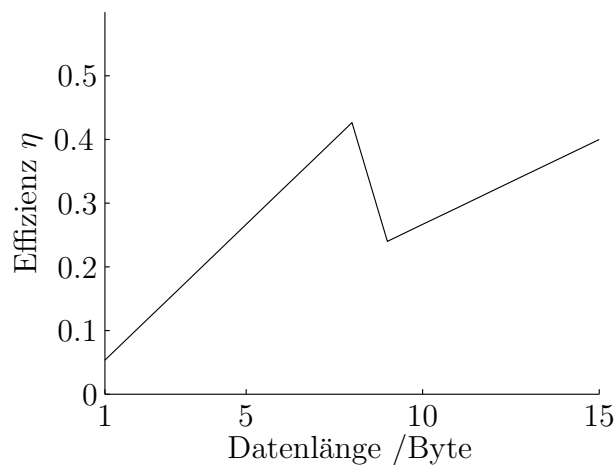


Abbildung 5.19: Effizienz von SafeNet

Da auch hier zwischen den Nachrichten über die 146 Byte hinaus keine Zeit eingeplant werden muss, ergibt sich die Nettodatenrate aus der Multiplikation der Effizienz mit der Bruttodatenrate. Im Unterschied zu CAN muss hierzu nicht die Topologie oder die Leitungslänge des Netzwerkes berücksichtigt werden.

Die Nettodatenrate der behandelten Systeme ist in Abbildung 5.20 vergleichend dargestellt. Die Ordinate ist dabei der Übersichtlichkeit halber logarithmisch aufgetragen. CAN ist hier für eine Bitübertragungsschicht, die

5 Simulation und Implementierung

mit 1 MBit/s arbeitet, dargestellt. FlexRay und TTP/C sind mit 10 MBit/s dargestellt, da diese Geschwindigkeit für FlexRay mit aktiven Sternen bereits erreicht wird [ODKC06] und 25 MBit/s für TTP/C lediglich in Planung ist [Kop01]. Für SafeNet wurden 100 MBit/s angenommen.

Aus dieser Graphik lässt sich erkennen, dass die Effizienz bei den betrachteten Systeme ähnlich ist und die Nettodatenrate hier daher hauptsächlich von der Bruttodatenrate abhängt. Grund dafür ist, dass, wie bereits mehrfach erwähnt, die Systeme in ihren optimalen Arbeitspunkten betrachtet werden. So ist beispielsweise für CAN ein 500 kBit/s Bus mit einer Buslast von $\rho \leq 0.7$ üblich. Werden sicherheitsrelevante Systeme betrachtet, sinkt ρ wie zuvor diskutiert, weiter. Die Nettodatenrate von CAN für sicherheitsrelevante Systeme wird hier also ungefähr um einen Faktor 3–6 überschätzt. Auch für FlexRay ist die Abschätzung optimistisch, da alle Werte, die Einfluss auf die Nettodatenrate haben, auf den optimalen Wert gesetzt wurden. Es ist also damit zu rechnen, dass in realen Systemen, die zum Beispiel den dynamischen Slot zur Fehlerbehandlung verwenden und für die die Puffer zwischen den einzelnen Frames zugunsten der Sicherheit länger gewählt wurden, die Nettodatenrate deutlich geringer ausfällt. Für TTP/C stehen nur wenige Daten zur Analyse zur Verfügung, weshalb eine detaillierte Aussage über die Nettodatenrate nicht möglich ist. Für SafeNet hängt die Nettodatenrate lediglich von der implementierten Bitübertragungsschicht ab. Wird hier eine Geschwindigkeit kleiner als 100 MBit/s gewählt, so sinkt die Nettodatenrate proportional.

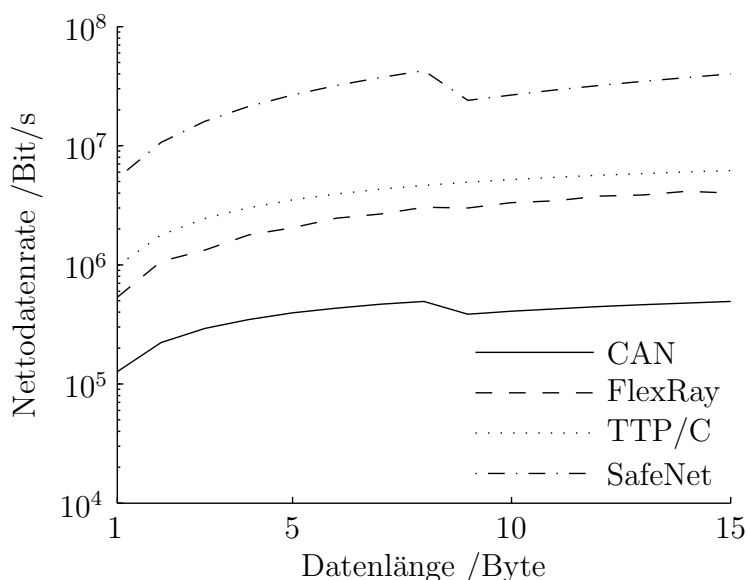


Abbildung 5.20: Nettodatenrate der betrachteten Systeme

Die absoluten Zeiten, die zur Auslieferung der Nachrichten benötigt werden, gerechnet ab Beginn der Nachricht auf dem Medium, hängen in CAN,

TTP und FlexRay lediglich von der Nettodatenrate ab. Da die Nachrichten in SafeNet jedoch durch das Netzwerk propagieren müssen, ist τ_{max} von dem kürzesten Weg im Netzwerk und damit von der Topologie abhängig, siehe auch Gleichung 3.28. Interessant ist der Vergleich von τ_{max} mit der Zykluszeit der TDMA Systeme, da sich hieraus Rückschlüsse auf die Echtzeitfähigkeit des Kommunikationssystems ziehen lassen.

Dieser Vergleich ist in Abbildung 5.21 zu sehen. Da diese Zeit für CAN nicht angegeben werden kann und für TTP keine genauen Daten vorliegen, werden lediglich FlexRay und SafeNet miteinander verglichen. Hierzu wird angenommen, dass jeder der im Netz vorhandenen Knoten eine Nachricht der Nutzdatenlänge 8 Byte versendet.

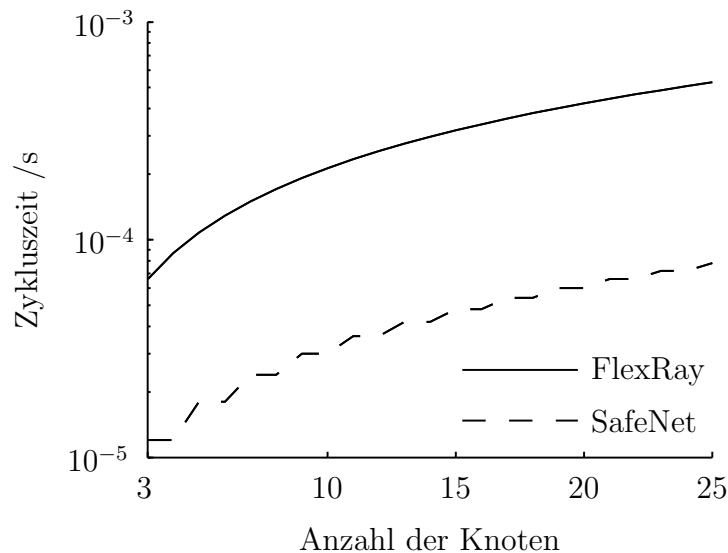


Abbildung 5.21: Zykluszeit von FlexRay und SafeNet im Vergleich

Es ist zu erkennen, dass sowohl FlexRay als auch SafeNet deutlich unter einer Zykluszeit von 1 ms bleiben, was den heute an Echtzeitsysteme gestellten Forderungen genügen sollte. Steigt die Nutzdatenlänge der einzelnen Nachrichten, so profitiert FlexRay im Vergleich zu SafeNet. Es ist hier jedoch anzumerken, dass sich die Zykluszeit bei 128 Byte Nutzdaten pro Nachricht für FlexRay schon für wenige Knoten (< 10) im Millisekundenbereich bewegt und daher die Echtzeitfähigkeit des Systems in Frage gestellt wird.

5.3.3 Fehlerbehandlung

CAN

Die Behandlung von Fehlern ist in CAN auf syntaktische Fehler beschränkt. Ein babbling idiot Fehler existiert beispielsweise in CAN per Definition nicht, da dauerhaftes Senden einer hochpriorigen Nachricht in CAN eine legale Aktion ist. Auch byzantinische Fehler werden in CAN nicht erkannt, da in CAN keine doppelte Auslegung der physikalischen Schicht vorgesehen ist. Eine Rückmeldung über falsch empfangene Nachrichten wird nur im Fall syntaktischen Fehlern mit einem Fehlerframe gegeben. Der Fall, dass zwei Geräte zwei syntaktisch korrekte aber unterschiedliche Versionen einer Nachricht erhalten, wird nicht berücksichtigt. Die Cliquenbildung wird also nur für den Fall, dass eine korrekte Nachricht durch eine Fehlerursache auf eine syntaktisch inkorrekte Nachricht abgebildet wird, verhindert.

Die Fehleraufdeckungszeiten sind in CAN relativ gering. Fehler, die die Bit-stuffing Regel verletzen, werden bereits nach sechs Bit erkannt. Andere Fehler werden am Ende der Nachricht, also nach 136 Bit, maximale Länge einer Nachricht plus Fehlerframe, erkannt. Durch den aktiven Fehlerframe wird der fehlerhafte Empfang allen anderen Knoten mitgeteilt.

Als Reaktion auf erkannte Fehler unterscheidet CAN zwischen drei Zustände in den Knoten, zwischen denen anhand von zwei Zählern (RX_Cnt und TX_Cnt) unterschieden wird.

fehleraktive Knoten (*error active*) Nach Reset und Konfiguration befindet sich ein Knoten im fehleraktiven Zustand und kann die Nachricht, die momentan gesendet wird, durch eine Fehlernachricht aktiv zerstören und damit alle anderen Knoten am Empfang dieser Nachricht hindern. Da ein fehlerhafter Knoten so dauerhaft den gesamten Bus lahmlegen könnte, wird gleichzeitig der RX_Cnt Zähler erhöht. Dieser Zähler ist ein Anhaltspunkt für die mittlere Fehlerrate der von einem Knoten empfangenen Nachrichten. Eine detaillierte Beschreibung der Funktionsweise der Zähler findet sich zum Beispiel in [Ets94]. Analog existiert ein Zähler für Fehler beim Senden (TX_Cnt). Übersteigt einer dieser Zähler einen festgelegten Wert, so wird der Knoten fehlerpassiv.

fehlerpassive Knoten (*error passive*) Knoten, die durch zu viele fehlerhafte Übertragungen vom Zustand fehleraktiv in den Zustand fehlerpassiv übergegangen sind, dürfen die Übertragung nicht mehr aktiv stören und senden im Fehlerfall eine Folge von sechs rezessiven Bit. Somit bestätigen sie den korrekten Empfang der Nachrichten nicht. Fallen beide Zähler wieder unter festgelegte Grenzwerte, so kann ein fehlerpassiver Knoten wieder fehleraktiv werden. Die Grenzwerte, die wieder

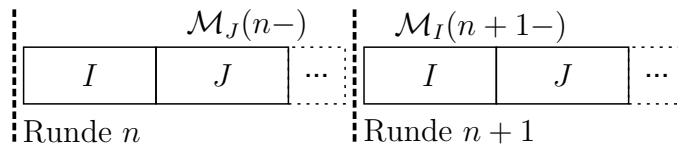


Abbildung 5.22: Runden, Slots und Memberships $\mathcal{M}_I(n-)$, $\mathcal{M}_J(n+1-)$ in TDMA Systemen

unterschritten werden müssen, sind dabei niedriger als die, die zum Übergang in den fehlerpassiven Zustand überschritten werden müssen, um ein Oszillieren zu verhindern.

vom Bus getrennte Knoten (*bus off*) Empfängt oder sendet ein fehlerpassiver Knoten weiter fehlerhafte Nachrichten, so wird er von dem Controller vom Bus getrennt. Durch Reset und Rekonfiguration kann der Knoten wieder an den Bus angeschlossen werden.

Die Fehlerbehandlung von CAN arbeitet unter der Annahme, dass sich ein fehlerhafter Knoten immer vom Bus trennen kann. Eine dem Bus Guardian vergleichbare Einheit ist in CAN nicht vorgesehen, kann jedoch ergänzt werden, siehe auch [Roo05].

TTP/C

Abweichend von der bisher verwendeten Reihenfolge, wird für die Diskussion der Fehlerbehandlung TTP vor FlexRay behandelt. Grund hierfür ist, dass für FlexRay ein Group Membership Algorithmus präsentiert wird, der auf dem Group Membership Algorithmus von TTP aufbaut. Um diese Algorithmen beschreiben zu können, werden zuerst einige formale Definitionen vorgenommen. Die hier verwendete Konvention orientiert sich dabei an [DMS95].

Ein Knoten I ist Element der Menge aller Knoten \mathcal{N} . Daher gilt $I \in \mathcal{N}$. Der Knoten I kann auch in Runde n Element des Memberships von Knoten $J \in \mathcal{N}$ sein. Dann gilt $I \in \mathcal{M}_J(n)$. Das \mathcal{M} -Membership ist ein Indikator dafür, ob Knoten die gleiche Sicht des Systems besitzen, also die gleichen Nachrichten empfangen haben. Da in TDMA Systemen zu jeder Zeit nur ein Knoten senden kann, wird die aktuellste Information, die einem anderen Knoten in Runde n über das Membership von Knoten I vorliegt als $\mathcal{M}_i(n-)$ bezeichnet. Diese Information kann aus Runde n stammen, falls I vor dem betreffenden Knoten sendet, oder aus der vorhergehenden Runde $n-1$, siehe auch Abbildung 5.22.

5 Simulation und Implementierung

Für die Modellierung der Kommunikation wird die Funktion $\text{send}_{I,J}(n)$ definiert, die zu wahr ausgewertet wird, falls Knoten I in Runde n an Knoten J eine korrekte Nachricht versendet hat.

Group Membership Service, Implicit Acknowledgement und Clique Detection

Eine Besonderheit von TTP ist die Tatsache, dass es im Gegensatz zu anderen hier vorgestellten Kommunikationssystemen auch Funktionalitäten höherer Schichten des OSI-Schichtenmodells zur Verfügung stellt und damit die Entwicklung von fehlertoleranten Systemen vereinfacht. Ein besonders wichtiger Dienst ist hierbei der so genannte *Group Membership Service*, bei dem die ECUs sich untereinander über empfangene Nachrichten abstimmen. Eine detailliertere Darstellung sowie eine formale Verifikation finden sich in [Pfe00]². Die Verwendung eines Group Membership Algorithmus ist nötig, da in einem TDMA-System die Bus Guardians die Einhaltung der Slots überwachen und daher kein ACK-Flag (siehe Unterabschnitt 2.4.1) gesetzt werden kann. Es wird also ein Verfahren benötigt, welches ähnlich dem Fehlerzähler in CAN den Steuergeräten eine Möglichkeit gibt, das Funktionieren der eigenen Kommunikation zu überwachen. Hierzu verfügt jeder Knoten I über eine Liste der seiner Meinung nach korrekt funktionierenden Knoten \mathcal{M}_I , die in der so genannten *Membership List* gespeichert ist. Im Allgemeinen wird diese Membership Liste nicht gesondert übertragen, sondern in den *erweiterten CRC* eingerechnet. Hierzu wird zur Berechnung des CRC im Sender dem eigentlichen Inhalt der Nachricht die Membership Liste angehängt. Der Empfänger hängt an die von ihm empfangenen Daten seine eigene Membership Liste an. Stimmt der so berechnete CRC mit dem empfangenen überein, so kann der Empfänger davon ausgehen, dass die Nachricht nicht gestört wurde und dass der Zustand des Senders und des Empfängers gleich sind. Im Falle einer Abweichung wird die Nachricht als inkorrekt verworfen und es kann nicht unterschieden werden, ob die Nachricht gestört wurde oder die Membership Listen abweichen.

Das Vorgehen zur Sicherstellung einer einheitlichen Sicht der Knoten unterscheidet sich dabei für die Detektion von Sende- und Empfangsfehlern. Für die Detektion von Sendefehlern wird *Implicit Acknowledgement* (implizite Bestätigung) angewandt:

1. Ein seiner eigenen Meinung nach fehlerfreies Steuergerät I, für das also $I \in \mathcal{M}_I$ gilt, sendet in seinem Zeitschlitz t_n eine Nachricht³.

²Die Verifikation berücksichtigt allerdings nur symmetrische Fehler und geht davon aus, dass die Geräte sich nach einer Selbstdiagnose noch selbst vom Kommunikationsmedium trennen.

³Der Sendezeitpunkt t_n darf hier nicht mit der Runde n verwechselt werden.

2. Das nächste fehlerfreie Gerät J mit $J \in \mathcal{M}_J$ sendet im Slot t_{n+1} seine Nachricht und die Liste der Geräte, die es für fehlerfrei hält, also \mathcal{M}_J . Stimmt diese Liste mit der des ersten Steuergeräts überein (insbesondere ist das zweite Gerät der Meinung, dass das Erste fehlerfrei ist, also $I \in \mathcal{M}_J$), so kann das Erste sicher sein, dass die Übertragung fehlerfrei bei allen fehlerfreien Geräten angekommen ist. Der Grund hierfür ist die Ein-Fehler-Hypothese, die TTP/C zugrunde liegt. Wäre die Sendefunktion des ersten Geräts I fehlerhaft, so müsste das zweite Gerät J dies erkennen. Da Nachrichten aufgrund des CRCs entweder korrekt oder gar nicht empfangen werden, kann das erste Gerät sicher sein, dass es fehlerfrei sendet.

Ist das erste Gerät nicht in der Membership Liste des zweiten, gilt also $I \notin \mathcal{M}_J$, so ist unklar, ob der Fehler beim ersten oder zweiten Gerät liegt. Diese Mehrdeutigkeit wird erst durch den dritten Knoten K aufgelöst.

3. Falls die Liste des dritten Geräts nun das erste enthält, das dritte Gerät also der Meinung ist, dass das erste fehlerfrei funktioniert $I \in \mathcal{M}_K$, so entfernt das erste Gerät das zweite aus seiner Membership Liste $J \notin \mathcal{M}_I$. Fehlt der erste Knoten in der Liste $I \notin \mathcal{M}_K$, so ist klar, dass das Versenden der Nachricht im Slot t_n nicht erfolgreich war und der erste Knoten sich daher aus seiner eigenen Liste entfernen muss $I \notin \mathcal{M}_I$. In der nächsten Runde wird er nicht mehr senden.

Ein korrespondierendes Vorgehen wäre auch für die Aufdeckung von Empfangsfehlern denkbar. TTP geht hier aber einen Weg, der den Sende- und Empfangszählern von CAN ähnelt, die sogenannte *Clique Avoidance* (Vermeidung von Cliquenbildung). Hierzu werden in den Zählern *Accept Counter* (AC) und *Failure Counter* (FC) lokal die Anzahl der korrekt beziehungsweise falsch empfangenen Nachrichten seit dem letzten Sendeslots eines Gerätes gezählt. Bevor ein Gerät sendet wird überprüft, ob $AC \leq FC$ gilt. Falls dies gilt nimmt der Knoten sich aus seiner Membership Liste, sendet einen Null-Frame und informiert somit die anderen Knoten im Netzwerk über seine Fehlfunktion. Der Vorteil dieses Mechanismus ist, dass Empfangsfehler, die nur einzelne Geräte betreffen, wie zum Beispiel elektromagnetische Störungen im Motorraum, nicht die Bildung von disjunkten Gruppen, den so genannten *Cliquen* bewirkt. Verdeutlicht wird das Vorgehen am folgenden Beispiel. Die Zustände der Geräte sind zum einfacheren Verständnis in Tabelle 5.1 gesondert aufgeführt.

Beispiel 2 (Membershipalgorithmus von TTP/C) Gegeben seien fünf Geräte $A - E \in \mathcal{N}$ an einem TTP/C Bus. Die Geräte senden in alphabetischer

5 Simulation und Implementierung

Reihenfolge. Zu Beginn gelte $I \in \mathcal{M}_J \forall I, J \in \mathcal{N}$ und die Zähler AC und FC seien auf den entsprechenden Werten. Sendet nun A und wird die von A gesendete Nachricht von B und E korrekt, von C und D jedoch falsch empfangen, so entfernen C und D das Gerät A aus ihrer Membership List

$$\neg \text{send}_{A,C} \Rightarrow A \notin \mathcal{M}_C \wedge \neg \text{send}_{A,D} \Rightarrow A \notin \mathcal{M}_D.$$

Gleichzeitig wird in C und D FC erhöht sowie in B und E der Zähler AC.

Betrachtet man nun die Membership Listen der einzelnen Geräte, so ist festzustellen, dass sich zwei Cliques gebildet haben. Jede Clique ist überzeugt, dass nur die Mitglieder der eigenen Clique ein richtiges Bild von dem Status des Systems haben. Nun greift der *Clique Avoidance*-Algorithmus. Die Geräte vergleichen vor dem Senden ihre AC/FC-Zähler. Die Mitglieder der kleineren Clique stellen dabei fest, dass sie in der Minderheit sind, nehmen sich selbst aus ihrer Membership Liste und senden eine Null-Frame, der die anderen Geräte über die Fehlfunktion in Kenntnis setzt. Nach einer weiteren Runde sind die Membership Listen des jetzt kleineren Systems wieder konsistent. Hiermit kann die in der Fehlerhypothese angegebene maximale Fehlerrate von einem Fehler in zwei Runden begründet werden⁴.

Nach dem Senden von B wird B ebenfalls aus den Membership Listen von C und D gelöscht, da ja aufgrund des extended CRC die Nachricht mit dem falschen Membership als inkorrekt ausgewertet wurde.

$$\neg \text{send}_{B,C} \Rightarrow B \notin \mathcal{M}_B \wedge \neg \text{send}_{B,D} \Rightarrow B \notin \mathcal{M}_D$$

Vor dem Senden überprüft nun C seine Fehlerzähler und da $AC_C = FC_C$ gilt, sendet C einen Null-Frame, nimmt sich selbst aus seinem Membership und geht in den freeze-Zustand über. Durch Senden des Null-Frames nehmen auch alle anderen Geräte C aus ihrer Membership Liste.

$$AC_C = FC_C \Rightarrow C \notin \mathcal{M}_I \forall I \in \mathcal{N}$$

Für D gilt nun, dass $AC_D < FC_D$, weshalb D ebenfalls einen Null-Frame sendet und in den freeze-Zustand übergeht.

$$AC_D = FC_D \Rightarrow D \notin \mathcal{M}_I \forall I \in \mathcal{N}$$

Da E ein mit A und B übereinstimmendes Membership besitzt, wird die von E gesendete Nachricht in A und B korrekt ausgewertet und E bleibt im Membership dieser beiden Knoten.

⁴In [BM02] wird bewiesen, dass der Group Membership Algorithmus auch bei häufigerem Auftreten von Fehlern die Cliquesbildung effektiv verhindern kann. Demonstriert wird dies an einem Beispiel, in dem nach dem Auftreten von zwei Fehlern in einer Runde nur noch ein Gerät sendeberechtigt ist. Zwar ist klar, dass hier nicht mehrere sendeberechtigte Cliques mehr existieren können; für eine reale Anwendung wäre dieser Fall wohl mit einem Totalausfall des Kommunikationssystems gleichzusetzen.

Tabelle 5.1: Beispiel zum Group Membership Algorithmus von TTP/C. Die Angaben für Listeneinträge und Zählerstände beziehen sich auf den Zustand nach dem Senden beziehungsweise dem Empfang einer Nachricht.

		Knoten A	Knoten B	Knoten C	Knoten D	Knoten E
A	\mathcal{M}	ABCDE	ABCDE	BCDE	BCDE	ABCDE
	AC/FC	0/0	4/0	2/1	1/1	1/0
B	\mathcal{M}	ABCDE	ABCDE	CDE	CDE	ABCDE
	AC/FC	1/0	0/0	2/2	1/2	2/0
C	\mathcal{M}	ABDE	ABDE	freeze	DE	ABDE
	AC/FC	1/0	0/0	—	1/2	2/0
D	\mathcal{M}	ABE	ABE	freeze	freeze	ABE
	\mathcal{M}	1/0	0/0	—	—	2/0
E	\mathcal{M}	ABE	ABE	freeze	freeze	ABE
	AC/FC	2/0	1/0	—	—	0/0
A	\mathcal{M}	ABE	ABE	freeze	freeze	ABE
	AC/FC	0/0	0/0	—	—	1/0
	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots

Es fällt auf, dass der Clique Avoidance Algorithmus von TTP sehr schnell greift, in Beispiel 2 bereits bei einer falschen Nachricht. Diese kann durch ein einzelnes gekipptes Bit hervorgerufen werden, was, wie in Abschnitt 2.3.2 gezeigt, im Automobil durchaus wahrscheinlich ist. Dieses Verhalten kann als „drakonisch“ [Rus03] bezeichnet werden, da es Geräte bei der kleinsten Fehlfunktion von der Kommunikation ausschließt. Einerseits führt dies dazu, dass nur verlässliche Geräte kommunizieren; andererseits können gerade bei schlechten Umweltbedingungen bereits sehr kurze transiente Fehler die Anzahl funktionierender Geräte drastisch reduzieren. Daher wird eine Strategie zur Wiederaufnahme von Geräten benötigt. In [TTT03] ist spezifiziert, dass ein Gerät, das sich als einer Minderheit zugehörig diagnostiziert in den *Freeze*-Zustand übergeht, aus dem es durch das *Cold Start*-Flag wieder in den Initialisierungszustand übergehen und versuchen kann sich durch Übernehmen der Membership Liste der funktionierenden Geräte wieder in das System einzugliedern. Ähnlich verläuft auch der Cold Start. Falls ein Knoten sich nicht innerhalb einer festgelegten Zeit auf einen Frame mit explizitem Status synchronisieren kann, sendet er selbst einen Frame, der es anderen Geräten erlaubt, sich zu synchronisieren. Um Cliquenbildung beim Start zu vermeiden, wird der so genannte *Big Bang* angewandt [TTT03].

FlexRay

Im Gegensatz zu TTP beschränkt sich die Fehlerbehandlung in FlexRay auf die Behandlung von syntaktischen Fehlern sowie Fehlern in dem Controller selbst. Es wird argumentiert, dass die Fehlerbehandlung mit Hilfe von zusätzlichem Wissen über das System sowie den Fehler auf einer höheren Ebene effektiver und effizienter erfolgen kann. Vor dem Hintergrund von Beispiel 2 ist klar, dass ein Algorithmus, der aufgrund einer fehlerhaften Nachricht fast die Hälfte der in einem Netzwerk vorhandenen Knoten von der Kommunikation ausschließt, nicht immer erwünscht ist. Insbesondere bei der Verwendung von Duplex-Paaren kann dieses Verhalten leicht zu einem vollständigen Verlust der Systemfunktionalität führen. Um zu verdeutlichen, dass der Ansatz der Verlagerung der Fehlerbehandlung berechtigt ist, wird im Laufe dieses Unterabschnittes der in [NBK07a] vorgestellte Membership Algorithmus präsentiert werden.

Für die Fehler innerhalb der FlexRay Controllers steht ein der Fehlerbehandlung von CAN zu vergleichendes Degradierungsmodell zur Verfügung. Im normalen Betrieb befindet sich der Controller in dem Zustand *normal active*, in dem er synchronisiert ist und Nachrichten sowohl senden als auch empfangen darf. Schlägt die Synchronisierung fehl da die Werte die festgelegten Grenzen überschreiten, geht der Controller in der Zustand *normal passive* über und empfängt Nachrichten. Dies ist sinnvoll, da so einerseits eine Störung der korrekt synchronisierten Geräte verhindert wird und andererseits davon ausgegangen werden kann, dass eine solche Abweichung nicht unbedingt den korrekten Empfang von Nachrichten stört. Durch korrekte Synchronisation kann ein Knoten aus *normal passive* wieder in *normal active* übergehen.

Fehlen die Daten zur Synchronisation jedoch vollständig, so liegen keine Synchronisationsdaten vor und der Knoten geht in den Zustand *halt* über, indem weder Senden noch Empfangen von Nachrichten möglich ist. Der Zustand *halt* kann nur durch ein Host Kommando wieder verlassen werden.

Durch den Bus Guardian beziehungsweise den aktiven Stern werden viele Fehler in FlexRay auf eine Verletzung des Sendeslots abgebildet. Hierzu gehören der babbling-idiot und der Masquerading Fehler. Treten diese Fehler auf, so werden sie effektiv gekapselt. Dies geschieht bereits in den Knoten selbst, beziehungsweise in dem aktiven Stern, bevor sich die fehlerhafte Nachricht ausbreiten kann. Es kann daher bei der Fehlerbehandlung davon ausgegangen werden, dass weder ein babbling-idiot noch ein Masquerading Fehler vorliegt. Byzantinische Fehler werden in FlexRay nicht auf der Protokollebene behandelt, weswegen lediglich syntaktische Fehler bei der Übertragung berücksichtigt werden müssen.

Diese syntaktischen Fehler werden in FlexRay lediglich entdeckt, die Behandlung erfolgt durch den Host. Insbesondere wird keine Information über den korrekten oder fehlerhaften Empfang an die anderen Geräte des Netzwerkes gesendet. Es kann daher ohne weitere Maßnahmen nicht von konsistenten Datensätzen in dem Netzwerk ausgegangen werden. Dies ist bei verteilten sicherheitsrelevanten Systemen aber notwendig, wie am Beispiel der Bremsen klar wird. Empfangen lediglich die Bremsen auf einer Seite des Fahrzeugs den Befehl zur Vollbremsung, kann dies zur Instabilität des Fahrzeugs führen.

Der Group Membership Algorithmus, der im Zusammenhang mit TTP vorgestellt wurde, ließe sich mit expliziter Übertragung des Memberships direkt auf FlexRay übertragen. Problematisch ist jedoch, dass dieser Algorithmus nicht zwischen Membership auf der logischen Ebene, dem \mathcal{M} -Membership, und der Fähigkeit zweier Geräte, fehlerfrei zu kommunizieren, unterscheidet. Relevant wird dieser Unterschied, wenn zwei Knoten sich in ihrer Sicht über ein drittes Gerät unterscheiden, dessen Ausgangsdaten keinen Einfluss auf die zwischen den beiden Geräten ausgetauschten Nachrichten haben.

Daher wird im Folgenden zwischen zwei Arten von Membership, dem bereits bekannten \mathcal{M} -Membership und dem \mathcal{C} -Membership unterschieden. Es sei $\mathcal{C}_I^X(n)$ die Menge aller Knoten $\in \mathcal{N}$, mit denen I in Runde n über den Kanal X , $X \in \{A, B\}$ sicher kommunizieren kann. Es gilt im normalen Betrieb

$$\begin{aligned} \text{send}_{J,I}(n-) \wedge I \in \mathcal{C}_J^X(n-) &\Rightarrow J \in \mathcal{C}_I^X(n) \\ \text{mit } I \in \mathcal{C}_I^X(n) \forall n, \forall I \in \mathcal{N}, X \in \{A, B\}. \end{aligned} \quad (5.5)$$

Die \mathcal{C} -Membership Mengen der beiden FlexRay Kanäle können dabei vereinigt werden, um ein einziges \mathcal{C} -Membership zu erhalten.

$$\mathcal{C}_I(n) = \mathcal{C}_I^A(n) \cup \mathcal{C}_I^B(n)$$

Durch diesen Mechanismus führt der Verlust der Kommunikationsfähigkeit auf lediglich einem der Kanäle nicht zu einem Verlust des \mathcal{C} -Memberships. Der Verlust ist jedoch für die höheren Schichten transparent und kann hier entdeckt und behandelt werden. Es ist jedoch zu beachten, dass hierbei byzantinische Fehler nicht berücksichtigt werden. Werden auf beiden Kanälen korrekte aber unterschiedliche Nachrichten empfangen, so ist dies wie der Empfang zweier fehlerhafter Nachrichten zu werten, da nicht entschieden werden kann, welche der beiden Nachrichten kontextuell korrekt ist⁵.

⁵Das Versenden unterschiedlicher Nachrichten auf den beiden Kanälen eines FlexRay Systems ist laut Spezifikation zur Erhöhung der Datenrate erlaubt [Fle05b]. Für sicherheitsrelevante Systeme ist jedoch eine redundante Übertragung der Daten notwendig.

5 Simulation und Implementierung

Durch Gleichung 5.5 ist es Knoten möglich festzustellen, mit welchen anderen Knoten eine bidirektionale Kommunikation möglich ist. Dies ist für zustandslose Daten, also beispielsweise Sollwerte für Steuerungen ausreichend. Für komplexere Operationen, wie zum Beispiel der Aufbau von FSUs, die synchronisierte Eingangsdaten benötigen, muss jedoch sichergestellt werden, dass die Geräte über die gleichen Eingangsdatensätze verfügen. Hierzu wird die Menge \mathcal{M} verwendet, die wiederum Information über die eventuell im System vorhandenen Cliques enthält. Es sei $\mathcal{M}_I(n)$ die Menge aller Knoten, die aus der Sicht von I Element der Clique, der auch I angehört, sind. Im normalen Betrieb gilt also

$$I \in \mathcal{M}_J(n) \wedge J \in \mathcal{C}_I(n) \Rightarrow J \in \mathcal{M}_I(n) \text{ mit } I \in \mathcal{M}_I(n) \forall n. \quad (5.6)$$

Aus Gleichung 5.5 und Gleichung 5.6 lässt sich erkennen, dass \mathcal{M}_I eine Untermenge von \mathcal{C}_I ist. Dies ist auch sinnvoll, da Knoten, die nicht miteinander kommunizieren können, auch nicht das gleiche \mathcal{M} -Membership besitzen können, insbesondere, da sie jeweils nicht Element des \mathcal{M} -Memberships des anderen Knoten sein können. Andererseits sind Knoten, die miteinander fehlerfrei kommunizieren können, aufgrund von Gleichung 5.6 nicht automatisch in der gleichen Clique. Die Menge \mathcal{M}_I erlaubt also eine Aussage darüber, welche Knoten mit I synchronisierte Datensätze besitzen, während \mathcal{C}_I lediglich die Fähigkeit von I, mit den enthaltenen Knoten zu kommunizieren, ausdrückt.

Da FlexRay die Ein-Fehler-Hypothese zugrunde liegt, können zu jedem Zeitpunkt lediglich zwei disjunkte Cliques existieren, die von dem Fehler betroffenen und diejenigen, die von dem Fehler nicht betroffen sind. Es ist aber wichtig anzumerken, dass zu einer sofortigen Identifizierung der Cliques direkt nach dem Auftreten des Fehlers globales Wissen über das System notwendig wäre. Da dies nicht möglich ist, verzögert sich die Aufdeckung des Fehlers jedoch. Eine Runde nach dem Auftreten des Fehlers kann aber davon ausgegangen werden, dass jeder Knoten ein Nachricht versendet hat. Da für das \mathcal{M} -Membership die Eigenschaft

$$I \in \mathcal{M}_L \wedge I \notin \mathcal{M}_J \Rightarrow J \notin \mathcal{M}_L$$

nach dem Austausch von Nachrichten gilt, ist das \mathcal{M} -Membership nach einer Runde stabil.

Für die Ausführung des Membership Algorithmus muss ein Knoten in der Lage sein festzustellen, ob er Teil der Mehrheits- oder der Minderheitsclique ist. Ist er Teil der Mehrheitsclique, so darf er weiter senden, andernfalls kann er versuchen sich in die Mehrheitsclique zu integrieren. Da zu einer Zeit höchstens zwei Cliques in einem System existieren und im normalen

Betrieb jeder Knoten selbst Teil seines eigenen \mathcal{M} -Memberships ist, also $I \in \mathcal{M}_I \forall I \in \mathcal{N}$ gilt, ist ein Knoten Element der Minderheitsclique, wenn

$$|J \notin \mathcal{M}_I| \geq |J \in \mathcal{M}_I|, I, J \in \mathcal{N}$$

gilt.

Durch das Senden einer *rejoin*-Nachricht signalisiert ein Knoten der Minderheitsclique, dass er diese verlassen und der Mehrheitsclique beitreten möchte.

$$I \notin \mathcal{M}_J(n) \forall J \in \mathcal{N} | I \in \mathcal{M}_J(n-) \wedge I \in \mathcal{M}_J(n) \forall J \in \mathcal{N} | I \notin \mathcal{M}_J(n-) \\ I \in \mathcal{C}_J \forall J \in \mathcal{N} | \text{send}_{I,J}$$

Falls beide Cliques gleich groß sind, sorgt das erste Gerät, welches eine *rejoin*-Nachricht sendet, dass Mehrheits- und Minderheitsclique klar voneinander getrennt sind. Zu diesem Zeitpunkt verfügt der Knoten, der die *rejoin*-Nachricht versendet hat, aber weder über eine aktuelle Membership-Liste der Mehrheitsclique, noch über konsistente Daten. Die muss sowohl dem Knoten selber als auch den anderen Knoten in der Mehrheitsclique bewusst sein. Nach der *rejoin*-Nachricht beginnt innerhalb einer Runde der dynamische Slot. Hier können nun alle Daten, die zur Eingliederung in die Mehrheitsclique benötigt werden, von deren Mitgliedern versendet werden, *sync-Nachricht*, siehe auch Abbildung 5.23.


Runde 1						dynamischer Slot
Runde 2			rejoin	rejoin		sync-Nachricht
Runde 3			✓	✓		dynamischer Slot

Abbildung 5.23: Die drei Runden von Fehler bis zur Wiedereingliederung in FlexRay

Nach drei Runden sind die Geräte der Minderheitsclique wieder synchronisiert und können so nach einem transienten Fehler ihre Arbeit wieder aufnehmen. Während dieser Zeit war es den Geräten der beiden Cliques meist möglich, miteinander zu kommunizieren. Ein Kommunikationsfehler schlägt sich nach maximal einer Runde in dem \mathcal{C} -Membership der Knoten nieder. Es ist nicht möglich aus lokalem Wissen der Knoten immer auf den Zustand des Kommunikationsmediums zu schließen, weshalb die Knoten auf die Informationen der letzten Runde zurückgreifen müssen. Dies ist keine Eigenschaft des Algorithmus, sondern vielmehr von TDMA Systemen im Allgemeinen.

5 Simulation und Implementierung

Um die Arbeitsweise des Algorithmus zu verdeutlichen, soll nun Beispiel 2 wieder aufgegriffen werden. Die einzelnen Memberships sind in Tabelle 5.2 aufgelistet.

Beispiel 3 Es werden wieder die Knoten A–E mit $A - E \in \mathcal{N}$ betrachtet, die während dem statischen Segment in alphabetischer Reihenfolge senden. An das statische Segment schließt sich, wie in Abbildung 5.23 dargestellt ein ausreichend langes dynamisches Segment an. Zu Beginn gelte

$$I \in \mathcal{C}_J \forall I, J \in \mathcal{N} \wedge I \in \mathcal{M}_J \forall I, J \in \mathcal{N}.$$

Die von A gesendete Nachricht wird von den Knoten C und D aufgrund eines transienten Fehlers falsch empfangen, während B und E die korrekte Version der Nachricht empfangen. Während B den korrekten Empfang der Nachricht bestätigt indem $A \in \mathcal{C}_B$ und $A \in \mathcal{M}_B$, entfernen C und D den Knoten A aufgrund der fehlerhaften Übertragung sowohl aus dem \mathcal{C} -Membership als auch aus dem \mathcal{M} -Membership.

Der Knoten B hingegen wird lediglich aus den \mathcal{M} -Memberships von C und D entfernt. Da seine Nachricht korrekt empfangen wurde, behalten sowohl C als auch D den Knoten B in ihrem \mathcal{C} -Membership. Zu diesem Zeitpunkt sind C und D noch überzeugt, der Mehrheit anzugehören und senden daher ihre Nachrichten. In folgenden Zeitschlitzen werden C und D aus den \mathcal{M} -Memberships von A, B und E, nicht aber aus deren \mathcal{C} -Membership gestrichen. Der Knoten E wird wie B aus den \mathcal{M} -Memberships von C und D gelöscht.

Die Knoten C und D wissen jetzt zwar, dass sie der Minderheitsclique angehören, sie können dies aber erst in der folgenden Runde durch eine rejoin-Nachricht der Mehrheitsclique mitteilen. In dem dynamischen Segment dieser Runde kann nun die Synchronisation der Eingangsdaten vorgenommen werden. Die Synchronisation erfolgt also bevor der Knoten den rejoin-Zustand verlassen und erneut senden kann, wodurch ein Versenden fehlerhafter Daten vermieden wird. Schlägt die Synchronisation während des dynamischen Segments fehl, so müssen die Knoten im rejoin-Zustand sich selbst aus ihrem Membership entfernen und dürfen nicht senden, da sonst unsynchronisierte Daten die Mehrheitsclique korrumpieren könnten.

Die Verwendung dieses Algorithmus stellt zwei Anforderungen an die Parametrierung von FlexRay:

- das dynamische Segment muss ausreichend lang sein
- zwischen zwei Frames muss es den Knoten möglich sein, ihr Membership zu aktualisieren.

Können diese Forderungen nicht eingehalten werden, so verzögert sich die Wiedereingliederung der Geräte der Minderheitsclique. Während dieser Zeit muss sichergestellt werden, dass kein weiterer Fehler auftritt. Es ist deutlich, dass die Annahmen zur Nettodatenrate von FlexRay, die im vorherigen Unterabschnitt getroffen wurden, verletzt werden.

Abschließend ist festzustellen, dass FlexRay Fehler in der Übertragung schnell lokal erkennt. Problematisch ist die Kommunikation der Fehlerentdeckung an die anderen Geräte in einem Netzwerk. Hierfür kann mit dem präsentierten Algorithmus eine maximale Zeit von zwei Runden angenommen werden. Nach einer weiteren Runde können diese Geräte nach einem transienten Fehler wieder integriert werden.

Vergleich

Die Fehlerbehandlungsalgorithmen von SafeNet wurden bereits in Abschnitt 3.8 ausführlich behandelt, weshalb an dieser Stelle ein Vergleich der Systeme vorgenommen wird. Hierfür wird ein System verwendet, welches aus acht Knoten besteht, die jeweils eine Nachricht mit 8 Byte Nutzdaten versenden. Die Knoten senden dabei so, dass in SafeNet und CAN eine Auslastung von $\rho = 1$ entsteht. In FlexRay wird kein dynamisches Segment verwendet.

An dieser Stelle sei darauf hingewiesen, dass die Länge einer FlexRay Runde mit der Anzahl der Geräte im Netz, also mit N wächst, während die Zeit τ_{max} , die eine Nachricht in SafeNet benötigt, um alle Knoten zu erreichen, topologieabhängig ist und beispielsweise im chordalen Ring der Skip-Länge zwei mit $N/2$ wächst.

5 Simulation und Implementierung

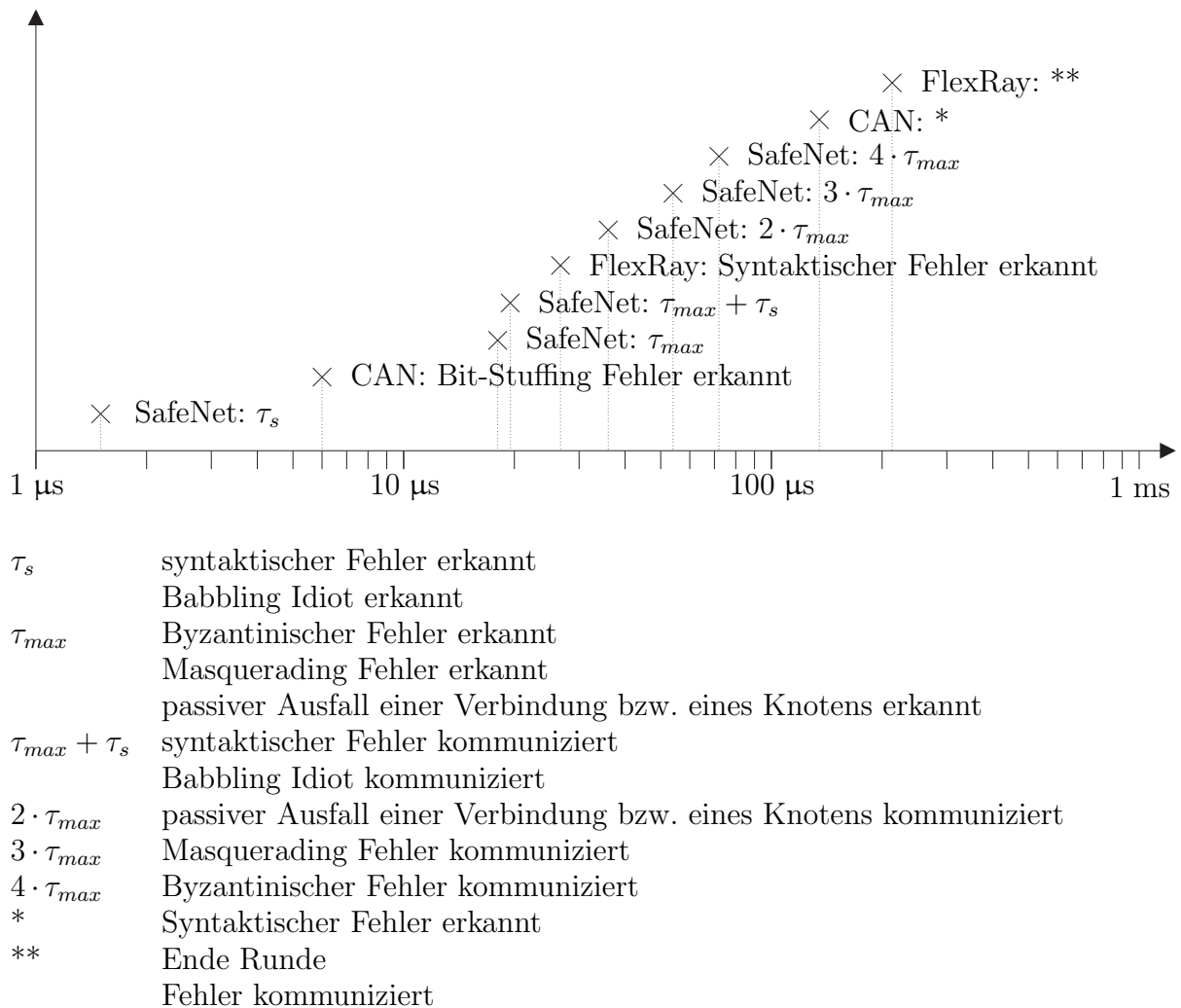


Abbildung 5.24: Worst-case Fehleraufdeckungszeiten im Vergleich

Tabelle 5.2: Group Membership in FlexRay (Beispiel 3)

		Knoten A	Knoten B	Knoten C	Knoten D	Knoten E
INIT	\mathcal{C}	ABCDE	ABCDE	ABCDE	ABCDE	ABCDE
	\mathcal{M}	ABCDE	ABCDE	ABCDE	ABCDE	ABCDE
A	\mathcal{C}	ABCDE	ABCDE	BCDE	BCDE	ABCDE
	\mathcal{M}	ABCDE	ABCDE	BCDE	BCDE	ABCDE
B	\mathcal{C}	ABCDE	ABCDE	BCDE	BCDE	ABCDE
	\mathcal{M}	ABCDE	ABCDE	CDE	CDE	ABCDE
C	\mathcal{C}	ABCDE	ABCDE	BCDE	BCDE	ABCDE
	\mathcal{M}	ABDE	ABDE	CDE	CDE	ABDE
D	\mathcal{C}	ABCDE	ABCDE	BCDE	BCDE	ABCDE
	\mathcal{M}	ABE	ABE	CDE	CDE	ABE
E	\mathcal{C}	ABCDE	ABCDE	BCDE	BCDE	ABCDE
	\mathcal{M}	ABE	ABE	CD	CD	ABE
A	\mathcal{C}	ABCDE	ABCDE	ABCDE	ABCDE	ABCDE
	\mathcal{M}	ABE	ABE	CD	CD	ABE
B	\mathcal{C}	ABCDE	ABCDE	ABCDE	ABCDE	ABCDE
	\mathcal{M}	ABE	ABE	CD	CD	ABE
C	\mathcal{C}	ABCDE	ABCDE	ABCDE	ABCDE	ABCDE
	\mathcal{M}	ABE	ABE	rejoin	CD	ABE
D	\mathcal{C}	ABCDE	ABCDE	ABCDE	ABCDE	ABCDE
	\mathcal{M}	ABE	ABE	rejoin	rejoin	ABE
E	\mathcal{C}	ABCDE	ABCDE	ABCDE	ABCDE	ABCDE
	\mathcal{M}	ABE	ABE	rejoin	rejoin	ABE
A	\mathcal{C}	ABCDE	ABCDE	ABCDE	ABCDE	ABCDE
	\mathcal{M}	ABCDE	ABCDE	ABCDE	ABCDE	ABCDE
	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots

6 Zusammenfassung und Ausblick

Der Einfluss von Elektroniksystemen auf die Automobilindustrie wird in den kommenden Jahren weiter wachsen. Im Vordergrund stehen dabei neben den bekannten Komfortfunktionen auch Systeme, die aktiv in die Fahrzeugdynamik eingreifen und daher sicherheitsrelevant sind. Hierbei kommt der Systemarchitektur dieser sicherheitsrelevanten Kraftfahrzeugelektroniksysteme eine Schlüsselrolle zu, da nur durch sie der notwendige Grad an Systemintegrität und damit die geforderte Sicherheit erreicht werden kann. Durch die besonderen Randbedingungen der Automobilindustrie ist jedoch besonderer Wert auf die Kosteneffizienz der Lösung zu legen. Im Rahmen dieser Arbeit wird dabei von der Ein-Fehler-Hypothese ausgegangen. Eine Aussage über die Ausfallwahrscheinlichkeit der Komponenten wird nicht getroffen. In der vorliegenden Arbeit werden zwei Ansätze für die Systemarchitektur vorgestellt, die diese Randbedingungen berücksichtigen und dennoch eine ausreichende Systemintegrität sicherstellen können.

Der Schwerpunkt der vorliegenden Arbeit ist das Kommunikationssystem SafeNet. SafeNet verzichtet im Gegensatz zu anderen gängigen (CAN) oder in der Entwicklung befindlichen Kommunikationssysteme (FlexRay) für Kraftfahrzeuganwendungen auf ein gemeinsames Element wie beispielsweise eine Busleitung oder einen aktiven Sternpunkt und ist vollständig verteilt aufgebaut. Als Verbindungen zwischen den Knoten eines SafeNets existieren lediglich unidirektionale Punkt zu Punkt Verbindungen, weshalb eine Arbitrierung oder ein zeitbasiertes Zugriffsschema nicht notwendig ist. Da auf Stichleitungen verzichtet werden kann, besteht die Möglichkeit, auf gut abgeschlossenen Leitungen eine hohe Übertragungsgeschwindigkeit zu erreichen.

Jeder Knoten besitzt je zwei Ein- und Ausgänge, über die er mit den anderen Knoten verbunden ist. Durch geringe Einschränkungen in der Topologie eines solchen Netzwerkes, unter anderem muss das Netzwerk auch ohne einen beliebigen Knoten zusammenhängend sein, ist es möglich, für sicherheitsrelevante Systeme wertvolle Eigenschaften zu erhalten. Einerseits bestehen zwischen zwei beliebigen Knoten immer redundante Verbindungen, die auch im Fehlerfall die Kommunikation zwischen den fehlerfreien Knoten des Netzwerkes garantieren. Andererseits ist es möglich, einen fehlerhaften Knoten durch die übereinstimmende Diagnose zweier Knoten auch im Falle eines aktiven Ausfalls ohne die Zurhilfenahme zusätzlicher Teilsysteme wie beispielsweise

6 Zusammenfassung und Ausblick

eines Bus Guardians sicher vom Netzwerk zu trennen.

Aufgrund der unidirektionalen Verbindungen muss jede Nachricht, die im Netz versendet wird, weitergereicht werden. Auf einen Routingalgorithmus wird dabei verzichtet, da es einerseits in Kraftfahrzeugelektroniknetzwerken der Regelfall ist, dass eine Nachricht von mehreren Knoten empfangen wird, und andererseits ein solcher Algorithmus die Komplexität eines SafeNet Knotens steigern würde. Es wird stattdessen ein einfacher Algorithmus verwendet, der jede Nachricht im fehlerfreien Fall von jedem Knoten aus genau ein Mal versendet und somit die Broadcast-Eigenschaft des Netzwerkes sicherstellt. Jeder Knoten empfängt jede Nachricht zwei Mal auf unabhängigen Wegen, wodurch die Redundanz des Systems gewährleistet ist.

Durch eine Erweiterung des Sendealgorithmus wird sichergestellt, dass in dem Netzwerk trotz des Fehlens einer zentralen Einheit und trotz des ereignisdiskreten Charakters von SafeNet eine deterministische worst-case Zeit zwischen Absenden einer Nachricht und dem Zeitpunkt, zu dem alle anderen Knoten diese Nachricht empfangen haben, angegeben werden kann. Diese Eigenschaft ist für sicherheitsrelevante Echtzeitsysteme von zentraler Bedeutung und wurde im Rahmen dieser Arbeit hergeleitet. Zudem stellt diese Erweiterung sicher, dass die Wartezeit der Nachrichten in dem System auch für hohe Auslastungen gegen den Auslastungsfaktor eins endlich bleiben, wodurch die Stabilität des Kommunikationssystems garantiert wird. Daneben wird auch ein im Vergleich zu herkömmlichen Systemen deutlich erhöhter Datendurchsatz und eine verringerte maximale Laufzeit ermöglicht.

Während FlexRay und CAN lediglich eine Teilmenge der möglichen Fehler auf Protokollebene entdecken und behandeln, wurde bei SafeNet der Ansatz gewählt, möglichst viele Fehler direkt in SafeNet selbst zu behandeln und so zu kapseln. Hierbei ist es möglich, passive Ausfälle wie den Ausfall eines Knotens oder einer Verbindung, syntaktisch fehlerhafte Nachrichten genauso wie aktive Ausfälle, beispielsweise Babbling Idiot Fehler oder Masquerading Fehler, aufzudecken und zu behandeln. Selbst byzantinische Fehler werden in SafeNet explizit berücksichtigt, ohne höher liegende Schichten zu belasten. Die Fehleranalyse basiert auf regulären Nachrichten und ist daher für die Anwendungen transparent. Durch effiziente Fehlerfindungs- und Fehlerkapselungsalgorithmen kann die Fehlerbehandlung in SafeNet um eine Größenordnung schneller erfolgen, als dies in herkömmlichen Kommunikationssystemen für Kraftfahrzeuge der Fall ist. Die Fehleranalyse wurde, wie die anderen Funktionalitäten von SafeNet auch, im Rahmen dieser Arbeit an einem Modell und in leicht eingeschränkter Form an einem Beispielsystem verifiziert.

Zusammenfassend ist zu sagen, dass SafeNet aus technischer Sicht eine Alternative zu den existierenden oder in Entwicklung befindlichen Kommu-

nikationssystemen darstellt. Durch den Verzicht auf ein zentrales Medium zugunsten einer Netzwerktopologie sowie den Einsatz einfacher verteilter Algorithmen ist es möglich, trotz geringer Komplexität ein System zu entwerfen, welches herkömmliche Systeme in allen für den Einsatz in sicherheitsrelevanter Kraftfahrzeugelektronik relevanten Kriterien übertrifft. Insbesondere ist es in einem verteilten ereignisdiskreten Kraftfahrzeugkommunikationssystem erstmals möglich, trotz stochastischer Nachrichtengenerierung eine deterministische worst-case Laufzeit anzugeben und so die Vorteile ereignisdiskreter Systeme mit der Eignung für sicherheitsrelevante Anwendungen zu verbinden.

6 Zusammenfassung und Ausblick

A Erweiterung des Systemkonzepts auf dynamische Verteilung

A.1 Der *State-Server*-Ansatz

Bereits in Abschnitt 2.1 wurde auf die verschiedenen Möglichkeiten eingegangen, mit deren Hilfe die Korrektheit von Daten in sicherheitsrelevanten Strukturen sichergestellt werden kann. Wie dort bereits erwähnt, ist der bitweise Vergleich der Ausgangsdaten der einzelnen Geräte dem Vergleich mit Toleranz vorzuziehen, da durch den bitweisen Vergleich eventuell auftretende Abweichungen direkt als Fehler interpretiert werden können und somit die Möglichkeit schnellerer Kapselung besteht. In Abbildung 2.4 sind die Ausfallraten der einzelnen Redundanzstrukturen dargestellt.

Problematisch für die Implementierung im Kraftfahrzeug ist nun vor allem der Kostenfaktor. Bereits ein TMR System benötigt verglichen mit einem Simplex System einen Overhead von zwei Geräten, verursacht also eine Verdreifachung der Kosten ohne im fehlerfreien Fall einen Mehrwert zu bieten.

Eine mögliche Lösung dieses Problems liegt in der dynamischen Verteilung der Systemfunktionen auf die vorhandenen Steuergeräte zur Laufzeit. Wie bereits in Unterabschnitt 2.2.1 argumentiert, ist in der Automobilindustrie der Stückpreis der ausschlaggebende Kostenfaktor. Ziel ist es also, den Einsatz von Hardware mit hohen Stückkosten durch den Einsatz von Software, deren Kosten hauptsächlich durch die Entwicklung verursacht werden, zu ersetzen. Im Folgenden wird ein Lösungsansatz für die dynamische Verteilung von Funktionen auf Steuergeräte im laufenden Betrieb untersucht und der *State-Server* vorgestellt, der einen reibungslosen Ablauf dieser Verteilung ermöglicht.

Basierend auf den eingangs beschriebenen Argumenten werden folgende Annahmen getroffen:

- die Ein-Fehler-Hypothese ist gültig
- es wird ein bitweiser Vergleich verwendet
- es wird eine fail-silent Strategie verfolgt
- Common-Mode Ausfälle werden nicht berücksichtigt.

A Erweiterung des Systemkonzepts auf dynamische Verteilung

Basierend auf diesen Annahmen ist es notwendig, Fail-silent Units (FSUs) aus zwei Geräten aufzubauen, die sich gegenseitig überwachen. Durch die Verwendung von Duplex Paaren ist einerseits möglich, eine gute Fehlerabdeckung für einen sicherheitsrelevanten Fehler zu erreichen, andererseits ist der benötigte Aufwand im Vergleich zu TMR gesenkt. Im Allgemeinen ist es bei dem Ausfall einer FSU nicht möglich festzustellen, welches der beiden Geräte den Ausfall verursacht hat, also fehlerhaft ist. Es ist daher sinnvoll, eine ausgefallene FSU auf eine funktionierende FSU zu verlagern. Als kleinste Einheiten der dynamischen Verlagerung von Funktionen wird daher zunächst die Funktionalität einer FSU angenommen.

Zusätzlich wird der Begriff der normalisierten Geschwindigkeit eingeführt. Die normalisierte Geschwindigkeit ξ gibt das Verhältnis zwischen der der Zeit, die einer ECU zugeordnete Funktion zur Ausführung benötigt und der Zykluszeit, also der Periode der Funktion an. Sie ist somit ein Maß für die Auslastung eines Gerätes durch die ihm zugeordnete Funktion. Ideal ist im Zusammenhang der Ressourcenausnutzung eine normalisierte Geschwindigkeit von $\xi = 1$ anzusehen, da in diesem Fall die Leistungsfähigkeit der ECU perfekt an die Aufgabe angepasst wurde. Hierbei ist jedoch die Stabilität des Systems zu berücksichtigen, da abhängig von dem verwendeten Schedulingmechanismus geringere normalisierte Geschwindigkeiten verwendet werden müssen. Eine genauere Analyse von Schedulingmechanismen und ihrem Einfluss auf die Systemstabilität findet sich in [Ram06]. Die normalisierte Geschwindigkeit ist vergleichbar mit der bereits zuvor eingeführten Netzwerkauslastung ρ , weshalb die für ρ angestellten Betrachtungen insbesondere für die Stabilität bei Vollauslastung für ξ analog durchgeführt werden können. Die Anzahl der Funktionen, die innerhalb eines Systems ausgeführt werden müssen, wird im Folgenden mit x bezeichnet. Des Weiteren wird im Folgenden davon ausgegangen, dass alle Geräte die gleiche absolute Geschwindigkeit besitzen, also im einfachsten Fall baugleich sind. Dies ist keine prinzipielle Einschränkung sondern dient nur der vereinfachten Analyse.

Werden nun Funktionen während der Laufzeit zwischen den Geräten verteilt, so muss sichergestellt werden, dass die zur Berechnung der Ausgangsdaten nötigen Eingangsdaten allen im Netzwerk vorhandenen Geräten zur Verfügung gestellt werden. Dies kann zum Beispiel wie in Abbildung A.1 dargestellt durch den verstärkten Einsatz von so genannten *networked sensors*, also Sensoren, die ihre Daten nicht einem speziellen Steuergerät zur Verfügung stellen, sondern über ein Kommunikationsnetzwerk an alle in diesem Netzwerk vorhandenen Geräte versenden. Entsprechendes gilt natürlich auch für die im System vorhandene Aktorik.

Damit Funktion einer defekten FSU nun auf eine andere, funktionierende FSU verlagert werden kann, darf diese FSU im fehlerfreien Fall keine aktive

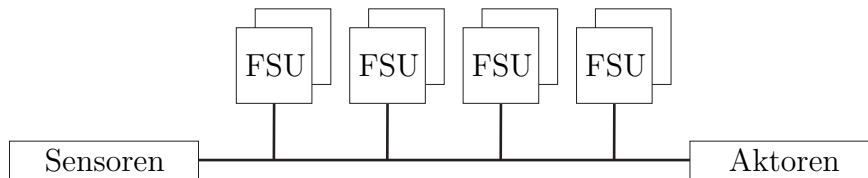


Abbildung A.1: Grundstruktur eines sicherheitsrelevanten Systems mit an das Netzwerk angeschlossener Sensorik und Aktorik

Funktion ausführen, muss also als *cold-standby* dienen. Es müssen daher in dem System insgesamt $2 \cdot (x + 1)$ Geräte vorhanden sein, also zwei Geräte mehr als bei Verwendung von Duplex Paaren. Durch diese Geräte wird jedoch das fail-silent Verhalten des Gesamtsystems auf ein fail-operational/fail-silent Verhalten, das einen sicherheitsrelevanten Fehler tolerieren kann, erweitert. Im Vergleich zu einer TMR Architektur, welche $3x$ Geräte benötigt, besteht also bereits bei kleinen Systemen ein Kostenvorteil.

Im Gegensatz zur Duo-Duplex Architektur gibt es nicht zu jeder FSU eine fest zugeordnete FSU, die im Fehlerfall die Funktion übernimmt. Es ist der Standby FSU also nicht möglich alle Funktionen parallel mitzurechnen, da dann für die normalisierte Geschwindigkeit $\xi = x$ gelten müsste.

Für einfache Funktionen wie zum Beispiel eine Umrechnung der Lenkradstellung in einen Radeinschlagwinkel ist dies unproblematisch. Jedoch schon für eine geschwindigkeitsabhängige Umrechnung stößt dieses Konzept an seine Grenzen, da zusätzlich sichergestellt werden muss, dass die Geschwindigkeit in der Standby FSU vorliegt. Allgemein ist die Verwendung dieses Konzeptes für *dynamische* Funktionen, wie sie in [KJ05] definiert¹ sind, problematisch. Hierunter fallen insbesondere Filter und Systeme in Zustandsraumdarstellung, die im Allgemeinen von der Vergangenheit der Eingangsdaten und der Ausgangsdaten abhängen. Insbesondere diese Funktionen sind aber für die Signalverarbeitung im Kraftfahrzeug von Bedeutung. Eine Eigenschaft dieser Systeme ist aber oft, dass die relevanten Daten, also Wissen über die Vergangenheit des Systems im aktuellen Zustand des Systems repräsentiert sind. Daher ist mit Kenntnis des aktuellen Zustands und der aktuellen Eingangsdaten eine reibungslose Fortsetzung einer Funktion auf einen anderen Steuergerät möglich.

Zu diesem Zweck wird der *State-Server* eingeführt, dessen Aufgabe es ist, die Zustände aller sicherheitsrelevanten Funktionen in einem Netzwerk zu speichern und deren Ausführung nach dem Ausfall der FSU, die diese Funktion ursprünglich ausführte, sicherzustellen. Hierfür muss der *State-Server*

¹In [KJ05] werden dynamische Systeme definiert. Um Verwechslung mit dem Gesamtsystem zu vermeiden, wird hier der Begriff dynamische Funktion verwendet.

A Erweiterung des Systemkonzepts auf dynamische Verteilung

einige Voraussetzungen erfüllen:

- selbst mindestens Duplex ausgelegt sein um die Korrektheit der Zustandsdaten sicherzustellen
- implizites Berechnung der Zustände aus Ein- und Ausgangsdaten oder Speicherung der explizit übertragenen Zustandsdaten
- Verteilung der Funktionen in Fehlerfall.

Es ist durchaus möglich, den State-Server, wie in [NRK05b] vorgeschlagen, den State-Server mit höherer Redundanz, beispielsweise quadruplex, auszuliegen um so die Funktionsfähigkeit des State-Servers im Falle eines internen Fehlers zu gewährleisten. Dies kann, abhängig von der Architektur des State-Servers, sinnvoll sein, um die Ausfallwahrscheinlichkeit des Gesamtsystems relativ kostengünstig zu senken. Allgemein muss das System jedoch weiterhin nur einen Fehler tolerieren können, wofür eine Duplex Auslegung des State-Servers ausreichend ist. Innerhalb dieses Duplex Paares müssen die inneren Zustände des State-Servers abgeglichen werden.

Die Berechnung der Zustände aus den Ein- und Ausgangsdaten ist prinzipiell für beobachtbare Funktionen möglich [FDK94]. Allgemein ist die Beobachtbarkeit beweisbar und zum Beispiel für das reduzierte nichtlineare Zweispurmodell gegeben [NH04]. Für nicht beobachtbare Funktionen müssen die Zustandsdaten explizit übertragen werden. Auch eine näherungsweise Berechnung der Zustandsdaten, die periodisch durch explizite Übertragung gestützt wird, kann unter Umständen akzeptabel sein. Dies widerspricht der Annahme über toleranzlosen bitweisen Vergleich nicht, da der Zustand der Funktion in den Geräten des State-Servers gleich, wenn auch abweichend von dem Zustand in der ausführenden FSU, gespeichert ist.

Wird eine gesonderte FSU als Standby verwendet, die im normalen Betrieb keine Aufgabe erfüllt, so kann der Fall eintreten, dass diese FSU bereits ausgefallen ist und dieser passive Fehler durch die Verlagerung aktiviert wird. Da der State-Server mindestens als Duplex Paar ausgelegt ist, kann er selbst die Funktionalität der ausgefallenen FSU übernehmen. Hierfür muss er jedoch eine absolute Geschwindigkeit besitzen, die den übrigen FSUs vergleichbar ist. Sind die Funktionen des Systems jedoch modular aufgebaut, so ist es möglich, die Funktion der ausgefallenen FSU auf mehrere FSUs zu verteilen, deren normalisierte Geschwindigkeit klein ist, also $\xi < 1$. Gilt für die Summe über die normalisierten Geschwindigkeiten der einzelnen Geräte

$$\sum_i \xi_i \leq x - 1, \tag{A.1}$$

für die normalisierte Geschwindigkeit jedes Geräts also

$$\xi \leq \frac{x - 1}{x}, \quad (\text{A.2})$$

so ist die Verlagerung theoretisch möglich.

Eine normalisierte Geschwindigkeit von $\xi = 1$ ist, wie zuvor schon bemerkt, als Idealfall zu betrachten. Daher ist es oft der Fall, dass die tatsächliche Auslastung der Geräte geringer ist. Ist es im normalen Betrieb nicht möglich, die Funktionen zu verlagern, so kann die nötige Ausführungszeit auf den funktionsierenden Geräten auch durch Funktionsdegradierung, also das Abschalten nicht sicherheitsrelevanter Funktionen, erreicht werden [Roo05].

Das Nachladen von Funktionen im laufenden Betrieb vergleichbar mit dem Software-Tankstellen Konzept [HS03], also die Verteilung des Codes zur Laufzeit von einem Codeserver, ist aufgrund der sicherheitsrelevanten Funktionen, die über das Netzwerk kommunizieren, auch mit schnellen Kommunikationsmedien nicht innerhalb der geforderten maximalen Laufzeiten mit dem aktuellen Stand der Technik nicht möglich. Daher ist es sinnvoll, mögliche Konfigurationen bereits beim Systementwurf zu berechnen und den Code zur Laufzeit lediglich mit dem aktuellen Zustand zu initialisieren und zu aktivieren. Diese Konfigurationen müssen, wie die ursprüngliche Konfiguration auch, die Randbedingungen des Systems erfüllen und auf ihre Gültigkeit hin getestet werden. Insbesondere ist hier auf die Einhaltung von worst-case Ausführungszeiten in den Geräten und worst-case Laufzeiten, die durch das Kommunikationsmedium verursacht werden, zu achten. In dem Fall, dass der State-Server oder eine gesonderte FSU direkt die Aufgaben der ausgefallenen FSU übernimmt, sind diese Überlegungen hinfällig, da die Konfiguration ja der ursprünglichen entspricht und daher von ihrer Korrektheit ausgegangen werden kann.

Welcher der beiden in Abbildung A.2 dargestellten Konfigurationen der Vorzug gegeben wird, hängt von den gegebenen Randbedingungen ab. Insbesondere ist der Aufwand, der auf den State Server selbst verwendet werden soll, von Bedeutung. Auch der Grad, in dem Funktionsdegradierung von dem System akzeptiert werden kann, ist ausschlaggebend. Ist es möglich, eine vollständige FSU als State Server zu reservieren, so ist diese Möglichkeit wohl mit weniger Aufwand zu implementieren. Ist hingegen ein kleiner und daher billig zu implementierender State Server gefragt, so muss sichergestellt sein, dass es im Fehlerfall möglich ist, eine gültige Konfiguration der Geräte aufzurufen.

Um einen Vergleich mit den in Abbildung 2.3 gezeigten Strukturen zu ermöglichen, sind in Abbildung A.3 die für die verschiedenen Strukturen benötigten Geräte in Abhängigkeit von den zu erfüllenden Aufgaben gezeigt.

A Erweiterung des Systemkonzepts auf dynamische Verteilung

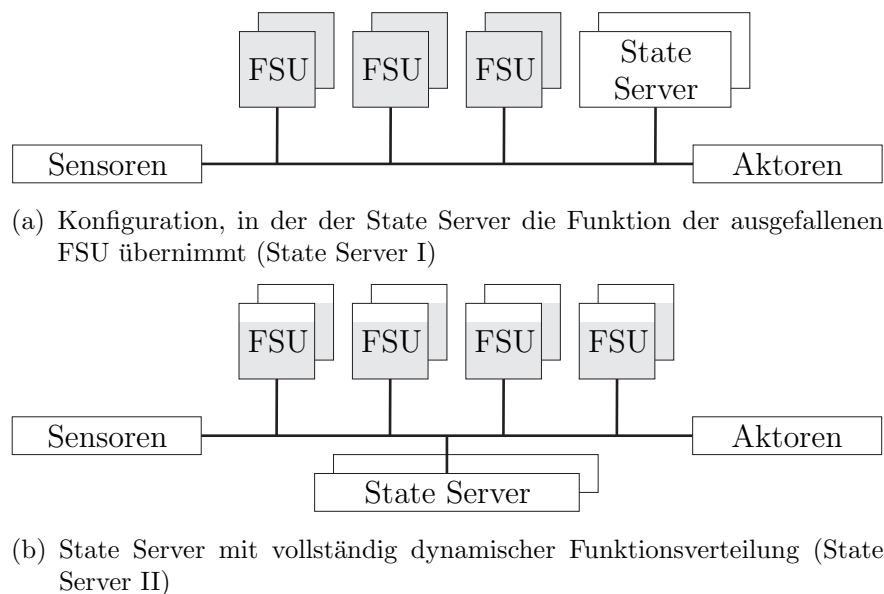


Abbildung A.2: Grundstruktur eines sicherheitsrelevanten Systems mit State Server

Für die State Server I Architektur gilt für die Anzahl der Geräte N

$$N = 2x + 2.$$

Für die State Server II Architektur gilt

$$N = 2x \frac{x}{x - 1},$$

wobei hier der durch den State Server verursachte Aufwand nicht berücksichtigt ist, da dieser wie zuvor beschrieben klein dimensioniert werden kann.

A.2 Implementierung und Verifikation

Im Rahmen dieser Arbeit wurde ein System implementiert, um die dem State Server zugrunde liegende Idee zu verifizieren. Ziel dieser Entwicklung war es, einen State Server unter den in der Automobilindustrie herrschenden Randbedingungen, also kurze Laufzeit und hohe Auslastung der ECUs, in Betrieb zu nehmen. Als Hardwareplattform wurde hierzu das am IIIT entwickelte SAPS/RC Board [Tor03] verwendet. Es basiert auf dem auch im Kraftfahrzeug eingesetzten 16 Bit Mikrocontroller C167CR von Infineon [Roo05], hier mit 25 MHz getaktet und weicht auch sonst nur leicht von industriell eingesetzten Systemen ab. Durch die Verwendung des C167CR steht dem Board

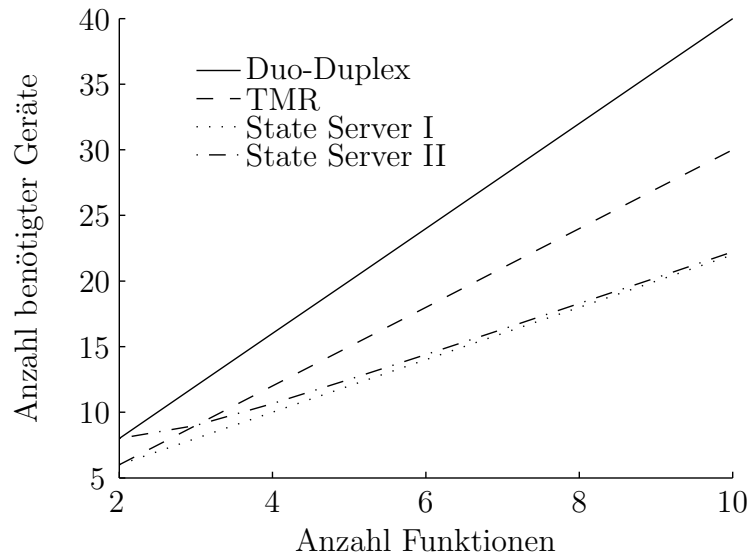


Abbildung A.3: Vergleich der benötigten Geräte für spezifizierte Aufgaben nach Architektur

auch ein CAN Controller zur Verfügung, der mit Hilfe einer Aufsatzplatine an ein CAN Netzwerk angeschlossen werden kann. Neben dem Mikrocontroller besitzt das SAPS/RC auch einen DSP zur schnellen Signalverarbeitung, der aber für die folgenden Betrachtungen irrelevant ist.

Wie bereits in Kapitel 1 erwähnt, sind die im Automobilbereich verwendeten Systeme aufgrund zu knapper Ressourcen nicht in der Lage, gängige, für nicht sicherheitsrelevante Anwendungen ausgelegte Plattformen zur dynamischen Funktionsverlagerung zu nutzen. Daher muss auf diesen Systemen ein Echtzeitbetriebssystem zum Einsatz kommen, welches an die speziellen Gegebenheiten angepasst ist. Üblich sind OSEK-basierte [KJ98] Betriebssysteme. Da diese jedoch zum größten Teil kommerziell vertrieben werden und keine Änderungen auf der für die Implementierung des State Servers nötigen Ebene erlauben, fiel die Wahl auf freeRTOS². Dieses Betriebssystem ist schon für kleine Mikrocontroller ab 8 Bit einsetzbar, gut dokumentiert und unter der GNU Public License (GPL) veröffentlicht, was bedeutet, dass der Quellcode frei zugänglich ist und auch verändert werden darf. Es existiert zudem ein kommerzielles Projekt (safeRTOS), welches auf freeRTOS basierend die Anforderungen an SIL 3 erfüllt. Daher wurde freeRTOS Version 2.1.1 als Basis für das State Server Projekt gewählt. Da freeRTOS in dieser Version jedoch nur für 8 Bit Mikrocontroller verfügbar war, wurde eine Portierung auf den C167CR vorgenommen.

Im Rahmen dieser Portierung wurde auch eine Speicherverwaltung imple-

²Siehe auch www.freertos.org.

A Erweiterung des Systemkonzepts auf dynamische Verteilung

mentiert, die sowohl die Möglichkeiten, die durch die Verwendung des C167 geboten werden, nutzt, als auch eine dynamische Speicherallokierung erlaubt. Die Speicherverwaltung ist in freeRTOS Teil der Portierungsschicht, siehe auch Abbildung A.4.

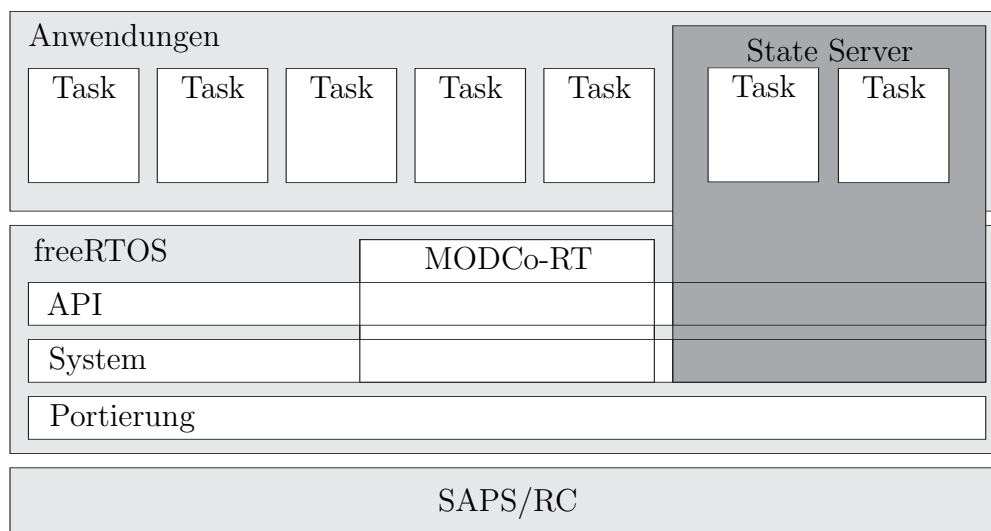


Abbildung A.4: Aufbau von freeRTOS

Zur einfacheren Verwaltung angeschlossener Geräte, die auch den abwechselnden Zugriff mehrerer Tasks unterstützt, wurde die Treiberumgebung MODCo-RT entwickelt. Sie erleichtert durch konsistente Schnittstellen und implementierte Grundfunktionen die Erstellung dynamischer Anwendungen, indem sie als überwachende Schicht zwischen Anwendungen und der Hardware, beispielsweise dem CAN Bus, agiert.

Der eigentliche State Server erstreckt sich über die System- und *application programming interface* (API)-Schicht. Im fehlerfreien Fall speichert der State Server lediglich die in einer speziellen CAN Nachricht enthaltenen Zustandsdaten der anderen Geräte. Die anderen im Netz vorhandenen Tasks sind im State Server initialisiert, werden aber nicht ausgeführt. Bleibt ein vordefiniertes Signal der FSUs aus, so aktiviert der State Server die betreffenden Tasks und übernimmt somit die Funktionen der ausgefallenen FSU. Implementiert ist dieses Signal in der periodischen Zustandsnachricht. Prinzipiell ließe sich aber auch die Nachricht, die die Ausgangsdaten enthält, verwenden. Mit Hilfe dieses Beispielsystems wurde die prinzipielle Eignung des State Server Ansatzes zur dynamischen Verteilung von Funktionen gezeigt.

A.3 State-Server in SafeNet

Das proof-of-concept System für den State Server Ansatz wurde auf der Basis von CAN implementiert um die Anwendbarkeit in heutigen Kraftfahrzeugelektroniksystemen nachzuweisen. Es nutzt aber in den Grundfunktionalitäten³ keine Eigenheiten von CAN aus. Daher ist auch eine Übertragung des State Server Konzeptes auf SafeNet denkbar. SafeNet erfüllt die Anforderung, dass alle Nachrichten, die von den FSUs versendet werden, durch den State Server empfangen werden, wodurch die Rekonstruktion der Zustände ermöglicht wird.

Es stellt sich hier die Frage, wie FSUs in SafeNet aufgebaut werden können. Ziel muss es sein, fehlerhafte Datensätze, in diesem Fall schließt dies abweichende Nachrichten der beiden ECUs einer FSU bezüglich desselben Datensatzes mit ein, möglichst schnell zu entdecken, wenn möglich die Ausbreitung zu verhindern und den Fehler so zu kapseln. Hierbei handelt es sich nicht um eine Aufgabe von SafeNet, da die Nachrichten trotz ihres fehlerhaften Inhalts korrekt versendet werden. Vielmehr zielt diese Frage darauf ab, wie ganze sicherheitsrelevante Systeme mit Hilfe von SafeNet aufgebaut werden können.

Prinzipiell lassen sich die Knoten einer FSU beliebig in einem SafeNet platzieren. Für diesen Zweck bietet sich aber die bereits in Abbildung 3.4 dargestellte Leiter-Struktur in SafeNet an, da hier ein Knotenpaar durch ein anderes überwacht werden kann, siehe auch Abbildung A.5.

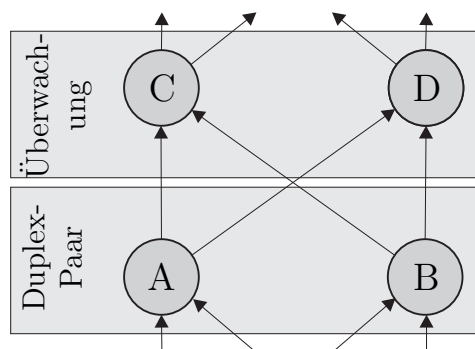


Abbildung A.5: Aufbau einer FSU in SafeNet

Die beiden ECUs einer FSU senden ihre Ausgangsdaten im fehlerfreien Fall quasi synchron. Wird nun eine Verzögerung in den Sendalgorithmus eingeplant, so haben die Knoten der Überwachungsschicht die Möglichkeit,

³Einzige Ausnahme ist der hier nicht beschriebene dynamische Start-up. Hier wird die Priorität von CAN Nachrichten ausgenutzt, um die Rollen der einzelnen ECUs innerhalb des State-Servers festzulegen.

A Erweiterung des Systemkonzepts auf dynamische Verteilung

die empfangenen Daten auf Abweichungen zu überprüfen. Um die Eigenschaft beizubehalten, dass die Reihenfolge der Nachrichten in einem Knoten erhalten bleibt, muss diese Verzögerung für alle Nachrichten in dem Netz gelten. Sollen auch noch die Überlegungen zur Berechenbarkeit der worst-case Laufzeit ihre Gültigkeit behalten, so muss diese Verzögerung in allen Knoten für alle Nachrichten stattfinden, was die worst-case Laufzeit verschlechtert. Hier wird also ein Eingriff in das Ausbreitungsverhalten von Nachrichten in SafeNet vorgenommen.

Ist dieser Effekt nicht erwünscht, so kann in den Geräten der Überwachungsschicht eine zusätzliche Nachricht für den Empfang gleicher Inhalte auf beiden Eingängen erzeugt werden. Diese Nachricht können die Geräte, die auf die Daten des Duplex-Paares angewiesen sind, als Eingangsdatum verwenden. Da hier aber der Fall auftreten kann, dass ein Knoten der Überwachungsschicht fälschlicherweise eine solche Nachricht versendet, ist eine solche Nachricht nur mit der ursprünglichen Datennachricht gültig. In diesem Fall kann davon ausgegangen werden, dass nach der Ein-Fehler-Hypothese nicht sowohl ein Knoten des Duplex-Paares als auch der Überwachungsschicht fehlerhaft sind. Der Vorteil gegenüber der Verwendung der ursprünglichen Nachrichten liegt lediglich in der schnelleren und zentralisierten Fehleraufdeckung. Diese kann jedoch auch dezentral, also separat in jedem Knoten, erfolgen.

In dieser Anordnung können die Knoten der Überwachungsschicht also im Fehlerfall die Knoten des Duplex-Paares über die Abweichung informieren. Vom Netzwerk getrennt werden dürfen die Knoten allerdings nicht, da sich der Fehler auf einer Ebene über SafeNet befindet, solange Nachrichten von beiden Knoten des Duplex-Paares korrekt versendet werden. Die Verschaltung von Knoten zu FSUs erfolgt also nicht in SafeNet direkt, sondern auf einer höheren Ebene. Daher muss zwischen Abweichungen zweier Versionen einer Nachricht Abweichung zweier Nachrichten, die eine Aussage über die gleiche Daten treffen, unterschieden werden.

Allgemein ist es also möglich, FSUs aus Duplex-Paaren in SafeNet mit einer beliebigen Topologie aufzubauen. Durch die Zielsetzung von SafeNet, durch die Bereitstellung redundanter Wege auch immer einen SPOF zu vermeiden, lässt sich das Versenden logisch falscher Daten nicht verhindern. Es kann jedoch, wie in Abschnitt 3.3 gezeigt, sichergestellt werden, dass trotz eines defekten Knotens in einer FSU die korrekt funktionierenden Knoten weiterhin kommunizieren können und durch die von dem defekten Knoten eingestreuten Nachrichten die Auslieferung der übrigen Nachrichten lediglich verzögern, wobei allerdings die worst-case Zeit weiterhin eingehalten wird.

Da SafeNet im Gegensatz zu CAN und FlexRay kein gemeinsames physikalisches Medium besitzt, gestaltet sich die Gewährleistung gleicher Eingangs-

datensätze zur Berechnung der Ausgangsdaten schwieriger. Ohne eine global synchronisierte logische Uhr muss auf ereignisdiskrete Ansätze zur Synchronisierung zurückgegriffen werden. Hierzu bietet sich die so genannte Vektorzeit an, siehe auch [Lam78] und [Kie06].

A Erweiterung des Systemkonzepts auf dynamische Verteilung

B Weitere Simulationsergebnisse

B.1 Leitertopologie mit sechs Knoten

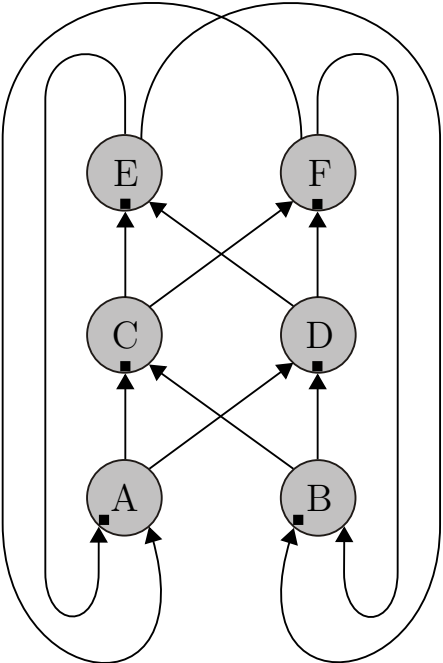
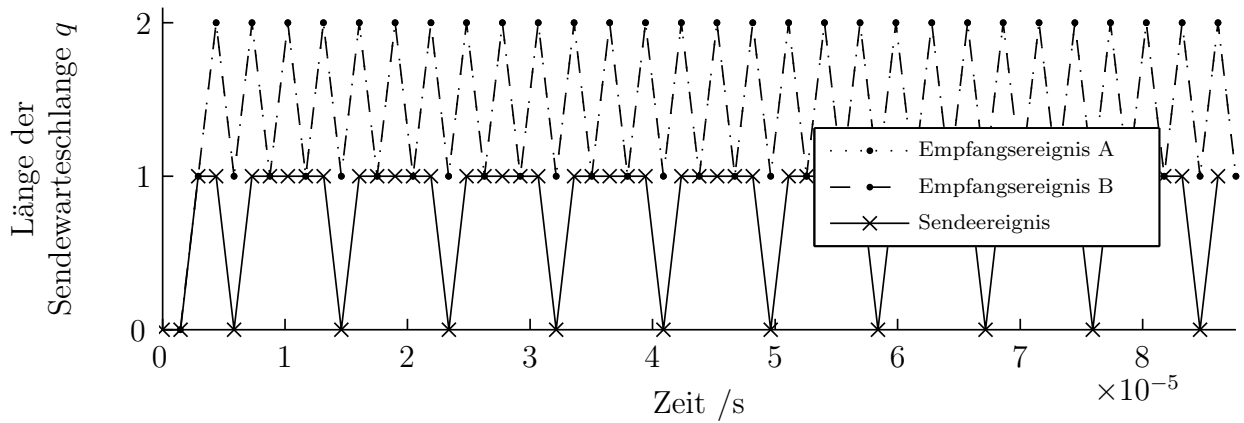
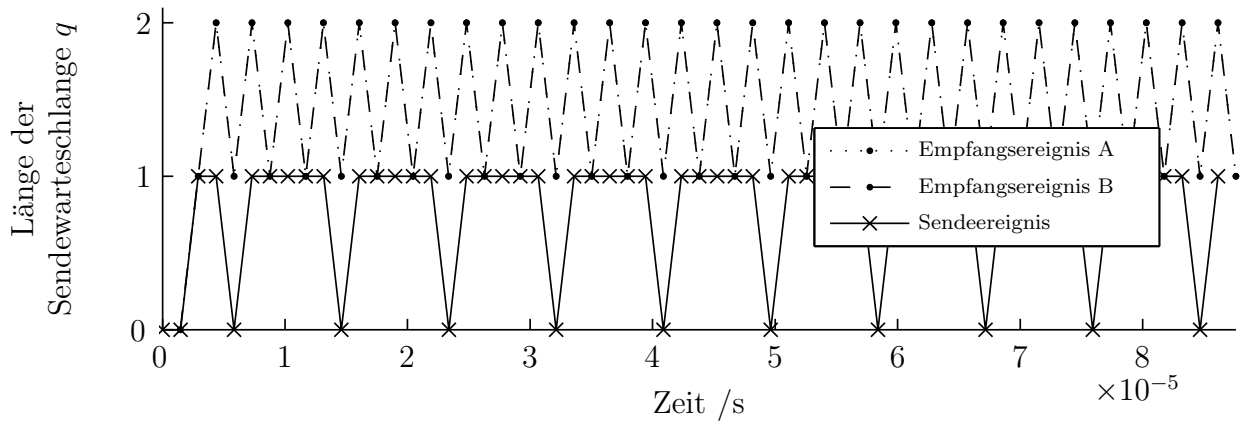


Abbildung B.1: Leitertopologie mit sechs Knoten

B Weitere Simulationsergebnisse



(a) Knoten A



(b) Knoten B

Abbildung B.2: Länge der Sendewarteschlange q über der Zeit

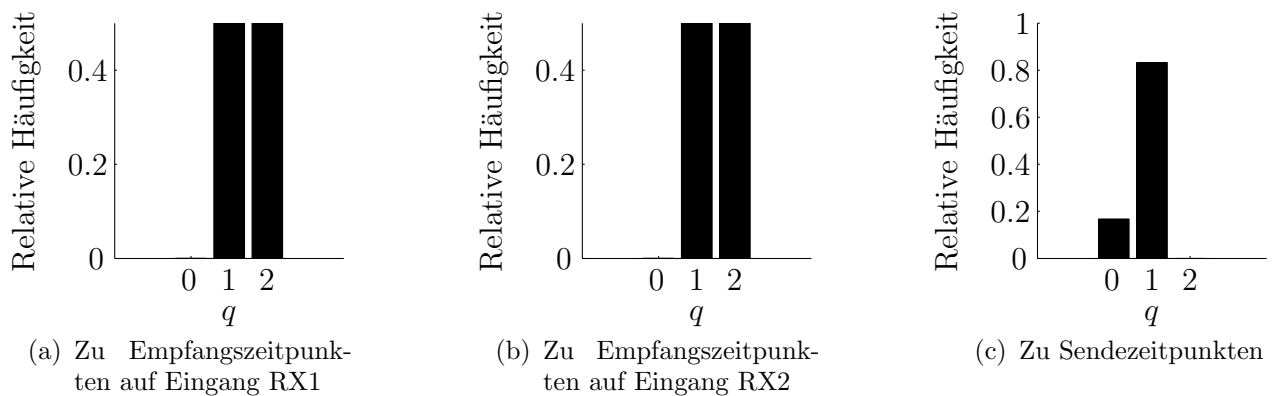


Abbildung B.3: Relative Häufigkeit der Warteschlangen-Längen q in Knoten A

B.1 Leitertopologie mit sechs Knoten

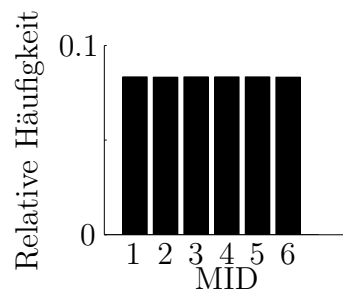


Abbildung B.4: Relative Häufigkeit des Auftretens aller MIDs in Knoten A

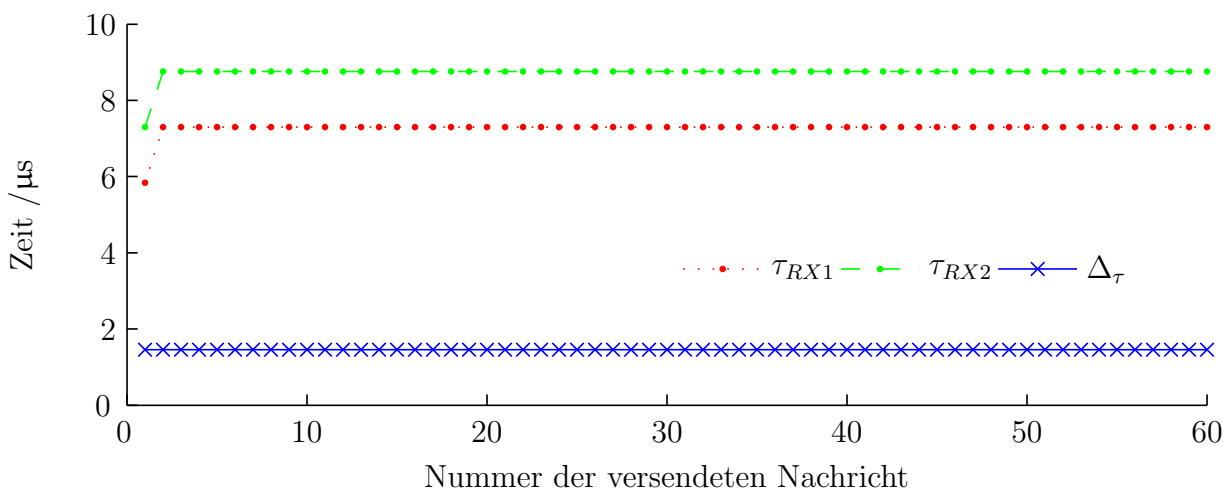


Abbildung B.5: Laufzeiten aller von Knoten A versendeten Nachrichten bis zu ihrem Empfang auf Eingang RX1 und Eingang RX2

B.2 Chordaler Ring mit zwölf Knoten, Skiplänge vier

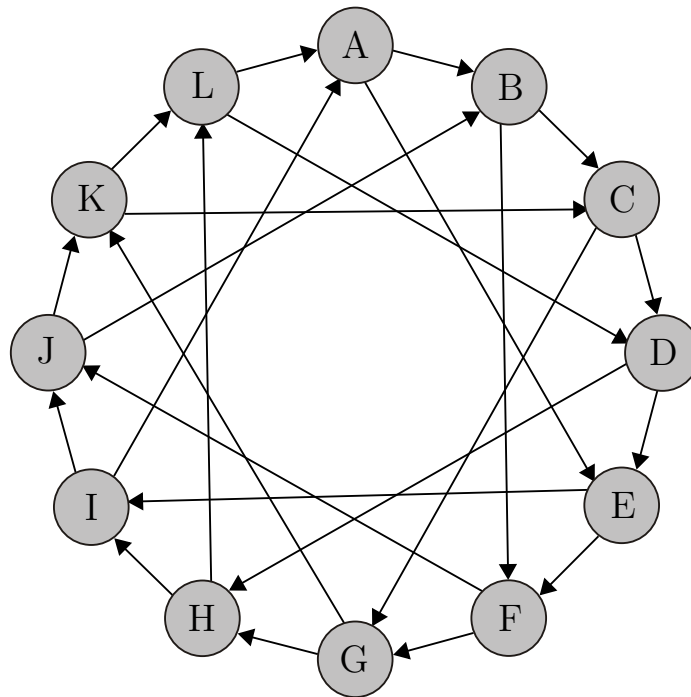
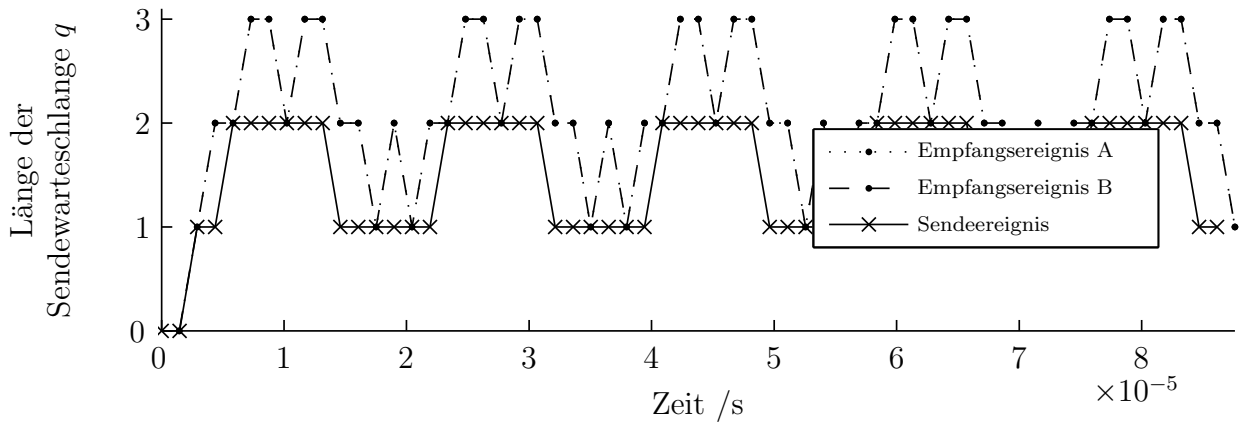
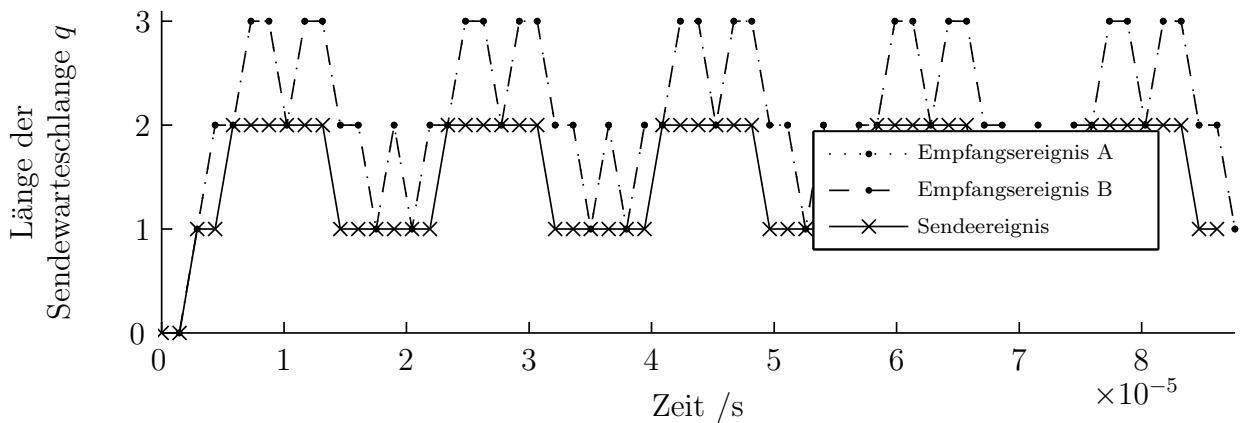


Abbildung B.6: Chordaler Ring mit zwölf Knoten, Skiplänge vier

B.2 Chordaler Ring mit zwölf Knoten, Skiplänge vier



(a) Knoten A



(b) Knoten B

Abbildung B.7: Länge der Sendewarteschlange q über der Zeit

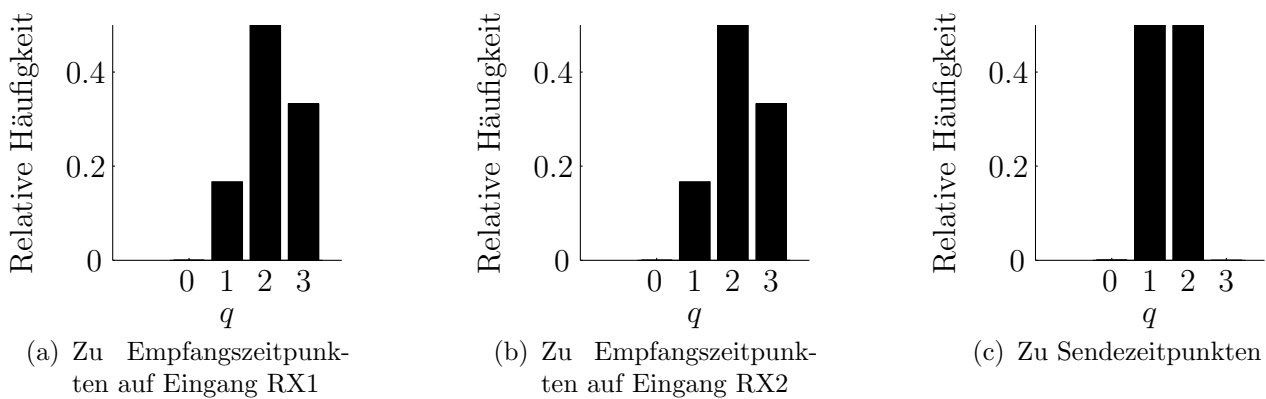


Abbildung B.8: Relative Häufigkeit der Warteschlangen-Längen q in Knoten A

B Weitere Simulationsergebnisse

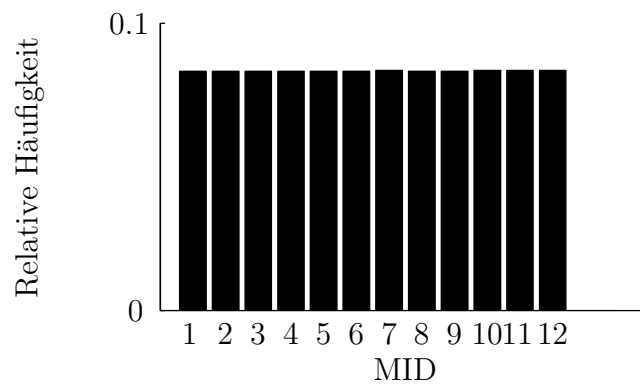


Abbildung B.9: Relative Häufigkeit des Auftretens aller MIDs in Knoten A

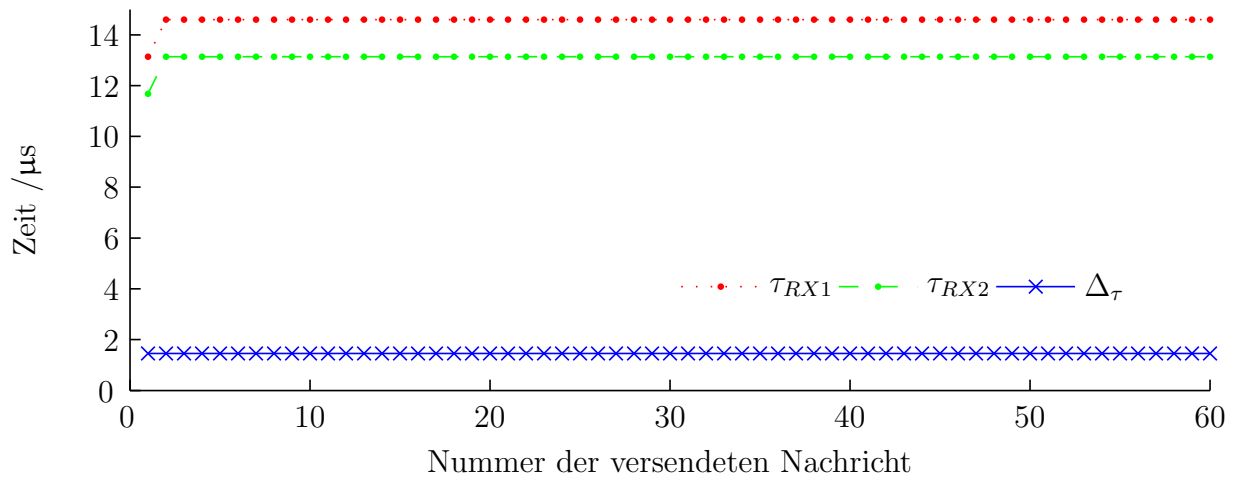


Abbildung B.10: Laufzeiten aller von Knoten A versendeten Nachrichten bis zu ihrem Empfang auf Eingang RX1 und Eingang RX2

B.3 Chordaler Ring mit zwölf Knoten in sechs Paaren

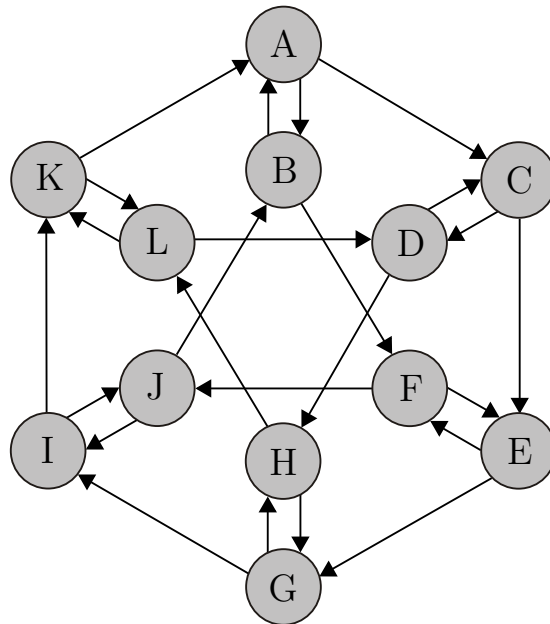
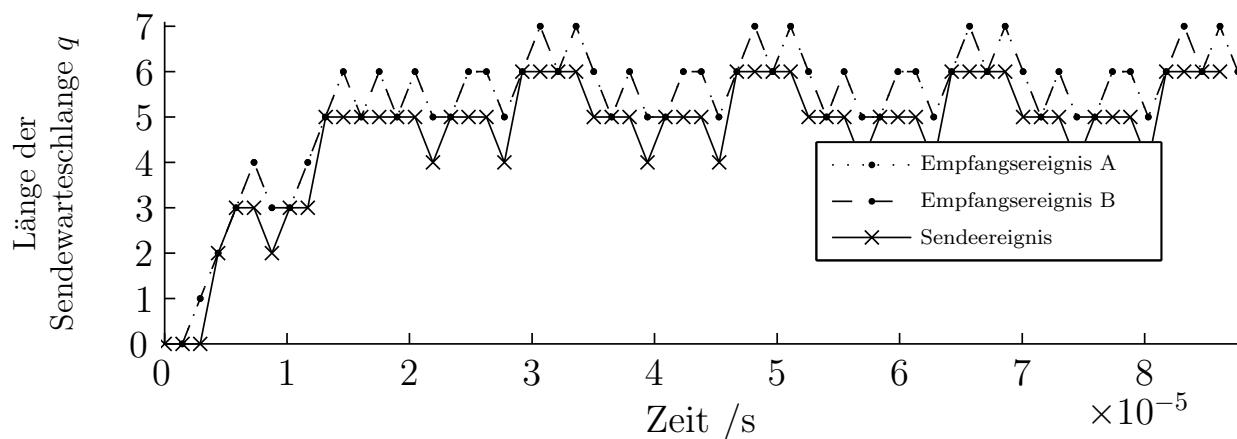
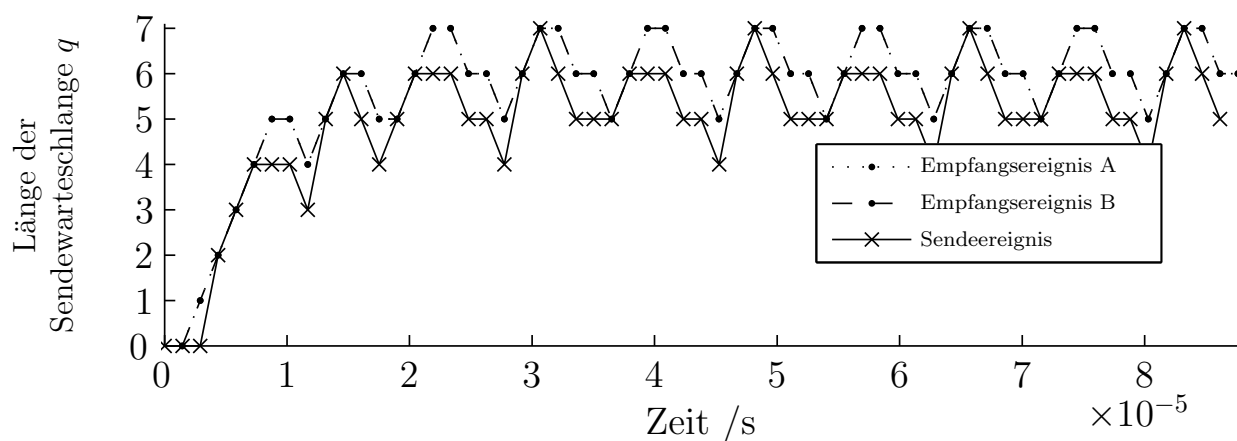


Abbildung B.11: Chordaler Ring mit zwölf Knoten in sechs Paaren

B Weitere Simulationsergebnisse

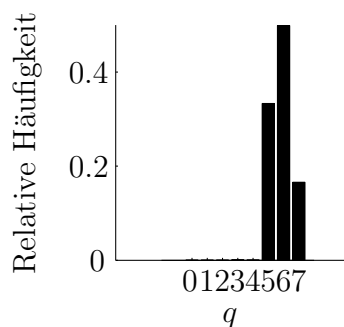


(a) Knoten A

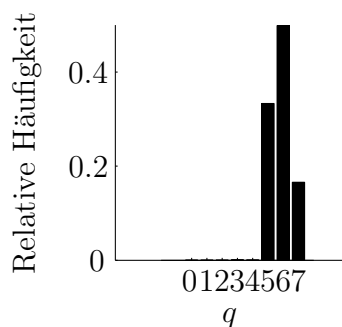


(b) Knoten B

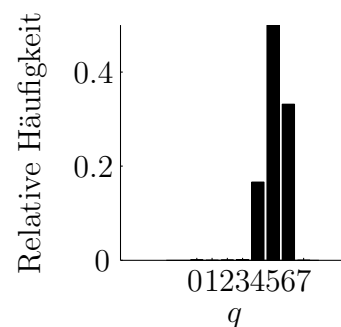
Abbildung B.12: Länge der Sendewarteschlange q über der Zeit



(a) Zu Empfangszeitpunkten auf Eingang RX1



(b) Zu Empfangszeitpunkten auf Eingang RX2



(c) Zu Sendezeitpunkten

Abbildung B.13: Relative Häufigkeit der Warteschlangen-Längen q in Knoten A

B.3 Chordaler Ring mit zwölf Knoten in sechs Paaren

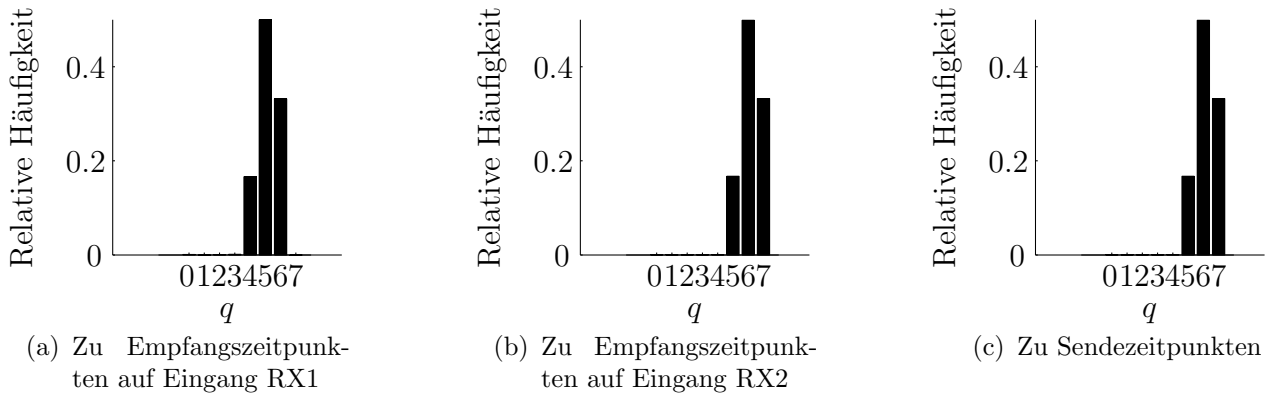


Abbildung B.14: Relative Häufigkeit der Warteschlangen-Längen q in Knoten B

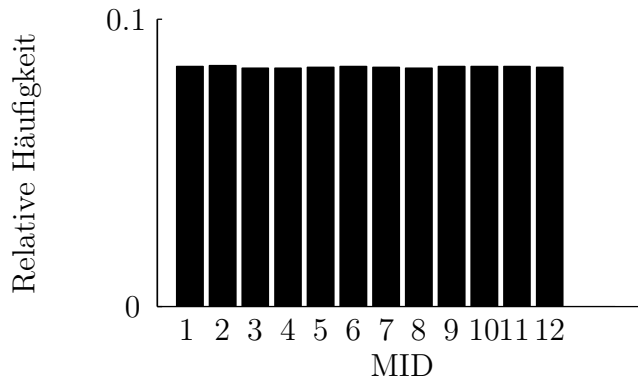


Abbildung B.15: Relative Häufigkeit des Auftretens aller MIDs in Knoten A

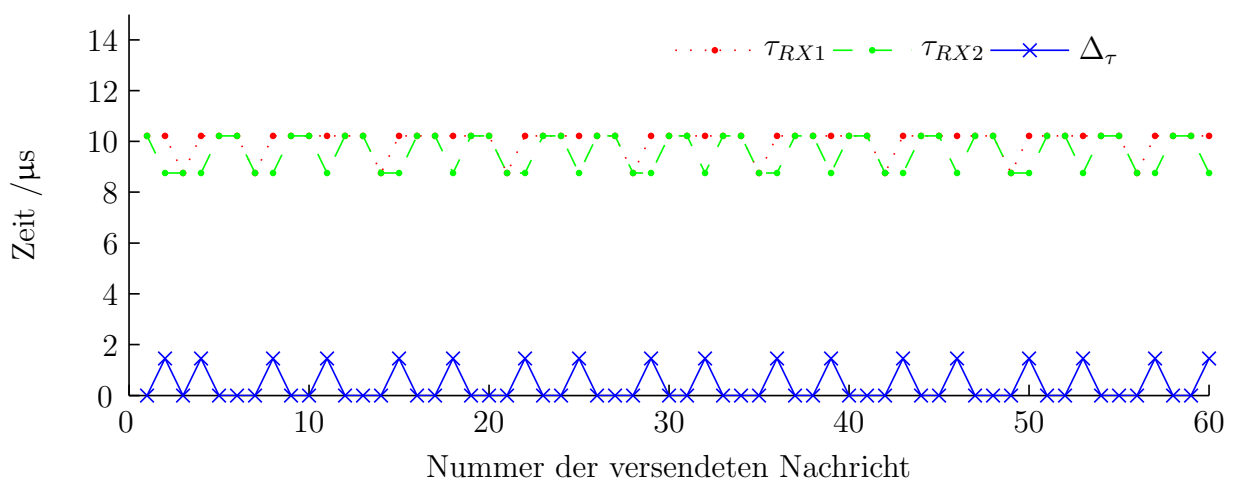


Abbildung B.16: Laufzeiten aller von Knoten A versendeten Nachrichten bis zu ihrem Empfang auf Eingang RX1 und Eingang RX2

B Weitere Simulationsergebnisse

C Nomenklatur

C.1 Formelzeichen

Symbol	Einheit	Bedeutung
a		Sendewort
A		Auslastungsfaktor
$d_H(x, y)$		Hammingdistanz der Worte x und y
d_{min}		Minimale Hammingdistanz eines Codes
e		Fehlerwort
k		Kundenzahl in einer Warteschlange
k_a		Anzahl der in einem Serviceintervall ankommenden Nachrichten
$l_{A,B}$		geometrischer Abstand der Knoten A und B
n_{netto}	Bit	Nettolänge einer Nachricht
$n_{overhead}$	Bit	Overhead, der einer Nachricht beim Versenden durch ein Kommunikationssystem hinzugefügt wird
N		Anzahl der Knoten in einem Netzwerk
$p_s(k_a)$		Marginalwahrscheinlichkeit, dass k_a Nachrichten in einem Serviceintervall ankommen
q_I		Queuelänge in Knoten I
q_{max}		Maximale Queuelänge einer Topologie
x		Anzahl der in einem System vorhandenen Funktionen
y		Empfangswort
\mathcal{C}_I		\mathcal{C} -Membership des Knotens I
\mathcal{M}_I		\mathcal{N} -Membership des Knotens I
\mathcal{N}		Menge aller Knoten in einem Netzwerk
X^+		Menge der direkten downstream Knoten von X
X_α^+		Menge der downstream Knoten von X bezüglich der Nachricht α
X^-		Menge der direkten upstream Knoten von X
X_α^-		Menge der upstream Knoten von X bezüglich der Nachricht α
Δ	s	Zeit, die in SAFEbus zwischen dem Beginn eines windows und dem Sendebeginn des ersten shadows vergehen muss
Δ_τ	s	Zeit zwischen dem ersten und dem zweiten Empfangen einer Nachricht in einem Knoten
η		Effizienz eines Kommunikationssystems
λ	s^{-1}	Ankunftsrate eines Poissonprozessen
λ_{ECU}	h^{-1}	Ausfallrate eines Teilsystems
λ_K	s^{-1}	Rate, mit der der Knoten K eigene Nachrichten generiert

C Nomenklatur

λ_{sys}	h^{-1}	Ausfallrate des Gesamtsystems
$\lambda_{TX,K}$	s^{-1}	Senderate des Knotens K
π_j		Zustandswahrscheinlichkeit für j Nachrichten in einer Warteschlange
$\pi_{k,j}$		Zustandswahrscheinlichkeit für j Nachrichten in k seriell geschalteten Warteschlangen
Π		Zustandsvektor einer Warteschlange
ρ		Auslastung
τ_a	s	Zwischenankunftszeit
τ_d	s	Durchlaufzeit
τ_{max}	s	maximale Laufzeit einer Nachricht
τ_p	s	Durch die maximale Ausbreitungsgeschwindigkeit verursachten Verzögerung der Nachrichten in einem Kommunikationssystem
τ_{RX1}	s	Zeit zwischen Ende des Sendens einer eignen Nachricht und dem Empfang dieser Nachricht auf RX1
τ_{RX2}	s	Zeit zwischen Ende des Sendens einer eignen Nachricht und dem Empfang dieser Nachricht auf RX2
τ_s	s	Servicezeit
τ_w	s	Wartezeit
ξ		normalisierte Geschwindigkeit einer ECU

C.2 Abkürzungen

Abkürzung	Bedeutung
ABS	Antiblockiersystem
AC	Accept Counter
ACE	Actuator Control Electronics
ACK	Acknowledge
AIMS	Airplane Information Management System
API	Application Programming Interface
ARQ	Automatic Repeat Request
ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
AWGN	Additive White Gaussian Noise
BCH	Bose-Chaudhuri-Hocquenghem
BIU	Bus Interface Unit
BIST	Built-in Self-Test
BTL	Bus Transceiver Logic
CA	Collision Avoidance
CAN	Controller Area Network
COTS	Commercial off-the-shelf
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
EMI	Electro-Magnetic Interference
FEC	Forward Error Control
FC	Failure Counter
FPGA	Field Programmable Gate Array
FSU	Fail-Silent Unit
GPL	GNU Public License
HD	Hamming Distance
IFG	Inter Frame Gap
ISO	International Organization for Standardization
LRM	Line Replaceable Module
MEDL	Message Descriptor List
MID	Message Identifier
MMU	Memory Management Unit
MSI	Message Sequence Identifier
PCU	Power Control Units
PFC	Primary Flight Computer
RS	Reed-Solomon
RTR	Remote Transmission
SBC	Sensotronic Brake Control
SCB	Self Checking Busses
SCRAM	Subsystem Control Reconfiguration Analysis and Management
SFF	Safe Failure Fraction
SIL	Safety Integrity Level
SOF	Start of Frame

C Nomenklatur

SOS	Slightly Off Specification
SPOF	Single Point of Failure
STP	Shield Twisted Pair
SYF	Synchronization Flag
TDMA	Time Division Multiple Access
TMR	Triple Modular Redundancy
TTA	Time-Triggered Architecture
TTP/A	Time-Triggered Protocol / Class A
TTP/C	Time-Triggered Protocol / Class C
UID	Unique Identifier
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Literaturverzeichnis

- [AHM04] ALANEN, Jarmo ; HIETIKKO, Marita ; MALM, Timo: Safety of Digital Communications in Machines / VTT Industrial Systems. 2004. – Forschungsbericht
- [ALR01] AVIŽIENIS, Algirdas ; LAPRIE, Jean-Claude ; RANDELL, Brian: Fundamental Concepts of Dependability / UCLA, LAAS-CNRS, University of Newcastle upon Tyne. 2001. – Forschungsbericht
- [ALS95] AVIŽIENIS, Algirdas ; LYU, Michael R. ; SCHÜTZ, Werner: In Search of Effective Diversity: A Six-Language Study of Fault-Tolerant Flight Control Software. In: *Symposium on Fault-Tolerant Computing* Bd. 3, 1995, S. 136–143
- [Alt07] Altera: *Cyclone Device Handbook Version 5.1*. 2007. – Version 5.1
- [ATJ01] AHLSTROM, Kristina ; TORIN, Jan ; JOHANNESSEN, Per: Design Method for Conceptual Design of By-Wire Control: Two Case Studies. In: *IEEE International Conference on Engineering of Complex Computer Systems* (2001)
- [ATZ05] *Die neue S-Klasse von Mercedes-Benz*. ATZ MTZ extra. 2005
- [Bab07] BABIC, Marko: *Analyse und Optimierung der codierten OFDM-Systeme für die schnelle Powerlinekommunikation*, Universität Karlsruhe (TH), Dissertation, 2007
- [Bau06] BAUMANN, Julian: *Einspritzmengenkorrektur in Common-Rail-Systemen mit Hilfe von magnetoelastischen Drucksensoren*, Universität Karlsruhe (TH), Dissertation, 2006
- [BBJ99] BERGER, Thorsten ; BORT, Peter ; JOHN, Dirk: Verteile Systeme im Kraftfahrzeug. In: *it+ti* 5 (1999), S. 7–11
- [BM02] BOUAJJANI, Ahmed ; MERCERON, Agathe: Parametric Verification of a Group Membership Algorithm / LIAFA, Universität Paris. 2002. – Forschungsbericht. Um Beweise erweiterte Version eines in den Proceedings der FTRTFT'02 erschienen Papers

Literaturverzeichnis

- [Bör02] BÖRCSÖK, Josef: IEC&EN 61508 - eine Norm für viele Fälle. In: *atp* 12 (2002), Nr. 44, S. 48–55
- [Bos98] BOSCH: *Autoelektrik Autoelektronik*, 3. Aufl. Vieweg, 1998
- [Bos04] BOSCH: *Sicherheits- und Komfortsysteme*, 3. Aufl. Vieweg, 2004
- [BSMM97] BRONSTEIN, Ilja ; SEMENDJAJEW, Konstantin ; MUSIOL, Gerhard ; MÜHLIG, Heiner: *Taschenbuch der Mathematik*, 3. Aufl. Verlag Harri Deutsch, 1997
- [Col99] COLLINSON, R.P.G.: Fly-by-wire flight control. In: *Computing & Control Engineering Journal* 10 (1999), S. 141–152
- [DHSZ03] DRISCOLL, Kevin ; HALL, Brendan ; SIVENCRONA, Håkan ; ZUMSTEG, Phil: Byzantine Fault Tolerance, from Theory to Reality. In: *Safecomp*, 2003
- [DMS95] DOLEV, Danny ; MALKI, Dalia ; STRONG, Ray: A Framework for Partitionable Membership Service / The Hebrew University of Jerusalem. 1995. – Forschungsbericht
- [EPZ02] ELLIMS, M. ; PARKER, S. ; ZURLO, J.: Design and analysis of a robust real-time engine control network. In: *IEEE Micro* 22 (2002), Nr. 4, S. 20–27
- [ESA96] ESA. *Ariane 501 - Presentation of Inquiry Board Report*. Pressemitteilung Nr 33-1996. 1996
- [Ets94] ETSCHBERGER, Konrad: *CAN*. Carl Hanser Verlag, 1994
- [FDK94] FÖLLINGER, Otto ; DÖRRSCHEIDT, Frank ; KLITTICH, Manfred: *Regelungstechnik*, 8. überarb. Aufl. Hüthig, 1994
- [Fle04] FLEXRAY CONSORTIUM: *FlexRay Communication System Bus Guardian Specification Version 2.0*, 2004
- [Fle05a] FLEXRAY CONSORTIUM: *FlexRay Communication System Electrical Physical Layer Version 2.1*, 2005
- [Fle05b] FLEXRAY CONSORTIUM: *FlexRay Communications System Protocol Specification Version 2.1*, 2005
- [Fri96] FRIEDRICHS, Bernd: *Kanalcodierung*, 1. Aufl. Springer Verlag, 1996

- [Ham02] HAMMETT, Robert: Design by Extrapolation: An Evaluation of Fault Tolerant Avionics. In: *IEEE AESS Systems Magazine* 17 (2002), S. 17–25
- [HD93] HOYME, Kenneth ; DRISCOLL, Kevin: SAFEbus. In: *Aerospace and Avionics Magazine* 3 (1993), Nr. 3, S. 34–39
- [Heb05] HEBERLE, Klaus: Austauschbare Logik. In: *Automotive Electronics* 3 (2005), S. 8–12
- [HS03] HEINISCH, Cornelia ; SIMONS, Martin: Adaptierbare Software-Architektur für den Software-Download in Kfz-Steuergeräten. In: *GI Jahrestagung*, 2003, S. 320–324
- [HSF⁺04] HEINECKE, Harald ; SCHNELLE, Klaus-Peter ; FENNEL, Helmut ; BORTOLAZZI, Jürgen ; LUNDH, Lennart ; LEFLOUR, Jean ; MATÉ, Jean-Luc ; NISHIKAWA, Kenji ; SCHARNHORST, Thomas: AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures. In: *Convergence 2004*, 2004
- [HSH05] HUCK, Thorsten ; SCHIRMER, Jürgen ; HOGENMÜLLER, Thomas: Statistical Evaluation of Impulsive Noise on Vehicular DC-Lines. In: KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 8. Shaker Verlag, 2005, S. 155–162
- [ISO94] ISO 11898. *Austausch digitaler Informationen – Steuergerätenetz (CAN) für schnellen Datenaustausch*. 1994
- [ISO02] ISO 61508. *Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme*. 2002
- [ISS02] ISERMANN, Rolf ; SCHWARZ, Ralf ; STÖLZL, Stefan: Fault Tolerant Drive-by-Wire Systems. In: *IEEE Control Systems Magazine* 22 (2002), S. 64–81
- [Jon01] JONDRAL, Friedrich: *Nachrichtensysteme*, 1. Aufl. J. Schlembach Fachverlag, 2001
- [KDL86] KIENCKE, Uwe ; DAIS, Siegfried ; LITSCHER, Martin: Automotive Serial Controller Area Network. In: *Proceedings of the SAE World Congress*, 1986

Literaturverzeichnis

- [Kie06] KIENCKE, Uwe: *Ereignisdiskrete Systeme*, 2. Aufl. R. Oldenbourg Verlag, 2006
- [KJ98] KIENCKE, Uwe ; JOHN, Dirk: On the way to an international standard for automotive applications-OSEK/VDX. In: *Proceedings of the International Congress on Transportation Electronics (Convergence)*, 1998, S. 95–100
- [KJ05] KIENCKE, Uwe ; JÄKEL, Holger: *Signale und Systeme*, 3. Aufl. R. Oldenbourg Verlag, 2005
- [Kop97] KOPETZ, Hermann ; STANKOVIC, John A. (Hrsg.): *Real-Time Systems Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997
- [Kop98] KOPETZ, Hermann: A Comparison of CAN and TTP / Institut für Technische Informatik, TU Wien. 1998. – Forschungsbericht
- [Kop01] KOPETZ, Hermann: A Comparison of TTP/C and FlexRay / Institut für Technische Informatik, TU Wien. 2001. – Forschungsbericht
- [KPF⁺03] KÖHN, Philip ; PAULY, Axel ; FLECK, Reidar ; PISCHINGER, Martin ; RICHTER, Thorsten ; SCHNABEL, Mark ; BARTZ, Rüdiger ; WACHINGER, Manfred ; SCHOTT, Stefan: Die Aktivlenkung – Das fahrdynamische Lenksystem des neuen 5er. In: *ATZ MTZ extra* 8 (2003), S. 96–105
- [KPS03] KLIMANT, Herbert ; PIOTRASCHKE, Rudi ; SCHÖNFELD, Dagmar: *Informations- und Kodierungstheorie*. Teubner, 2003
- [Lam78] LAMPORT, Leslie: Time, clocks, and the ordering of events in a distributed system. In: *Communications of the ACM* 21 (1978), Nr. 7, S. 558–565
- [Lap92] LAPRIE, Jean-Claude: *Dependability: basic concepts and terminology*. Springer, 1992
- [Law99] LAWRENZ, Wolfhard: *Controller Area Network*, 3. Aufl. Hüthig, 1999
- [LC83] LIN, Shu ; COSTELLO, Daniel J.: *Error control coding*, 1. Aufl. Prentice-Hall, 1983

- [LH02] LEEN, Gabriel ; HEFFERNAN, Donal: Expanding Automotive Electronic Systems. In: *IEEE Computer* 35 (2002), S. 88–93
- [LPS01] LITTLEWOOD, Bev ; POPOV, Peter ; STRINGINI, Lorenzo: Modeling Software Design Diversity - A Review. In: *ACM Computing Surveys* 33 (2001), Nr. 2, S. 177–208
- [LSP82] LAMPORT, Leslie ; SHOSTAK, Robert ; PEASE, Marshall: The Byzantine Generals Problem. In: *ACM Transactions on Programming Languages and Systems* 4 (1982), Nr. 3, S. 382–401
- [Mah00] MAHMOUD, Rachad: *Sicherheits- und Verfügbarkeitsanalyse komplexer Kfz-Systeme*, Universität-Gesamthochschule Siegen, Dissertation, 2000
- [ODKC06] ORTEGA, Lorena D. ; DÖRING, Martin ; KRAFT, Karl H. ; CLAUS, Lothar: Mit 10 Mbit/s über den FlexRay-Bus. In: *Elektronik automotive* 6 (2006), S. 47–51
- [Pfe00] PFEIFER, Holger: Formal Verification of the TTP Group Membership Algorithm. In: BOLOGNESI, Tommaso (Hrsg.) ; LATELLA, Diego (Hrsg.): *Formal Methods for Distributed System Development Proceedings of FORTE XIII / PSTV XX 2000*, Kluwer Academic Publishers, 2000, S. 3–18
- [PK99] POLEDNA, Stefan ; KROISS, Georg: TTP: “Drive by Wire” in greifbarer Nähe. In: *Elektronik* 14 (1999)
- [PS04] PROAKIS, John G. ; SALEHI, Masoud: *Grundlagen der Kommunikationstechnik*, 3. Aufl. Pearson Studium, 2004
- [Ram06] RAMBOW, Thomas: *Design and Optimization of Distributed Real-Time Systems*, Universität Karlsruhe (TH), Dissertation, 2006
- [Rie03] RIETH, Peter E.: Sicherer als ein Flugzeug. In: *Automobil-Elektronik* (2003), S. 16–18
- [RLH05] RYAN, Colin ; LEEN, Donal ; HEFFERNAN, Gabriel: Additional Communication System Services for Safety-Critical Applications. In: *IEE Proceedings - ISSC*, 2005, S. 308–313
- [Roo05] ROOKS, Oliver: *Softwarebasierte Sicherheitsmechanismen in Drive-by-Wire Fahrzeugrechnern*, Universität Karlsruhe (TH), Dissertation, 2005

Literaturverzeichnis

- [Rus03] RUSHBY, John: A Comparison of Bus Architectures for Safety-Critical Embedded Systems / NASA, SRI International. 2003. – Forschungsbericht
- [Sch98] SCHNEIDER, Sandra: *Methodische Entwicklung und Leistungsanalyse der Steuergeräte im Automobil*, Universität Karlsruhe (TH), Dissertation, 1998
- [Sch04] SCHNEIDER, Etienne: *A Middleware Approach for Dynamic Real-Time Software Reconfiguration on Distributed Embedded Systems*, Université Louis Pasteur Strasbourg, Dissertation, 2004
- [SKA04] STRUNK, Elisabeth A. ; KNIGHT, John C. ; AIELLO, M. A.: Distributed Reconfigurable Avionics Architectures. In: *Digital Avionics Systems Conference*, 2004
- [SM97] STANTON, Neville ; MARSDEN, Phil: Drive-by-wire systems: some reflections on the trend to automate the driver role. In: *Journal of Automobile Engineering* 211 (1997), Nr. 4, S. 267–276
- [Spi04] SPIEGELBERG, Gernot: Vom Airbus lernen. In: *Automobil Industrie* 3 (2004), S. 68–73
- [Sta06] STATISTISCHES BUNDESAMT: Verkehr, Verkehrsunfälle / DE-STATIS. 2006. – Forschungsbericht
- [StV77] *Übereinkommen über den Straßenverkehr (StVÜbk) vom 8. November 1968*. Veröffentlicht im BGBl. 1977 II S. 811ff. 1977
- [SV05] SEIFFERT, Ulrich ; VARCHMIN, Jörn-Uwe: Der Nutzen von mehr Fahrzeugelektronik. In: *Automotive Electronics* 3 (2005), S. 46–49
- [Tak93] TAKAGI, Hideki: *Queueing Analysis*. North Holland, 1993
- [Tor03] TORKZADEH, Dara D.: *Echtzeitsimulation der Verbrennung und modellbasierte Reglersynthese am Common-Rail-Dieselmotor*, Universität Karlsruhe (TH), Dissertation, 2003
- [TTT03] TTTech: *Time-Triggered Protocol TTP/C High-Level Specification Document Protocol Version 1.1*. 2003
- [Tur96] TURAU, Volker: *Algorithmische Graphentheorie*. Addison-Wesley, 1996

- [Yeh96] YEH, Y. C.: Triple-Triple Redundant 777 Primary Flight Computer. In: *Aerospace Applications Conference 1* (1996), S. 293–307

Eigene Veröffentlichungen

- [BKNK06] BRUMMUND, Stephan ; KEHL, Natalja ; NENNINGER, Philipp ; KIENCKE, Uwe: ISODATA Clustering for Optimized Software Allocation in Distributed Automotive Electronic Systems. In: *Proceedings of the SAE World Congress, 2006*
- [BNK⁺06] BAUMANN, Julian ; NENNINGER, Philipp ; KIENCKE, Uwe ; SCHLEGEL, Thomas ; FRITSCH, Jürgen: Compensating the Influence of Pressure Waves on Injection Accuracy in Common Rail Systems by Use of a Magneto-Elastic Pressure Sensor. In: *Proceedings of the FISITA World Automotive Congress, 2006*
- [BNSK05] BRUMMUND, Stephan ; NENNINGER, Philipp ; SAXENA, Simmi ; KIENCKE, Uwe: Design of Automotive Complex Electronic Systems Based on SOM Clustering. In: *Proceedings of the World SOM Congress, 2005*
- [NB07] NENNINGER, Philipp ; BAUER, Michael: Algorithms for Error Detection and Error Containment in Network-based Automotive Communication Media. In: KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 10. Shaker Verlag, 2007
- [NBK05] NENNINGER, Philipp ; BRUMMUND, Stephan ; KIENCKE, Uwe: Fault Detection in Distributed Automotive Electronic Systems using Hierarchical Colored Bayesian Petri-Nets. In: *Proceedings of the SAE World Conference, 2005*
- [NBK06] NENNINGER, Philipp ; BRUMMUND, Stephan ; KIENCKE, Uwe: Fault Detection in Distributed Automotive Electronic Systems Using Hierarchical Colored Bayesian Petri-Nets. In: *Journal of Passenger Cars: Electronic and Electrical Systems* 114 (2006), Nr. 7, S. 74–82
- [NBK07a] NENNINGER, Philipp ; BAUER, Michael ; KIENCKE, Uwe: On Reliable Communication and Group Membership in Safety-Relevant Automotive Electronic Systems. In: *Angenommen zu SAE World Congress, 2007*

Eigene Veröffentlichungen

- [NBK07b] NENNINGER, Philipp ; BRUMMUND, Stephan ; KIENCKE, Uwe: On Timing Issues in Network-based Automotive Communication Media for Safety-Relevant Applications. In: *Angenommen zu Proceedings of the 11th European Automotive Congress EAEC*, 2007
- [NBS05] NENNINGER, Philipp ; BRUMMUND, Stephan ; SAXENA, Simmi: Clustering of Control Units in the Design of Complex Electronic Systems Using Self Organising Maps. In: KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 8. Shaker Verlag, 2005, S. 95–105
- [Nen05] NENNINGER, Philipp: System Inherent Backup for the Steering Mechanism in Drive-by-Wire Vehicles. In: KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 8. Shaker Verlag, 2005, S. 107–114
- [Nen06] NENNINGER, Philipp: On Byzantine Fault Tolerance in Automotive Communication Systems. In: KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 9. Shaker Verlag, 2006, S. 141–152
- [NH04] NENNINGER, Philipp ; HIEMER, Marcus: Observability of a Reduced Nonlinear Two-Track Model. In: KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 7. Shaker Verlag, 2004, S. 41–50
- [NK06] NENNINGER, Philipp ; KIENCKE, Uwe: A Contribution towards an Electronic Network for Safety Relevant Automotive Electronic Systems. In: *Proceedings of the FISITA World Automotive Congress*, 2006
- [NK07] NENNINGER, Philipp ; KIENCKE, Uwe: A Model-Based Comparison of Time-Division Multiplex Access and Network Based Communication Systems for Safety-Relevant Automotive Electronic Architectures. In: *Eingereicht zu Fifth IFAC Symposium on Advances in Automotive Control*, 2007
- [NMBK06] NENNINGER, Philipp ; MERZ, Benedikt ; BRUMMUND, Stephan ; KIENCKE, Uwe: A Network Approach to Connecting Safety-Relevant Automotive Electronic Systems. In: *Proceedings of the SAE World Congress*, 2006

- [NMBK07] NENNINGER, Philipp ; MERZ, Benedikt ; BRUMMUND, Stephan ; KIENCKE, Uwe: A Network Approach to Connecting Safety-Relevant Automotive Electronic Systems. In: *Journal of Passenger Cars: Electronic and Electric Systems* (2007)
- [NR05] NENNINGER, Philipp ; RAMBOW, Thomas: An Analysis of the Requirements for Safety Relevant Automotive Systems. In: KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 8. Shaker Verlag, 2005, S. 57–64
- [NRK04a] NENNINGER, Philipp ; RAMBOW, Thomas ; KIENCKE, Uwe: Automatic Model Based Partitioning of Distributed Automotive Electronic Systems. In: *Proceedings of the SAE World Congress*, 2004
- [NRK04b] NENNINGER, Philipp ; ROOKS, Oliver ; KIENCKE, Uwe: A Novel Approach for Dynamic Distribution in Safety Relevant Automotive Systems. In: *Proceedings of the IAR Annual Meeting*, 2004, S. 47–52
- [NRK05a] NENNINGER, Philipp ; RAMBOW, Thomas ; KIENCKE, Uwe: Clustering of Complex Electronic Systems with Self-Ordering Maps. In: *Proceedings of the SAE World Conference*, 2005
- [NRK05b] NENNINGER, Philipp ; ROOKS, Oliver ; KIENCKE, Uwe: Improved System Architecture for Safety-Relevant Systems Using Dynamic Distribution and State Buffering. In: *Proceedings of the IFAC World Congress*, 2005
- [NRK06] NENNINGER, Philipp ; RAMBOW, Thomas ; KIENCKE, Uwe: Clustering of Complex Electronic Systems with Self-Ordering Maps. In: *Journal of Passenger Cars: Electronic and Electric Systems* 114 (2006), Nr. 7, S. 259–264
- [NRR04] NENNINGER, Philipp ; RAMBOW, Thomas ; ROOKS, Oliver: Clustering of Automotive Electronic Systems in n-Dimensional Space Using Self-Organizing Neural Networks. In: KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 7. Shaker Verlag, 2004, S. 117–126
- [RBN04] ROOKS, Oliver ; BARTHLOTT, Jürgen ; NENNINGER, Philipp: Neuro-Fuzzy Modelling in Fault Diagnosis Applications. In:

Betreute Studien- und Diplomarbeiten

- KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 7. Shaker Verlag, 2004, S. 61–72
- [RN04] RAMBOW, Thomas ; NENNINGER, Philipp: Analysis of Preemptive, Periodic Real Time Systems Based on a Timed Event Graph Model. In: KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 7. Shaker Verlag, 2004, S. 127–136
- [RNS⁺05] RAMBOW, Thomas ; NENNINGER, Philipp ; SCHLÖR, Rainer ; BUSCH, Rainer ; SEIBERTZ, Achim: Effective Response Time Analysis for Distributed Real-Time Systems with Communication on a Controller Area Network. In: KIENCKE, Uwe (Hrsg.) ; DOSTERT, Klaus (Hrsg.): *Reports on Industrial Information Technology* Bd. 8. Shaker Verlag, 2005, S. 83–93

Betreute Studien- und Diplomarbeiten

- [Bar04] BARBIER, Sophie: *Konzeption und Implementierung einer optischen Geschwindigkeitsmessung für Schüttgutförderbänder*, Universität Karlsruhe (TH) in Kooperation mit INSA, Strasbourg, Frankreich, Internship Report, 2004
- [Bau06a] BAUER, Michael: *Algorithmen zur Sicherstellung der Datenintegrität in verteilten Kraftfahrzeug-Kommunikationssystemen*, Universität Karlsruhe (TH), Diplomarbeit, 2006
- [Bau06b] BAUER, Michael: *In-Car Schedule and Location Information System with Conversation-Oriented User Interface*, Universität Karlsruhe (TH) in Kooperation mit Denso IT Lab, Tokyo, Japan, Studienarbeit, 2006
- [Bel03] BELTRAND, Nicolas: *Konstruktion eines Schlittens für Crashtestversuche*, Universität Karlsruhe (TH) in Kooperation mit INSA, Strasbourg, Frankreich, Internship Report, 2003
- [Bra05a] BRAUN, Andreas: *Implementierung eines State-Servers auf Basis des C167CR*, Universität Karlsruhe (TH), Studienarbeit, 2005
- [Bra05b] BRAUN, Daniel: *Konzeption und Implementierung einer Ressourcenverwaltung für ein Echtzeit-Betriebssystem in dynamisch ver-*

teilten Kfz-Anwendungen, Universität Karlsruhe (TH), Diplomarbeit, 2005

- [Bra06] BRAUN, Andreas: *Modellierung und Vergleich von netzwerkbasier-ten Kommunikationsmedien*, Universität Karlsruhe (TH), Diplomarbeit, 2006
- [Bru04a] BRUMMUND, Stephan: *Erstellung und Inbetriebnahme einer rapid prototyping Umgebung für Crashtestversuche*, Universität Karlsruhe (TH), Studienarbeit, 2004
- [Bru04b] BRUMMUND, Stephan: *Fehlerfindung in dynamisch verteilten Systemen*, Universität Karlsruhe (TH), Diplomarbeit, 2004
- [Har06] HARTMANN, Bastian: *Vision-based pedestrian detection and estimation with a blind corner camera*, Universität Karlsruhe (TH) in Kooperation mit Denso IT Lab, Tokyo, Japan, Studienarbeit, 2006
- [Hil04] HILLENBRAND, Tobias: *Evaluierung von Fuzzy/Neuro-Methoden zur Verarbeitung digitaler Sensorinformationen*, Universität Karlsruhe (TH) in Kooperation mit Robert Bosch GmbH, Diplomarbeit, 2004
- [Kai03] KAISER, Andreas: *Simplification and Improvement of an Extended Kalman Filter*, Universität Karlsruhe (TH) in Kooperation mit Hitachi America R&D, Detroit, USA, Studienarbeit, 2003
- [Kal03] KALLENBERGER, Christoph: *Modelling neurons in the cochlear nucleus receiving convergent auditory nerve fiber input*, Universität Karlsruhe (TH) in Kooperation mit McMasters University, Kanada, Studienarbeit, 2003
- [Li06] LI, Lei: *Integration eines State Servers in ein Steuergerätenetzwerk*, Universität Karlsruhe (TH), Diplomarbeit, 2006
- [Lu07] LU, Waner: *Erstellung eines Kanalmodells für den automotive baseband Kanal*, Universität Karlsruhe (TH), Studienarbeit, 2007
- [Mer05] MERZ, Benedikt: *Redundanzstrukturen in dynamisch verteilten, sicherheitsrelevanten Kfz-Elektroniksystemen*, Universität Karlsruhe (TH), Diplomarbeit, 2005

Betreute Studien- und Diplomarbeiten

- [Met04] METTETAL, Virginie: *Konzeption und Aufbau eines Versuches zur digitalen Messwertaufnahme*, Universität Karlsruhe (TH) in Kooperation mit INSA, Strasbourg, Frankreich, Internship Report, 2004
- [Rüt06] RÜTTERS, René: *High Frequency Simulation of Telematic Antennas*, Universität Karlsruhe (TH) in Kooperation mit Hitachi America R&D, Detroit, USA, Studienarbeit, 2006
- [Sax05] SAXENA, Simmi: *Clustering of Complex Electronic Systems*, Universität Karlsruhe (TH) in Kooperation mit IIT, Madras, Indien, Masters Thesis, 2005
- [Stö04] STÖLKER, Jürgen: *Embedded Betriebssysteme für das SAPS-RC Board*, Universität Karlsruhe (TH), Studienarbeit, 2004
- [Stö05] STÖLKER, Jürgen: *Fehlersuche in dynamisch verteilten, nicht sicherheitsrelevanten Kfz-Elektroniksystemen mit Hilfe von hierarchical colored Bayesian Petri-Netzen (HCBPN)*, Universität Karlsruhe (TH), Diplomarbeit, 2005
- [Thi04] THIBAUT, Guillaume: *Adaption und Erweiterung eines Praktikumsversuches zur Kalman-Filterung*, Universität Karlsruhe (TH) in Kooperation mit INSA, Strasbourg, Frankreich, Internship Report, 2004
- [Vir04] VIRLOUVET, Jérôme: *Implementation of a Thread-Based Control and Monitoring Application for a CAN-Based ECU Network*, Universität Karlsruhe (TH) in Kooperation mit INSA, Strasbourg, Frankreich, Internship Report, 2004
- [Xu06] XU, Cheng: *Aufbau eines FPGA-basierten proof-of-concept Systems für ein Kommunikationsnetzwerk für automobile Echtzeitanwendungen*, Universität Karlsruhe (TH), Diplomarbeit, 2006
- [Zho06] ZHOU, Wenwang: *Redesign und Erweiterung der Implementierung eines netzwerkbasierenden Kommunikationsmediums*, Universität Karlsruhe (TH), Diplomarbeit, 2006
- [Zim05] ZIMMERMANN, Bernd: *Modellierung einer systeminhärenten Rückfallebene für Drive-by-Wire Systeme*, Universität Karlsruhe (TH), Studienarbeit, 2005

- [Zim06] ZIMMERMANN, Bernd: *Analyse von forward error correction Algorithmen zur Verwendung in Kraftfahrzeug-Elektroniknetzwerken*, Universität Karlsruhe (TH), Diplomarbeit, 2006

Betreute Studien- und Diplomarbeiten

Lebenslauf

Persönliche Daten

Name Philipp Nenninger
Geburtsdatum 17.02.1976
Geburtsort Lahr/Schwarzwald

Schulbildung

1982 – 1986 Grundschule Griesheim
1986 – 1992 Oken-Gymnasium Offenburg
1992 – 1993 Troy High School, Troy, TX, USA
1993 – 1996 Oken-Gymanasium Offenburg
1996 Abitur

Zivildienst

07.1996 – 07.1997 Spastikerverein Offenburg

Studium und Beruf

10.1997 – 02.2003 Studium der Elektrotechnik und Informationstechnik an der Universität Karlsruhe (TH), Vertiefungsrichtung Industrielle Informationstechnik

seit 03.2003 Wissenschaftlicher Angestellter am Institut für industrielle Informationstechnik (IIIT) der Universität Karlsruhe (TH)

ISBN: 978-3-86644-134-7

www.uvka.de